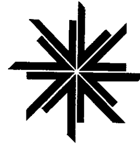


# Control Data®
# 7600 Computer System

## Preliminary Reference Manual

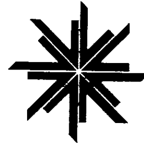
# INSTRUCTION INDEX


CENTRAL PROCESSOR

| Code | Instruction | Page |
|---|---|---|
| 00000 | Error exit to EEA | 3-2 |
| 0100K | Return jump to K | 3-30 |
| 011jK | Block copy K + (Bj) words from LCM to SCM | 3-39 |
| 012jK | Block copy K + (Bj) words from SCM to LCM | 3-36 |
| 013jk | Exchange exit to NEA if exit flag clear | 3-31 |
| 013jK | Exchange exit to K + (Bj) if exit flag set | 3-32 |
| 014jk | Read LCM at (Xk) to Xj | 3-42 |
| 015jk | Write (Xj) into LCM at (Xk) | 3-44 |
| 0160k | Reset channel (Bk) input buffer if j=0 | 3-45 |
| 016jk | Read channel (Bk) input status to Bj if j≠0 | 3-46 |
| 0170k | Reset channel (Bk) output buffer if j=0 | 3-48 |
| 017jk | Read channel (Bk) output status to Bj if j≠0 | 3-49 |
| 02i0k | Jump to K + (Bi) | 3-33 |
| 030jK | Branch to K if (Xj) = 0 | 3-34 |
| 031jK | Branch to K if (Xj) ≠0 | 3-34 |
| 032jK | Branch to K if (Xj) positive | 3-34 |
| 033jK | Branch to K if (Xj) negative | 3-34 |
| 034jK | Branch to K if (Xj) in range | 3-34 |
| 035jK | Branch to K if (Xj) not in range | 3-34 |
| 036jK | Branch to K if (Xj) definite | 3-34 |
| 037jK | Branch to K if (Xj) indefinite | 3-34 |
| 04ijK | Branch to K if (Bi) = (Bj) | 3-35 |
| 05ijK | Branch to K if (Bi) ≠(Bj) | 3-35 |
| 06ijK | Branch to K if (Bi) > (Bj) | 3-35 |
| 07ijK | Branch to K if (Bi) < (Bj) | 3-35 |
| 10ij0 | Copy (Xj) to Xi | 3-8 |
| 11ijk | Logical product of (Xj) and (Xk) to Xi | 3-9 |
| 12ijk | Logical sum of (Xj) plus (Xk) to Xi | 3-9 |
| 13ijk | Logical difference of (Xj) minus (Xk) to Xi | 3-10 |
| 14i0k | Copy complement of (Xk) to Xi | 3-10 |
| 15ijk | Logical product of (Xj) and comp (Xk) to Xi | 3-11 |
| 16ijk | Logical sum (Xj) plus comp (Xk) to Xi | 3-11 |
| 17ijk | Logical difference of (Xj) minus comp (Xk) to Xi | 3-12 |
| 20ijk | Left shift (Xi) by jk | 3-12 |
| 21ijk | Right shift (Xi) by jk | 3-13 |
| 22ijk | Left shift (Xk) by (Bj) to Xi | 3-14 |
| 23ijk | Right shift (Xk) by (Bj) to Xi | 3-14 |
| 24ijk | Normalize (Xk) to Xi and Bj | 3-15 |
| 25ijk | Round and normalize (Xk) to Xi and Bj | 3-16 |
| 26ijk | Unpack (Xk) to Xi and Bj | 3-17 |
| 27ijk | Pack (Xk) and (Bj) to Xi | 3-18 |
| 30ijk | Floating sum of (Xj) plus (Xk) to Xi | 3-19 |
| 31ijk | Floating difference of (Xj) minus (Xk) to Xi | 3-20 |
| 32ijk | Floating DP sum of (Xj) plus (Xk) to Xi | 3-21 |
| 33ijk | Floating DP difference of (Xj) minus (Xk) to Xi | 3-22 |
| 34ijk | Round floating sum of (Xj) plus (Xk) to Xi | 3-22 |
| 35ijk | Round floating difference of (Xj) minus (Xk) to Xi | 3-23 |
| 36ijk | Integer sum of (Xj) plus (Xk) to Xi | 3-7 |
| 37ijk | Integer difference of (Xj) minus (Xk) to Xi | 3-7 |
| 40ijk | Floating product of (Xj) times (Xk) to Xi | 3-24 |
| 41ijk | Round floating product of (Xj) times (Xk) to Xi | 3-26 |
| 42ijk | Floating DP product of (Xj) times (Xk) to Xi | 3-27 |
| 43ijk | Form mask of jk bits to Xi | 3-18 |
| 44ijk | Floating divide (Xj) by (Xk) to Xi | 3-27 |
| 45ijk | Round floating divide (Xj) by (Xk) to Xi | 3-29 |
| 46000 | Pass | 3-3 |
| 47i0k | Population count of (Xk) to Xi | 3-8 |
| 50ijK | Increment (Aj) plus K to Ai | 3-4 |
| 51ijK | Increment (Bj) plus K to Ai | 3-4 |
| 52ijK | Increment (Xj) plus K to Ai | 3-4 |
| 53ijk | Increment (Xj) plus (Bk) to Ai | 3-4 |
| 54ijk | Increment (Aj) plus (Bk) to Ai | 3-4 |
| 55ijk | Increment (Aj) minus (Bk) to Ai | 3-4 |
| 56ijk | Increment (Bj) plus (Bk) to Ai | 3-4 |
| 57ijk | Increment (Bj) minus (Bk) to Ai | 3-4 |
| 60ijK | Increment (Aj) plus K to Bi | 3-5 |
| 61ijK | Increment (Bj) plus K to Bi | 3-5 |
| 62ijK | Increment (Xj) plus K to Bi | 3-5 |
| 63ijk | Increment (Xj) plus (Bk) to Bi | 3-5 |
| 64ijk | Increment (Aj) plus (Bk) to Bi | 3-5 |
| 65ijk | Increment (Aj) minus (Bk) to Bi | 3-5 |
| 66ijk | Increment (Bj) plus (Bk) to Bi | 3-5 |
| 67ijk | Increment (Bj) minus (Bk) to Bi | 3-5 |
| 70ijK | Increment (Aj) plus K to Xi | 3-6 |
| 71ijK | Increment (Bj) plus K to Xi | 3-6 |
| 72ijK | Increment (Xj) plus K to Xi | 3-6 |
| 73ijk | Increment (Xj) plus (Bk) to Xi | 3-6 |
| 74ijk | Increment (Aj) plus (Bk) to Xi | 3-6 |
| 75ijk | Increment (Aj) minus (Bk) to Xi | 3-6 |
| 76ijk | Increment (Bj) plus (Bk) to Xi | 3-6 |
| 77ijk | Increment (Bj) minus (Bk) to Xi | 3-6 |

PERIPHERAL PROCESSORS

| Code | Instruction | Page |
|---|---|---|
| 00 | Error stop | 6-5 |
| 01 | Long jump to m + (d) | 6-19 |
| 02 | Return jump to m + (d) | 6-19 |
| 03 | Unconditional jump d | 6-17 |
| 04 | Zero jump d | 6-17 |
| 05 | Nonzero jump d | 6-18 |
| 06 | Positive jump d | 6-18 |
| 07 | Negative jump d | 6-19 |
| 10 | Shift d | 6-11 |
| 11 | Logical difference d | 6-11 |
| 12 | Logical product d | 6-12 |
| 13 | Selective clear d | 6-12 |
| 14 | Load d | 6-5 |
| 15 | Load complement d | 6-5 |
| 16 | Add d | 6-8 |
| 17 | Subtract d | 6-8 |
| 20 | Load dm | 6-7 |
| 21 | Add dm | 6-10 |
| 22 | Logical product dm | 6-13 |
| 23 | Logical difference dm | 6-13 |
| 24 | Pass | 6-5 |
| 25 | Pass | 6-5 |
| 26 | Pass | 6-5 |
| 27 | Pass | 6-5 |
| 30 | Load (d) | 6-6 |
| 31 | Add (d) | 6-9 |
| 32 | Subtract (d) | 6-9 |
| 33 | Logical difference (d) | 6-12 |
| 34 | Store (d) | 6-6 |
| 35 | Replace add (d) | 6-14 |
| 36 | Replace add one (d) | 6-14 |
| 37 | Replace subtract one (d) | 6-15 |
| 40 | Load ((d)) | 6-6 |
| 41 | Add ((d)) | 6-9 |
| 42 | Subtract ((d)) | 6-10 |
| 43 | Logical difference ((d)) | 6-13 |
| 44 | Store ((d)) | 6-7 |
| 45 | Replace add ((d)) | 6-15 |
| 46 | Replace add one ((d)) | 6-16 |
| 47 | Replace subtract one ((d)) | 6-16 |
| 50 | Load (m+(d)) | 6-7 |
| 51 | Add (m+(d)) | 6-10 |
| 52 | Subtract (m+(d)) | 6-11 |
| 53 | Logical difference (m+(d)) | 6-14 |
| 54 | Store (m+(d)) | 6-8 |
| 55 | Replace add (m+(d)) | 6-16 |
| 56 | Replace add one (m+(d)) | 6-16 |
| 57 | Replace subtract one (m+(d)) | 6-17 |
| 60 | Jump on input word flag | 6-20 |
| 61 | Jump if no input word flag | 6-20 |
| 62 | Jump on input record flag | 6-20 |
| 63 | Jump if no input record flag | 6-21 |
| 64 | Jump on output word flag | 6-20 |
| 65 | Jump if no output word flag | 6-20 |
| 66 | Jump on output record flag | 6-20 |
| 67 | Jump if no output record flag | 6-21 |
| 70 | Input to A from channel d | 6-21 |
| 71 | Input (A) words to m from channel d | 6-22 |
| 72 | Output from A on channel d | 6-21 |
| 73 | Output (A) words from m on channel d | 6-24 |
| 74 | Output record flag on channel d | 6-26 |
| 75 | Pass | 6-5 |
| 76 | Pass | 6-5 |
| 77 | Error stop | 6-5 |



# Control Data®
# 7600 Computer System

## Preliminary Reference Manual

# RECORD of REVISIONS

| REVISION | NOTES |
|---|---|
| 01 | Preliminary edition. |
| (11-1-68) | |
| 02 | Manual revised; Engineering Change Order 21882, publications change only. Pages iii, 1-4, 1-6, |
| (3-21-69) | 1-7, 2-2, 2-3, 2-4, 2-5, 2-6, 2-14, 2-19, 3-2, 3-4, 3-5, 3-7, 3-12, 3-13, 3-15, 3-25, 3-32, 3-33, 3-35, |
| | 3-36, 3-37, 3-43, 3-45, 3-46, 4-1, 4-2, 4-3, 4-4, 4-7, 4-8, 4-10, 4-11, 5-1, 6-1, 6-17, 7-2, 7-4, A-1, A-2 |
| | A-3, A-4, A-6, A-7, B-6 and B-8 revised. Delete page 7-5. |

FORM CA230 REV. 1-67

Pub No. 60258200
© 1968, 1969
by Control Data Corporation
Printed in United States of America

Address comments concerning this manual to:

Control Data Corporation
Technical Publications Department
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

# CONTENTS


1. SYSTEM DESCRIPTION

| | |
|---|---|
| Introduction | 1-1 |
| System Characteristics | 1-1 |
| Central Processor Unit Characteristics | 1-1 |
| Peripheral Processor Unit Characteristics | 1-3 |
| Basic System Description | 1-3 |
| Central Processor Unit (CPU) | 1-3 |
| Peripheral Processor Unit (PPU) | 1-5 |
| Maintenance Control Unit (MCU) | 1-6 |
| System Communication | 1-6 |

2. CENTRAL PROCESSOR UNIT

| | |
|---|---|
| Computation Section | 2-1 |
| Operating Registers | 2-1 |
| CPU Instruction Formats | 2-4 |
| Instruction Word Stack | 2-6 |
| Functional Units | 2-8 |
| Exchange Jump | 2-10 |
| Program Status Designations | 2-14 |

3. CENTRAL PROCESSOR INSTRUCTIONS

| | |
|---|---|
| Introduction | 3-1 |
| Error Exit and No Operation | 3-2 |
| Increment | 3-4 |
| Fixed Point Arithmetic | 3-7 |
| Logical | 3-8 |
| Shift | 3-12 |
| Floating Point Arithmetic | 3-19 |
| Branch | 3-30 |
| Input/Output | 3-36 |

4. CENTRAL PROCESSOR MEMORY

| | |
|---|---|
| Introduction | 1-1 |
| Memory Protection | 1-1 |
| Small Core Memory | 4-2 |
| Organization | 4-2 |
| Address Format | 4-3 |
| Small Core Memory Access | 4-3 |
| Memory Reference | 4-4 |
| Central Processor Unit Access | 4-5 |
| I/O Multiplexer | 4-5 |
| Large Core Memory | 4-10 |
| Organization | 4-10 |
| Large Core Memory Access | 4-10 |

5. PERIPHERAL PROCESSOR UNIT

| | |
|---|---|
| Organization | 5-1 |
| Processor | 5-1 |
| Memory | 5-3 |
| Input/Output | 5-3 |

6. PERIPHERAL PROCESSOR INSTRUCTIONS

| | |
|---|---|
| Instruction Formats | 6-1 |
| Address Modes | 6-1 |
| No Address | 6-1 |
| Direct Address | 6-2 |
| Indirect Address | 6-2 |
| Description of Peripheral Instructions | 6-4 |
| Error Stop | 6-5 |
| No Operation | 6-5 |

| | | | |
|---|---|---|---|
| Data Transmission | 6-5 | Introduction | 7-1 |
| Arithmetic | 6-8 | Maintenance Control Unit | 7-1 |
| Shift | 6-11 | PPU Dead Start | 7-1 |
| Logical | 6-11 | Dead Dump | 7-3 |
| Replace | 6-14 | CPU Dead Start | 7-3 |
| Branch | 6-17 | Parity Error Register | 7-4 |
| Input/Output | 6-20 | Program Error | 7-4 |
| | | Console | 7-4 |
| 7. MANUAL CONTROL | | Keyboard Input | 7-4 |

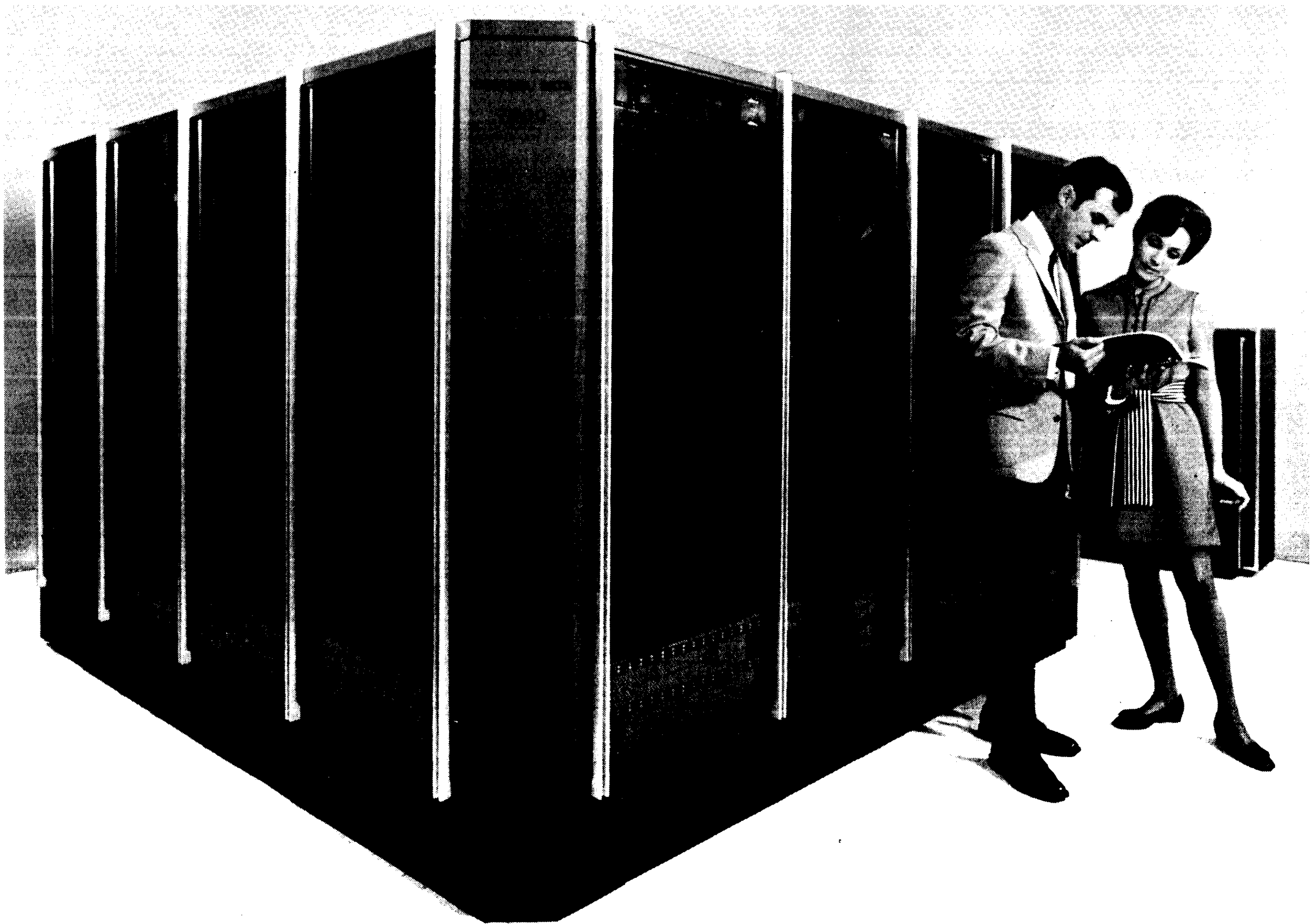
Appendix A Timing Notes
Appendix B Floating Point Arithmetic
Appendix C Mnemonic Codes

## FIGURES


1-1 System Communication Paths 1-7
2-1 CPU Information Flow 2-2
2-2 Parcel Instruction Arrangements 2-5
2-3 Exchange Package 2-11
2-4 Flag and Register Arrangement 2-15
4-1 Memory Map 4-2
4-2 SCM Address Format 4-3
4-3 I/O Exchange Package Areas 4-6
4-4 Buffer Area Arrangements 4-7
5-1 Signal for One PPU Channel (fully Duplexed) 5-5
5-2 Controller/PPU Communications 5-8
6-1 71 Flow Chart 6-23
6-2 73 Flow Chart 6-25
7-1 MCU Configuration 7-2
7-2 Display Console 7-5


## TABLES


3-1 Central Processor Instruction Designators 3-1
6-1 Addressing Modes for Peripheral Processor Instructions 6-3
6-2 Peripheral Processor Instruction Designators 6-4



CONTROL DATA 7600 COMPUTER

# 1. SYSTEM DESCRIPTION

## INTRODUCTION

The CONTROL DATA® 7600 computing system is a large-scale solid-state, general purpose digital computing system. The 7600 is the result of a development program to provide computing capacity significantly beyond that of the 6000 systems. The advanced design techniques incorporated in this system provide for extremely fast and efficient solutions for large scale, general purpose processing.

The 7600 system comprises a Central Processor Unit (CPU) and a number of Peripheral Processor Units (PPU). Some of the PPU are physically located with the CPU and others may be remotely located. The PPU provides a communication and message switching function between the CPU and individual peripheral equipment. Each PPU may have a number of high speed data links to individual peripheral equipment as well as a data link to the CPU.

## SYSTEM CHARACTERISTICS

### Central Processor Unit Characteristics

Computation Section

- 60-bit internal word
- binary computation in fixed point and floating point format
- nine independent functional units

| | |
|---|---|
| Floating Multiply | Boolean |
| Floating Divide | Shift |
| Floating Add | Increment |
| Long Add | Population Count |
| Normalize | |

- twelve-word instruction stack
- synchronous internal logic with 27.5 nanosecond clock period

Small Core Memory

- 65,536 of 60-bit words of coincident current memory with five parity bits per 60-bit word
- organized into 32 independent banks (2,048 words per bank)
- 275 nanosecond read/write cycle time
- 27.5 nanosecond per word maximum transfer rate

Large Core Memory

- 512,000 60-bit words of linear select memory with four parity bits per 60-bit word
- organized into 8 independent banks (64,000 words per bank)
- 1,760 nanosecond read/write cycle time
- 8 words read simultaneously each reference
- 27.5 nanosecond per word maximum transfer rate

Multiplexer

- 15 independent 12-bit channels
- each channel fully duplex
- fixed SCM buffer areas for each channel; normally 128 words
- two clock periods per 60-bit word maximum transfer rate between SCM and the Multiplexer

## Peripheral Processor Unit Characteristics

Computation Section

- 12-bit internal word
- binary computation in fixed-point
- synchronous internal logic with 27.5 nanosecond clock period

Core Memory

- 4,096 12-bit words of coincident current memory with a parity bit for each 12-bit word
- organized into 2 independent banks (2,048 words per bank)
- 275 nanosecond read/write cycle time

Input/Output Section

- 8 independent channels (asynchronous)
- each channel fully duplex (12-bit)
- nine clock periods per 12-bit word maximum transfer rate

# BASIC SYSTEM DESCRIPTION

The 7600 mainframe includes a Central Processor Unit (with its associated memory), a Maintenance Control Unit, and may contain up to 13 Peripheral Processor Units. Additional PPU's may be mounted externally. Overall system operation depends on the integral operation of these elements. Following are brief descriptions of the system elements; detailed descriptions appear in subsequent chapters.

## Central Processor Unit (CPU)

The CPU is a single integrated processing unit. It consists of a computation section, small core memory, large core memory and an input/output multiplexer. These sections are all contained in one main frame cabinet and operate in a tightly synchronous mode. Communication outside the main frame cabinet is asynchronous.

## Computation Section

The computation section of the CPU contains 9 functional units, and 24 operating registers. These units work together to execute a CPU program. Data moves into, and out of, the computation section of the CPU through the operating registers.

## Core Memory

The CPU contains three types of internal memory arranged in a hierarchy of speed and size.

1. The instruction stack which is a register memory of 12, 60-bit words holds instructions. Small program loops can be contained within this stack thus avoiding memory references.
2. The Small Core Memory (SCM) containing 65,536 60-bit words arranged in 32 banks of 2,048 60-bit words each. Each bank is phased; that is, consecutive addresses go to different banks giving a marked decrease in memory conflicts and allows overlapping of memory cycles. Each bank is made up of ten stacks. Each stack contains 1,024 12-bit words plus one parity bit per 12-bit word. These stacks are identical with the Peripheral Processing Unit memory. Either instructions or data may be held in SCM.
3. The Large Core Memory (LCM) contains 500K of 60-bit words arranged in eight phased banks. This memory is a linear select memory with one parity bit for each 15 bits. The LCM words contain eight 60-bit words and is designed for rapid transfer of blocks of data. However, individual 60-bit words may be accessed. Instructions cannot be executed directly from LCM.

The SCM performs certain basic functions in system operation which the LCM cannot effectively perform. These functions are essentially those requiring rapid random access to unrelated fields of data. The first 4K addresses in SCM are reserved for the input/output control and data transfer to service the communication channels to the PPU. CPU object programs do not have access to these areas. The remainder of the SCM may be divided between fields of CPU program code and fields of data for the currently executing program. A small portion will contain a resident monitor program.

Input/Output Multiplexer

The CP input/output Multiplexer (MUX) includes the mechanism to buffer data to (or from) a PPU that is directly connected to the CPU. The PPU communicates with the CPU over a 12-bit fully duplex channel. There are a total of 15 such channels in the MUX. Each channel has assembly/disassembly registers to convert 12-bit channel data to 60-bit CP words (and vice versa). The function of the MUX is to deliver these 60-bit words to SCM for incoming data, read 60-bit words from SCM for outgoing data, and provide the capability to interrupt the CP program for monitor action on the buffer data.

Each channel has a SCM buffer area for incoming data and a separate buffer area for outgoing data. Each channel also has separate exchange packages for incoming and outgoing data. The MUX exchange package areas and the buffer areas are permanently assigned in the lowest order addresses of SCM. The buffer areas may be changed both in size and order (by a wiring change) to accommodate various types of channel volume.

## Peripheral Processor Unit (PPU)

The Peripheral Processor Units (PPU) are separate and independent computers, some of which may reside in the main frame cabinet. Others may be remotely located. The PPU may be connected to SCM, another PPU, a peripheral device or a mix of these. Each PPU has a computation section which performs binary computation in fixed point arithmetic. A PP memory provides storage for 4, 096 12-bit words. This storage is arranged in two independent banks, each with a cycle time of 275 nanoseconds. The two stacks used in a bank contain 1024 12-bit words each. These same stacks are used in groups of five (60 bits) to form SCM.

The PP instruction set combined with the high speed memory and channel flexibility enables a PPU to drive many types of peripherals without the necessity of an intermediate controller. There are eight input data paths and eight output paths connecting the PPU to other devices. The PP input/output facility provides a flexible arrangement for very high speed communication with a variety of I/O devices. The duplex channels allow additional Peripheral Processor Units to be added to the system by linking PPU to PPU.

## Maintenance Control Unit (MCU)

The Maintenance Control Unit is a mainframe PPU with specially connected I/O channels. It has the capability of selecting any PPU's channel which is connected to the scanner, and dead starting this PPU. It can write into any part of SCM by specifying the SCM address. It can dead start CPU by entering a program into SCM and initiating an exchange jump to start execution.

With these capabilities it may perform system initialization and basic recovery of the system. The MCU also serves as a maintenance station for describing and monitoring system maintenance activity.

# SYSTEM COMMUNICATION

System communication paths are illustrated in Figure 1-1. All input data enters and leaves the system via peripheral equipment. The PPU serves to gather input data from the peripheral equipment controllers for delivery to the CPU for processing, and distribute processed data to the output devices. Communication between the PPU and the I/O devices is generally limited by the rate at which the equipment controller can deliver or accept data.

Communication between the PPU and the CPU is over a channel identical to that used for communication between the PPU and peripheral equipment. All 15 CP I/O channels may be in operation at the same time. Data may be sent to or from the CPU on long records. These records can exceed the size of the SCM buffer area which is filled and emptied in a circular mode. This is done by interrupts which initiate a CPU program that can empty or fill the buffer area some fifty times faster than a PPU.

Example:

The PPU starts filling the buffer area at its lowest address and continues entering words until the midpoint of the buffer is reached. This causes an interrupt to a CPU program which empties the lower half of the buffer. Meanwhile the PPU continues filling the buffer. At the end of the buffer another interrupt occurs to initiate the CPU program. Meanwhile, the PPU starts to refill the buffer at the lower address again.

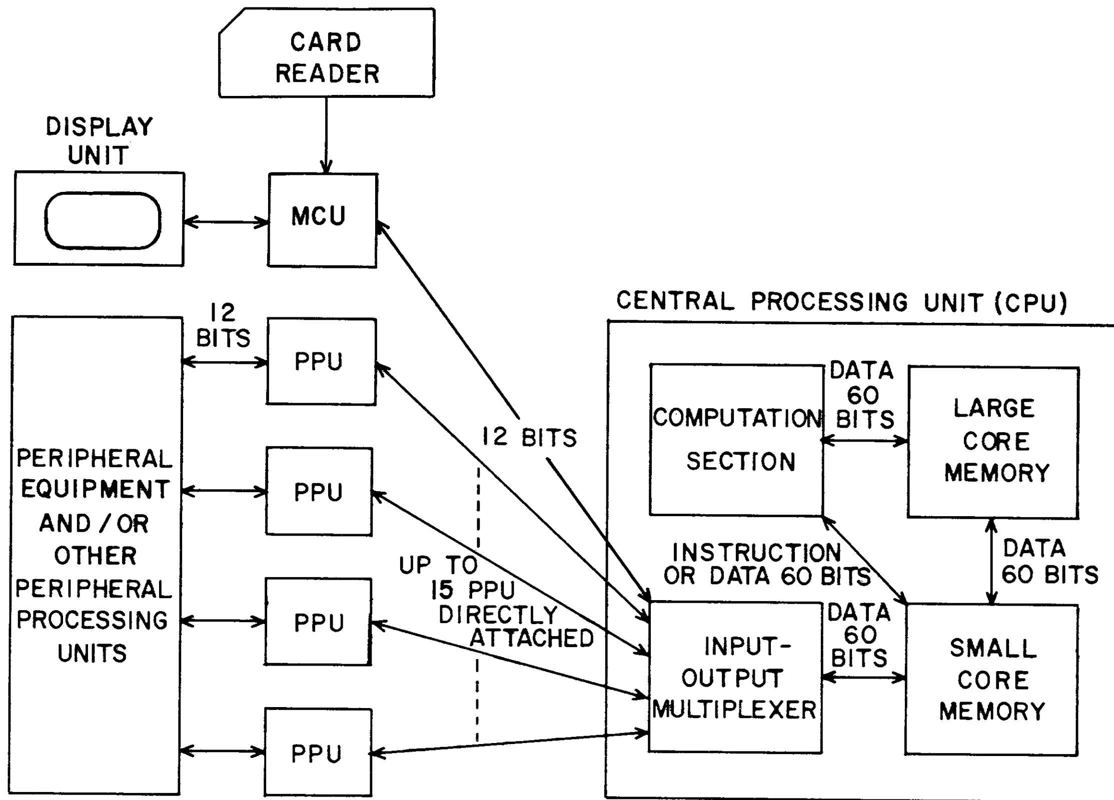



Figure 1-1. System Communication Paths

# 2. CENTRAL PROCESSOR UNIT

## COMPUTATION SECTION

The computation section of the Central Processor Unit contains all logic necessary to execute program instructions stored in Small Core Memory. It includes the registers and control logic to direct the arithmetic operations and provide interface between the arithmetic units, SCM, and LCM. In addition to instruction execution, the Central Processor Unit performs instruction fetching, address preparation, memory protection, and data fetching and storing. Figure 2-1 illustrates the general flow of information.

Program execution is begun by an exchange jump. The operating system can use an exchange jump to switch program execution between two SCM programs, leaving the first program in a usable state for later re-entry.

The Central Processor Unit reads 60-bit words from SCM and stores them in an instruction stack capable of holding up to twelve 60-bit words. Each instruction word in turn leaves the stack, enters a Current Instruction Word register for interpretation and testing. The Current Instruction Word register holds four 15-bit instructions, two 30-bit instructions or combinations of the two types of instructions. The 15- or 30-bit instructions issue individually from the Current Instruction Word register to one of nine functional units. The functional units obtain the instruction operands from and store results in 24 operating registers. Reservation control keeps an account of active operating registers to avoid conflicts.

### Operating Registers

Twenty-four registers are provided to minimize memory references for arithmetic operands and results. These 24 are divided into:

| Function | Identity | Length | Number |
|---|---|---|---|
| Operand Registers | X0 - X7 | 60 Bits | 8 |
| Address Registers | A0 - A7 | 18 Bits | 8 |
| Increment Registers | B0 - B7 | 18 Bits | 8 |

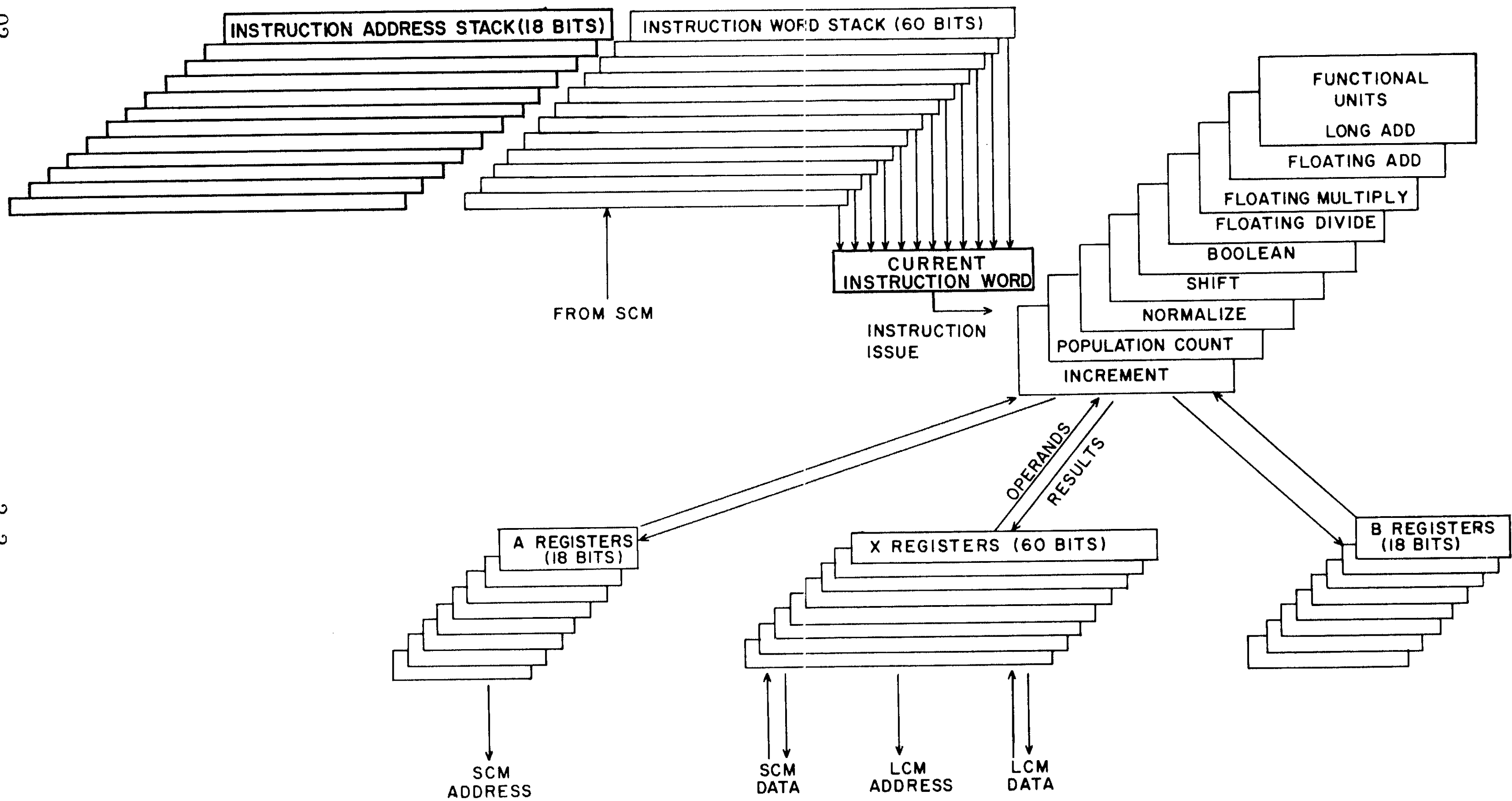



Figure 2-1. CPU Information Flow

X Registers

There are eight 60-bit X registers in the computation section of the CPU. These registers (X0, X1, . . . X7) are the principal data handling registers for computation. Data flows from these registers to the SCM and the LCM. Data also flows from SCM and LCM into these registers. All 60-bit operands involved in computation must originate and terminate in these registers.

Operands and results transfer between SCM and these registers as a result of placing a quantity into a corresponding address register.

The X registers also serve as address registers for referencing single words from LCM. X0 is used as the LCM relative starting address in a block copy operation.

A Registers

There are eight 18-bit A registers in the computation section of the CPU. These registers (A0, A1, . . . A7) are essentially SCM operand address registers. The registers are associated one-for-one with the X registers. Placing a quantity into an address register A1 - A5 causes an immediate SCM read reference to that relative address and sends the SCM word to the corresponding operand register X1 - X5. Similarly, placing a quantity into address register A6 or A7 causes the word in the corresponding X6 or X7 operand register to be written into that relative address of SCM. Only the Lower 16 bits are used; the remainder are ignored.

The A0 and X0 registers operate independently of each other and have no connection with SCM. A0 is used as the relative SCM starting address in a block copy operation and for scratch pad or intermediate results.

B Registers

There are eight 18-bit B registers in the computation section of the CPU. These registers (B0, B1, . . . B7) are primarily indexing registers for controlling program execution. Program loop counts may be incremented or decremented in these registers.

Program addresses may be modified on the way to an A register by adding or subtracting B register quantities. The B register also holds shift counts for pack and normalize operations and the channel number for channel status requests.

B0 always contains positive zero; that is, B0 is held clear.

## CPU Instruction Formats

Program instruction words are divided into 15-bit fields called parcels. The first parcel is the highest order 15 bits of the 60-bit word. The second, third, and fourth parcels follow in order. A CPU instruction may occupy either one or two parcels, depending on the type of instruction. The possible arrangements of one and two parcel instructions are shown in Figure 2-2. If an instruction requires two parcels it should not begin in the fourth parcel of the word. When a two parcel instruction begins in the last parcel of an instruction word it will be executed as if there were a fifth parcel in the instruction word and this parcel in the instruction word and this parcel contained all zeros; it will not obtain its seconds half of the instruction word from the next instruction word. For example, an 02 Jump instruction in the fourth parcel may be acceptable if the programmer wishes K to be zero.

One parcel Pass instructions may be used to complete a 60-bit word in order to place a particular instruction in the first parcel of a word. It may also be used to avoid starting a two parcel instruction in the fourth parcel of a word. Note that a 60, 61, 62 instructions with i equal to zero become pass instructions, (Page 3-5). Since these are 30 bit instructions they may be used as two parcel pass instruction. Pass instruction may be necessary for branch entry points because a branch instruction destination address must begin with a new word.

Groups of bits in an instruction are identified by the letters g, h, i, j, k, and K. Each letter represents an octal digit except K, which represents 6 octal digits. The designators are arranged in one and two parcel words as shown in Figure 2-2.

The g and h designators form the operation code. The g designator generally identifies the type of instruction and frequently specifies the functional unit. The h designator completes the function code specification for all but a few instructions by specifying the functional unit mode.

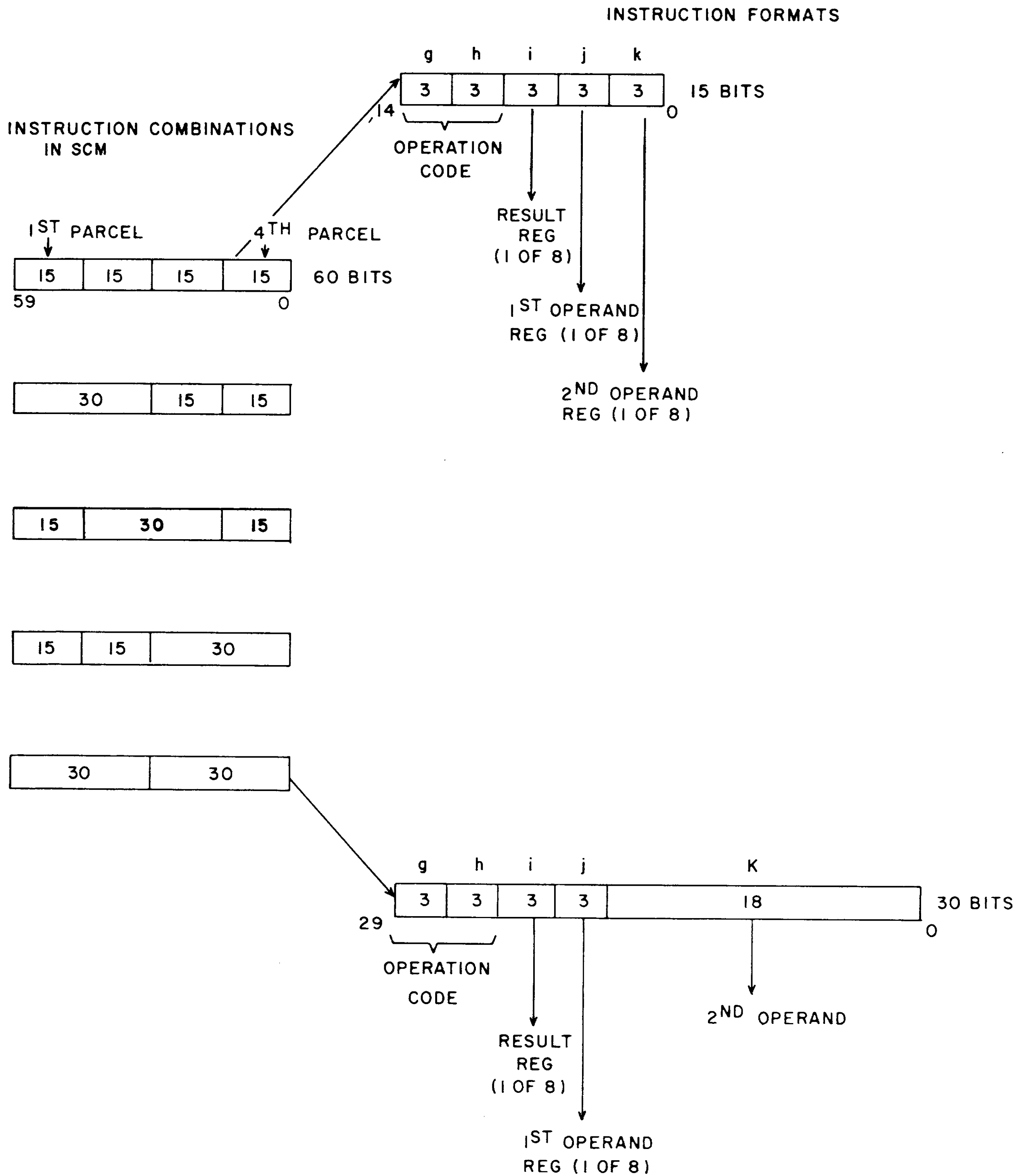



Figure 2-2. Parcel Instruction Arrangements

The i, j, and k designators are the operand source and destination indicators. They specify which one of the eight possible A, B, or X registers is referenced. The i designator is normally the destination indicator. If there are two destinations required for the instruction, both the i and j designators specify destination. In some 15-bit instructions the j and k designators specify shift count.

The K designator in a 30-bit instruction is an 18-bit operand for branch destination addresses and for small integer constants.

## Instruction Word Stack

The Instruction Word Stack (IWS) is a group of twelve 60-bit registers in the CPU computation section that hold program instruction words for execution. The instruction stack information is essentially a moving window in the program code. The stack is filled two words ahead of the program address currently being executed. A small program loop may frequently be entirely contained within the instruction stack. When this happens, the loop may be executed repeatedly without further references to SCM.

A group of twelve 18-bit address registers are associeted with the Instruction Word Stack. These registers, called the instruction address stack (IAS), hold relative SCM program addresses on a one-for-one basis with the program words in the instruction word stack. The rank one register contains the oldest address in the stack and the SCM address from which the word in rank one of the instruction word stack was read.

When a shift stack condition exists each rank is cleared and simultaneously entered with information from the next highest order rank. The information in rank one is discarded. New information arriving from SCM is entered in rank 12.

The twelve registers are individually identified by rank. The rank one register contains the oldest data in the stack. If the Instruction Word Stack contains sequential program instruction words, the contents of the rank one register in the stack correspond with the lowest storage address in the instruction address stack. The rank 12 register contains the last word to enter the stack. This register is loaded directly from SCM.

### Program Address Register

An 18-bit P register serves as a program address counter and holds the relative address for each program step. P is advanced to the next program step in the following ways:

1. P is advanced by one when the associated instruction word is sent to the Current Instruction Word register.
2. P is set to the address specified by a Go To ... (branch) instruction. If the instruction is a Return Jump, (P)+1 is stored before entering P with the new value to allow a return to the original sequence.
3. P is set to the address specified in the Exchange package.

Instruction Issue

Program instruction words are read one at a time from the instruction stack into the Current Instruction Word (CIW) register for execution. An instruction "issues" from the CIW register when the conditions in the functional units and operating registers are such that the functions required for execution may be performed to completion without conflicting with a previously issued instruction. Once an instruction has issued it must be completed in a fixed time framework. No delays are allowed from issue to delivery of data to the destination operating registers.

Since each instruction word is divided into four 15-bit parcels, there may be as many as four instructions in the CIW register at one time. These instructions are executed in sequence and the proper allowance made for the mixture of one and two parcel instruction formats.

Program Branching

When program execution reaches a branch instruction, the action taken depends upon whether the destination address is already in the instruction address stack. If the destination address is in the instruction address stack the P register is altered to the new program address and the corresponding word is read from the instruction stack to the CIW register. The jump is then completed without an SCM reference for a new instruction word.

If the destination address is out of the stack two new words, located at the destination address and the destination address plus one, are requested from SCM to begin the new program sequence. Instruction execution continues upon receipt of the words from SCM.

### Duplicate Entries In Stack

It is possible for a branch out of IWS to occur when the destination address corresponds to a program word that has already been requested from SCM as a result of the sequential two-word read ahead. If the word has not actually arrived at the IWS at the time of the branch test, the jump occurs and a duplicate of the first word in the new sequence is read from SCM. Execution of the new sequence begins as soon as the earlier word arrives at the instruction stack.

Duplicate entries in the IWS cause no problems unless an instruction is modified during execution. Since this modification occurs only in SCM, and since duplicate entries are merged in a logical sum network, an erroneous instruction will result. Therefore, the IWS should be voided by executing a Return Jump (01) instruction after instruction modification has been performed.

### Holes In The Stack

It may happen that several small program sequences reside in the instruction stack at the same time. Program execution may branch back and forth between two such sequences. The execution of the sequence occupying the lower ranks of the instruction stack may branch in such a way as to continue sequential execution into a program area not loaded into the stack on the initial pass. When this happens it is possible for the next sequential instruction word to be missing in the stack and no request has been made for it because rank 11 or 12 were not involved.

This situation is equivalent to a branch out of stack with no branch instruction involved. Two new words are requested from SCM to continue the program sequence.

## Functional Units

There are nine functional units in the computation section of the CPU. Each is a specialized unit with algorithms for a portion of the CPU instructions. The general function of each unit is listed. A number of functional units may be in operation at the same time.

| UNIT | GENERAL FUNCTION |
|---|---|
| Boolean | Handles the basic logical operations of transfer, logical product, logical sum, and logical difference. It also performs the pack and unpack floating point operations. |
| Shift | Executes operations basic to shifting. This includes left (circular) and right (end-off sign extension) shifting, and mask generation. |
| Normalize | Performs the normalize operations. |
| Long Add Unit | Performs integer addition or subtraction of two 60-bit fixed point operands. |
| Floating Add | Performs single or double precision floating point addition or subtraction on floating point operands. |
| Floating Multiply | Performs single or double precision floating point multiplication on floating point operands. |
| Floating Divide | Performs single precision floating point division of floating point operands. |
| Population Count | Counts the number of 1 bits in a 60-bit word. |
| Increment | Performs one's complement addition and subtraction of 18-bit operands. |

A functional unit receives one or two operands from operating registers at the beginning of instruction execution and delivers the result to the operating registers after performing the function. The functional units do not retain any information for reference in subsequent instructions. The units operate in three address mode with source and destination addressing limited to the operating register.

Except for the floating multiply and divide units, all functional units have one clock period segmentation. This means that the information arriving at the unit, or moving within the unit, is captured and held in a new set of registers at the end of every clock period. It is therefore possible to start a new set of operands for unrelated computation

into a functional unit each clock period even though the unit may require more than one clock period to complete the calculation. This process may be compared to a delay line in which data moves through the unit in segments to arrive at the destination in the proper order but at a later time. All functional units perform their algorithms in a fixed amount of time. No delays are possible once the operands have been delivered to the front of the unit.

The floating multiply unit has a two clock period segmentation. Operands may enter the multiply unit in any clock period providing there was no multiply operation initiated in the preceding clock period. There is a one clock period delay in initiating a multiply instruction if another multiply instruction has just been started.

The floating divide unit is the only functional unit in which an iterative algorithm is executed. There is essentially no segmentation possible in this unit. Therefore, to maximize execution speed, the beginning of a new divide operation should follow a previous divide operation by at least 18 clock periods.

## Exchange Jump

The CPU exchange jump is a mechanism for switching CPU execution between programs.

The execution of an exchange jump involves the simultaneous storing of all pertinent information in the CPU operating registers and control registers into SCM and writing new information from SCM into these same registers. This block of data is called an exchange package. An exchange package (Figure 2-3) provides the following information on a program to be executed:

1. Program address (P)
2. Reference address for Small Core Memory (RAS)
3. Field length of program for Small Core Memory (FLS)
4. Reference address for Large Core Memory (RAL)
5. Field length of program for Large Core Memory (FLL)
6. Program Status Designation register (PSD)

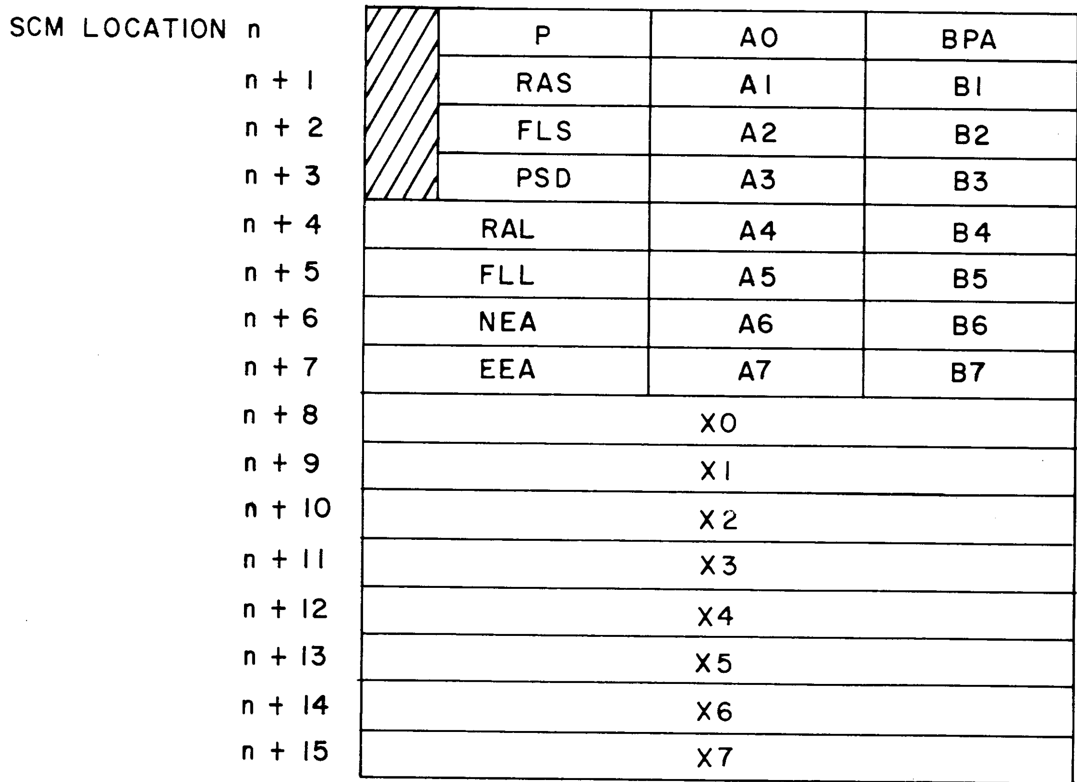



Figure 2-3. Exchange Package

7. Normal exit address (NEA)
8. Error exit address (EEA)
9. Breakpoint address (BPA)
10. Initial contents of the eight A registers
11. Initial contents of the eight X registers
12. Initial contents of B registers B1 through B7.

The period of time during which a particular exchange package resides in the CPU hardware registers is termed the execution interval. The execution interval begins with an exchange jump that reads the exchange package from SCM and enters these parameters into the CPU registers. It ends with another exchange jump that stores the exchange package back into SCM.

Several instructions or conditions initiate exchange jumps and select the exchange package that is to begin execution.

1. Exchange exit instructions (01300 and 013jK)
2. Error exit
3. Input/Output interrupt
4. Real time interrupt
5. Program breakpoint
6. Step mode

## Exchange Exit Instructions

The normal termination for an exchange package execution interval is caused by an exchange exit instruction (01300 or 013jK) in the associated program. The exit mode flag in the PSD register determines the source of the exchange package.

The exit mode flag is intended to indicate a privileged monitor program and is normally not set for an object program execution interval. When the flag is not set and the

object program terminates the execution interval with an 01300 instruction, the normal exit address (NEA) is the absolute address of the exchange package. When this flag is set and program terminates the execution interval with an 013jK instruction, the absolute SCM address for the exchange package is formed by adding (Bj) + K + (RAS).

An overflow of the lowest order 16 bits of this result causes an error condition that is not sensed in the hardware. Should a program erroneously execute an exchange exit instruction with an overflow condition, the exchange jump sequence will begin at the absolute SCM address corresponding to the lowest order 16 bits of this sum.

Error Exit

An object program terminates execution with an exchange jump to the Error Exit Address (EEA) upon encountering an error exit instruction (00) or under certain conditions defined by the Program Status Designation register (PSD). Some of these conditions may be selected by the programmer, and some are unconditional. In general, errors caused by arithmetic overflow, underflow, or indefinite results during computation may be allowed to proceed through the calculation, or may cause an error exit, depending on mode selection. Errors caused by hardware failure or program addressing out of an assigned field in storage cause unconditional error exits. In any error exit case the programmer may allow the object program to continue where the error can be corrected or ignored.

The error condition flags and mode selection flags are all contained in the Program Status Designation register (PSD), which is loaded from the exchange package for each program execution interval. The mode selections are made in the exchange package prior to the execution interval of the program. If an error condition occurs during the execution interval the type of error can be determined by analyzing the terminating exchange package parameters. Each bit in the PSD register has significance either as a mode selection or an error condition flag. For a detailed description of the PSD register refer to Program Status Designation (page 2-14).

Input/Output Interrupt

Refer to Small Core Memory (page 4-2)

### Real Time Interrupt

CPU programs may be timed precisely by using the CPU clock period counter which is advanced one count each clock period of 27.5 nanoseconds. Since the clock advances synchronously with program execution, a program may be timed to an exact number of CPU clock periods.

The CPU clock period counter contains a 17-bit register that can be read by a CPU program with a Read Clock instruction (016j0). This register contains the lowest order 17 bits of the real time count. An overflow of the highest order bit in this register sets a real time interrupt flag, which can be seen as the 18th bit when the time is read. It also attempts an interrupt of the CPU program to absolute address 0020 in SCM every 3.6 milliseconds (approximate). The real time exchange package at this SCM address executes a CPU program that performs operations associated with the clock.

### Program Breakpoint

A program may be executed in small sections during a debugging phase by using the Breakpoint Address register (BPA). This is a hardware register in the computation section of the CPU that is loaded from the program exchange package. A coincidence test is made between (BPA) and the Program Address register (P) as each program instruction word is read from the instruction word stack. When coincidence occurs the program execution terminates with an exchange jump to the Error Exit Address (EEA). If the BPA is equal to (P) in the initiating exchange jump package, no instructions are executed. Normally, no instructions are executed at address BPA (see P2-18).

### Step Mode

A program may be executed in Step mode by setting the step mode flag in the Program Status Designation register for the program execution interval. Step mode causes the program to be interrupted at the end of each program instruction word with an exchange jump to the Error Exit Address (EEA).

## Program Status Designations

The Program Status Designation register (PSD) is a collection of 18 program status flags. Six of these flags are mode designators and 12 are condition designators. The arrangement of these flags in the register is shown in Figure 2-4.

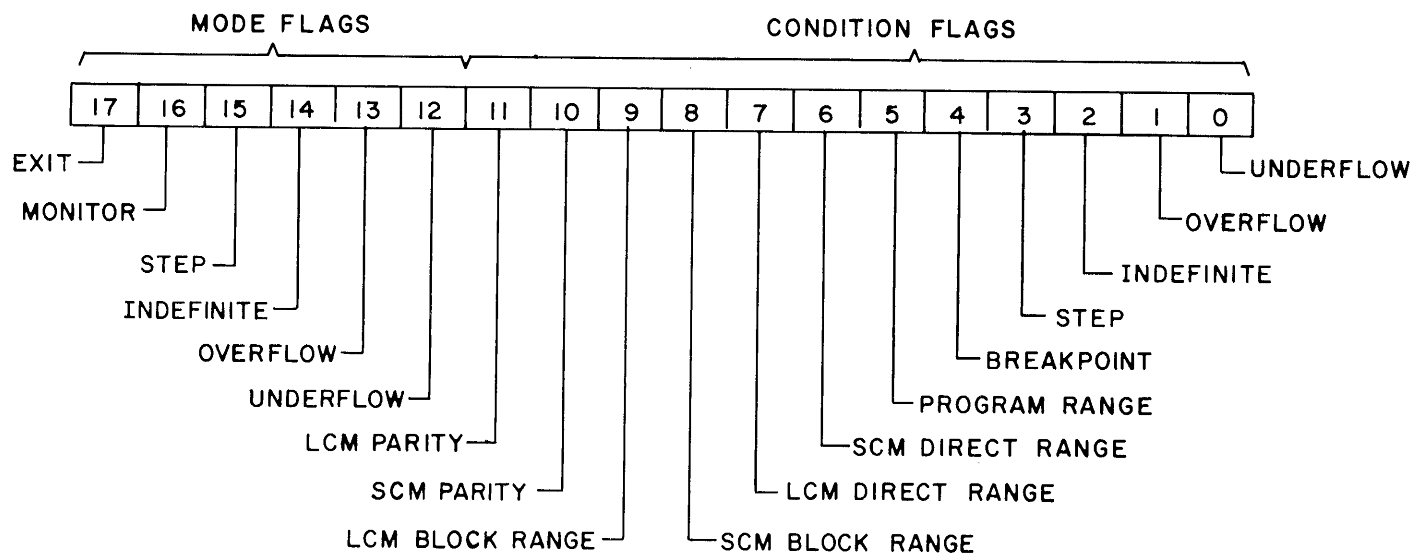



Figure 2-4. Flag and Register Arrangement

The PSD register is loaded from the exchange package during an exchange jump sequence. All 18 bits are entered in the register at this time. The six mode designators remain unaltered throughout the execution interval for the exchange package. The 12 condition designators may be set by conditions that occur during the execution interval. All flags are stored in the SCM exchange package at the end of the execution interval.

The execution interval for an exchange package may be terminated by an error condition that occurred during this interval.

Mode Flags

Exit Mode Flag (Bit 17): The exit mode flag controls the source of the exchange package address for the execution of an exchange exit instruction (013). If this flag is set, the exchange package absolute address is (Bj) + K + (RAS). If this flag is not set, the exchange package absolute address is (NEA).

Monitor Mode Flag (Bit 16): The monitor mode flag controls the mode of input/output activity. If this flag is set, the program currently being executed cannot be interrupted by an I/O interrupt request. If an I/O interrupt occurs, it will not be honored until the end of the execution interval for the current exchange package.

The monitor flag also controls the execution of the reset buffer instructions (0160, 0170). If the monitor mode flag is set, the reset buffer instructions are executed (Page 5-45). Otherwise, the reset buffer instructions are executed as pass instructions. This flag prevents an object program from interfering with I/O activity.

Step Mode Flag (Bit 15): The step mode flag, if set, causes the current program to be interrupted at the end of each program instruction word. The terminating exchange package is at absolute address (EEA) in SCM.

Indefinite Mode Flag (Bit 14): The indefinite mode flag enables interruption of the current program on the condition of an indefinite floating point result. The combination of this flag set and the indefinite condition flag set terminates the execution interval at

the end of the current program instruction word. The terminating exchange package is located at absolute address (EEA) in SCM.

Overflow Mode Flag (Bit 13): The overflow mode flag enables interruption of the current program on the condition of an overflow of a floating point result. The combination of this flag set and the overflow condition flag set terminates the execution interval at the end of the current program instruction word. The terminating exchange package is located at absolute address (EEA) in SCM.

Underflow Mode Flag (Bit 12): The underflow mode flag enables interruption of the current program on the condition of an underflow of a floating point result. The combination of this flag set and the underflow condition flag set terminates the execution interval at the end of the current program instruction word. The terminating exchange package is located at absolute address (EEA) in SCM.

Condition Flags

LCM Parity Condition Flag (Bit 11): The LCM parity condition flag is set whenever an LCM parity error is detected during an LCM reference. When this flag is set the execution interval for the exchange package terminates at the end of the current program word. The terminating exchange package is located at absolute address (EEA) in SCM.

SCM Parity Condition Flag (Bit 10): The SCM parity condition flag is set whenever an SCM parity error is detected during an SCM read/write cycle. When this flag is set the execution interval for the exchange package terminates at the end of the current program instruction word. The terminating exchange package is located at absolute address (EEA) in SCM.

LCM Block Range Condition Flag (Bit 9): The LCM Block range condition flag is set whenever a block copy instruction is issued that would cause an LCM reference to an address equal to or greater than (FLL). The block copy instruction is issued as a pass instruction in this case. When this flag is set the execution interval for the exchange package terminates at the end of the current program instruction word. The terminating exchange package is located at absolute address (EEA).

SCM Block Range Condition Flag (Bit 8): The SCM block range condition flag is set whenever a block copy instruction is issued that would cause an SCM reference to an address equal to or greater than (FLS). The block copy instruction is issued as a pass instruction in this case. When this flag is set the execution interval for the exchange package terminates at the end of the current program instruction word. The terminating exchange package is located at absolute address (EEA) in SCM.

LCM Direct Range Condition Flag (Bit 7): The LCM direct range condition flag is set whenever a read LCM (014) or write LCM (015) instruction causes an LCM reference to an address equal to or greater than (FLL). Writing into LCM is inhibited in such a case. When this flag is set the execution interval for the exchange package terminates at the end of the current program instruction word. The terminating exchange package is located at absolute address (EEA) in SCM.

SCM Direct Range Condition Flag (Bit 6): The SCM direct range condition flag is set whenever an SCM reference other than a block copy instruction occurs with an address equal to or greater than (FLS) or whenever the P register is greater than or equal to (FLS). Writing into SCM is inhibited in such a case. When this flag is set the execution interval for the exchange package terminates at the end of the current program instruction word. The terminating exchange package is located at absolute address (EEA) in SCM.

Program Range Condition Flag (Bit 5): The program range condition flag is set when the P register equals zero or an error exit instruction is issued. When this flag is set the execution interval for the exchange package terminates immediately. The terminating exchange package is located at absolute address (EEA) in SCM.

Breakpoint Condition Flag (Bit 4): The breakpoint condition flag is set whenever (P) equals (BPA). When this flag is set the execution interval for the exchange package terminates at the end of the current program instruction word. The terminating exchange package is located at absolute address (EEA) in SCM.

This condition flag normally sets in time to terminate the execution interval before the instruction word located at program address (BPA) is executed. If two increment

instructions with 30-bit formats are contained in the instruction word at (BPA) -1, however, it is possible for execution of the instruction word at (BPA) to begin before the breakpoint condition flag has taken effect. In this case the execution interval for the exchange package terminates at the end of the execution of the instruction word located at address (BPA). If the breakpoint condition flag is set at a word immediately following a branch instruction, the program will terminate whether or not the branch is taken.

Step Condition Flag (Bit 3): The step condition flag is set whenever the step mode flag is set and an instruction issues. This combination of conditions allows only one instruction word to be executed during this execution interval for the exchange package. When this flag is set the execution interval for the exchange package terminates at the end of the current program instruction word. The terminating exchange package is located at absolute address (EEA) in SCM.

Indefinite Condition Flag (Bit 2): The indefinite condition flag is set whenever an indefinite floating point value is detected by a floating point functional unit. An indefinite value may occur during execution of instructions 30, 31, 32, 33, 34, 35, 40, 41, 42, 44, and 45. When this flag is set and the indefinite mode flag is also set, the execution interval for the exchange package terminates at the end of the current program instruction word. Note that this program instruction word is not necessarily the word containing the instruction that caused the indefinite condition. Rather, it is the current instruction word at the time the error condition is generated in the functional unit. The terminating exchange package is located at absolute address (EEA) in SCM.

Overflow Condition Flag (Bit 1): The overflow condition flag is set whenever an overflow of the floating point range is detected by a functional unit. A floating point overflow may occur in the execution of instructions 30, 31, 32, 33, 34, 40, 41, 42, 44, and 45. When this flag is set and the overflow mode flag is also set, the execution interval for the exchange package terminates at the end of the current program instruction word. Note that this program instruction word is not necessarily the word containing the instruction that caused the overflow condition. Rather, it is the current instruction word at the time the error condition is generated in the functional unit. The terminating exchange package is located at absolute address (EEA) in SCM.

Underflow Condition Flag (Bit 0): The underflow condition flag is set whenever an underflow of the floating point range is detected by a functional unit. A floating point underflow may occur in the execution of instructions 32, 33, 40, 41, 42, 44, and 45. When this flag is set and the underflow mode flag is also set, the execution interval for the exchange package terminates at the end of the current program instruction word. Note that this program instruction word is not necessarily the word containing the instruction that caused the underflow condition. Rather, it is the current instruction word at the time the error condition is generated in the functional unit. The terminating exchange package is located at absolute address (EEA) in SCM.

# 3. CENTRAL PROCESSOR INSTRUCTIONS

## INTRODUCTION

This section describes the Central Processor Unit instructions. The CPU instructions tend to fall into two distinct categories: those causing computation, and those causing storage references or program branching. The CPU instructions causing computation are generally executed in a fixed amount of time after they have issued from the Current Instruction Word register. Instructions involving storage references for operands or program branching cannot be precisely timed. Program branching within the instruction stack causes no storage references and small program loops can therefore be precisely timed.

Careful coding of critical program loops can produce substantial improvements in execution time. Detailed timing information is provided in the appendix section of this manual to allow a complete analysis of these situations warranting the programming effort.

Preceding the description of each instruction is the octal code, the instruction name and length. Table 3-1 defines the Central Processor Unit instruction designators.

TABLE 3-1. CENTRAL PROCESSOR INSTRUCTION DESIGNATORS

| DESIGNATOR | USE |
|---|---|
| A | Specifies one of eight 18-bit address registers. |
| B | Specifies one of eight 18-bit index registers; B0 is fixed and equal to zero. |
| gh | A 6-bit instruction code. |
| ghi | A 9-bit instruction code. |
| i | A 3-bit code specifying one of eight designated registers (e.g., Ai). |
| j | A 3-bit code specifying one of eight designated registers (e.g., Bj). |

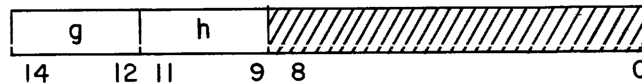
TABLE 3-1. CENTRAL PROCESSOR INSTRUCTION DESIGNATORS (Cont'd)

| DESIGNATOR | USE |
|---|---|
| jk | A 6-bit constant, indicating the number of shifts to be taken. |
| k | A 3-bit code specifying one of eight designated registers (e.g., Bk). |
| K | An 18-bit constant, used as an operand or as a branch destination (address). |
| X | Specifies one of eight 60-bit operand registers. |

Instruction formats are also given; parallel lines within a format indicate these bits are not used in the operation.

## Error Exit And No Operation

***00*** ***Error exit to EEA*** ***(15 Bits)***




This instruction is treated as an error condition and will set the program range condition flag in the PSD register. The condition flag will then generate an error exit request which will cause an exchange jump to address (EEA). All instructions issued prior to this instruction will be run to completion. Any instructions following this instruction in the current instruction word will not be executed. When all operands have arrived at the operating registers as a result of previously issued instructions, an exchange jump will occur to the exchange package designated by (EEA).

The i, j, and k designators are ignored. The program address stored in the exchange package on the terminating exchange jump is advanced one count from the address of the current instruction word. This is true no matter which parcel of the current instruction word contains the Error Exit instruction.

This instruction format is intended to be a privileged instruction. The program range condition flag is set in the PSD register to indicate that the program has jumped to an

area of the SCM field which may be in range but is not valid program code. This should occur when an incorrectly coded program jumps into an unused area of the SCM field or into a data field. The program range condition flag is also set on the condition of a jump to address zero. These conditions can be determined on the basis of the register contents in the exchange package. The existence of an Error Exit condition resulting from execution of this instruction format may thus be deduced.

An error exit is treated as an SCM range error which blocks a write operation in SCM as soon as the error is detected. Thus, a legitimate SCM write operation may be blocked by the error condition even though the instruction causing the write operation issues substantially before the error exit.

***46*** **_No operation (Pass)_** ***(15 Bits)***

| g | h | (unused) |
|---|---|---|
| 14–12 | 11–9 | 8–0 |

This instruction is a "do-nothing" instruction that is typically used to pad the program between certain program steps.

EXAMPLE:

| | 59 | | | 0 |
|---|---|---|---|---|
| P | 30-BIT INST. | 15-BIT INST. | PASS | |
| P+1 | 30-BIT INST. | 30-BIT INST. | | |

In this example, a Pass instruction is used to pad the remainder of the word at P. Since the next instruction is 30 bits, it cannot fit in P and must be placed in P + 1.

Increment

| | | |
|---|---|---|
| 50 | *Set Ai to Aj + K* | *(30 Bits)* |
| 51 | *Set Ai to Bj + K* | *(30 Bits)* |
| 52 | *Set Ai to Xj + K* | *(30 Bits)* |

| g | h | i | j | K |
|---|---|---|---|---|
| 29 27 | 26 24 | 23 21 | 20 18 | 17 0 |

| | | |
|---|---|---|
| 53 | *Set Ai to Xj + Bk* | *(15 Bits)* |
| 54 | *Set Ai to Aj + Bk* | *(15 Bits)* |
| 55 | *Set Ai to Aj — Bk* | *(15 Bits)* |
| 56 | *Set Ai to Bj + Bk* | *(15 Bits)* |
| 57 | *Set Ai to Bj — Bk* | *(15 Bits)* |

| g | h | i | j | k |
|---|---|---|---|---|
| 14 12 | 11 9 | 8 6 | 5 3 | 2 0 |

These instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in address register i. Operands are obtained from address (A), increment (B), and operand (X) registers as well as the instruction itself (K = 18-bit signed constant). Operands obtained from an Xj operand register are the truncated lower 18 bits of the 60-bit word. The highest order bits are ignored.

These instructions are intended for fetching operands from storage for computation and for delivering results back into storage. If the i designator is non-zero, a storage reference is made to SCM using the lower 16 bits of the resulting sum or difference as the relative storage address. The type of storage reference is a function of the i designator value.

i = 0; no storage reference

i = 1, 2, 3, 4, 5; read from SCM to register Xi

i = 6, 7; write into SCM from register Xi

If this instruction makes a storage reference to SCM the address is compared with (FLS) to determine if the reference is within the assigned SCM field. If the address is out of range the SCM direct range condition flag is set in the PSD register. This flag will cause the current program sequence to terminate with an exchange jump to (EEA). If the reference involved writing in to an X register, the out of range word addressed will be written into the X register before the interrupt occurs. If the reference involved writing into SCM the memory sequence will be aborted to avoid altering the designated storage quantity. If the quantity placed in A is larger than 16 bits, only the lower 16 bits will be sent to SCM.

| | | |
|---|---|---|
| *60* | *Set Bi to Aj + K* | *(30 Bits)* |
| *61* | *Set Bi to Bj + K* | *(30 Bits)* |
| *62* | *Set Bi to Xj + K* | *(30 Bits)* |

| g | h | i | j | K |
|---|---|---|---|---|
| 29 27 | 26 24 | 23 21 | 20 18 | 17 0 |

| | | |
|---|---|---|
| *63* | *Set Bi to Xj + Bk* | *(15 Bits)* |
| *64* | *Set Bi to Aj + Bk* | *(15 Bits)* |
| *65* | *Set Bi to Aj — Bk* | *(15 Bits)* |
| *66* | *Set Bi to Bj + Bk* | *(15 Bits)* |
| *67* | *Set Bi to Bj — Bk* | *(15 Bits)* |

| g | h | i | j | k |
|---|---|---|---|---|
| 14 12 | 11 9 | 8 6 | 5 3 | 2 0 |

These instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in increment register Bi. An overflow condition is ignored.

Operands are obtained from address (A), increment (B), and operand (X) registers as well as the instruction itself (K = 18-bit signed constant). Operands obtained from an

Xj operand register are the truncated lower 18 bits of the 60-bit word. The highest order bits are ignored. If the designator i is a zero, these instructions become pass instructions.

| | | |
|---|---|---|
| *70* | *Set Xi to Ai + K* | *(30 Bits)* |
| *71* | *Set Xi to Bj + K* | *(30 Bits)* |
| *72* | *Set Xi to Xj + K* | *(30 Bits)* |

| g | h | i | j | K |
|---|---|---|---|---|
| 29 27 | 26 24 | 23 21 | 20 18 | 17 0 |

| | | |
|---|---|---|
| *73* | *Set Xi to Xj + Bk* | *(15 Bits)* |
| *74* | *Set Xi to Aj + Bk* | *(15 Bits)* |
| *75* | *Set Xi to Aj — Bk* | *(15 Bits)* |
| *76* | *Set Xi to Bj + Bk* | *(15 Bits)* |
| *77* | *Set Xi to Bj — Bk* | *(15 Bits)* |

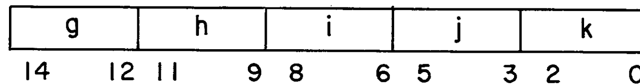
| g | h | i | j | k |
|---|---|---|---|---|
| 14 12 | 11 9 | 8 6 | 5 3 | 2 0 |

These instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result into the lower 18 bits of operand register Xi. The sign of the result is extended to the upper 42 bits of operand register Xi. An overflow condition is ignored.

Operands are obtained from address (A), increment (B), and operand (X) registers as well as the instruction itself (K = 18-bit signed constant). Operands obtained from an Xj operand register are the truncated lower 18 bits of the 60-bit word. The highest order bits are ignored.

## Fixed Point Arithmetic

*36* *Integer sum of Xj and Xk to Xi* *(15 Bits)*

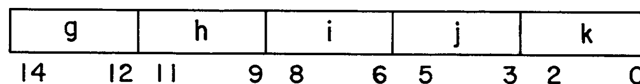



This instruction forms a 60-bit one's complement sum of the quantities from operand registers Xj and Xk and stores the result in operand register Xi. An overflow condition is ignored.

This instruction is intended for addition of integers too large for handling in the increment unit. It is also useful for merging and comparing data fields during data processing.

If both operands are zero the result is zero. If either zero operand is positive,the result is positive zero. If both operands are negative zero the result is negative zero.

*37* *Integer difference of Xj and Xk to Xi* *(15 Bits)*

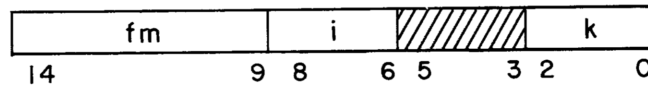



This instruction forms the 60-bit one's complement difference of the quantities from operand registers Xj (minuend) and Xk (subtrahend) and stores the result in operand register Xi. An overflow condition is ignored.

This instruction is intended for subtraction of integers too large for handling in the increment unit. This instruction is also useful in comparing data fields during data processing.

If (Xj) is a negative zero quantity, and (Xk) is a positive zero quantity, the result is a negative zero quantity. The other three combinations of positive and negative zero operands result in a positive zero quantity.

**47** **Count the number of "1's" in Xk to Xi** **(15 Bits)**

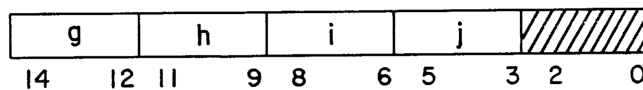



This instruction counts the number of "1 bits" in operand register Xk and stores the count in the lower order 6 bits of operand register Xi. Bits 6 through 59 are cleared to zero.

If Xk is a word of all ones, a count of 60 (decimal) is delivered to the Xi register. If Xk is a word of all zeros, a zero word is delivered to the Xi register.

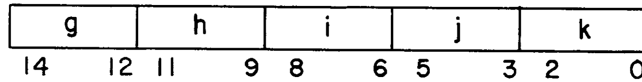
## Logical

**10** **Transmit Xj to Xi** **(15 Bits)**




This instruction transfers a 60-bit word from operand register Xj to operand register Xi. It is intended for moving data from one X register to another X register as rapidly as possible. No logical function is performed on the data.

11 ***Logical Product of Xj and Xk to Xi*** *(15 Bits)*

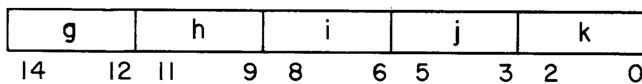



This instruction forms the logical product (AND function) of 60-bit words from operand registers Xj and Xk and places the product in operand register Xi. Bits of register Xi are set to "1" when the corresponding bits of the Xj and Xk registers are "1" as in the following example:

Xj = 0101
Xk = 1100
Xi = 0100

This instruction is intended for extracting portions of a 60-bit word during data processing. If the j and k designators have the same value, the instruction degenerates into a copy instruction.

12 ***Logical sum of Xj and Xk to Xi*** *(15 Bits)*

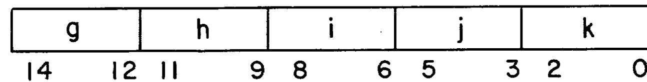



This instruction forms the logical sum (inclusive OR) of 60-bit words from operand registers Xj and Xk and places the sum in operand register Xi. Bits of register Xi are set to "1" if the corresponding bit of the Xj or Xk register is a "1" as in the following example:

Xj = 0101
Xk = 1100
Xi = 1101

This instruction is intended for merging portions of a 60-bit word into a composite word during data processing. If the j and k designators have the same value, the instruction degenerates into a copy instruction.

13 *Logical difference of Xj and Xk to Xi* (15 Bits)

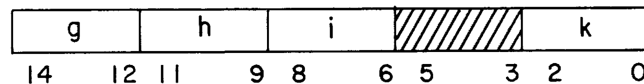



This instruction forms the logical difference (exclusive OR) of 60-bit words from operand registers Xj and Xk and places the difference in operand register Xi. Bits of register Xi are set to "1" if the corresponding bits in the Xj and Xk registers are unlike as in the following example:

Xj = 0101
Xk = 1100
Xi = 1001

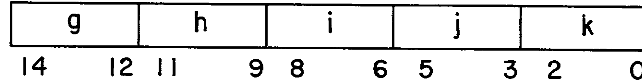
This instruction is intended for comparing bit patterns or for complementing bit patterns during data processing. If the j and k designators have the same value the result will be a word of all zeros written into register Xi.

14 *Transmit the complement of Xk to Xi* (15 Bits)




This instruction extracts the 60-bit word from operand register Xk, complements it, and transmits this complemented quantity to operand register Xi. It is intended for changing the sign of a fixed point or floating point quantity as quickly as possible.

*15* *Logical product of Xj and complement of Xk to Xi* *(15 Bits)*

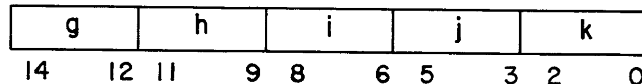



This instruction forms the logical product (AND function) of the 60-bit quantity from operand register Xj and the complement of the 60-bit quantity from operand register Xk, and places the result in operand register Xi. Thus, bits of Xi are set to "1" when the corresponding bits of the Xj register and the complement of the Xk register are "1" as in the following example:

Xj = 0101
Complemented Xk = 0011
Xj = 0001

This instruction is intended for extracting portions of a 60-bit word during data processing. If the j and k designators have the same value, a logical product is formed between two complementary quantities. The result will be a word of all zeros.

*16* *Logical sum of Xj and complement of Xk to Xi* *(15 Bits)*

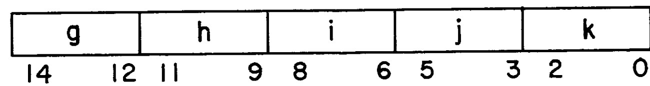



This instruction forms the logical sum (inclusive OR) of the 60-bit quantity from operand register Xj and the complement of the 60-bit word from operand register Xk, and places the result in operand register Xi. Thus, bits of Xi are set to "1" if the corresponding bit of the Xj register or complement of the Xk register is a "1" as in the following example:

Xj = 0101
Complemented Xk = 0011
Xi = 0111

This instruction is intended for merging portions of a 60-bit word into a composite word during data processing. If the j and k designators have the same value the result will be a word of all ones.

**17** *Logical difference of Xj and complement of Xk to Xi* *(15 Bits)*



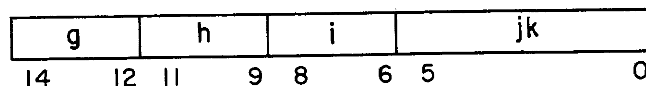

This instruction forms the logical difference (exclusive OR) of the quantity from operand register Xj and the complement of the 60-bit word from operand register Xk, and places the result in operand register Xi. Thus, bits of Xi are set to "1" if the corresponding bits of Xj and the complement of register Xk are unlike as in the following example:

| | | |
|---|---|---|
| Xj | = | 0101 |
| Complemented Xk | = | 0011 |
| Xi | = | 0110 |

This instruction is intended for comparing bit patterns or for complementing bit patterns during data processing. If the j and k designators have the same value, a logical difference is formed between two complementary quantities. The result is a word of all ones.

## Shift

**20** *Left shift Xi, jk places* *(15 Bits)*




This instruction shifts the 60-bit word in operand register Xi left circular jk places. Bits shifted off the left end of operand register Xi replace those shifted from the right end.

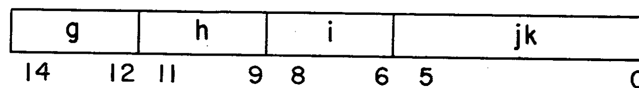
The 6-bit shift count jk allows a complete circular shift of register Xi.

In the example below the j designator has a value of 1 and the k designator a value of 2. These octal quantities are treated as a shift count of 12 octal or 10 decimal.

Example: Initial (Xi) = 2323 6600 0000 0000 0111
jk = 12
Final (Xi) = 7540 0000 0000 0022 2464

If the shift count is greater than the 60-bit register length the shift is performed modulo 60. For example, if the shift count is 63 (decimal) the result is a three bit position left shift.

**21** *Arithmetic right shift Xi, jk places*

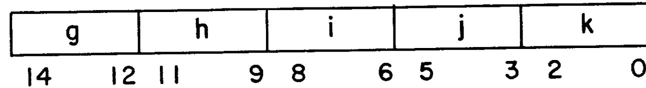



This instruction shifts the 60-bit word in operand register Xi right jk places. The rightmost bits of Xi are discarded and the sign bit is extended.

Example: Initial (Xi) = 2004 7655 0002 3400 0004
jk = 30
Final (Xi) = 0000 0000 2004 7655 0002

If the shift count is greater than the 60-bit register length the result will contain 60 copies of the sign bit. If the operand was positive, a positive zero word will result. If the operand was negative, a negative zero word will result.

22 *Left shift Xk nominally Bj places to Xi* (15 Bits)

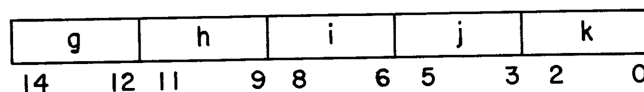



This instruction shifts the 60-bit quantity from operand register Xk the number of places specified by the quantity in increment register Bj and places the result in operand register Xi.

1. If Bj is positive (i.e., bit 17 of Bj = 0), the quantity from Xk is shifted left-circular. The lower order six bits of Bj specify the shift count. The higher order bits are ignored.
2. If Bj is negative (i.e., bit 17 of Bj = 1), the quantity from Xk is shifted right (end off with sign extension). The one's complement of the lower order 12 bits of Bj specify the shift count. The higher order bits are ignored. If the shift count is greater than 60 (decimal) the result stored in the Xi register will consist of 60 copies of the operand sign bit.

The contents of Bj might be the result of an unpack operation; in which case it is the unbiased exponent and (Xi) is the coefficient. This instruction is used for shifting a coefficient from a floating point number to the integer position after an unpack operation.

23 *Arithmetic right shift Xk nominally Bj places to Xi* (15 Bits)




This instruction shifts the 60-bit quantity from operand register Xk the number of places specified by the quantity in increment register Bj and places the result in operand register Xi.

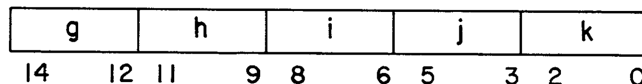
1. If Bj is positive (i.e., bit 17 of Bj = 0), the quantity from register Xk is shifted right (end off with sign extension). The lower order 12 bits of Bj

specify the shift count. The higher order bits are ignored. If the shift count is greater than 60 (decimal) the result stored in the Xi register will consist of 60 copies of the operand sign bit.

2. If Bj is negative (i.e., bit 17 of Bj = 1), the quantity from register Xk is shifted left circular. The complement of the lower order six bits of Bj specify the shift count. The higher order bits are ignored.

This instruction is intended for use in data processing where the amount of shift is derived in the computation. This instruction is also useful for adjusting the coefficient of a floating point number while it is in its unpacked form.

***24*** ***Normalize Xk in Xi and Bj*** ***(15 Bits)***




This instruction normalizes the floating point quantity from operand register Xk and places it in operand register Xi. Normalizing consists of left shifting the coefficient the minimum number of positions required to make bit 47 different from bit 59. This places the most significant bit of the coefficient in the highest order position of the coefficient portion of the word. The exponent portion of the word is then decreased by the number of bit positions shifted. The number of left shifts necessary to normalize the quantity is entered in increment register Bj.

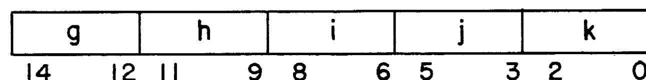
If a complete underflow occurs (i.e., the unpacked exponent is more negative than -1777), a zero word is delivered to the Xi register. The sign of the operand is preserved and (Xi) is either all zero bits or all one bits, depending on the sign of the original operand. The shift count delivered to the Bj register is a result of considering the coefficient field of (Xk) without regard to the exponent. This quantity is therefore the value that would be appropriate for normalizing the operand if the exponent were in range.

If a partial underflow occurs (i.e., the unpacked exponent equals -1777), the result is delivered to the Xi register and the Bj register as for a normal case even though subsequent computation may detect this operand as an underflow case.

Normalizing either a plus or minus zero coefficient sets the shift count (Bj) to $48_{10}$ and clears Xi to all zeros. The sign of the operand is preserved and (Xi) is either all zero bits or all one bits.

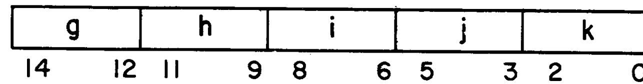
If Xk contains an infinite quantity (3777X....X or 4000X....X) or an indefinite quantity (1777X....X or 6000X....X) no shift takes place. The contents of Xk are copied into Xi and Bj is set equal to zero. No flags are set in the PSD register by the normalize unit.

**25** ***Round and normalize Xk in Xi and Bj*** ***(15 Bits)***



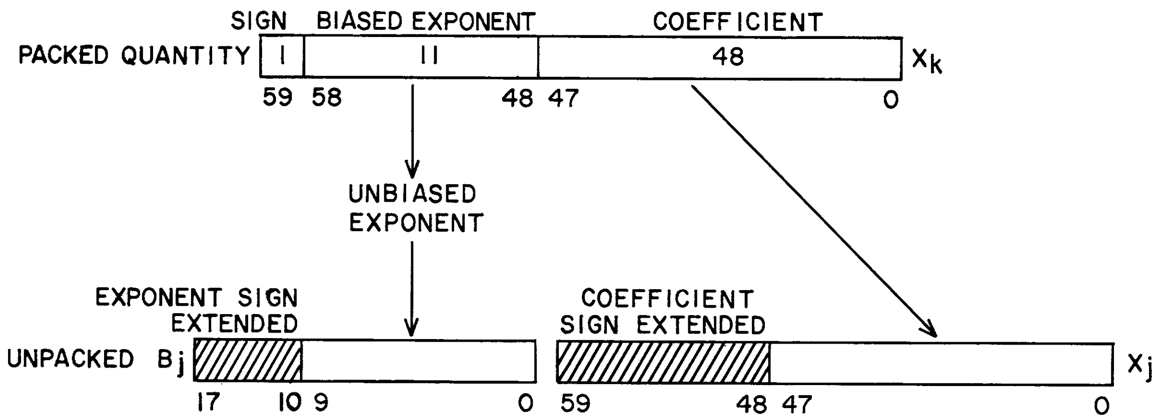

This instruction performs the same operation as instruction 24 except that the quantity from the operand register Xk is rounded before it is normalized. Rounding is accomplished by placing a "1" round bit immediately to the right of the least significant coefficient bit. The resulting coefficient is increased by one-half the value of the least significant bit. Normalizing a zero coefficient places the round bit in bit 47 and reduces the exponent by 48. Note that the same rules apply for underflow, overflow, and indefinite results.

If Xk contains an infinite quantity (3777X....X or 4000X....X) or an indefinite quantity (1777X....X or 6000X....X), no shift takes place. The contents of Xk are copied into Xi and Bj is set equal to zero. No flags are set in the PSD register.



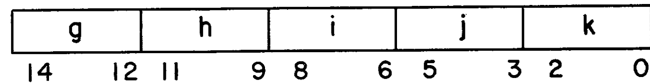

This instruction unpacks the floating point quantity from operand register Xk and sends the 48-bit coefficient to operand register Xi and the 11-bit exponent to increment register Bj. The exponent bias is removed during Unpack so that the quantity in Bj is the true one's complement representation of the exponent.

The exponent and coefficient are sent to the low-order bits of the respective registers as shown below:




Special operand formats are treated in the same manner as normal operands. No flags are set in the PSD register by this instruction.

**27** *Pack Xi from Xk and Bj* *(15 Bits)*



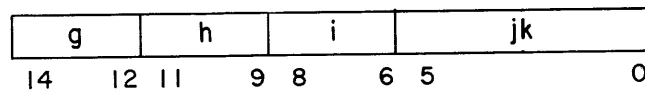

This instruction packs a floating point number in operand register Xi. The coefficient of the number is obtained from operand register Xk and the exponent from increment register Bj. Bias is added to the exponent during the Pack operation. The instruction does not normalize the coefficient.

Exponent and coefficient are obtained from the proper low-order bits of the respective registers and packed as shown in the illustration for the Unpack (26) instruction. Thus, bits 48 to 58 of Xk and bits 11 to 17 of Bj are ignored. There is no test for overflow or underflow. No flags are set in the PSD register by this instruction.

Note that if Xk is positive, the packed exponent occupying positions 48 to 58 of Xi is obtained from bits 0 to 10 of Bj by complementing bit 10; if Xk is negative, bit 10 is not complemented but bits 0 to 9 are.

The j designator may be set to zero in this instruction to pack a fixed point integer into floating point format without using one of the active B registers.

**43** *Form mask in Xi, jk bits* *(15 Bits)*




The instruction forms a mask in operand register Xi. The 6-bit quantity jk defines the number of "1's" in the mask as counted from the highest order bit in Xi. The completed masking word consists of "1's" in the high order bit positions of the word and "0's" in the remainder of the word.

Example: j = 2

k = 4

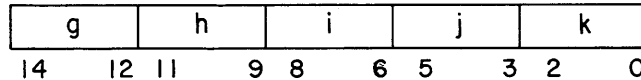
(Xi) = 7777 7760 0000 0000 0000

The contents of operand register i = 0 when jk = 0. The contents of operand register i are all "1's" when jk is 60 (decimal) or greater.

This instruction is intended for generating variable width masks for logical operations. Used with the shift instruction, this instruction will create an arbitrary field mask faster than by reading a pre-generated mask from storage.

## Floating Point Arithmetic

**30** ***Floating sum of Xj and Xk to Xi*** ***(15 Bits)***




This instruction forms the sum of the floating point quantities from operand registers Xj and Xk and packs the result in operand register Xi. The packed result is the upper half of a double precision sum.

At the start both arguments are unpacked, and the coefficient of the argument with the smaller exponent is entered into the upper half of a 99-bit accumulator. The coefficient is shifted right by the difference of the exponents. The other coefficient is then added into the upper half of the accumulator. If overflow occurs, the sum is right-shifted one place and the exponent of the result increased by one. The upper half of the accumulator holds the coefficient of the sum, which is not necessarily in normalized form. The exponent and upper coefficient are then repacked in operand register Xi.

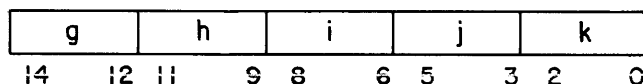
If the two operands are of equal magnitude and opposite sign the resulting sum will have a zero coefficient. The exponent delivered to the Xi register will be the same as

the exponent for the operands even though the coefficient is zero. The sign of the result will be positive.

If one of the operands is at the upper limit of the floating point range, the resulting sum may be exactly +1777 unbiased. In this case the resulting exponent will indicate the overflow condition, but the coefficient will be processed in a normal manner. No error indication is made for this case and no condition flags will be set in the PSD register. Subsequent use of this number as an operand in a floating point unit will, however, result in overflow detection.

If the exponents of both operands are zero (i.e., 2000), and no overflow occurs, the instruction effects an ordinary integer addition.

***31*** ***Floating difference Xj and Xk to Xi*** ***(15 Bits)***




This instruction forms the difference of the floating point quantities from operand registers Xj and Xk and packs the result in operand register Xi. Alignment and overflow operations are similar to the Floating Sum (30) instruction, and the difference is not necessarily normalized. The packed result is the upper half of a double precision difference.

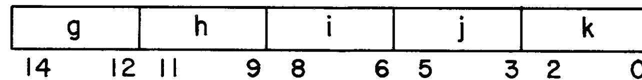
If the two operands are identical the resulting difference will have a zero coefficient. The exponent delivered to the Xi register will be the same as the exponent for the operands. The sign of the result will be positive.

If one of the operands is at the upper limit of the floating point range, the resulting difference may be exactly +1777 unbiased. In this case the resulting exponent will indicate the Overflow condition, but the coefficient will be processed in a normal

manner. No error indication is made for this case and no condition flags are set in the PSD register. Subsequent use of this number as an operand in a floating point unit will, however, result in overflow detection.

When the exponents of both operands are zero (i.e., 2000), an ordinary integer subtraction is performed.

*32* *Floating DP sum of Xj and Xk to Xi* *(15 Bits)*




This instruction forms the sum of two floating point numbers as in the Floating Sum (30) instruction, but packs the lower half of the double precision sum with an exponent 48 less than the upper sum. The result is not necessarily normalized.

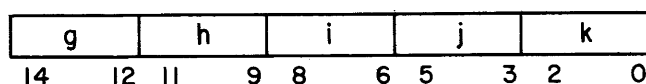
If one operand is at the upper limit of the floating point range, the resulting double precision sum may overflow and cause the exponent for the upper half to go out of range. Since the exponent for the lower half of the double precision sum is 48 less than this overflow value, the result delivered to the Xi register is processed as a normal floating point result and no error condition flags are set in the PSD register.

If the two operands are near the lower limit of the floating point range, the exponent for the lower half of the double precision sum may be exactly -1777 unbiased. This result is processed as a normal floating point number and no error condition flags are set in the PSD register. Subsequent use of this number as an operand in a floating point unit may, however, result in underflow detection.

If the exponent for the lower half of the double precision sum is less than -1777 unbiased, the result delivered to the Xi register is a complete underflow word with a zero coefficient. The sign of the result will be the same as the sign of the operand with the larger exponent. If the two operands have identical exponents the sign of the result

is the same as the sign of (Xk). The underflow condition flag is set in the PSD register for this case.

**33** *Floating DP difference of Xj and Xk to Xi* *(15 Bits)*

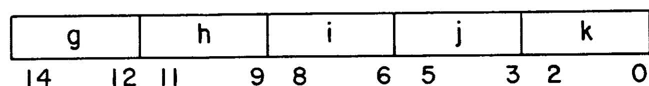



This instruction forms the difference of two floating point numbers as in the Floating Difference (31) instruction, but packs the lower half of the double precision difference with an exponent of 48 less than the upper difference. The result is not necessarily normalized.

If the two operands are identical the resulting double precision coefficient difference will be zero. This condition is not sensed as a special case and the exponent will be the same value as for a nonzero coefficient. The sign of the resulting zero coefficient will be positive.

For treatment of other special situations and operands refer to the description of instruction 32 Floating DP Sum.

**34** *Round floating sum of Xj and Xk to Xi* *(15 Bits)*

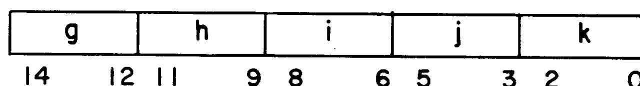



This instruction forms the round sum of the floating point quantities from operand registers Xj and Xk and packs the upper sum of the double precision result in operand register Xi. This instruction is intended for use in floating point calculations involving single precision accuracy. The result is not necessarily normalized.

Rounding of the operand coefficients occurs just prior to the double precision add operation. At this time the two 48-bit coefficients are positioned in the 99-bit ones complement adder with an offset corresponding to the difference of the exponents. A round bit is always added to the coefficient corresponding to the larger exponent. If the exponents are equal the round bit is added to the coefficient for (Xk). The round bit is equal to the complement of the sign bit and is inserted immediately to the right of the lowest order bit in the coefficient. This has the effect of increasing the magnitude of the coefficient by one-half of the least significant bit. A second round bit is added in a corresponding manner to the other coefficient if both operands were normalized, or if the operands had unlike signs.

For treatment of special situations, refer to instruction 30 Floating Sum.

***35*** ***Round floating difference of Xj and Xk to Xi*** ***(15 Bits)***



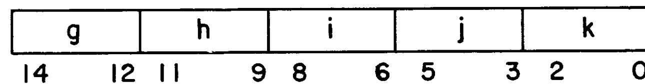

This instruction forms the round difference of the floating point quantities from operand registers Xj and Xk and packs the upper difference of the double precision result in operand register Xi. This instruction is intended for use in floating point calculations involving single precision accuracy. The result is not necessarily normalized.

Rounding of the operand coefficients occurs just prior to the double precision subtract operation. At this time the two 48-bit coefficients are positioned in the 99-bit ones complement adder with an offset corresponding to the difference of the exponents. A round bit is always added to the coefficient corresponding to the larger exponent. If the exponents are equal the round bit is added to the coefficient for (Xk). The round bit is equal to the complement of the sign bit and is inserted immediately to the right of the lowest order bit in the coefficient. This has the effect of increasing the magnitude of the coefficient by one-half of the least significant bit. A second round bit is

added in a corresponding manner to the other coefficient if both operands were normalized, or if the operands had like signs.

For treatment of special situations, refer to instruction 31, Floating Difference.

***40*** ***Floating product of Xj and Xk to Xi*** ***(15 Bits)***




This instruction multiplies two floating point quantities obtained from operand registers Xj (multiplier) and Xk (multiplicand) and packs the upper product result in operand register Xi.

In this operation, the exponents of the two operands are unpacked from the floating point format and are added with a correction factor of 48 to form the exponent for the result. The coefficients are multiplied as signed integers to form a 96-bit integer product. The upper half of this product is then extracted to form the coefficient for the result. An alternate output path is provided with a one-bit position displacement to normalize the result coefficient if the original operands were normalized and the double precision product has only 95 significant bits. The exponent for the result is corrected by one count in this case.

If the two operands are not both normalized the resulting double precision product will have less than 96 significant bits. No test is made for the position of the most significant bit in the product for this case. The upper 48 bits are read from the 96-bit positions in the double precision product register, and leading zeros will occur in the result coefficient. The alternate path is not used in this case even though the one-bit displacement may have normalized the result.

If the two operands are not both normalized the upper half of the double precision product may be all zeros. This situation is not sensed, and the exponent for the result

will be processed without regard to the zero coefficient. This will result in a zero coefficient and a nonzero exponent. No error flags are set in the PSD register for this case.

A partial overflow occurs for this instruction whenever the exponent computation results in exactly +1777 octal and the result coefficient is taken from a double precision product with 96 bits of significance. There are no error condition flags set in the PSD register for this case, and the result is delivered to the Xi register in a normal manner. Subsequent use of this result as an operand in a floating point unit will, however, result in overflow detection. However, if the coefficient is shifted one position to normalize it, the exponent delivered to the Xi register will be reduced one count and the result will be in floating point range.

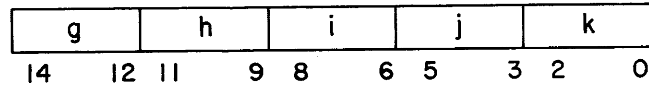
A complete overflow occurs for this instruction whenever the exponent computation results in an exponent greater than +1777 octal. This situation is sensed as a special case, and a complete overflow word with proper sign, overflow exponent, and zero coefficient is delivered to the Xi register. The coefficient calculation is ignored for this case, and the overflow condition flag is set in the PSD register.

A partial underflow occurs for this instruction whenever the exponent computation results in exactly -1776 octal and the result coefficient is shifted one position to normalize. The exponent delivered to the Xi register is reduced one count, creating an underflow exponent with a valid coefficient. There are no condition flags set in the PSD register for this case. Subsequent use of this result in a floating point unit may, however, result in underflow detection.

A complete underflow occurs for this instruction whenever the exponent computation results in less than -1776 octal. This situation is sensed as a special case, and a complete zero word with proper sign is delivered to the Xi register. The coefficient calculation is ignored in this case, and the underflow condition flag is set in the PSD register.

If either operand is zero, the underflow condition flag will set.

**41** *Round floating product of Xj and Xk to Xi* *(15 Bits)*



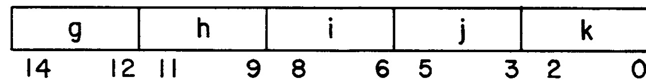

This instruction multiplies the floating point number from operand register Xk (multiplicand), by the floating point number from operand register Xj. The upper product result is packed in operand register Xi. (No lower product is available.) The multiply operation is identical to that of instruction 40 except that a rounding bit is added in bit position 46 of the product. The upper half of the product is then extracted to form the coefficient for the result. An alternate output path is provided with a one-bit position displacement to normalize the result coefficient if the original operands were normalized and the double precision product has only 95 bits of significance. The exponent for the result is corrected by one count in this case. The following rounded result is the net effect of this action:

- for products $\geq 2^{95}$, round is by one-fourth
- for all other products, round is by one-half

The result is a normalized quantity only when both operands are normalized; the exponent in this case is the sum of the exponents plus 47 (or 48).

The result is unnormalized when either or both operands are unnormalized; the exponent in this case is the sum of the exponents plus 48. For treatment of special situations and operands, refer to instruction 40, Floating Product.

42 *Floating DP product of Xj and Xk to Xi* (15 Bits)

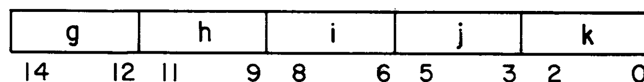



This instruction multiplies two floating point quantities obtained from operand registers Xj and Xk and packs the lower product in operand register Xi. The two 48-bit coefficients are multiplied together to form a 96-bit product. The lower-order 48 bits of this product (bits 47-00) are then packed together with the resulting exponent. The result is not necessarily a normalized quantity. The exponent of this result is 48 less than the exponent resulting from a 40 instruction using the same operands.

This instruction is intended for use in multiple precision floating point calculations. It may also be used to form the product of two integers providing the resulting product will not exceed 48 bits of significance. The operands must be packed in floating point format before executing this instruction. The result must be unpacked to obtain the integer product.

For treatment of special situations and operands, refer to instruction 40 Floating Product.

44 *Floating divide Xj by Xk to Xi* (15 Bits)




This instruction divides two normalized floating point quantities obtained from operand registers Xj (dividend) and Xk (divisor) and packs the quotient in operand register Xi.

The exponent of the result in a no-overflow case is the difference of the dividend and divisor exponents minus 48.

A one-bit overflow is compensated for by adjusting the exponent and right shifting the quotient one place. In this case the exponent is the difference of the dividend and divisor exponents minus 47. The result is a normalized quantity when both the dividend and divisor are normalized.

If the divisor is not normalized and the dividend coefficient is larger than the divisor by a factor of two or more, the quotient coefficient will be incorrect. The quotient is disregarded in this case, and the word delivered to the Xi register is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

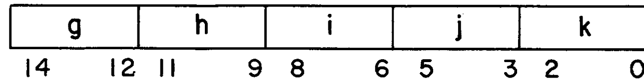
A partial overflow occurs for this instruction whenever the exponent computation results in exactly +1777 octal. There are no error condition flags set in the PSD register for this case, and the result is delivered to the Xi register in a normal manner. Subsequent use of this result as an operand in a floating point unit will, however, result in overflow detection.

A complete overflow occurs for this instruction whenever the exponent computation results in an exponent greater than +1777 octal. This situation is sensed as a special case, and a complete overflow word with proper sign, overflow exponent, and zero coefficient is delivered to the Xi register. The coefficient calculation is ignored for this case, and the overflow condition flag is set in the PSD register.

A partial underflow occurs for this instruction whenever the exponent computation results in exactly -1777 octal. There are no error condition flags set in the PSD register for this case, and the result is delivered to the Xi register in a normal manner. Subsequent use of this result as an operand in a floating point unit may, however, result in underflow detection.

A complete underflow occurs for this instruction whenever the exponent computation results in an exponent less than -1777 octal. This situation is sensed as a special case, and a complete zero word with proper sign is delivered to the Xi register. The coefficient calculation is ignored in this case, and the underflow condition flag is set in the PSD register.

**45** **_Round floating divide Xj by Xk to Xi_** **_(15 Bits)_**




This instruction divides the floating quantity from operand register j (dividend) by the floating point quantity from operand register Xk (divisor) and packs the round quotient in operand register Xi. The operation is the same as for a Floating Divide except that a round bit is added just below the lowest order bit of the coefficient from Xj. This round bit has the effect of increasing the magnitude of the dividend by one-half the value of the least significant bit.

The result is a normalized quantity when both the dividend and the divisor are normalized.

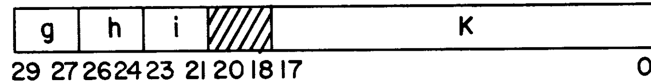
The result exponent in a no-overflow case is the difference of the dividend and divisor exponents minus 48.

A one-bit overflow is compensated for by adjusting the exponent and right shifting the quotient one place; in this case the exponent is the difference of the dividend and divisor exponents minus 47.

For treatment of special situations and operands, refer to instruction 44 Floating Divide.

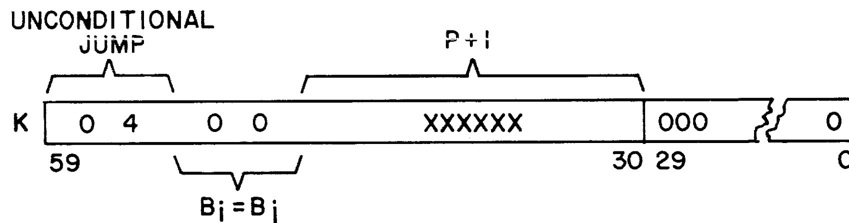
Branch

*010* **Return jump to K** *(30 Bits)*




This instruction stores an 04 unconditional jump and the current address plus one [(P) + 1] in the upper half of address K in SCM, then branches to K + 1 for the next instruction. The lower half of the stored word is all zeros. This instruction always branches out of the instruction stack and voids all instructions presently in the instruction stack.

The octal word at K after the instruction appears as follows:




This instruction is intended for executing a subroutine between execution of the current instruction word and the following instruction word. Instructions appearing after the return jump instruction in the current instruction word will not be executed. The called subroutine entrance address must be K + 1 in SCM. The called subroutine must exit at address K in SCM. A jump to address K of the branch routine returns the program to the original sequence.

Special Situations

If the value of K in a return jump instruction is greater than the SCM field length, the instruction is executed with the store of the exit word in SCM inhibited. The program

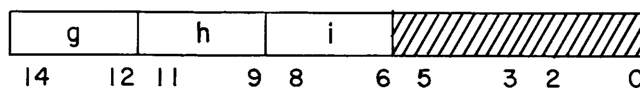
address is altered to the value K and advanced by one count in a normal manner. The program range condition flag is set in the PSD register to indicate the jump is out of range. The program sequence is then terminated with an exchange jump to (EEA). The resulting exchange package will contain a program address equal to K + 1, and a bit set in the PSD area corresponding to the range condition flag.

If the value of K in the return jump instruction is zero, the instruction is executed in a normal manner, and the exit word is stored at address zero in the SCM field. In the process of executing the instruction (P) is momentarily set to zero. This is sensed as an error condition, and the program range condition flag is set in the PSD register. As a result, the program sequence will be terminated at the completion of the return jump instruction with an exchange jump to (EEA). The return jump instruction will have advanced the program address one count so that the exchange package will indicate a program address of one rather than zero.

If the value of K in the return jump instruction is equal to (BPA), in the process of executing the instruction (P) will momentarily be set equal to (BPA). This will be detected as a breakpoint condition, and the breakpoint condition flag will set in the PSD register. The return jump instruction will advance (P) one count in the process of completing execution. This final value of (P) will appear in the exchange package when the breakpoint interrupt occurs.

An I/O section interrupt request may occur during the execution of a return jump sequence. In such a case the return jump instruction is completed, and an exchange jump to the proper I/O channel exchange package occurs with the program address equal to K + 1 from the return jump instruction.

***01300*** ***Exchange exit to NEA (Exit mode flag cleared)*** ***(15 Bits)***




An exchange exit instruction executed with the exit mode flag cleared causes the current program sequence to terminate with an exchange jump to address (NEA). This is an

absolute address in SCM and is generally not in the SCM field for the current program. The j and k designators in the instruction are ignored.

This instruction is intended for use in calling a system monitor program for input/output request, library calls etc. All operating register values, program address, and mode selections are preserved in the exchange package for the object program in order that the object program may be continued at a later time. The program address in the object program exchange package will be advanced one count from the address of the instruction word containing the exchange exit instruction. The monitor program will normally resume the object program at this address.

This instruction has priority over all other types of exchange jump requests. If an I/O interrupt request or an error exit request has occurred prior to the execution of this instruction, it is denied and the exchange jump specified by this instruction is executed. The rejected interrupt request is not lost, however, as the conditions that caused it will be reinstated when the exchange package enters its next execution interval.

The current contents of the instruction word stack are voided by the execution of this instruction.

There are no protective tests made on the exchange jump address for this instruction. The assignment of (NEA) is a responsibility of the system monitor program. If (NEA) has more than 16 bits of significance, the upper bits are discarded and the lower 16 bits used as the absolute address in SCM for the exchange jump.

**013** | ***Exchange exit to (Bj) ± K*** ***(Exit mode flag set)*** | ***(30 Bits)***

| g | h | i | j | K |
|---|---|---|---|---|
| 29 27 | 26 24 | 23 21 | 20 18 | 17 0 |

If the exit mode flag is set, this instruction causes the current program sequence to terminate with an exchange jump to an address in the SCM field for the current program. The exchange package is located at relative address (Bj) + K.

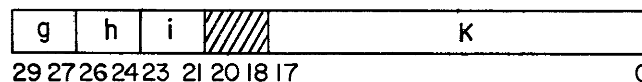
This form of the exchange exit instruction is intended to be privileged to a monitor program.

This instruction has priority over all other types of exchange jump requests. If an I/O interrupt request or an error exit request has occurred prior to the execution of this instruction, it is denied and the exchange jump specified by this instruction is executed. The rejected interrupt request is not lost, however, as the conditions that caused it will be reinstated when the exchange package enters its next execution interval.

The current contents of the instruction word stack are voided by the execution of this instruction.

There are no protective tests made on the exchange jump address for this instruction.

***02*** ***Jump to Bi + K*** ***(30 Bits)***




This instruction adds the contents of increment register Bi to K and branches to the relative SCM address specified by the sum. The remaining instructions, if any, in the current instruction word will not be executed. The branch address is K when i = 0.

Addition is performed in an 18-bit ones complement mode. The instruction word stack is not altered by execution of this instruction. The instruction is intended to allow computed branch point destinations. It is the only CPU instruction in which a computed parameter can specify a program branch destination address. All other jump instructions have preassigned destination addresses. Program modification to implement changes in a branch point destination address is not recommended in general because of complications associated with the instruction stack.

Special Situations

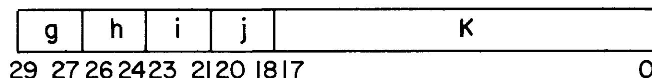
If an I/O interrupt request or an error exit request exists at the time this instruction is executed, the instruction is executed to completion before the interrupt occurs.

If the branch point destination address is greater than the SCM field length, the SCM range condition flag is set in the PSD register. The instruction will execute to completion, but the first instruction word for the next program sequence will not read from the IWS to the CIW register. At this point an Error interrupt will occur as a result of the SCM range condition flag, and an exchange jump will occur to address (EEA) in the SCM. The terminating exchange package will contain the out-of-range address in the program address field.

A jump to relative address zero in the SCM field causes the program range condition flag to set in the PSD register. The program will be terminated with an error exit to address (EEA). The terminating exchange package will contain a zero quantity in the program address field.

A jump to address (BPA) will set the breakpoint condition flag in the PSD register. The instruction will be executed to completion, and the exchange jump to address (EEA) will occur before the first instruction is executed at the branch point destination address.

| | | |
|---|---|---|
| 030 | *Jump to K if $Xj = 0$* | *(30 Bits)* |
| 031 | *Jump to K if $Xj \neq 0$* | *(30 Bits)* |
| 032 | *Jump to K if Xj is positive* | *(30 Bits)* |
| 033 | *Jump to K if Xj is negative* | *(30 Bits)* |
| 034 | *Jump to K if Xj is in range* | *(30 Bits)* |
| 035 | *Jump to K if Xj is out of range* | *(30 Bits)* |
| 036 | *Jump to K if Xj is definite* | *(30 Bits)* |
| 037 | *Jump to K if Xj is indefinite* | *(30 Bits)* |




These instructions cause the program sequence to branch to K or to continue with the current program sequence depending on the contents of operand register Xj. The decision will not be made until the Xj register is free.

The following applies to tests made in this instruction group:

1. The 030 and 031 operations test the full 60-bit word in Xj. The words 00.....00 and 77.....77 are treated as zero. All other words are non-zero. Thus, these instructions are not a valid test for floating point zero coefficients However, they can be used to test for underflow of floating point quantities.

2. The 032 and 033 operations examine only the sign bit ($2^{59}$) of Xj. If the sign is zero, the word is positive; if the sign bit is one, the word is negative. Thus, the sign test is valid for fixed point worde or for coefficients in floating point words.

3. The 034 and 035 operations examine the upper-order 12 bits of Xj. The following quantities are detected as being out of range:

   3777 X.....X (Positive Overflow)
   4000 X.....X (Negative Overflow)
   1777 X.....X (Positive Indefinite)
   6000 X.....X (Negative Indefinite)

   All other words are in range. An underflow quantity is considered in range. The value of the coefficient is ignored in making this test.

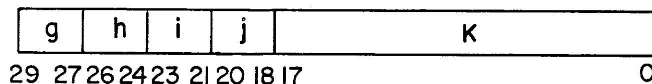
4. The 036 and 037 operations examine the upper-order 12 bits of Xj. Both positive and negative indefinite forms are detected:

   1777 X.....X and 6000 X......X are indefinite.

   All other words are definite. The value of the coefficient is ignored in making this test.

   For special situations, refer to the 02 Jump instruction.

| | | |
|---|---|---|
| *04* | *Jump to K if $Bi = Bj$* | *(30 Bits)* |
| *05* | *Jump to K if $Bi \neq Bj$* | *(30 Bits)* |
| *06* | *Jump to K if $Bi \geq Bj$* | *(30 Bits)* |
| *07* | *Jump to K if $Bi < Bj$* | *(30 Bits)* |



These instructions test an 18-bit word from register Bi against an 18-bit word from register Bj for the condition specified and branch to address K on a successful test. Otherwise, the program sequence continues. All tests against zero (all zeros) can be made by setting Bj = B0. The decision is not made until both B registers are free.

The following rules apply in the tests made by these instructions:

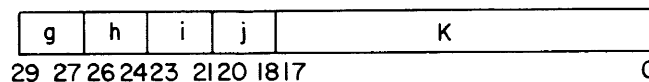
1. Positive zero is recognized as unequal to negative zero, and
2. Positive zero is recognized as greater than negative zero, and
3. A positive number is recognized as greater than a negative number.

The 06 and 07 instructions are intended for branching on an index threshold test. The tests are made in a 19-bit ones complement mode. The quantity (Bi) and the quantity (Bj) are sign extended one bit to prevent an erroneous result caused by exceeding the modulus of the comparison device. The quantity (Bj) is then subtracted from the quantity (Bi). The branch decision is based on the sign bit in the 19-bit result.

For special situations, refer to the 02 Jump instruction.

## Input/Output

*012* *Block Copy SCM to LCM* *(30 bits)*




This is a two parcel instruction in which the lower order 18 bits are used as an operand K. This instruction reads a sequence of 60-bit words from consecutive addresses in SCM and copies them into a block of consecutive addresses in LCM. The block of words begins at address (A0) in the SCM field. The words are stored in the LCM field beginning at address (X0). The number of words to be copied is determined by the sum of K + (Bj). This quantity, (K+(Bj)) cannot exceed $1777_8$ words. If a quantity larger

than this is used LCM truncates the quantity to the 10-bit maximum. Thus a block count of $3000_8$ words will transfer $1000_8$ words. No error indications are given when this occurs.

This instruction is intended to move a quantity of data from SCM into the LCM as quickly as possible. All other activity in the CPU, with the exception of I/O word requests, is stopped during this block transfer of data. All instructions which have issued prior to this instruction are executed to completion. No further instructions are issued until this block transfer is nearly completed. As a result of these restrictions the data flow from SCM to LCM can proceed at the rate of one 60-bit word each clock period. When an I/O multiplexer request for SCM occurs during this transfer, the data flow is interrupted for one clock period. The I/O word address is inserted in the stream of addresses to the SAS, and the addresses for the block transfer are resumed with a one clock period delay.

The length of the block is determined by adding the quantity K from the instruction to the contents of register Bj. Either quantity may be used to increment, or decrement, the other. The addition is performed in an 18-bit ones complement mode. The resultant sum is treated as an 18-bit positive integer. This 18-bit quantity is truncated to a 10-bit quantity by LCM. A zero result will cause this instruction to be executed as a pass instruction.

Three of the parameters for this instruction reside in operating registers (A0, X0, Bj). The contents of these registers are not altered by the execution of this instruction.

The lowest order 19 bits of (X0) are used to determine the initial address in the LCM field for the block copy. The higher order bits are ignored. If (X0) is negative the lowest order 19 bits are masked out and treated as a positive integer.

A test against LCM field length is made at the beginning of the block copy sequence. The length of the block is determined by adding the quantity K to (Bj) in an 18-bit ones complement mode. The resulting sum is treated as an 18-bit positive integer. This integer is added to the lowest order 19 bits of (X0), also treated as a positive integer. The resulting sum is compared with (FLL). If the resulting sum is greater than (FLL), indicating that the block copy will go beyond the assigned LCM field, the block copy is not executed. In this case the LCM block range condition flag is set in the PSD regis-

ter and the block copy instruction is issued as a pass. The exchange jump to (EEA) resulting from setting the LCM block range condition flag will not occur before execution of the next program instruction word unless a delay is introduced by subsequent instructions in the current instruction word.

A test against SCM field length is made at the beginning of the block copy sequence. The length of the block is determined by adding the quantity K to (Bj) in an 18-bit ones complement mode. The resulting sum is treated as an 18-bit positive integer. This integer is added to (A0), also treated as an 18-bit positive integer. The resulting sum is compared with (FLS). If the resulting sum is greater than (FLS), indicating that the block copy will go beyond the assigned SCM field, the block copy is not executed. In this case the SCM block range condition flag is set in the PSD register and the block copy instruction is issued as a pass. The exchange jump to (EEA) resulting from setting the SCM block range condition flag will not occur before execution of the next program instruction word unless a delay is introduced by subsequent instructions in the current instruction word.

Block Length Negative

The length of the block is determined by adding the quantity K from the instruction to the contents of register Bj. The addition is performed in an 18-bit ones complement mode. The resultant sum is treated as an 18-bit positive integer. A negative result will therefore appear as a large positive integer. In this case the SCM block range condition flag, and possibly the LCM block range condition flag, will set in the PSD register, indicating too large a block for the assigned fields. The block copy instruction will issue as a pass. The exchange jump to (EEA) resulting from setting the SCM block range condition flag will not occur before execution of the next program instruction word unless a delay is introduced by subsequent instructions in the current instruction word.

Block Length Zero

A zero block length is treated as a normal situation. No error flags are set. A block copy instruction is executed as a pass.

Last Parcel

The block copy instruction requires two parcels of an instruction word for normal use. If this instruction begins in the first, second, or third parcel of an instruction word the following parcel completes the instruction. If a block copy instruction begins in the last parcel of an instruction word it will not be continued in the following word. In this case the instruction will be executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

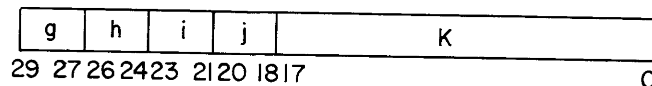
Error Condition During Execution

A LCM or SCM parity error may occur during the execution of a block copy instruction. An arithmetic error from a previous instruction may also occur during the beginning of the block copy sequence. If any error conditions occur, the proper flags are set in the PSD register, and the block copy instruction is executed to completion. There are no error conditions which will interrupt the instruction before completion.

Multiplexer Interrupt During Execution

An I/O section interrupt request may occur during the execution of a block copy instruction. In this case the interrupt request is not honored until the block copy instruction has been completed and any subsequent instructions in the current instruction word have been completed.

*011* *Block Copy LCM to SCM* *(30 bits)*




This is a two parcel instruction in which the lower order 18 bits are used as an operand K. This instruction reads a sequence of 60-bit words from consecutive addresses in LCM and copies them into a block of consecutive addresses in SCM. The block of words begins at address (X0) in the LCM field. The words are stored in the SCM field beginning at address (A0). The number of words to be copied is determined by the sum of K + (Bj). This quantity, (K + (Bj)) cannot exceed $1777_8$ words. If a quantity larger than this is used LCM truncates the quantity to the 10-bit maximum. Thus a block count of 3000 words will transfer $1000_8$ words. No error indications are given when this occurs.

This instruction is intended to move a quantity of data from the large core memory into SCM as quickly as possible. All other activity in the CPU, with the exception of I/O word requests, is stopped during this block transfer of data. All instructions which have issued prior to this instruction are executed to completion. No further instructions are issued until this block transfer is nearly completed. As a result of these restrictions the data flow from LCM to SCM can proceed at the rate of one 60-bit word each clock period. When an I/O Multiplexer word request for SCM occurs during this transfer, the data flow is interrupted for one clock period. The I/O word address is inserted in the stream of addresses to the SAS, and the addresses for the block transfer are resumed with a one clock period delay.

The length of the block is determined by adding the quantity K from the instruction to the contents of register Bj. Either quantity may be used to increment, or decrement, the other. The addition is performed in an 18-bit ones complement mode. The resultant sum is treated as an 18-bit positive integer. This 18-bit quantity is truncated to ten bits by LCM. A zero result will cause this instruction to be executed as a pass instruction.

Three of the parameters for this instruction reside in operating registers (A0, X0, Bj). The contents of these registers are not altered by the execution of this instruction.

The lowest order 19 bits of (X0) are used to determine the initial address in the LCM field for the block copy. The higher order bits are ignored. If (X0) is negative the lowest order 19 bits are masked out and treated as a positive integer.

<u>LCM Out of Range</u>

A test against LCM field length is made at the beginning of the block copy sequence. The length of the block is determined by adding the quantity K to (Bj) in an 18-bit ones complement mode. The resulting sum is treated as an 18-bit positive integer. This integer is added to the lowest order 19 bits of (X0), also treated as a positive integer. The resulting sum is compared with (FLL). If the resulting sum is greater than (FLL), indicating that the block copy will go beyond the assigned LCM field, the block copy is not executed. In this case the LCM block range condition flag is set in the PSD register, and the block copy instruction is issued as a pass with a four clock period execution time. The exchange jump to (EEA) resulting from setting the LCM block range condition flag will not occur before execution of the next program instruction word unless a

delay is introduced by subsequent instructions in the current instruction word.

### SCM Out of Range

A test against SCM field length is made at the beginning of the block copy sequence. The length of the block is determined by adding the quantity K to (Bj) in an 18-bit ones complement mode. The resulting sum is treated as an 18-bit positive integer. This integer is added to (A0), also treated as an 18-bit positive integer. The resulting sum is compared with (FLS). If the resulting sum is greater than (FLS), indicating that the block copy will go beyond the assigned SCM field, the block copy is not executed. In this case the SCM block range condition flag is set in the PSD register, and the block copy instruction is issued as a pass with a four clock period execution time. The exchange jump to (EEA) resulting from setting the SCM block range condition flag will not occur before execution of the next program instruction word unless a delay is introduced by subsequent instructions in the current instruction word.

### Block Length Negative

The length of the block is determined by adding the quantity K from the instruction to the contents of register Bj. The addition is performed in an 18-bit ones complement mode. The resultant sum is treated as an 18-bit positive integer. A negative result will therefore appear as a large positive integer. In this case the SCM block range condition flag, and possibly the LCM block range condition flag, will set in the PSD register, indicating too large a block for the assigned fields. The block copy instruction will issue as a pass with a four clock period execution time. The exchange jump to (EEA) resulting from setting the SCM block range condition flag will not occur before execution of the next program instruction word unless a delay is introduced by subsequent instructions in the current instruction word.

### Block Length Zero

A zero block length is treated as a normal situation. No error flags are set. The block copy instruction is executed as a pass with a four clock period execution time.

### LCM Words Already in Bank Operand Register

The LCM words required for the block copy instruction may already be in one of the LCM bank operand registers from the execution of a previous instruction. This

situation is not sensed. The words in the LCM bank operand register are discarded and are reread from the LCM bank.

## Last Parcel

The block copy instruction requires two parcels of an instruction word for normal use. If this instruction begins in the first, second, or third parcel of an instruction word the following parcel completes the instruction. If a block copy instruction begins in the last parcel of an instruction word it will not be continued in the following word. In this case the instruction will be executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

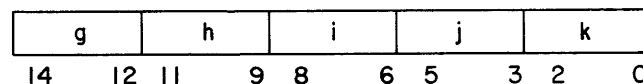
## Error Condition During Execution

A LCM or SCM parity error may occur during the execution of a block copy instruction. An arithmetic error from a previous instruction may also occur during the beginning of the block copy sequence. If any error conditions occur, the proper flags are set in the PSD register and the block copy instruction is executed to completion. There are no error conditions which will interrupt the instruction before completion.

## I/O Interrupt During Execution

An I/O multiplexer interrupt request may occur during the execution of a block copy instruction. In this case the interrupt request is not honored until the block copy instruction has been completed and any subsequent instructions in the current instruction word have been completed.

***014*** ***Read LCM*** ***(15 Bits)***




This instruction reads one word from the LCM and enters this word in an X register. The word is read from the LCM field at relative address (Xk). The word is then entered in register Xj. The SCM is not involved in this process.

This instruction is intended for direct addressing of the LCM for individual words. It may also be used to advantage in addressing a string of words in consecutive storage locations. This is particularly true if a string of words is to be read, modified, and written back into the same storage locations. The process of reading and writing will proceed in this case without a LCM read/write cycle delay until the addressing crosses a LCM bank boundary.

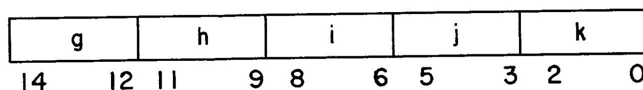
The lowest order 19 bits of (Xk) are used to determine the address in the LCM field. The higher order bits are ignored. If (Xk) is negative the lowest order 19 bits are masked out and treated as a positive integer. No error flags are set for these conditions unless the resulting address is out of range. The X0 register may be used for either Xj or Xk in this instruction. The j and k designators may have the same value in this instruction. In this case the requested address is lost when the word arrives at the Xj register.

This instruction is buffered to the extent that it issues in one minor cycle unless a previous LCM reference is in process. When this instruction issues the LCM busy flag is set and remains set until the requested word has been delivered to the designated X register. The destination X register is reserved in a manner analogous to a reference in that only one LCM read or write may be in process at one time.

Address Out of Range

The lowest order 19 bits of (Xk) are compared with (FLL) to determine if the requested address is in the assigned LCM field. If the requested address is greater than, or equal to, (FLL) the LCM direct range condition flag is set in the PSD register. This flag will cause an error exit request to interrupt the program with an exchange jump to address (EEA). The instruction will be executed in this case with a LCM read reference beyond the assigned field, and a word will be entered in the Xj register from this location. The absolute address in LCM for this reference will be the lowest order 19 bits in the sum resulting from adding (RAL) to the lowest order 19 bits of (Xk). The exchange jump resulting from the error exit request will generally not occur before one or more subsequent instructions have been executed.

**015** | *Direct Write LCM* | *(15 Bits)*




This instruction writes one word directly into LCM from an X register. The word is read from register Xj and is written into the LCM field at relative address (Xk). The SCM is not involved in this process.

This instruction is intended for direct addressing of the LCM for individual words. It may also be used to advantage in addressing a string of words in consecutive storage locations. This is particularly true if a string of words is to be read, modified, and written back into the same storage locations. The process of reading and writing will proceed in this case without a LCM bank read/write cycle delay until the addressing crosses a LCM bank boundary.

The lowest order 19 bits of (Xk) are used to determine the address in the LCM field. The higher order bits are ignored. If (Xk) is negative the lowest order 19 bits are masked out and treated as a positive integer. No error flags are set for these conditions unless the resulting address is out of range. The j or k designators may be zero or both may be the same value.

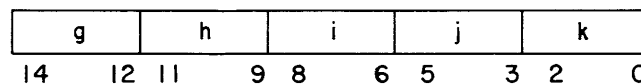
This instruction is buffered to the extent that it issues in one minor cycle unless a previous LCM reference is in process. When this instruction issues the LCM busy flag is set and remains set until the word has been delivered to the proper LCM bank operand register. No X register reservations are made for this instruction. The following instruction may issue in the next clock period and may use either of the X registers designated in this instruction. If the word cannot be entered immediately in the proper LCM bank operand register it is held in the LCM write register until the LCM bank operand register is free. This process differs from a SCM write reference in that only one LCM read or write may be in process at one time.

Address Out of Range

The lowest order 19 bits of (Xk) are compared with (FLL) to determine if the requested

address is in the assigned LCM field. If the requested address is greater than, or equal to, (FLL) the LCM direct range condition flag is set in the PSD register. This flag will cause an error exit request to interrupt the program with an exchange jump to address (EEA). In this case the word will not be written into LCM. The exchange jump resulting from the error exit condition will generally not occur before one or more subsequent instructions have been executed.

***0160*** ***Reset input buffer*** ***(15 Bits)***




This instruction is used to initiate a new record transmission from a PP to SCM. It resets the channel (Bk) input buffer in the multiplexer (MUX) in preparation for the next incoming record. The channel (Bk) input buffer address register is cleared to zero. The channel input assembly register is reset to first position.

This instruction is intended to be privileged to an input routine; that is,one which terminates a record of incoming data and prepares for the next record. A routine is called by an I/O section interrupt request when the record flag is set on the channel input data path. The data in the channel input buffer is removed and this instruction is executed to clear the buffer for the next incoming record.

This instruction is effective only if the monitor mode flag is set in the Program Status register. If the monitor mode flag is cleared this instruction becomes a pass instruction. When this instruction issues it will execute the required channel functions without regard to the current status or activity at the channel input register.

This instruction is intended to be executed in response to a multiplexer interrupt request resulting from the setting of the channel input record flag. The record flag is cleared when the interrupt request is generated. Further entries to the channel input buffer are not locked out by the interrupt request flag in the channel access control during the execution interval for the interrupt exchange package. The PPU should wait for a positive response through a programmed output over the output channel before beginning the next record.

The lowest order four bits of (Bk) are used in this instruction. The higher order bits are ignored. If higher order bits are set in (Bk) the lowest order four bits are masked out and used to determine the channel number. If (Bk) = 0, this instruction becomes a pass instruction.

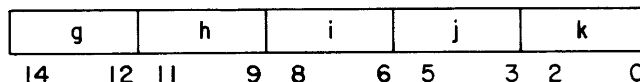
If the monitor mode flag is not set in the PSD register when this instruction is executed, this instruction becomes a pass instruction.

The channel input buffer is normally inactive when this instruction is executed because the PP has transmitted a record flag and is waiting for monitor response on the output channel. If the PPU has for some reason continued transmitting data, a word may be waiting to enter the channel input buffer and a word request flag may be set. These two operations may occur in the same clock period with conflicting commands to the registers from the multiplexer. In this case the commands associated with this instruction take priority, and the result is a loss of data in the input buffer for the incoming record. The incoming record will continue in this case with no indication of error except that the record will be shortened by the lost data.

Two or more reset input buffer instructions may occur in consecutive program instruction locations referencing different channels. These instructions may issue in consecutive clock periods, and no interference will result in the multiplexer.

Two or more reset input buffer instructions may occur in consecutive program instruction locations referencing the same channel. These instructions will issue in consecutive clock periods and repeatedly perform the same functions. No interference will occur other than the obvious repetitive functions.

**016** ***Read channel input status: j nonzero*** ***(15 Bits)***
***(Read real time clock: (Bk) = 0)***




This instruction reads the current value of the channel (Bk) input buffer address register contents to register Bj. The status of the channel (Bk) input buffer address register is not altered.

This instruction is intended for use in monitoring the progress of the channel input buffer in the multiplexer. The channel input buffer area is divided into two fields by the threshold testing mechanism. The first half of the buffer area constitutes one field and the last half of the buffer area the other field. An I/O threshold interrupt request is generated by the threshold testing mechanism whenever the channel input buffer address is advanced across a field boundary. This will occur at the center of the buffer area and at the end of the buffer area.

This instruction is the only vehicle for a program to determine whether an I/O section interrupt request was generated by a buffer threshold test or by a record flag. The program must retain the buffer address from one interrupt period to the next. If the buffer address is in the same field as for the previous interrupt, the interrupt request was from a record flag. If the buffer address is in the opposite field from the previous interrupt, the interrupt request was from a threshold test.

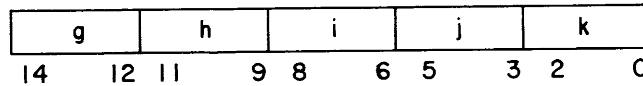
The lowest order four bits of (Bk) are used in this instruction. The higher order bits are ignored. If higher order bits are set in (Bk) the lowest order four bits are masked out and used to determine the channel number. If (Bk) = 0, this instruction reads the contents of the CPU clock period counter.

Two or more read channel input status instructions may occur in consecutive program instruction locations referencing the same or different channels. These instructions may issue in consecutive clock periods providing the Bj register reservations do not cause a delay. No interference will result in the multiplexer in these situations.

Real Time Clock

This instruction has a special use if the channel number (Bk) is zero. There are no buffer areas for the MCU which use the I/O section channel zero access position. In this case the current contents of the CPU clock period counter are read into the Bj register. This is a 17-bit counter which is advanced one count in a two's complement mode each clock period. This count is intended for timing measurements in the CPU program. Timing considerations for this special use are the same as the normal timing from a channel input buffer address register.

*0170* **Reset output buffer** *(15 Bits)*




This instruction is used to initiate a new record transmission from SCM to a PP. It resets the channel (Bk) output buffer in preparation for the next record transmission. The channel (Bk) output buffer address register is cleared to zero. A record pulse is transmitted over the channel output data path and a SCM reference is initiated.

This instruction is intended for execution in an output routine to initiate a new record transmission over a channel output data path. The channel output buffer is normally inactive when this instruction is executed. The channel output buffer is loaded with the data for the next record, and this instruction is executed to initiate the transmission. A record pulse is transmitted at the time this instruction is executed to indicate the beginning of a new record. The first word of data will follow as soon as the SCM word is entered in the channel output disassembly register.

This instruction is effective only if the monitor mode flag is set in the Program status register. If the monitor mode flag is cleared this instruction becomes a pass instruction. When this instruction issues it will execute the required channel functions without regard to the current status or activity at the channel output register. The channel output disassembly register is reset by the channel output word request flag.

The lowest order four bits of (Bk) are used in this instruction. The higher order bits are ignored. If higher order bits are set in (Bk) the lowest order four bits are masked out and used to determine the channel number. If (Bk) = 0, this instruction becomes a pass instruction.

If the monitor mode flag is not set in the PSD register when this instruction is executed, this instruction becomes a pass instruction.

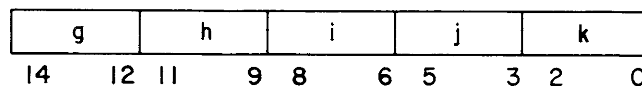
The channel output buffer is normally inactive when this instruction is executed because

a program should have checked for completion of the previous record before beginning this routine. There are two methods that a program can use to detect end of record. One method is to read the channel output buffer address and compare with a known record length. The other is a positive response from the peripheral unit over the corresponding channel input data path. If for some reason the channel output buffer is actively moving data over the channel output data path at the time this instruction is executed, conflicting commands may be sent to the channel registers. In this case the commands associated with this instruction have priority, and the result is a loss of data in the previous record.

Two or more reset output buffer instructions may occur in consecutive program instruction locations referencing different channels. These instructions may issue in consecutive clock periods and no interference will result in the multiplexer control.

Two or more reset output buffer instructions may occur in consecutive program instruction locations referencing the same channel. These instructions will issue in consecutive clock periods and repeatedly perform the same functions. A record pulse will be transmitted over the channel output data path for each instruction execution. The channel output buffer will be repeatedly restarted, and a data word may, or may not, be transmitted over the channel output data path depending on the timing of the instructions and the conflicts that occur.

***017*** ***Read channel output status: j nonzero*** ***(15 Bits)***




This instruction reads the current value of the channel (Bk) output buffer address register contents to register Bj. The status of the channel (Bk) output buffer address register is not altered.

This instruction is intended for use in monitoring the progress of the channel output buffer. The channel output buffer area is divided into two fields by the threshold testing mechanism. The first half of the buffer area constitutes one field and the last half of the buffer area the other field. A multiplexer interrupt request is generated by

the threshold testing mechanism whenever the channel output buffer address is advanced across a field boundary. This will occur at the center of the buffer area and at the end of the buffer area.

The lowest order four bits of (Bk) are used in this instruction. The higher order bits are ignored. If higher order bits are set in (Bk) the lowest order four bits are masked out and used to determine the channel number. If (Bk) = 0, this instruction reads all zeros into Bj.

Two or more read channel output status instructions may occur in consecutive program instruction locations referencing the same or different channels. These instructions may issue in consecutive clock periods providing the Bj register reservations do not cause a delay. No interference will result in the multiplexer in these situations.

# 4. CENTRAL PROCESSOR MEMORY

## INTRODUCTION

The Central Processor Unit contains two memories: Large Core Memory (LCM) and Small Core Memory (SCM). In the following description the term Central Processor Unit (CPU) implies that part of the Central Processor hardware which does not contain the memory or its directly associated hardware.

## MEMORY PROTECTION

All Central Processor Unit references to either SCM or LCM are made relative to a reference address (Figure 4-1). The reference address defines the lower limit of the program and/or data. All references from the program must lie within a range defined at its lower limit by the reference address and at its upper limit by the reference address added to program field length. The field length is the number of 60-bit words comprising the program and it is established by the programmer prior to execution.

During an Exchange Jump, the reference addresses and the field lengths are loaded from the exchange jump package into the respective registers to define the limits of the program.

When the program specifies an address, it is automatically added to the reference address. This new address is then checked to see if it lies within the bounds specified above. If it does, the program proceeds normally; if it does not then an unconditional exit is made and the program is terminated. These constraints are applied to both SCM and LCM addresses. Therefore, two reference addresses and two field lengths are required; one for each memory.

Two error-conditions are noted. One will occur if the requested address is outside the limits defined above. The other will occur if a block transfer between SCM and LCM will, during its execution, cause a reference to an address outside of the limits defined above. When one of the error conditions occurs one of two flags is set to indicate whether that error is in LCM or SCM.

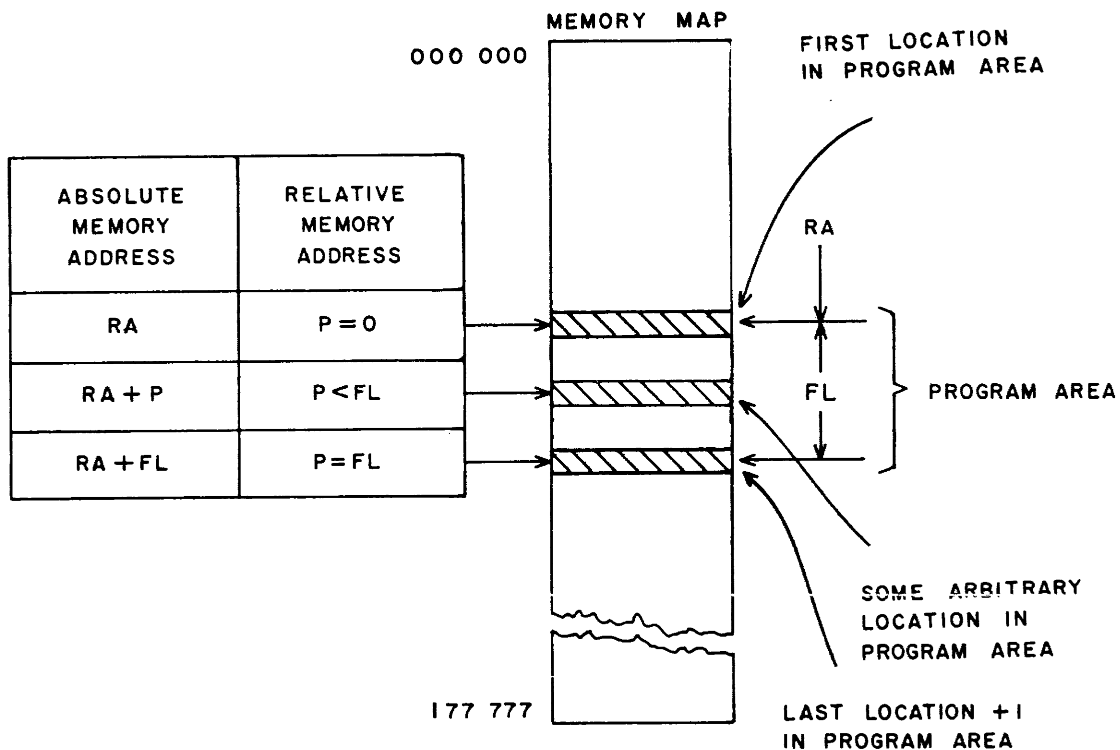



Figure 4-1. Memory Map

# SMALL CORE MEMORY

## Organization

Small Core Memory (SCM) is organized into 65K, 60-bit words, (plus 5 parity bits) in 32 banks of 2,048 words each. The banks are logically independent and consecutive

addresses go to different banks. Banks may be phased into operation at clock period intervals, resulting in very high operating speed. Up to 10 banks may be in operation at one time. The SCM address and data control permit a word to move to or from SCM every clock period. A parity error in SCM sets a flag in the Program Status register.

## Address Format

The location of each word in SCM is identified by 16-bit address. The address format is shown below (Figure 4-2). Within the address format, the lowest 5 bits specify one of 32 banks. The 11-bit address defines one of 2048 separate locations within the specified bank. Addresses that are numerically consecutive reference consecutive banks and hence make most efficient use of the bank phasing.

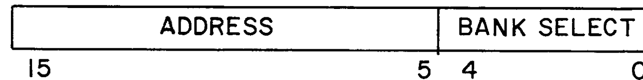



Figure 4-2. SCM Address Format

## Small Core Memory Access

<u>Introduction</u>

Logically, SCM is the center of the Computer System with communication to the CPU, LCM and the PPU.

PPU access is limited to certain buffer areas in the low order addresses of SCM. These are used for data transfers and for monitor communication between the CPU. The PPU can write into these areas at any time. It is the responsibility of the CPU programs to empty or fill the PPU buffer areas. The PPU can read the buffer areas only when directed to do so by the CPU.

The CPU can reference any part of SCM subject to the constraints of Field Length and Reference Addresses.

The LCM can communicate with any part of SCM. However, it is designed to copy large blocks of data to, or from, SCM. Although a block of data may be as small as one word, such transfers are time consuming. For this reason, there is the capability to reference single words from LCM directly into the CPU. LCM access is described under the LCM Section (page 4-10).

## Memory Reference

When a storage reference is initiated the address is sent to all banks in the memory, and the correct bank, if free, accepts the address. If the bank is busy the request waits until that bank is free. Requests for 3 addresses may be waiting for SCM at the same time. Instruction issue stops when the second address is sent to SCM and the first address sent has not been accepted. Instruction issue does not start again until all unaccepted addresses have been accepted by SCM (up to 3 addresses).

It is possible to abort a valid SCM memory write which is followed by an SCM write out of range. The case in which this can happen is as follows: write SCM bank X, write SCM bank X, write SCM out of range. Due to the 2 valid writes going to the same bank, the second write to bank X is held up in the SCM access control. However, the range check is made immediately after the out of range address is formed by the increment unit. Thus SCM direct range error bit in PSD register will be set before the second write to bank X can be initiated (page 2-18). Since the SCM direct range error stops any write into SCM both the second write to bank X (which is valid), and the write out of range will be aborted.

All addresses presented to SCM are processed in the order in which they are received. SCM requests from various parts of the computer are given a priority which determines which addresses shall be allowed access first.

These priorities are as follows:

1. Exchange Sequence Request
2. Increment Unit Request
3. Return Jump Exit Request
4. Input/Output Section Request
5. Read Next Instruction Request (RNI)
6. Read Next Instruction Request (RNI)

All memory references appear the same to SCM. The hardware provides tags that identify the source or destination of any word that is stored or read.

## Central Processor Unit Access

The CPU can access SCM in two ways, either from an A register or from the instruction stack.

### A Register Access

The eight A registers are divided into five read registers (A1-A5) and two write registers (A6, A7). A0, the remaining register, cannot be used in this manner. Placing a quantity into an A register causes a reference to that SCM location. If the A register is A6 or A7, the corresponding X register is stored in SCM. If the A register is A1-A5, the corresponding X register is loaded with the contents of that memory location.

### Instruction Stack Access

References to SCM are made by the stack advancing, by a branch out of the stack or an instruction request.

## I /O Multiplexer

### Introduction

The Multiplexer buffers data to (or from) the directly connected PPU. PPU's connected with the CPU communicate over a 12-bit full duplex channel. In the CPU, each channel has assembly and disassembly registers to convert the 12-bit channel data to 60-bit CPU words.

There are a total of 15 channels in the Multiplexer. These channels are numbered beginning with $01_8$ and ending with $17_8$. Each channel has a SCM buffer area for incoming data and a separate SCM buffer for outgoing data. In addition each channel has an exchange package for incoming data and an exchange package for outgoing data. Each buffer area is divided into two fields, a lower field and an upper field. Data is entered (or removed) from the buffer area in a circular mode. The last word in the lower field is followed by the first word in the upper field. The last word in the upper

field is followed by the first word in the lower field. Whenever a buffer area has been filled (or emptied) to the point where a field boundary is crossed, the CPU is interrupted and the associated exchange package initiates a program to process the buffer data. The channel continues to fill (or empty) the other buffer field while the CPU is processing this buffer data.

The I/O exchange package areas are permanently assigned in the lower order addresses of SCM. These areas are arranged as shown in Figure 4-3. The I/O section buffer areas are assigned in higher order address positions of SCM. These areas may be changed both in size and order (wiring change) to accommodate various types of channel volume. A typical arrangement for the buffer areas is shown in Figure 4-4. Total I/O section space in SCM cannot exceed absolute address $10,000_8$.

The Maintenance Control Unit is assigned to channel 0 but is different from the normal PPU access (see Manual Control, page 7-1).

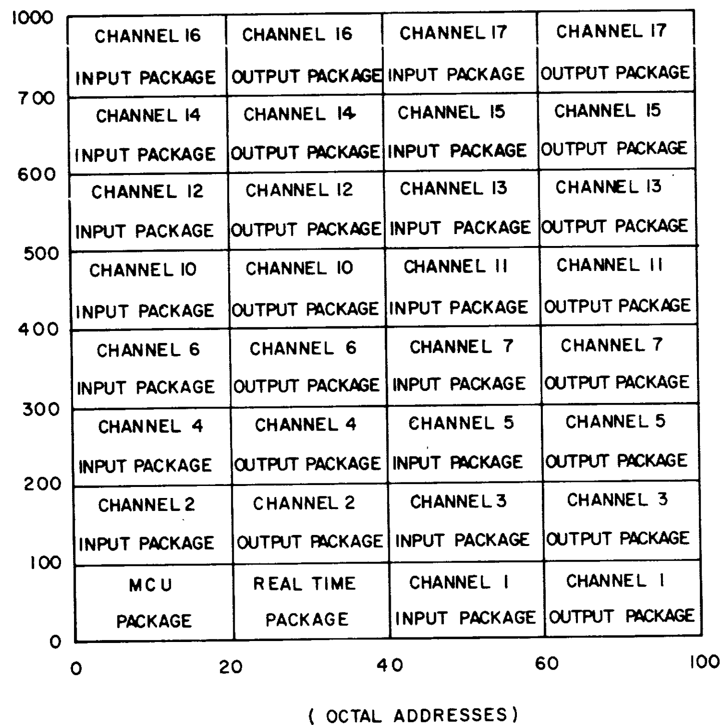



Figure 4-3. I/O Exchange Package Areas

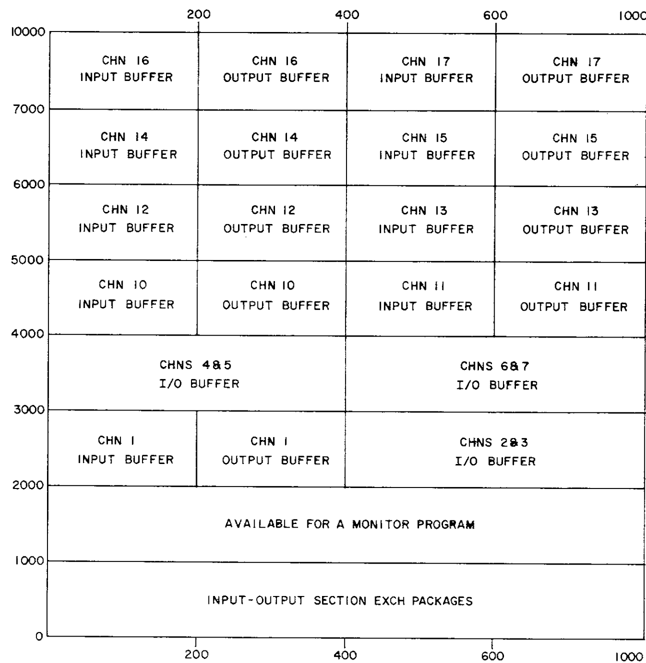



Figure 4-4. Buffer Area Arrangements

Example of PPU to SCM Data Transfer

Assume that the PPU has been notified that it must enter data into SCM. This must be done by the monitor transmitting a message over the associated output channel to the PPU. The CPU must clear the I/O control with a Reset Input Buffer instruction (0160). This sets the assembly counter and the address counter to zero. Each transfer should be preceded by this instruction.

The PPU transmits 12-bit words to the control which assembles them into 60-bit words. Each 12-bit transfer is accomplished by the usual scheme of word and record flags and resume signals. (See Peripheral Processors, page 5-3.) When a 60-bit word has been assembled from five 12-bit words a request is made for SCM access.

The PPU must wait until this is accomplished, which may take from a few clock periods to a considerable length of time depending on SCM conflicts. Once the word has been accepted by SCM the transmit and assemble procedure is repeated again.

When the PPU has transmitted enough words to half fill its assigned buffer area, a midpoint threshold interrupt is generated in the SCM access control. This causes an exchange jump to the channel input package which then proceeds to empty the buffer into LCM. When the CPU begins to copy data from SCM to LCM the issue of all new instructions is stopped, but existing instructions are allowed to complete execution. This allows the CPU to transfer data one 60-bit word every clock period. The only interruptions allowed are those from any PPU that is writing into SCM. Because of this the CPU should be able to empty the buffer area fifty times faster than the PPU can fill it.

When the PPU comes to the end of its buffer area a second interrupt request will be transmitted to the CPU. The CPU will again start emptying the buffer area into LCM. Since PPU records may be of any lengths, the PPU will now start entering new data at the lower end of the buffer area. At this point, the CPU may have not responded to the midpoint threshold interrupt. Therefore, should this condition occur (both threshold interrupt flags set) no further PPU data transmission is allowed.

The PPU terminates the data transfer by setting the Input Record flag on the CPU channel. If the PPU has not transmitted sufficient 12-bit words to form a complete 60-bit word then the 60-bit word is sent to SCM with the remaining parcels filled with zeros. Before setting the Record flag the PPU should check to see that the last word was accepted by SCM.

The PPU must not begin transmitting a new record of input data before the CPU has completed processing the data in the SCM buffer area. There is no hardware provision to prevent the PPU from doing this. Therefore, the PPU program must wait for communication through its input channel before starting a new record. Should it start a new record, it could destroy the End of Record condition already established in the SCM buffer area.

Note that an End of Record will establish an interrupt condition of the CPU. However, the CPU cannot determine whether this is a Threshold interrupt or an End of Record interrupt. The CPU program has to keep track of which buffer threshold boundary caused the last interrupt. If the interrupt occurs without crossing another buffer threshold boundary it is an End of Record interrupt.

Example of a SCM to PP Data Transfer

Assume that the PPU has been notified that it must read data from SCM. The CPU must clear the I/O section with a Reset Output Buffer instruction (0170). This sets the Assembly counter and the Address counter to zero. It also initiates a SCM reference to bring the first 60-bit word from the buffer area to the disassembly register. At the same time, the instruction sends a Record pulse to the PPU to indicate that the transmission is able to start. The upper 12-bit parcel of the 60-bit word is then held on the data lines and a Word Pulse signal is sent to set the Word flag in the PPU. The PPU accepts the word and returns a Resume signal to SCM. When the fifth (last) parcel is ready to be transmitted to the PPU a SCM reference is started to bring the second 60-bit word from the buffer area to the channel register. This sequence is accomplished by the hardware and once initiated by the CPU requires no further CPU activity instructions to be executed. As the thresholds are passed in the buffer area the CPU exchange package refills the buffer. This will happen as long as PPU requests data. It is therefore necessary that the PPU program must know the record length by prearranged data communication by the monitor or by some convention.

When the PP has received the expected amount of data it stops reading from the PPU input channel and this terminates SCM activity. However, the PP program must sense the Record flag on its input channel to determine whether the CPU has cleared the SCM buffer area and begun a new record transmission. This, the CP program must do by executing a Reset Output Buffer instruction again.

# LARGE CORE MEMORY

## Organization

Large Core Memory is a 2-wire, word organized memory of 1.76 μsec cycle time. Parity checking is provided with one parity bit for each 15 data bits. It is designed to provide a large amount of storage that emphasizes rapid transfer of data rather than high speed random access. It has a capacity of 512,000 60-bit words. These are arranged in 8 banks of 64,000 words each. Within the bank, eight 60-bit words are grouped into one LCM word and a parity bit is added for each 15 bits. A memory reference reads all eight 60-bit words simultaneously, which are then held in a 480-bit register. Since there is one 480-bit register per bank these 8 registers can hold a total of 64, 60-bit words.

The banks are phased to provide maximum data transfer rate for block copy instructions. Each bank is started at intervals of eight clock periods. Since each bank provides eight 60-bit words per reference this means that a maximum transfer rate of one 60-bit word per clock period is possible. LCM is provided with 4 reserve sense lines per bank (stack). Also, there are 12,288 reserve 60-bit words for LCM (1525 words per bank). These can be exchanged in groups of 128 60-bit words with groups in the bank and is done by changes to the stack wiring. A parity error in LCM sets a bit in the Program Status register (page 2-14).

## Large Core Memory Access

LCM can be accessed in two ways: either by block transfers between the SCM or by single word transfers between the CPU.

### SCM Access (Block Transfers)

Block transfers or copies are done by a block read or by a block write instruction. Upon issue of a block copy instruction, all previously issued instructions are allowed to complete execution but no new instructions are issued. This ensures that the copy will proceed at maximum speed. However, the PPU access channels may still read

or write into SCM. Since LCM will have ten SCM banks active when transferring at maximum rate, this leaves 22 banks free for PPU access. Note that there is no constraint on a PPU accessing the same portion of SCM that is being used for a block transfer. However, an I/O interrupt will not be processed until the block copy is complete.

CPU Access (Single Word Transfers)

Single word transfers store, or load, one 60-bit word from, or into, the designated X register. The first reference to a bank will require a complete clock period; however, subsequent references to the next seven words do not require the memory reference because these words are held in the bank registers. In this case, the access time is only 3 clock periods. Furthermore, there are 8 banks of LCM giving 8 60-bit registers; effectively a 64-word register memory. By loading instructions into SCM and retaining operands in LCM the operands can be used at the following rate:

| | |
|---|---|
| First word | 16 clock periods |
| Second through eighth word (Register Access) | 3 clock periods each |

Repeat for remaining eight word groups.

This means that operands are available, on an average, every 5 clock periods as opposed to operands from SCM at 8 clock periods.

# 5. PERIPHERAL PROCESSOR UNIT

## ORGANIZATION

Each Peripheral Processor Unit (PPU) is a completely independent and self-contained computer. Therefore, each PPU may be executing a different program at the same time. A PPU's primary function is to perform I/O tasks at the request of the Central Processor Unit. The standard system configuration has 7 peripheral processor units. One of these seven is designated the Maintenance Control Unit (MCU). It is identical to the other PPU's except it has specific, invariant channel connections. These channel connections may be made to other PPU's and to the CPU to monitor error conditions or to dead start them (see MCU, Manual Control, page 7-1).

The PPU has three major sections: the processor, memory, and I/O sections.

### Processor

The processor is of conventional organization with an accumulator directing operands to, or accepting operands from an adder. Two other adders are provided for use in arithmetic operations associated with indirect addressing.

The processor has seven basic registers. These are the A, P, Q, X, Sk, K and Fd. registers.

#### A Register (18 bits)

The arithmetic or A register is the principal operand register. The contents of A are treated as signed operands. If bit 17 is set the operand is negative. Overflows are ignored although an end around carry may show in the register at the end of an instruction execution. No sign extension is provided for 6-bit or 12-bit quantities which are entered in the low order bits. However, the unused upper bits are cleared to zero. Zero is represented by all zeros. The A register is used in the shift, logical arithmetic and four I/O instructions.

P Register (12 bits)

The Program Address register or P register holds the address of the current instruction. During the execution of the current instruction the contents of P are advanced by 1 or 2 to provide the address of the next instruction in the program for 12- or 24-bit instructions. If a jump is called for, the Jump address is entered in P.

Q Register (12 bits)

The Q register has two major functions. It is primarily used for holding the address of an operand during instruction execution. The secondary purpose is to hold the upper 6 bits of an 18-bit operand in the lower 6 bits of the register during operand arithmetic.

X Register (12 bits)

This register holds all data read from memory. It also is used during 18-bit arithmetic operations in the A register. It holds the lower 12 bits of the operand during these instructions.

Sk Register (6 bits)

The Sk register contains a shift count during shift instructions.

Fd Register (12 bits)

The Fd register holds the current instruction word for translation.

K Register (3 bits)

The K register is the instruction cycle counter and is used to count the number of memory references required during indirect addressing.

Note that, of all the registers listed above, only the A register is used directly by the programmer.

NOTE

Program loops in the PPU should be four words or longer to avoid reducing memory margins.

## Memory

Each processor has its own 12-bit, 4096-word, magnetic core, random access memory with a cycle time of 275 nanoseconds. Each 12-bit word has a parity bit attached.

The memory is organized into two banks and consecutive addresses alternate between these banks to increase processing speed. The memory consists of four 12-bit modules, each of 1024 words. Two of these modules form one memory bank. Associated with each bank is an S register which holds the address of the operand in storage, a Z register which holds operands to be stored and the X register which receives operands read from either bank. There are, therefore, two Z and two S registers for each PPU. Associated with each Z register is a parity generating circuit that generates an odd parity bit and this is stored in the memory with the operand. Parity is checked on reading operands from memory. In the event of a parity error, the PPU sends a parity error signal to the Maintenance Control Unit (MCU).

## Input/Output

The PPU's communicate with the CPU and with other devices over channels which operate in a full duplex mode. That is, information may be transmitted from the PPU to a particular device at the same time that information is being received from that device. Each full duplex channel consists of an input data path and an output data path plus the associated control lines for each path. The full duplex channel consists of two physical cables. Each cable handles data moving in one direction and contains the control lines associated with that data. The two cables are completely symmetrical.

### Word Flag

A word flag is normally a one clock period pulse transmitted over a cable to notify the receiving device that the 12 data lines contain new information which is ready to be sampled. In special situations the word flag is forced to a continuously set condition. In this case the data may be sampled by the receiving device at any time. There is no coordination between transmitter and receiver in this case, and the receiver must interpret the data to exact information on time changes.

Record Flag

A record flag is normally a one clock period pulse transmitted over a cable to notify the receiving device that a record of data transmission has been completed. In cases where the word flag is continuously set the record flag is not used.

Resume

A resume is normally a one clock period pulse transmitted from the data receiving device back to the data transmitting device to indicate that the data on the cable has been sampled and the next data may be placed on the lines. In special situations the resume may be forced to a continuously set condition. In this case the data transmitting device may send new data at its own rate. There is no coordination between transmitter and receiver in this case, and the receiver must be ready to accept each 12-bit data word as transmitted.

Input Channels

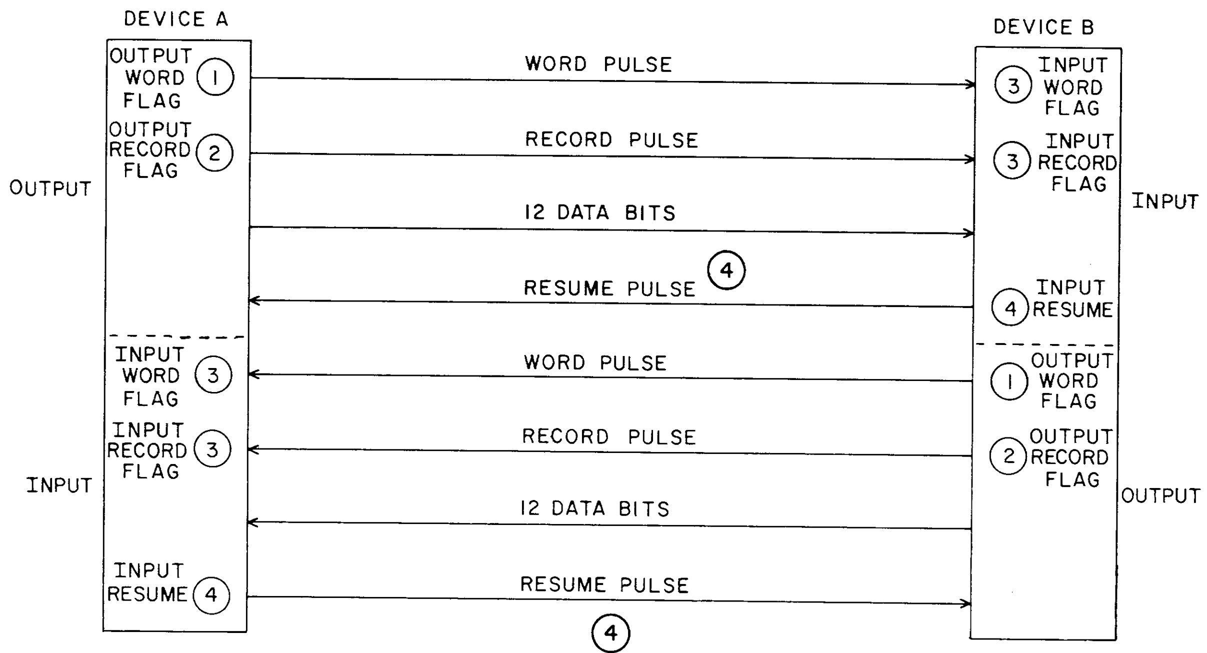
There are provisions for eight input cables in each PPU. Each input cable provides 12 bits of incoming data and the associated control lines for that data. The PPU may sample the data on any one of these eight input cables at any one time. The selection of which one of the eight cables is determined by the lowest order three bits in the d register. The input channels are numbered zero through seven to correspond with the value of the lowest order three bits in the d portion of the Fd register. The input data and input control lines are illustrated in Figure 5-1.

Input Word Flag

This flag is set when a word pulse is transmitted over the input cable to the PPU. The flag is cleared when the PPU has sampled the data on the cable and sends a resume pulse to the transmitting device at the other end of the cable. This flag is also forced to a cleared state during a Dead Start condition.

Input Record Flag

The flag is set when a record pulse is transmitted over the input cable to this PPU. The flag is cleared when the PPU has sampled the next following input data word and




Key

| | | |
|---|---|---|
| 1. | Set by any output instruction | cleared by a resume pulse |
| 2. | Set by output record flag instruction (74) | cleared by a resume pulse |
| 3. | Set by corresponding output flag | cleared by a resume pulse |
| 4. | Transmitted by any input instruction | cleared after one clock period |

Figure 5-1. Signals for one PPU Channel (Fully Duplexed)

sends a resume pulse to the transmitting device at the other end of the cable. This flag is also forced to a cleared state during a Dead Start condition.

Input Resume Flag

The flag is set for one clock period when the PPU has sampled the input data and is ready for the next word to be transmitted. This flag is also set during a Dead Start condition. A resume pulse is transmitted from this PPU over the input cable during the time in which this flag is set.

Output Channels

There are provisions for eight output cables in each PPU. Each output cable provides a path for 12 bits of outgoing data plus the associated control lines for that data. The PPU may enter data on any one of these eight output cables at any one time. The selection of which of the eight cables is determined by the lowest order three bits in the d register. The output channels are numbered zero through seven to correspond with the value on the lowest order three bits in the d register.

Output Word Flag

The flag is set when a word pulse is transmitted over the associated output cable. The flag is cleared when a resume pulse is returned over this output cable. This flag is also forced to a cleared position during a Dead Start condition. A one clock period wide word pulse is formed for transmission over an output cable. This pulse is associated with the output channel word flag and sets the word flag in addition to transmitting the pulse over the output cable.

Output Record Flag

The flag is set when a record pulse is transmitted over the associated output cable. The flag is cleared when a resume pulse is returned over this output cable. This flag is also forced to a cleared position during a Dead Start condition.

A one clock period wide record pulse is formed for transmission over an output cable. This pulse is associated with the output channel record flag and sets the record flag in addition to transmitting the record pulse over the output cable.

Example 1

Assume that there are two PPU's with an inter-connecting channel and that a series of one-word transfers is required. The PPU X is the output and PPU Y is the input PPU.

1. PPU X loads its A register. If the PPU Y is expecting a transfer, it executes a jump on the Input Word Flag instruction. If the flag is set it will go to the next instruction. If clear, it will repeat this instruction.

2. PPU X executes an Output from the A register instruction. This will set the output word flag in PPU X and the input word flag in PPU Y.

3. Since PPU Y's input word flag is now set it goes to the next instruction which is an Input to the A register. This accepts the word that was transmitted by PPU X and returns a Resume signal. This clears both the input and output word flags. While PPU Y was involved in this activity PPU X was executing a Jump on the Output Word flag instruction. If the flag is set and the word was not accepted, then it repeats the instruction. If a Resume signal was received then the flag would be clear and PPU X exits to the next instruction.

Note that the programmer had to indicate to the PPU that it should prepare for a data input. This could be done by having PPU X set its Output Record flag which, in turn, would set the Input Record flag in PPU Y. PPU Y could intermittently monitor this flag and when it became set it would then go to the sequence outlined above.

In the case of a block transfer PPU X and PPU Y would perform the signal exchange outlined above. The transfer is terminated when one of two conditions occurs. Either the correct number of words will have been transferred or a record flag will be set by PPU X and acknowledged by PPU Y to terminate the transfer. Note that there are two ways a channel can be hung through an incorrect instruction sequence in a FF. First, if the inputting device failed to look at its data channel, the outputting device would remain hung until its word flag was clear. Secondly, if PPU X would attempt a block transfer of more words than PPU Y then PPU X would hang up. The reverse is also true.

Example 2

In communicating with a peripheral device there is no way of telling the device whether data or a function code is being transmitted to it. However, this may be accomplished in a number of ways. In one case, one channel may be assigned to transmit data between the peripheral device and the PPU and a second channel may be assigned to control and status lines. The following example describes communication between a PP and a disk controller.

One channel is assigned to data transfers, a second channel carries control and status information (Figure 5-2).

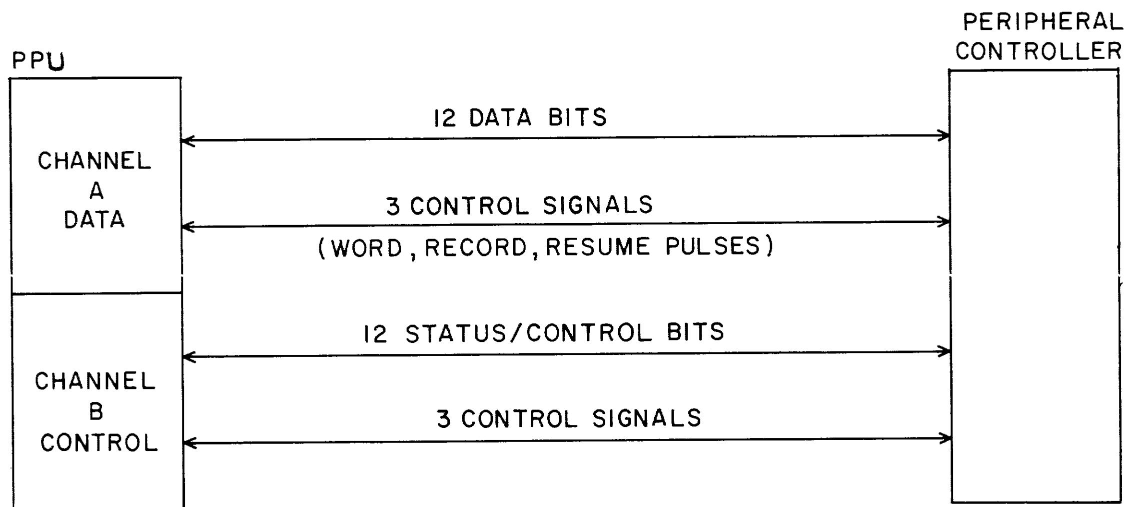



Figure 5-2. Controller/PPU Communication

1. The PPU executes a Record flag on channel A instruction (74) that initializes the controller. The controller sends a resume on channel A to clear the Record flag in the PPU.
2. The PPU reads channel B data which is 12 bits of status. These are always available because the word flags are held set.
3. Assuming status conditions are correct, the PPU transmits the function codes to the controller on channel B. For example, with a disk file the PPU would send a track select code, with the track address in the lower bits.

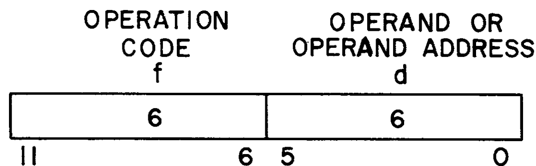
It would then send a sector select code and so forth. All outputs to the controller on channel B would be treated as function codes.

4. The controller is now ready to transfer data. This transfer would then p proceed over channel A as example 1. However, once the controller starts sending data it will not wait for the resume signal from the PPU. If the PPU does not accept the data at once, the controller will replace that data with new data. Therefore the controller's Word flag and Record flag are under hardware control.

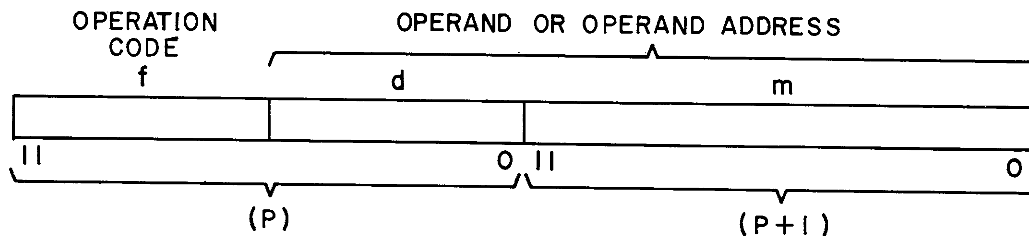
# 6. PERIPHERAL PROCESSOR INSTRUCTIONS

## INSTRUCTION FORMATS

An instruction may have a 12-bit or a 24-bit format. The 12-bit format has a 6-bit operation code f and a 6-bit operand or operand address d.




The 24-bit format uses the 12-bit quantity m, which is the contents of the next program address (P + 1), with d to form an 18-bit operand or operand address.




## ADDRESS MODES

Program indexing is accomplished and operands manipulated in several modes. The two instruction formats provide for 6-bit or 18-bit operands and 6-bit, 12-bit or 18-bit addresses.

### No Address

In this mode d or dm is taken directly as an operand. This mode eliminates the need for storing many constants in storage. The d quantity is considered as a 12-bit number the upper six bits of which are zero. The dm quantity has d as the upper six bits and m as the lower 12 bits.

## Direct Address

In this mode, d or m + (d) is used as the address of the operand. The d quantity specifies one of the first 64 addresses in memory ($0000\text{-}0077_8$). The m + (d) quantity generates a 12-bit address for referencing all possible memory locations but one ($0000\text{-}7776_8$). It is not possible to reference address $7777_8$. If d ≠ 0, the content of address d is added to m to produce an operand address (indexed addressing). If d = 0, m is taken as the operand address.

EXAMPLE: Address Modes

Given: d = 25
m = 100
contents of location 25 = 0150
contents of location 150 = 7776
contents of location 250 = 1234

Then:

| MODE | INSTRUCTION | A REGISTER |
| --- | --- | --- |
| No Address | 14 | 000025 |
| | 20 | 250100 |
| Direct Address | 30 | 000150 |
| | 50 | 001234 |
| Indirect Address | 40 | 007776 |

## Indirect Address

In this mode, d specifies an address the content of which is the address of the desired operand. Thus, d specifies the operand address indirectly. Indirect addressing and indexed addressing require an additional memory reference over direct addressing.

The Description of Instructions section uses the expression (d) to define the contents of memory location d. An expression with double parentheses ( (d) ) refers to indirect addressing. The expression (m + (d)) refers to direct addressing when d = 0 and to indexed direct addressing when d ≠ 0. Table 6-1 summarizes the addressing modes used for the various Peripheral Processor instructions.

TABLE 6-1. ADDRESSING MODES FOR PERIPHERAL PROCESSOR INSTRUCTIONS

| INSTRUCTION TYPE | ADDRESSING MODE | | |
|---|---|---|---|
| | DIRECT | INDIRECT | NO ADDRESS |
| Load | 30, 50 | 40 | 14, 20 |
| Add | 31, 51 | 41 | 16, 21 |
| Subtract | 32, 52 | 42 | 17 |
| Logical Difference | 33, 53 | 43 | 11, 23 |
| Store | 34, 54 | 44 | |
| Replace Add | 35, 55 | 45 | |
| Replace Add One | 36, 56 | 46 | |
| Replace Subtract One | 37, 57 | 47 | |
| Long Jump | 01 | | |
| Return Jump | 02 | | |
| Unconditional Jump | | | 03 |
| Zero Jump | | | 04 |
| Non-Zero Jump | | | 05 |
| Positive Jump | | | 06 |
| Minus Jump | | | 07 |
| Shift | | | 10 |
| Logical Product | | | 12, 22 |
| Selective Clear | | | 13 |
| Load Complement | | | 15 |

# DESCRIPTION OF PERIPHERAL INSTRUCTIONS

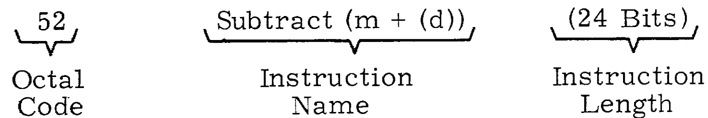
This section describes the Peripheral Processor instructions. Table 6-2 lists designators used throughout the section.

TABLE 6-2. PERIPHERAL PROCESSOR INSTRUCTION DESIGNATORS

| DESIGNATOR | USE |
|---|---|
| A | The A register. |
| d | A 6-bit operand or operand address. |
| f | A 6-bit instruction code. |
| m | A 12-bit quantity used with d to form an 18-bit operand or operand address. |
| P | The Program Address register. |
| Q | The Q register. |
| ( ) | Contents of a register or location. |
| ( () ) | Refers to indirect addressing. |

Preceding the description of each instruction is the octal code, the instruction name and instruction length.

EXAMPLE :




Instruction formats are also given; hashed lines within a format indicate these bits are not used in the operation.

## Error Stop

| | | |
|---|---|---|
| *00* | *Error Stop* | *(12 bits)* |
| *77* | *Error Stop* | *(12 bits)* |

The instructions cause the peripheral processor program to stop and to indicate a program error condition to the Monitor Control Unit. The Peripheral Processor Unit can be restarted only by a dead start sequence from the Maintenance Control Unit.

## No Operation

| | | |
|---|---|---|
| *24* | *Pass* | *(12 Bits)* |
| *25* | *Pass* | *(12 Bits)* |
| *26* | *Pass* | *(12 Bits)* |
| *27* | *Pass* | *(12 Bits)* |
| *75* | *Pass* | *(12 Bits)* |
| *76* | *Pass* | *(12 Bits)* |

| f | (unused) |
|---|---|
| 11 ... 6 | 5 ... 0 |

These instructions specify that no operation be performed. They provide a means of padding out a program.

## Data Transmission

| | | |
|---|---|---|
| *14* | ***Load d*** | *(12 Bits)* |

| f | d |
|---|---|
| 11 ... 6 | 5 ... 0 |

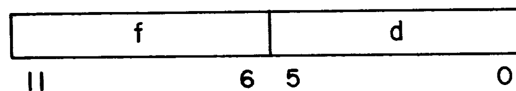
This instruction clears the A register and then loads it with d. The upper 12 bits of A are zero.

| | | |
|---|---|---|
| *15* | ***Load Complement d*** | *(12 Bits)* |

| f | d |
|---|---|
| 11 ... 6 | 5 ... 0 |

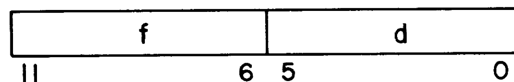
This instruction clears the A register and loads the complement of d. The upper 12 bits of A are set to one.

30 *Load (d)* (12 Bits)



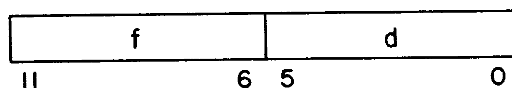

This instruction clears the A register and loads the contents of location d. The upper six bits of A are zero.

34 *Store (d)* (12 Bits)



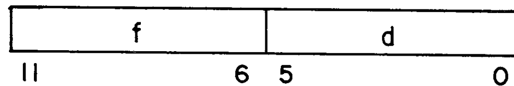

This instruction stores the lower 12 bits of A in location d.

40 *Load ((d))* (12 Bits)



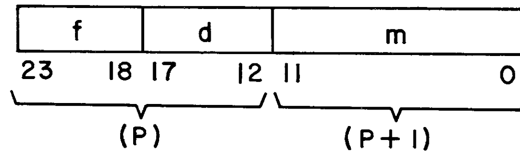

This instruction clears the A register and loads a 12-bit quantity that is obtained by indirect addressing. The upper six bits of A are zero. Location d is read out of memory, and the word obtained is used as the operand address.

44 *Store ((d))* *(12 Bits)*



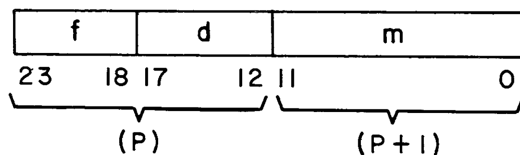

This instruction stores the lower 12 bits of A in the location specified by the contents of location d.

20 *Load dm* *(24 Bits)*




This instruction clears the A register and loads an 18-bit quantity consisting of d as the higher six bits and m as the lower 12 bits. The contents of the location following the present program address are read out to provide m.

50 *Load (m + (d))* *(24 Bits)*




This instruction clears the A register and loads a 12-bit quantity. The upper six bits of A are zero. The 12-bit operand is obtained by indexed direct addressing. The quantity "m", read out of memory location P + 1 serves as the base operand address to which (d) is added. If d = 0, the operand address is simply m, but if d ≠ 0, then m + (d) is the operand address. Thus location d may be used for an index quantity to modify operand addresses.

**54** *Store (m + (d))* *(24 Bits)*

| f | d | m |
|---|---|---|
| 23 18 | 17 12 | 11 0 |
| (P) | | (P + 1) |

This instruction stores the lower 12 bits of A in the location determined by indexed addressing (see instruction 50).

## Arithmetic

**16** *Add d* *(12 Bits)*

| f | d |
|---|---|
| 11 6 | 5 0 |

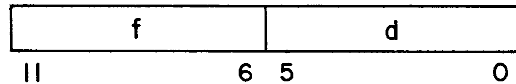
This instruction adds d (treated as a 6-bit positive quantity) to the content of the A register.

**17** *Subtract d* *(12 Bits)*

| f | d |
|---|---|
| 11 6 | 5 0 |

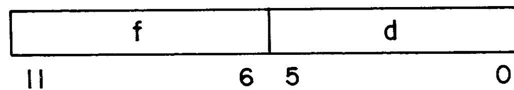
This instruction subtracts d (treated as a 6-bit positive quantity) from the content of the A register.

*31* *Add (d)* *(12 Bits)*



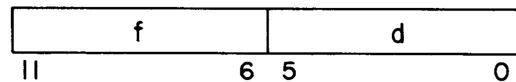

This instruction adds to the A register the contents of location d (treated as a 12-bit positive quantity).

*32* *Subtract (d)* *(12 Bits)*



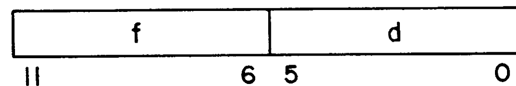

This instruction subtracts from the A register the contents of location d (treated as a 12-bit positive quantity).

*41* *Add ((d))* *(12 Bits)*



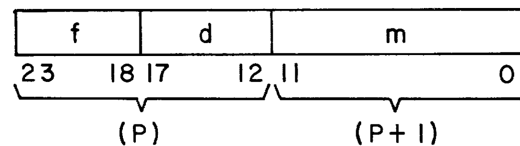

This instruction adds to the content of A a 12-bit operand (treated as a positive quantity) obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address.

42 *Subtract ((d))* (12 Bits)



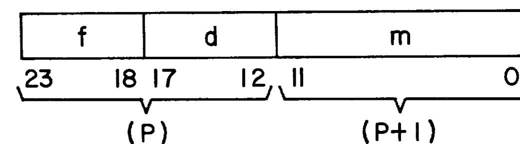

This instruction subtracts from the A register a 12-bit operand (treated as a positive quantity) obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address.

21 *Add dm* (24 Bits)



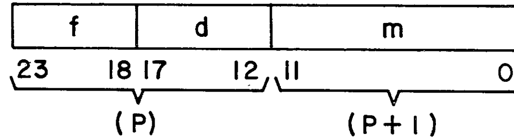

This instruction adds to the A register the 18-bit quantity consisting of d as the higher six bits and m as the lower 12 bits. The contents of the location following the present program address are read out to provide m.

51 *Add (m + (d))* (24 Bits)




This instruction adds to the content of A a 12-bit operand (treated as a positive quantity) obtained by indexed direct addressing (see instruction 50).

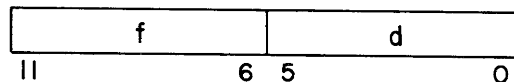
52 *Subtract (m + (d))* *(24 Bits)*




This instruction subtracts from the A register a 12-bit operand (treated as a positive quantity) obtained by indexed direct addressing (see instruction 50).

## Shift

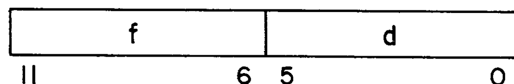
10 *Shift d* *(12 Bits)*




This instruction shifts the contents of A right or left d places. If d is positive (00-37) the shift is left circular; if d is negative (40-77) A is shifted right (end off with no sign extension). Thus, d = 06 requires a left shift of six places. A right shift of six places results when d = 71.

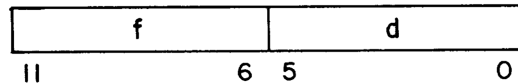
## Logical

11 *Logical difference d* *(12 Bits)*



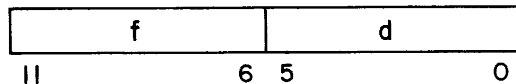

This instruction forms in A the bit-by-bit logical difference of d and the lower six bits of A. This is equivalent to complementing individual bits of A that correspond to bits of d that are one. The upper 12 bits of A are not altered.

12 **Logical product d** (12 Bits)



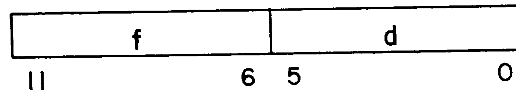

This instruction forms the bit-by-bit logical product of d and the lower six bits of the A register, and leaves this quantity in the lower 6 bits of A. The upper 12 bits of A are zero.

13 **Selective clear d** (12 Bits)



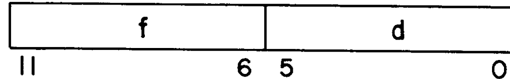

This instruction clears any of the lower six bits of the A register where there are corresponding bits of d that are one. The upper 12 bits of A are not altered.

33 **Logical difference (d)** (12 Bits)



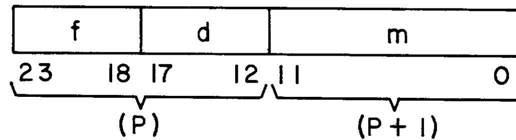

This instruction forms in A the bit-by-bit logical difference of the lower 12 bits of A and the contents of location d. This is equivalent to complementing individual bits of A which correspond to bits of (d) that are one. The upper six bits of A are not altered.

43 *Logical difference ((d))* *(12 Bits)*



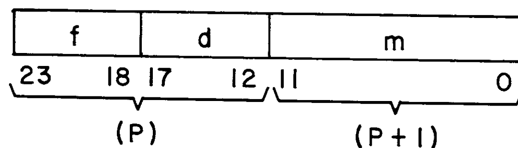

This instruction forms in A the bit-by-bit logical difference of the lower 12 bits of A and the 12-bit operand obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address. The upper six bits of A are not altered.

22 *Logical product dm* *(24 Bits)*



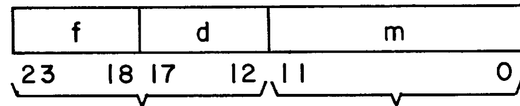

This instruction forms in the A register the bit-by-bit logical product of the contents of A and the 18-bit quantity dm. The upper six bits of this quantity consist of d and the lower 12 bits are the content of the location following the present program address.

23 *Logical difference dm* *(24 Bits)*




This instruction forms in A the bit-by-bit logical difference of the contents of A and the 18-bit quantity dm. This is equivalent to complementing individual bits of A which correspond to bits of dm that are one. The upper six bits of the quantity consist of d, and the lower 12 bits are the content of the location following the present program address.

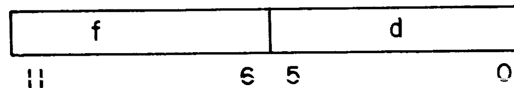
53 *Logical difference (m + (d))* (24 Bits)




This instruction forms in A the bit-by-bit logical difference of the lower 12 bits of A and a 12-bit operand obtained by indexed direct addressing. The upper six bits of A are not altered.

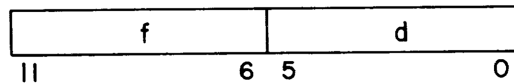
## Replace

35 *Replace add (d)* (12 Bits)



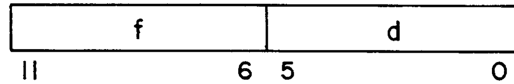

This instruction adds the quantity in location d to the contents of A and stores the lower 12 bits of the result at location d. The resultant sum is left in A at the end of the operation and the original contents of A are destroyed.

36 *Replace add one (d)* (12 Bits)



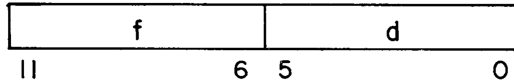

The quantity in location d is replaced by its original value plus one. The resultant sum is left in A at the end of the operation, and the original contents of A are destroyed.

*37* *Replace subtract one (d)* *(12 Bits)*



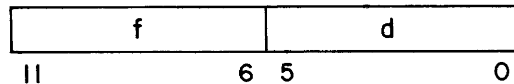

The quantity in location d is replaced by its original value minus one. The resultant difference is left in A at the end of the operation, and the original contents of A are destroyed.

*45* *Replace add ((d))* *(12 Bits)*



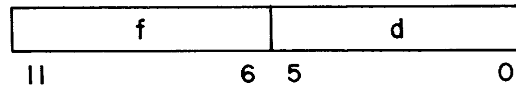

The operand which is obtained from the location specified by the contents of location d, is added to the contents of A, and the lower 12 bits of the sum replace the original operand. The resultant sum is also left in A at the end of the operation.

*46* *Replace add one ((d))* *(12 Bits)*



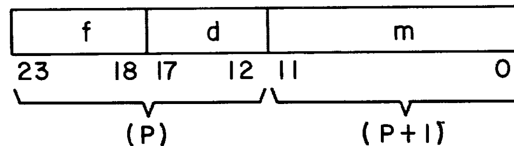

The operand, which is obtained from the location specified by the contents of location d, is replaced by its original value plus one. The resultant sum is also left in A at the end of the operation, and the original contents of A are destroyed.

47 *Replace subtract one ((d))* (12 Bits)



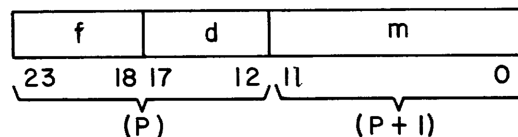

The operand, which is obtained from the location specified by the contents of location d, is replaced by its original value minus one. The resultant difference is also left in A at the end of the operation, and the original contents of A are destroyed.

55 *Replace add (m + (d))* (24 Bits)



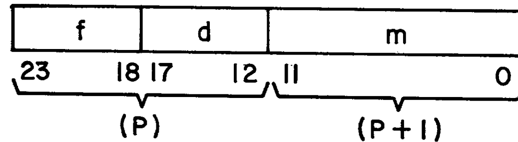

The operand, which is obtained from the location determined by indexed direct addressing, is added to the contents of A, and the lower 12 bits of the sum replace the original operand in memory. The resultant sum is also left in A at the end of the operation, and the original contents of A are destroyed.

56 *Replace add one (m + (d))* (24 Bits)




The operand, which is obtained from the location determined by indexed direct addressing, is replaced by its original value plus one (see instruction 50, for explanation of addressing). The resultant sum is also left in A at the end of the operation, and the original contents of A are destroyed.

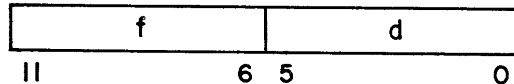
**57** *Replace subtract one (m + (d))* *(24 Bits)*




The operand, which is obtained from the location determined by indexed direct addressing, is replaced by its original value minus one (see instruction 50, for explanation of addressing). The resultant difference is also left in A at the end of the operation, and the original contents of A are destroyed.

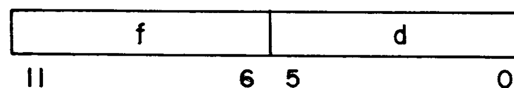
## Branch

**03** *Unconditional jump d* *(12 Bits)*




This instruction provides an unconditional jump to any instruction up to 31 steps forward or backward from the current program address. The value of d is added to the current program address. If d is positive (01 - 37), then 0001 (+1) - 0037 (+31) is added and the jump is forward. If d is negative (40 - 76) then 7740 (-31) - 7776 (-1) is added and the jump is backward. The value of d must not equal 00 or 77. Either of these values cause the PPU to hang in a loop on the 03 instruction. This will violate the restriction on program loops (see page 5-2). Note that the MCU cannot monitor this looping. Should the PPU be hung in such a loop a Dead Start is necessary to restart the PPU.

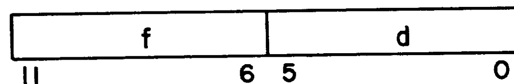
**04** *Zero jump d* *(12 Bits)*




This instruction provides a conditional jump to any instruction up to 31 steps forward

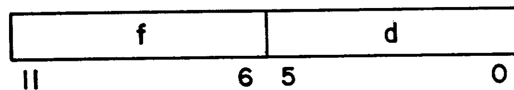
or backward from the current program address. If the content of the A register is zero, the jump is taken. If the content of A is non-zero, the next instruction is executed. Negative zero (777777) is treated as non-zero. For interpretation of d see instruction 03.

**05** **Nonzero jump d** **(12 Bits)**



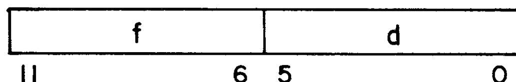

This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is nonzero, the jump is taken. If A is zero, the next instruction is executed. Negative zero (777777) is treated as nonzero. For interpretation of d see instruction 03.

**06** **Plus jump d** **(12 Bits)**



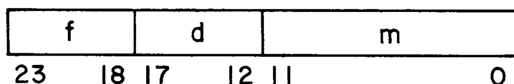

This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is positive, the jump is taken. If A is negative, the next instruction is executed. Positive zero is treated as a positive quantity; negative zero is treated as a negative quantity. For interpretation of d see instruction 03.

07 *Minus jump d* *(12 Bits)*



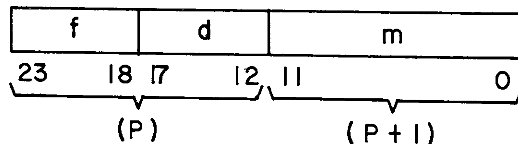

This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is negative, the jump is taken. If A is positive, the next instruction is executed. Positive zero is treated as a positive quantity; negative zero is treated as a negative quantity. For interpretation of d see instruction 03.

01 *Long jump to m + (d)* *(24 Bits)*




This instruction jumps to the sequence beginning at the address given by m + (d). If d = 0, then m is not modified.

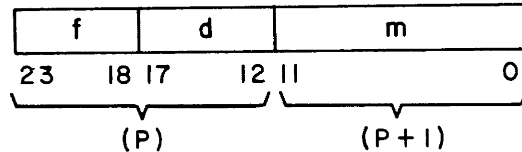
02 *Return jump to m + (d)* *(24 Bits)*




This instruction jumps to the sequence beginning at the address given by m + (d). If d = 0 then m is not modified. The current program address (P) plus two is stored at the jump address. The new program commences at the jump address plus one. This program should end with a long jump to, or normal sequencing into, the jump address minus one, which should in turn contain a long jump, 0100. The latter returns the original program address plus two to the P register.

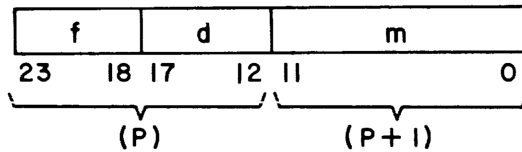
## Input/Output

| | | |
|---|---|---|
| *60* | *Jump on input word flag* | *(24 Bits)* |
| *64* | *Jump on output word flag* | *(24 Bits)* |



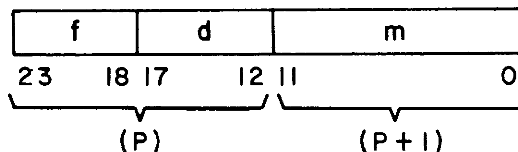

These instructions are conditional jumps. The current program sequence is continued if the flag on channel d is clear. If the flag on channel d is set a new program sequence is begun at address m.

| | | |
|---|---|---|
| *61* | *Jump on no input word flag* | *(24 Bits)* |
| *65* | *Jump on no output word flag* | *(24 Bits)* |




These instructions are identical to the 60 and 64 instructions except that the jump occurs if the flag is clear.

| | | |
|---|---|---|
| *62* | *Jump on input record flag* | *(24 Bits)* |
| *66* | *Jump on output record flag* | *(24 Bits)* |




These instructions are identical to the 60 and 64 instructions except that the jump is conditioned by the status of the record flag.

63 *Jump on no input record flag* (24 Bits)
67 *Jump on no output record flag* (24 Bits)

| f | d | m |
|---|---|---|
| 23 18 | 17 12 | 11 0 |
| (P) | | (P+1) |

These instructions are identical to the 61 and 65 instructions except that the jump is conditioned by the status of the record flag.

70 *Input to A from channel d* (12 Bits)

| f | d |
|---|---|
| 11 6 | 5 0 |

This instruction reads one word from input channel d and enters the word in the A register. This instruction will not be executed until the input channel d word flag is set. If the flag is not set at the time the instruction is read from storage the PPU program will stop with the instruction in the f and d registers and wait until the flag is set by an external signal. The input channel d record flag does not affect execution of this instruction. This instruction clears the input channel d word flag and record flag and transmits a resume signal over the input cable after the word has been read into the A register. The upper six bits of A remain clear.

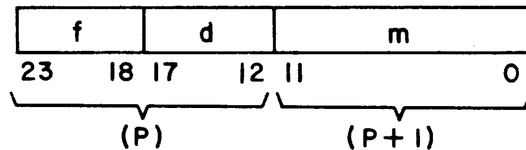
72 *Output from A on channel d* (12 Bits)

| f | d |
|---|---|
| 11 6 | 5 0 |

This instruction transmits one word over output channel d from the lower 12 bits of A. The A register content is not altered. This instruction will not be executed while the output channel d word flag is set. If the flag is set from a previous output instruction

the PPU program will stop with this instruction in the f and d registers and wait for an external resume signal to clear the output channel d word flag. When this instruction is executed the output channel d word flag is set and a word pulse is transmitted over the output channel d cable.

*71* *Input (A) words to m from channel d* *(24 Bits)*




This instruction reads a block of data arriving on input channel d and stores the data in consecutive address locations in memory. The initial storage location for the block is specified by the m designator. The length of the block is specified by the initial contents of the A register or by the setting of the record flag on the input channel during a data transfer.

The starting address is obtained from address m and is entered in the Q register which therefore contains the address for the first word of the data block. The d register contains the channel number, and the A register contains a word count for the block. If (A) is zero at this time the instruction sequence is terminated and the next instruction word is read from storage.

The input channel d word flag must be set before the first word of the block can be entered in storage. If this flag is not set when the instruction is initiated the PPU program will stop with the instruction in the f and d registers and wait until the flag is set by an external signal. The presence of an input channel d record flag is ignored for the first word of the block.

When the input channel d word flag is set the word on the input channel data lines is read into PPU storage at location (Q). The content of the A register is reduced by one count. The content of the Q register is increased by one count in a 12-bit ones

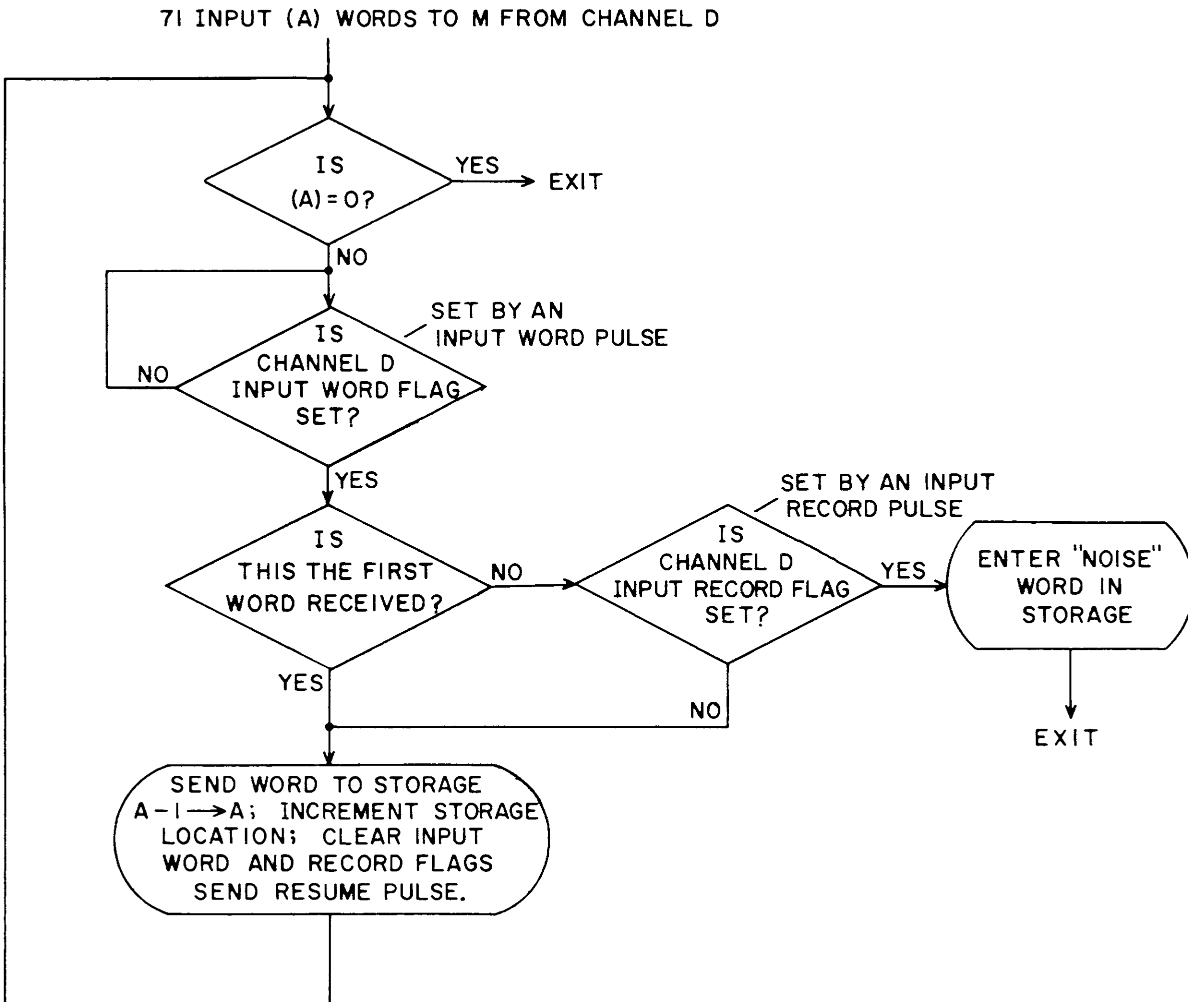
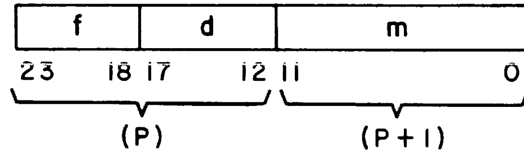



Figure 6-1. 71 Flow Chart

complement mode. The input channel d word flag and input channel d record flag are cleared, and a resume pulse is transmitted over the input cable. If the content of the A register is now zero the instruction sequence is terminated and the next instruction word is read from storage. If (A) is not zero the PPU program waits for the setting of the input channel d word flag for the next word of the block.

The setting of the input channel d record flag terminates the block input at any word after the first word. In this case the sequence is terminated with (A) decremented by the number of words actually transmitted over the input channel. A "noise" word is entered in the next sequential storage location in the PPU block input storage area. The remaining locations in the PPU storage area are unaltered. Note that Q may be incremented through location $7776_8$ to $0000_8$, which may destroy existing data or a program.

***73*** ***Output (A) words from m on channel d*** ***(24 Bits)***




This instruction transmits a block of data over output channel d from consecutive storage locations beginning at address m. The length of the block is specified by the initial contents of the A register. A zero length will cause the instruction to be executed as a pass instruction.

The starting address is obtained from the location defined by m and is entered in the Q register. The Q register now contains the address for the first word of the data block. The d register contains the channel number, and the A register contains the word count for the block. If (A) is zero at this time the instruction sequence is terminated and the next instruction word is read from storage.

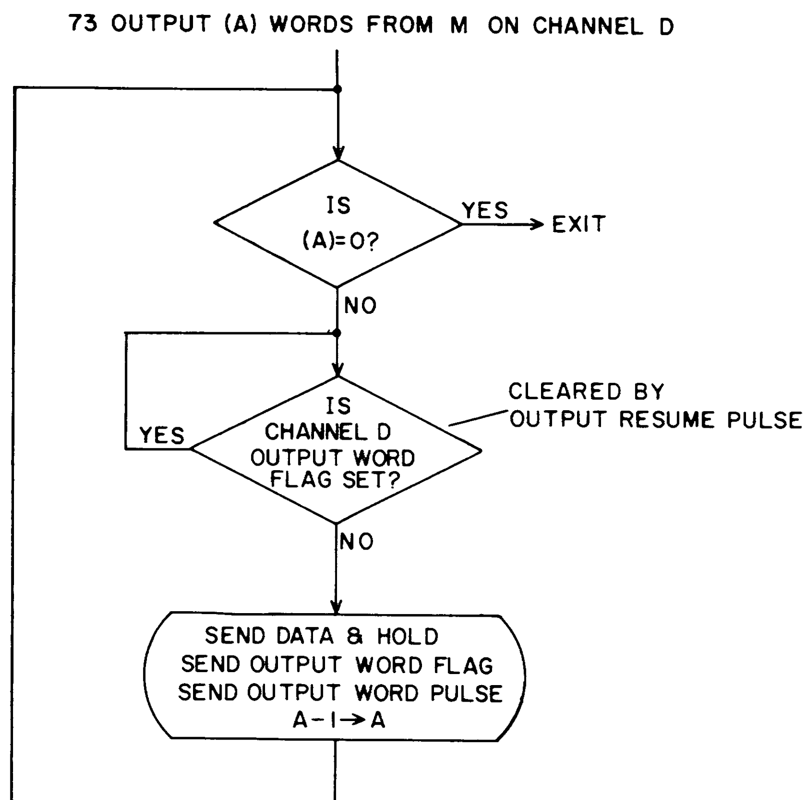
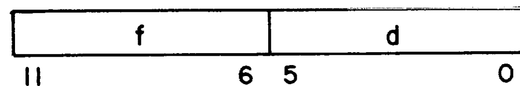



Figure 6-2. 73 Flow Chart

The output channel d word flag must be cleared before the first word of the block can be transmitted over the channel. If this flag is set when the instruction is initiated the PPU program will stop with the instruction in the f and d registers and wait until the flag is cleared by a resume pulse over the output channel d cable. The presence of an output channel d record flag has no effect on the execution of this instruction.

When the output channel d word flag is cleared a word is read from storage location (Q) and is entered in the channel d output register. The output channel d word flag is set, and a word pulse is transmitted over the output cable. The content of the A register is reduced by one count. The content of the Q register is increased by one count in a 12-bit ones complement mode. If the content of the A register is now zero the instruction is terminated and the next instruction is read from storage. If (A) is not zero the PPU program waits for the output channel d word flag to clear and repeats the sequence for the next word of the block.

*74* *Output record flag on channel d* *(12 Bits)*




This instruction sets the output channel d record flag and transmits a record pulse over the output channel d cable. The previous status of the output channel d flags is ignored in this process. The instruction will be executed and a record pulse transmitted even though the output channel d record flag was already set.

# 7. MANUAL CONTROL

## INTRODUCTION

Manual control is provided by the Maintenance Control Unit (MCU) and the console keyboard. The MCU is a PPU that is dedicated to system maintenance. The MCU enables the entering of programs into the system with no prior program in the system (Dead Start). It has secondary functions of monitoring memory parity errors, program errors and generate dead dumps.

## MAINTENANCE CONTROL UNIT

The MCU is connected with up to 15 PPU's through the scanner as shown in Figure 7-1. The scanner is a device that, upon translation of the scanner selector bits from the MCU, connects MCU channel 7 to channel 0 of any one of 15 PPU's. It also connects dead start, dead dump, and clear parity error to the selected PPU's control cable and the PPU program and parity errors to MCU channel 3. The MCU-scanner interface consists of MCU output channel 0, input channel 3, and input and output channel 7. Output channel 0 sends four PPU selector bits and one bit each for dead start, dead dump, and clear parity error. Channel 3 has five inputs to the MCU: a program error signal and four stack parity bits. Channel 7 is a normal data channel operating in a full duplex mode. The PPU-scanner interface consists of full-duplex channel 0 and a control cable containing the dead start, dead dump, and clear parity error inputs and the program error and four stack parity error outputs.

### PPU Dead Start

The following PPU registers are set to the indicated values by a Dead Start signal:

(A) = 007777
(P) = 0000
(X) = 0000
(f) = 71
(d) = 00
(k) = 1

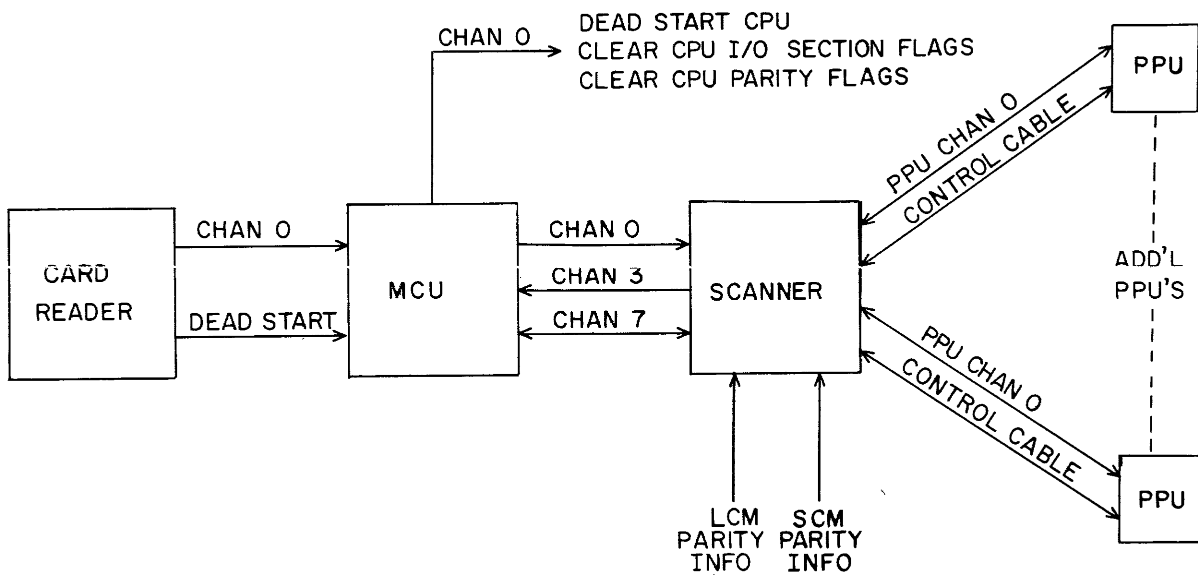



Figure 7-1. MCU Configuration

In addition to the values forced in the register, the PPU flags for the input and output channel control are all forced to a cleared condition and a continuous resume signal is sent over all input cables while the Dead Start signal is up. Dropping of the Dead Start signal then allows a program to be loaded into the PPU over input channel 0 beginning at storage address 0. The loading, if not terminated at some point by a record flag, will terminate at 7777 (octal) words. The PPU then begins execution of the program at address 0001. The Dead Start signal and the input program are sent from the MCU under control of the MCU program.

The MCU is dead started by the card reader. A switch on the card reader generates the Dead Start signal which sets the MCU registers to the values described above for the PPU. The card reader will then input a program to the MCU over channel 0 until the A register count has decremented to zero (memory full) or until the end of file switch on the card reader is manually depressed generating a record flag.

## Dead Dump

One of the control lines from the MCU to the PPU (through SCANNER selection) is a dead dump line. A signal is programmed on this line when the PPU program has failed and a dump of PPU memory is desired to analyze the cause of failure. In this case a Dead Start signal must be programmed first, followed by a Dead Dump signal. This changes the 71 input code in the f register to a 73 output code. The Dead Dump signal is synchronized in the PPU and begins a data transmission of the entire PPU storage over output channel 0. The transmission starts at address 0 and terminates at address $7776_8$.

## CPU Dead Start

The MCU outputs the CPU Dead Start signal and, while the Dead Start signal remains up, writes into SCM. The MCU has access to any part of SCM. Therefore, each word sent to the CPU is given a specific address. When the Dead Start signal drops, the

CPU executes an exchange jump using an exchange package starting at absolute SCM location 0. The MCU must have written the exchange package and a program into SCM. How the CPU loads the system programs is software determined.

### Parity Error Register

Each PPU contains a four bit parity error register. The storage stack in the PPU which has failed is indicated by a set bit. A bit in the parity error register remains set until a clear Parity Error signal is transmitted over the control cable from the MCU by way of the scanner. This condition is programmed in the MCU and clears all four bits in the selected PPU parity error register.

### Program Error

A Program Error condition is transmitted from the PPU to the MCU over the control cable. This condition is a translation of the f register contents and is present when the (f) are either 00 or 77.

## CONSOLE

This display console consists of a cathode ray tube display and a keyboard for manual entry of data. A system may have several display consoles for controlling independent programs simultaneously.

### Keyboard Input

The console may be selected for input to allow manual entry of data or instructions to the computer. The first part of an operating system program may select keyboard input to allow the programmer to manually select a routing from the operating system. Data entered via the keyboard may be displayed on the display tubes if desired. Assembly and display of keyboard entries is done by a routine in the operating system.

Figure 7-2. Display Console

# APPENDIX A

# TIMING NOTES

## TIMING NOTES

1. Times given include clock period known to occur before instruction issue, but do not consider register conflict conditions that might delay issue.

   Except for the multiply and divide units, all functional units permit new instructions to enter them every clock period. A new instruction may enter the multiply unit in any clock period, provided there was no operation initiated in the preceding clock period. A new instruction can enter the divide unit two clock periods prior to completion of a previous divide operation. Once an instruction issues to a functional unit, it is executed in a fixed amount of time. No delays are possible.

   Times given for instructions 01 to 07 and 50 to 57 do not consider memory conflict conditions or SAS backup conditions caused by bank conflicts.

2. Execution of Block Copy instructions (011 and 012) will be delayed until the following conditions are satisfied:

   a. All operating registers are free.

   b. No SCM bank conflicts exist.

   c. LCM is not busy.

   d. All LCM banks have completed previously initiated read/write cycles.

3. A delay will occur during instructions 011, 012, and 013 when an I/O section word request is made. A minimum delay of one clock period is required to enter the I/O word address in the address stream to the SAS. An additional delay will occur if the I/O reference causes a bank conflict in SCM.

4. A delay will occur in the execution of the Exchange Exit instruction (013) until two conditions are satisfied:

   a. All operating registers are free.

   b. No SCM bank conflicts exist.

5. The Read LCM and Write LCM instructions (014 and 015) will not issue until three conditions are satisfied:

   a. LCM is not busy.

   b. Xj register is free.

   c. Xk register is free.

6. A Read LCM instruction (014) for a word already residing in an LCM bank operand register as a result of a previous instruction will require three clock periods. For a word not currently residing in one of the LCM bank operand registers, the instruction requires 6 clock periods.

7. The Reset Buffer instructions and Read Channel Status instructions (016 and 017) will not issue and begin execution until the required B registers are free.

8. Jump instruction 02i0K will not begin execution until the Bi register is free. Instruction execution will also be delayed if an instruction fetch is in process.

9. The execution of a branch instruction (030 to 037, 04ijk, 05ijk, 06ijk, and 07ijk) maybe delayed if an instruction fetch is in process.

10. Instructions 10 to 47 and 60 to 77 will not issue until the following conditions are satisfied:

    a. The required A, B, and X registers are free.

    b. X and B register input paths will be free during the required clock period.

    c. No SAS backup condition exists.

    d. The multiply unit is free (instructions 40, 41, and 42 only).

    e. The divide unit is free (instructions 44 and 45 only).

11. Instructions 50 to 57 will not issue until the following conditions are satisfied:

    a. The required A, B, and X registers are free.

    b. No SAS backup condition exists.

12. A delay may occur in the execution of the Return Jump instruction (0100K) if the instruction stack control has requested one or more instruction words that have not arrived at the instruction stack (likely to occur in straight line coding). Average execution time is 18 clock periods.

## CENTRAL PROCESSOR INSTRUCTIONS

| INSTRUCTION CODE | NAME | EXECUTION TIME (CLOCK PERIODS) | FUNCTIONAL UNIT |
|---|---|---|---|
| 00000 | Error exit to EEA | - | - |
| 0100K | Return jump to K | Min 13* | - |
| 011jK | Block copy K + (Bj) words from LCM to SCM | Min = N + 15** | - |
| 012jK | Block copy K + (Bj) words from SCM to LCM | Min = N + 11** | - |
| 01300 | Exchange exit to NEA if exit flag clear | Min = 28 | - |
| 013jK | Exchange exit to K + (Bj) if exit flag set | Min = 28 | - |
| 014jk | Read LCM at (Xk) to Xj | 3, 16* | - |
| 015jk | Write (Xj) into LCM at (Xk) | 3 | - |
| 0160k | Reset channel (Bk) input buffer if j = 0 | 4 | - |
| 016jk | Read channel (Bk) input status to Bj if j ≠ 0 | 3 | - |
| 0170k | Reset channel (Bk) output buffer if j = 0 | 16 | - |
| 017jk | Read channel (Bk) output status to Bj if j ≠ 0 | 3 | - |
| 02i0K | Jump to K + (Bi) | Min 3 (in stack jump)<br>Min 11 (out of stack jump) | - |
| 030jK | Branch to K if (Xj) = 0 | Min 2 (branch fall through)<br>Min 3 (branch in stack)<br>Min 11 (branch out of stack) | - |
| 031jK | Branch to K if (Xj) ≠ 0 | | - |
| 032jK | Branch to K if (Xj) positive | | - |
| 033jK | Branch to K if (Xj) negative | | - |
| 034jK | Branch to K if (Xj) in range | Same as above (031jK–037jK) | - |
| 035jK | Branch to K if (Xj) not in range | | - |
| 036jK | Branch to K if (Xj) definite | | - |
| 037jK | Branch to K if (Xj) indefinite | | - |

* Refer to Timing Notes.

** 1≤ = Number of words in the block. (4 clock periods if N=0)

CENTRAL PROCESSOR INSTRUCTIONS (Cont'd)

| INSTRUCTION CODE | NAME | EXECUTION TIME (CLOCK PERIODS) | FUNCTIONAL UNIT |
|---|---|---|---|
| 041jK | Branch to K if (Bi) = (Bj) | Min 2 (branch fall through)<br>Min 3 (branch in stack)<br>Min 11 (branch out of stack) | - |
| 051jK | Branch to K if (Bi) ≠ (Bj) | Same as above | - |
| 061jK | Branch to K if (Bi) ≥ (Bj) | Same as above | - |
| 071jK | Branch to K if (Bi) < (Bj) | Same as above | - |
| 10ij0 | Copy (Xj) to Xi | 2 | Boolean |
| 11ijk | Logical product of (Xj) and (Xk) to Xi | 2 | Boolean |
| 12ijk | Logical sum of (Xj) plus (Xk) to Xi | 2 | Boolean |
| 13ijk | Logical difference of (Xj) minus (Xk) to Xi | 2 | Boolean |
| 14i0k | Copy complement of (Xk) to Xi | 2 | Boolean |
| 15ijk | Logical product of (Xj) and comp (Xk) to Xi | 2 | Boolean |
| 16ijk | Logical sum (Xj) plus comp (Xk) to Xi | 2 | Boolean |
| 17ijk | Logical difference of (Xj) minus comp (Xk) to Xi | 2 | Boolean |
| 20ijk | Left shift (Xi) by jk | 2 | Shift |
| 21ijk | Right shift (Xi) by jk | 2 | Shift |
| 22ijk | Left shift (Xk) by (Bj) to Xi | 2 | Shift |
| 23ijk | Right shift (Xk) by (Bj) to Xi | 2 | Shift |
| 24ijk | Normalize (Xk) to Xi and Bj | 3 | Normalize |
| 25ijk | Round and normalize (Xk) to Xi and Bj | 3 | Normalize |
| 26ijk | Unpack (Xk) to Xi and Bj | 2 | Boolean |
| 27ijk | Pack (Xk) and (Bj) to Xi | 2 | Boolean |
| 30ijk | Floating sum of (Xj) plus (Xk) to Xi | 4 | Floating Add |
| 31ijk | Floating difference of (Xj) minus (Xk) to Xi | 4 | Floating Add |

CENTRAL PROCESSOR INSTRUCTIONS (Cont'd)

| INSTRUCTION CODE | NAME | EXECUTION TIME (CLOCK PERIODS) | FUNCTIONAL UNIT |
|---|---|---|---|
| 32ijk | Floating DP sum of (Xj) plus (Xk) to Xi | 4 | Floating Add |
| 33ijk | Floating DP difference of (Xj) minus (Xk) to Xi | 4 | Floating Add |
| 34ijk | Round floating sum of (Xj) plus (Xk) to Xi | 4 | Floating Add |
| 35ijk | Round floating difference of (Xj) minus (Xk) to Xi | 4 | Floating Add |
| 36ijk | Integer sum of (Xj) plus (Xk) to Xi | 2 | Long Add |
| 37ijk | Integer difference of (Xj) minus (Xk) to Xi | 2 | Long Add |
| 40ijk | Floating product of (Xj) times (Xk) to Xi | 5 | Floating Multiply |
| 41ijk | Round floating product of (Xj) times (Xk) to Xi | 5 | Floating Multiply |
| 42ijk | Floating DP product of (Xj) times (Xk) to Xi | 5 | Floating Multiply |
| 43ijk | Form mask of jk bits to Xi | 2 | Shift |
| 44ijk | Floating divide (xj) by (Xk) to Xi | 20 | Floating Divide |
| 45ijk | Round floating divide (Xj) by (Xk) to Xi | 20 | Floating Divide |
| 46000 | Pass | 2 | - |
| 47i0k | Population count of (Xk) to Xi | 2 | Population Count |
| 50ijK | Increment (Aj) plus K to Ai | 2 (Set Aj)<br>8 (Read to Xj)<br>1 (Store from Xj) | Increment |
| 51ijK | Increment (Bj) plus K to Ai | Same as above | Increment |
| 52ijK | Increment (Xj) plus K to Ai | Same as above | Increment |
| 53ijk | Increment (Xj) plus (Bk) to Ai | Same as above | Increment |

CENTRAL PROCESSOR INSTRUCTIONS (Cont'd)

| INSTRUCTION CODE | NAME | EXECUTION TIME (CLOCK PERIODS) | FUNCTIONAL UNIT |
|---|---|---|---|
| 54ijk | Increment (Aj) plus (Bk) to Ai | 2 Set (Aj)<br>8 (Read to Xj)<br>1 (Store from (Xj) | Increment |
| 55ijk | Increment (Aj) minus (Bk) to Ai | Same as above | Increment |
| 56ijk | Increment (Bj) plus (Bk) to Ai | Same as above | Increment |
| 57ijk | Increment (Bj) minus (Bk) to Ai | Same as above | Increment |
| 60ijK | Increment (Aj) plus K to Bi | 2 | Increment |
| 61ijK | Increment (Bj) plus K to Bi | 2 | Increment |
| 62ijK | Increment (Xj) plus K to Bi | 2 | Increment |
| 63ijk | Increment (Xj) plus (Bk) to Bi | 2 | Increment |
| 64ijk | Increment (Aj) plus (Bk) to Bi | 2 | Increment |
| 65ijk | Increment (Aj) minus (Bk) to Bi | 2 | Increment |
| 66ijk | Increment (Bj) plus (Bk) to Bi | 2 | Increment |
| 67ijk | Increment (Bj) minus (Bk) to Bi | 2 | Increment |
| 70ijK | Increment (Aj) plus K to Xi | 2 | Increment |
| 71ijK | Increment (Bj) plus K to Xi | 2 | Increment |
| 72ijK | Increment (Xj) plus K to Xi | 2 | Increment |
| 73ijk | Increment (Xj) plus (Bk) to Xi | 2 | Increment |
| 74ijk | Increment (Aj) plus (Bk) to Xi | 2 | Increment |
| 75ijk | Increment (Aj) minus (Bk) to Xi | 2 | Increment |
| 76ijk | Increment (Bj) plus (Bk) to Xi | 2 | Increment |
| 77ijk | Increment (Bj) minus (Bk) to Xi | 2 | Increment |

## PERIPHERAL PROCESSOR INSTRUCTIONS

| INSTRUCTION CODE (OCTAL) | NAME | EXECUTION TIME (CLOCK PERIODS) |
|---|---|---|
| 00 | Error stop | - |
| 0100 | Long jump to m | 10 or 15 |
| 01XX | Long jump to m + (d) | 15, 20, 25 |
| 0200 | Return jump to m | 15 or 20 |
| 02XX | Return jump to m + (d) | 20, 25, 30 |
| 03 | Unconditional jump d | 8, 10 |
| 04 | Zero jump d | 5 |
| 05 | Nonzero jump d | 5 |
| 06 | Positive jump d | 5 |
| 07 | Negative jump d | 5 |
| 10 | Shift d | Minimum 6, Maximum 34 |
| 11 | Logical difference d | 5 |
| 12 | Logical product d | 5 |
| 13 | Selective clear d | 5 |
| 14 | Load d | 5 |
| 15 | Load complement d | 5 |
| 16 | Add d | 5 |
| 17 | Subtract d | 5 |
| 20 | Load dm | 10 |
| 21 | Add dm | 10 |
| 22 | Logical product dm | 10 |
| 23 | Logical difference dm | 10 |

NOTES:

1. Where more than one time is given, the shorter time is obtained when full use of bank phasing (back-to-back storage references to alternate banks) is made.
2. Conditional jump instructions list times for the "jump not taken" case. Add 3 or 5 clock periods for the "jump taken" case, depending on the value of d.
3. For the 10 (shift) instruction: Minimum time is required if the shift count $< 3$; for shift counts $> 3$, add 1 clock period per shift beyond 3 to the minimum time.

PERIPHERAL PROCESSOR INSTRUCTIONS (Cont'd)

| ꞉STRUCTION CODE (OCTAL) | NAME | EXECUTION TIME (CLOCK PERIODS) |
|---|---|---|
| 24 | Pass | 5 |
| 25 | Pass | |
| 26 | Pass | |
| 27 | Pass | |
| 30 | Load (d) | 15 |
| 31 | Add (d) | 15 |
| 32 | Subtract (d) | 15 |
| 33 | Logical difference (d) | 15 |
| 34 | Store (d) | 15 |
| 35 | Replace add (d) | 25 |
| 36 | Replace add one (d) | 25 |
| 37 | Replace subtract one (d) | 25 |
| 40 | Load ((d)) | 15, 25 |
| 41 | Add ((d)) | 15, 25 |
| 42 | Subtract ((d)) | 15, 25 |
| 43 | Logical difference ((d)) | 15, 25 |
| 44 | Store ((d)) | 15, 25 |
| 45 | Replace add ((d)) | 25, 35 |
| 46 | Replace add one ((d)) | 25, 35 |
| 47 | Replace subtract one ((d)) | 25, 35 |
| 5000 | Load (m) | 20 |
| 50XX | Load (m + (d)) | 20, 30 |
| 5100 | Add (m) | 20 |
| 51XX | Add (m + (d)) | 20, 30 |
| 5200 | Subtract (m) | 20 |
| 52XX | Subtract (m + (d)) | 20, 30 |
| 5300 | Logical difference (m) | 20 |
| 53XX | Logical difference (m + (d)) | 20, 30 |
| 5400 | Store (m) | 20 |

PERIPHERAL PROCESSOR INSTRUCTIONS (Cont'd)

| INSTRUCTION CODE (OCTAL) | NAME | EXECUTION TIME (CLOCK PERIODS) |
|---|---|---|
| 54XX | Store (m + (d)) | 20, 30 |
| 5500 | Replace add (m) | 30 |
| 55XX | Replace add (m + (d)) | 30, 40 |
| 5600 | Replace add one (m) | 30 |
| 56XX | Replace add one (m + (d)) | 30, 40 |
| 5700 | Replace subtract one (m) | 30 |
| 57XX | Replace subtract one (m + (d)) | 30, 40 |
| 60 | Jump on input word flag | 10* |
| 61 | Jump if no input word flag | 10 |
| 62 | Jump on input record flag | 10 |
| 63 | Jump if no input record flag | 10 |
| 64 | Jump on output word flag | 10 |
| 65 | Jump if no output word flag | 10 |
| 66 | Jump on output record flag | 10 |
| 67 | Jump if no output record flag | 10 |
| 70 | Input to A from channel d | 9** |
| 71 | Input (A) words to m from channel d | + |
| 72 | Output from A on channel d | 9++ |
| 73 | Output (A) words from m on channel d | + |
| 74 | Output record flag on channel d | 5 |
| 75 | Pass | 5 |
| 76 | Pass | 5 |
| 77 | Error Stop | - (restart only by a Dead Start) |

* Jump instruction times are for the "jump not taken" case. The "jump taken" execution time is identical if the jump is to an alternate bank. If the jump is taken to the same bank, add 5 clock periods.

** Assume input channel d word flag is set; if not set, add the time waiting for flag to set.

++ Assumes output channel d word flag is clear; if not clear, add the time waiting for flag to clear.

\+ Timing for these instructions are sample times only for various cases. Assumptions made for each case are stated on the following page.

71 Instruction:

Case 1: Assume -

a. a block input terminated by a record flag rather than by decrementing (A) to zero.

b. a 2 clock period response time between the resume and the word flag.

c. a 3-word block followed by a record flag.

d. the channel d input word flag is set at instruction initiation, and

e. the first data reference is to the alternate storage bank.

Execution Time = 42 Clock Periods

Case 2: Assume -

a. a block input terminated by reducing (A) to zero.

b. same response as in Item b, Case 1.

c. a count of 2 in the A register, and

d. items d and e in Case 1 are true.

Execution Time = 24 Clock Periods

Case 3: Assume -

a. a block input initiated with (A) = zero.

Execution Time = 10 Clock Periods

73 Instruction:

Case 1: Assume -

a. a count of 3 in the A register.

b. the device has a 2 clock period response time from receipt of word pulse to transmission of resume pulse.

c. the output channel d word flag is clear, and

d. the first word of the block is read from the alternate storage bank.

Execution Time = 34 Clock Periods

Case 2: Assume - a. a block output initiated with (A) = zero.

<u>Execution Time = 10 Clock Periods</u>

# APPENDIX B

# FLOATING POINT ARITHMETIC

## FLOATING POINT ARITHMETIC

### FORMAT

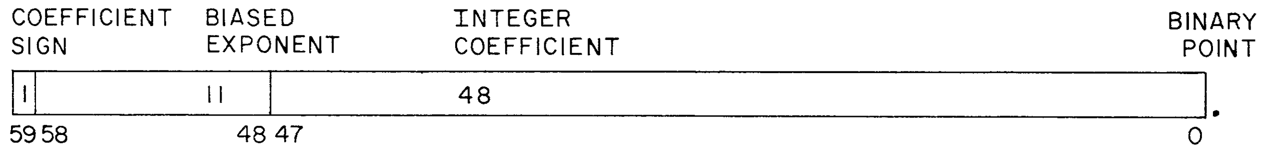
Floating point arithmetic takes advantage of the ability to express a number with the general expression $kB^n$, where:

k = coefficient

B = base number

n = exponent, or power to which the base number is raised

The base number (B) is assumed to be 2 for binary-coded quantities. In the 60-bit floating-point format shown below, the binary point is considered to be to the right of the coefficient. The lower 48 bits express the integer coefficient, which is the equivalent of about 15 decimal digits. The sign of the coefficient is separated from the rest of the coefficient and appears in the highest order bit of the packed word. Negative numbers are represented in one's complement notation.




The exponent portion of the floating point format is biased by complementing the exponent sign bit. This particular format for floating point numbers is chosen so that the packed form may be treated as a 60-bit integer for sign, threshold, equality, and zero tests.

The following table (B-1) summarizes the configurations of bits $2^{58}$ and $2^{59}$ and the implications regarding signs, of the possible combinations.

TABLE B-1. BIT $2^{58}$ AND $2^{59}$ CONFIGURATIONS

| $2^{59}$ | $2^{58}$ | COEFFICIENT SIGN | EXPONENT SIGN |
|---|---|---|---|
| 0 | 1 | Positive | Positive |
| 0 | 0 | Positive | Negative |
| 1 | 0 | Negative | Positive |
| 1 | 1 | Negative | Negative |

## PACKING

Packing refers to the conversion of numbers in the form $kB^n$ to floating point format. A short-cut method of packing exponents can be derived by considering the representation of negative and positive zero exponents. Assuming a positive coefficient, zero exponents are packed as follows:

Positive zero exponent = 2000X--------X

Negative zero exponent = 1777X--------X

Since positive exponents are expressed in true form, start with a "bias" of 2000 (positive zero) and add the magnitude of the exponent. The range of positive exponents is:

0000 through 1777

Or, in packed form:

2000 through 3777.

Negative exponents are expressed in complement form. Hence, start with a bias of 1777 (negative zero) and subtract the magnitude of the exponent. The range of negative exponents is:

-0000 through -1777

Or, in packed form:

1777 through 0000.

Some examples of packed and unpacked floating point numbers are shown below in octal notation to illustrate the packing process. The first two examples are different forms of the integer +1. The third example is +100 decimal and the fourth example is -100 decimal. The last two examples are of very large and very small positive numbers. The unpacked values are shown as they might appear in X and B registers prior to a pack operation.

1. unpacked coefficient = 0000 0000 0000 0000 0001
   unpacked exponent = 00 0000
   packed format = 2000 0000 0000 0000 0001

2. unpacked coefficient = 0000 4000 0000 0000 0000
   unpacked exponent = 77 7720
   packed format = 1720 4000 0000 0000 0000

3. unpacked coefficient = 0000 6200 0000 0000 0000
   unpacked exponent = 77 7726
   packed format = 1726 6200 0000 0000 0000

4. unpacked coefficient = 7777 1577 7777 7777 7777
   unpacked exponent = 77 7726
   packed format = 6051 1577 7777 7777 7777

5. unpacked coefficient = 0000 4771 3000 0044 7021
   unpacked exponent = 00 1363
   packed format = 3363 4771 3000 0044 7021

6. unpacked coefficient = 0000 6301 0277 4315 6033
   unpacked exponent = 77 6210
   packed format = 0210 6301 0277 4315 6033

## OVERFLOW

Overflow of the floating point range is indicated by an exponent value of +1777 octal (3777 or 4000 in packed form). This is the largest exponent value that can be represented in the floating point format (see Table B-2). This exponent value may result from the calculation in a floating point unit in which this exponent value, together with the computed coefficient value, is a correct representation of the result. This situation is called a "partial overflow" in this manual. An Overflow Error condition is not indicated by the functional unit generating this result. However, further computation in floating point functional units using this result will generate an overflow.

TABLE B-2. FLOATING POINT REPRESENTATION

| | POSITIVE COEFFICIENT | | NEGATIVE COEFFICIENT | |
|---|---|---|---|---|
| OVERFLOW | Complete Overflow<br>Partial Overflow | = 3777 0------0<br>= 3777 X----- X | Complete Overflow<br>Partial Overflow | = 4777 7----7<br>= 4000 X--- X |
| INTEGERS | Largest:<br>7------7. x $2^{+1776}$<br>Smallest:<br>1. x $2^{0}$ | = 3776 7------7<br><br>= 2000 0-----01 | *Largest:<br>-7------7. x $2^{-1776}$<br>*Smallest:<br>-1. x $2^{0}$ | = 4001 0----0<br><br>= 5777 7---76 |
| ZERO | Positive Zero | = 2000 0------0 | Negative Zero | = 5777 7----7 |
| INDEFINITE OPERANDS | Indefinite Operand | = 1777 0------0 | **Indefinite Operand | = 6000 7----7 |
| FRACTIONS | Largest:<br>7------7. x $2^{-60}$<br>Smallest:<br>1. x $2^{-1777}$ | = 1717 7------7<br><br>= 0000 0-----01 | *Largest:<br>-7------7. x $2^{-60}$<br>*Smallest:<br>-1. x $2^{-1777}$ | = 6060 0----0<br><br>= 7777 7---76 |
| UNDERFLOW | Complete Underflow<br>Partial Underflow | = 0000 0------0<br>= 0000 X----- X | Complete Underflow<br>Partial Underflow | = 7777 7----7<br>= 7777 X---X |

* In absolute value.

** An indefinite operand with a negative sign can only occur from packing or Boolean operations.

A "complete overflow" occurs whenever a floating point functional unit computes a result that requires an exponent larger than +1777 octal. In this case the functional unit indicates an Overflow Error condition and packs a "complete overflow" value for the result. This result has a +1777 exponent and a zero coefficient. The sign of the coefficient will be the same as that which would have been generated if the result had not overflowed the floating point range.

## UNDERFLOW

Underflow of the floating point range is indicated by an exponent value of -1777 octal (0000 or 7777 in packed form). This is the smallest exponent value that can be represented in the floating point format. This exponent value may result from the calculation in a floating point unit in which this exponent value, together with the computed coefficient value, is a correct representation of the result. This situation is called a "partial underflow" in this manual. An Underflow Error condition is not indicated by the functional unit generating this result. However, further computation in floating point functional units using this result may be detected as an underflow.

A "complete underflow" occurs whenever a floating point functional unit computes a result that requires an exponent smaller than -1777 octal. In this case the functional unit indicates an underflow error condition and packs a "complete underflow" value for the result. This result has a -1777 exponent and a zero coefficient. The sign of the coefficient will be the same as that which would have been generated if the result had not underflowed the floating point range. Thus, the complete underflow indicator is a word of all zero bits, or all one bits, depending on the sign. It is the same as a zero word in integer format.

## INDEFINITE RESULT

An indefinite result indicator is generated by a floating point functional unit whenever the calculation cannot be resolved. This is the case in division when the divisor is zero and the dividend is also zero. It is also the case in multiplication of an overflow number

times an underflow number. The indefinite result indicator is a value that cannot occur in normal floating point calculations. This indicator corresponds to a minus zero exponent and a zero coefficient (17770------0 in packed form). An Indefinite Error condition is indicated by the functional unit generating this result. Any floating point functional unit receiving an indefinite indicator as an operand will generate an indefinite result no matter what the other operand value. Although indefinite indicators are always generated with a positive sign by the floating arithmetic units, they may occur as operands with negative sign because of complementation in the Boolean unit.

## NON-STANDARD FLOATING POINT ARITHMETIC

In summary, the special operand forms in octal are:

positive underflow (+0) = 0000X------X
negative underflow (-0) = 7777X------X
positive overflow (+∞) = 3777X------X
negative overflow (-∞) = 4000X------X
positive indefinite (+IND) = 1777X------X
negative indefinite (-IND) = 6000X------X

If a functional unit generates the above special forms while using normal operands, it usually gives an error indication. When a floating point arithmetic unit uses one of these six special forms as an operand, however, only the following octal words can occur as results and the associated flag is set in the Program Status Designation (PSD).

| | | |
|---|---|---|
| positive overflow (+∞) | = 37770------0 | Overflow condition flag |
| negative overflow (-∞) | = 40007------7 | Overflow condition flag |
| positive indefinite (+IND) | = 17770------0 | Indefinite condition flag |
| positive underflow (+0) | = 00000------0 | Underflow condition flag |
| negative underflow (-0) | = 77777------7 | Underflow condition flag |

The following tabulations show the Add, Subtract, Multiply and Divide operations using various combinations of underflow, indefinite, and overflow quantities as operands. In the tabulations the designations W and N are defined as follows:

W = Any word except ±∞, ±IND

N = Any word except ±∞, ±IND, or ±0.

ADD

Xi=Xj+Xk

(Instructions 30, 32, 34)

| | | Xk | | | |
|---|---|---|---|---|---|
| | | W | +∞ | -∞ | ±IND |
| Xj | W | - | +∞ | -∞ | IND |
| | +∞ | +∞ | +∞ | IND | IND |
| | -∞ | -∞ | IND | -∞ | IND |
| | ±IND | IND | IND | IND | IND |

SUBTRACT

Xi=Xj-Xk

(Instructions 31, 33, 35)

| | | Xk | | | |
|---|---|---|---|---|---|
| | | W | +∞ | -∞ | ±IND |
| Xj | W | - | -∞ | +∞ | IND |
| | +∞ | +∞ | IND | +∞ | IND |
| | -∞ | -∞ | -∞ | IND | IND |
| | ±IND | IND | IND | IND | IND |

MULTIPLY

Xi=Xj*Xk

(Instructions 40, 41, 42)

| | | Xk | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | +N | -N | +0 | -0 | +∞ | -∞ | ±IND |
| Xj | +N | - | - | 0 | 0 | +∞ | -∞ | IND |
| | -N | - | - | 0 | 0 | -∞ | +∞ | IND |
| | +0 | 0 | 0 | 0 | 0 | IND | IND | IND |
| | -0 | 0 | 0 | 0 | 0 | IND | IND | IND |
| | +∞ | +∞ | -∞ | IND | IND | +∞ | IND | IND |
| | -∞ | -∞ | +∞ | IND | IND | IND | +∞ | IND |
| | ±IND | IND | IND | IND | IND | IND | IND | IND |

DIVIDE

Xi=Xj/Xk

(Instructions 44, 45)

| | | Xk | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | +N | -N | +0 | -0 | +∞ | -∞ | ±IND |
| Xj | +N | - | - | +∞ | -∞ | 0 | 0 | IND |
| | -N | - | - | -∞ | +∞ | 0 | 0 | IND |
| | +0 | 0 | 0 | IND | IND | 0 | 0 | IND |
| | -0 | 0 | 0 | IND | IND | 0 | 0 | IND |
| | +∞ | +∞ | -∞ | +∞ | -∞ | IND | IND | IND |
| | -∞ | -∞ | +∞ | -∞ | +∞ | IND | IND | IND |
| | ±IND | IND | IND | IND | IND | IND | IND | IND |

## NORMALIZED FLOATING POINT

A floating point number in packed format is normalized if the coefficient sign bit is different from bit 47. This condition implies that the coefficient has been shifted to the left as far as possible, and therefore the floating point number has no leading zeros in the coefficient.

The normalize unit performs this function. The floating multiply and floating divide units deliver normalized results when provided with normalized operands. The floating add unit may deliver un-normalized results even when both operands are normalized. It is therefore necessary to perform the normalize operation in the normalize unit after each sequence of floating add or subtract operations if the result is to be kept in a normalized form.

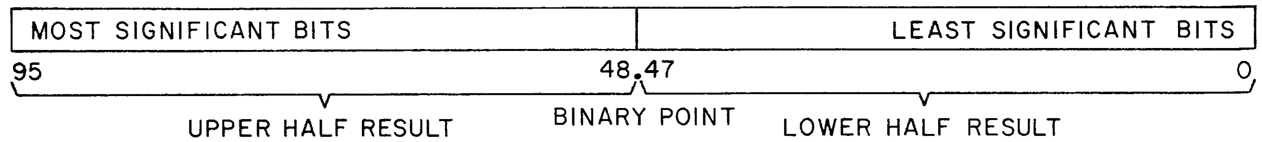
## ROUNDED COMPUTATION

Optional floating point instructions are provided to round the results in single precision computation. These instructions are executed in the same amount of time as the unrounded versions. The operands are modified in the functional units to accomplish the rounding function. The amount of bias introduced by the rounding operation varies from unit to unit and is affected by the coefficient value in the operands. The descriptions of the round instructions in Section 3 define the effects of rounding, in detail.

## DOUBLE PRECISION

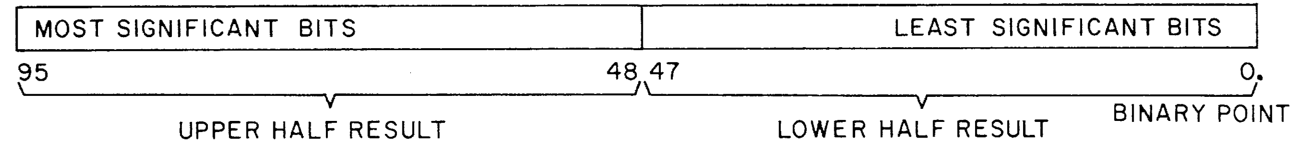
The floating point arithmetic instructions generate double-precision results. Use of unrounded instructions allows separate recovery of upper and lower half results with proper exponents; rounded instructions allow only upper half results to be obtained. The position of the binary point and the exponent of the double precision result depend upon the arithmetic operation chosen.

To add or subtract two floating point numbers, the floating point Add unit enters the coefficient having the smaller exponent into the upper half of an accumulator and shifts it right by the difference of the exponents. Then it adds the other coefficient into the upper half of the accumulator. The result is a double length register with the following format:




If single precision is selected, the upper 48 bits of the 96-bit result and the larger exponent is the result. Selecting double precision causes the lower 48 bits of the 96-bit result and the larger exponent minus $60_8$ to be returned as the result. The subtraction of $60_8$ is necessary because the binary point is effectively moved from the right of bit 48 to the right of bit 0 .

The Multiply units generate 96-bit products from two 48-bit coefficients. The result of a multiply is a double length register with the following format:




If single precision is selected, the upper 48 bits of the product and the sum of the exponents plus $60_8$ are returned as the result. The addition of $60_8$ is necessary because the binary point is effectively moved from the right of bit 0 to the right of bit 48 when the upper half of the 96-bit result is selected. If double precision is selected, the lower 48 bits of the product and the sum of the exponents is the result.

## INTEGER ARITHMETIC

There are no CPU integer multiply or divide instructions. Integer multiplication and division must be performed in the floating multiply and divide units. Integer arithmetic is accomplished by packing the integers into floating point format using the pack instruction with a zero exponent value.

In integer multiplication, a product can be formed for small integers without normalizing the operands by using the double precision multiply instruction. The result does not need to be unpacked if the destination is an A or a B register because the increment unit extracts only the lowest order 18 bits of the 60-bit word.

In integer division the divisor must be normalized with a Normalize instruction but the dividend need not be normalized. The resulting quotient must be unpacked and the coefficient shifted by the amount of the unpacked exponent using the Left Shift Nominally instruction to obtain the integer quotient.

# APPENDIX C

# MNEMONIC CODES

## MNEMONIC CODES

### CENTRAL PROCESSOR

| | | |
|---|---|---|
| ES | 00000 | Error exit to EEA |
| | | |
| RJ | 0100K | Return jump to K |
| RL | 011jK | Block copy K + (Bj) words from LCM to SCM |
| WL | 012jK | Block copy K + (Bj) words from SCM to LCM |
| MC | 013jk | Exchange exit to NEA if exit flag clear |
| ME | 013jK | Exchange exit to K + (Bj) if exit flag set |
| RX | 014jk | Read LCM at (Xk) to Xj |
| WX | 015jk | Write (Xj) into LCM at (Xk) |
| RI | 0160k | Reset channel (Bk) input buffer if j=0 |
| IB | 016jk | Read channel (Bk) input status to Bj if j≠0 |
| TB | 016j0 | Set Bj to current clock time |
| RO | 0170k | Reset channel (Bk) output buffer if j=0 |
| OB | 017jk | Read channel (Bk) output status to Bj if j≠0 |
| | | |
| JP | 02i0K | Jump to K + (Bi) |
| | | |
| ZR | 030jK | Branch to K if (Xj)=0 |
| NZ | 031jK | Branch to K if (Xj)≠0 |
| PL | 032jK | Branch to K if (Xj) positive |
| NG | 033jK | Branch to K if (Xj) negative |
| IR | 034jK | Branch to K if (Xj) in range |
| OR | 035jK | Branch to K if (Xj) not in range |
| DF | 036jK | Branch to K if (Xj) definite |
| ID | 037jK | Branch to K if (Xj) indefinite |
| | | |
| EQ | 04ijK | Branch to K if (Bi)=(Bj) |
| NE | 05ijK | Branch to K if (Bi)≠(Bj) |
| GE | 06ijK | Branch to K if (Bi)≥ (Bj) |
| LT | 07ijK | Branch to K if (Bi)<(Bj) |

| | | |
|---|---|---|
| BX | 10ij0 | Copy (Xj) to Xi |
| BX | 11ijk | Logical product of (Xj) and(Xk) to Xi |
| BX | 12ijk | Logical sum of (Xj) plus (Xk) to Xi |
| BX | 13ijk | Logical difference of (Xj) minus (Xk) to Xi |
| BX | 14i0k | Copy complement of (Xk) to Xi |
| BX | 15ijk | Logical product of (Xj) and comp (Xk) to Xi |
| BX | 16ijk | Logical sum (Xj) plus comp (Xk) to Xi |
| BX | 17ijk | Logical difference of (Xj) minus comp (Xk) to Xi |
| | | |
| LX | 20ijk | Left shift (Xi) by jk |
| AX | 21ijk | Right shift (Xi) by jk |
| LX | 22ijk | Left shift (Xk) by (Bj) to Xi |
| AX | 23ijk | Right shift (Xk) by (Bj) to Xi |
| NX | 24ijk | Normalize (Xk) to Xi and Bj |
| ZX | 25ijk | Round and normalize (Xk) to Xi and Bj |
| UX | 26ijk | Unpack (Xk) to Xi and Bj |
| PX | 27ijk | Pack (Xk) and (Bj) to Xi |
| | | |
| FX | 30ijk | Floating sum of (Xj) plus (Xk) to Xi |
| FX | 31ijk | Floating difference of (Xj) minus (Xk) to Xi |
| DX | 32ijk | Floating DP sum of (Xj) plus (Xk) to Xi |
| DX | 33ijk | Floating DP difference of (Xj) minus (Xk) to Xi |
| RX | 34ijk | Round floating sum of (Xj) plus (Xk) to Xi |
| RX | 35ijk | Round floating difference of (Xj) minus (Xk) to Xi |
| IX | 36ijk | Integer sum of (Xj) plus (Xk) to Xi |
| IX | 37ijk | Integer difference of (Xj) minus (Xk) to Xi |
| | | |
| FX | 40ijk | Floating product of (Xj) times (Xk) to Xi |
| RX | 41ijk | Round floating product of (Xj) times (Xk) to Xi |
| DX | 42ijk | Floating DP product of (Xj) times (Xk) to Xi |
| MX | 43ijk | Form mask of jk bits to Xi |
| FX | 44ijk | Floating divide (Xj) by (Xk) to Xi |
| RX | 45ijk | Round floating divide (Xj) by (Xk) to Xi |
| NO | 46000 | Pass |
| CX | 47i0k | Population count of (Xk) to Xi |

| | | |
|---|---|---|
| SA | 50ijK | Increment (Aj) plus K to Ai |
| SA | 51ijK | Increment (Bj) plus K to Ai |
| SA | 52ijK | Increment (Xj) plus K to Ai |
| SA | 53ijk | Increment (Xj) plus (Bk) to Ai |
| SA | 54ijk | Increment (Aj) plus (Bk) to Ai |
| SA | 55ijk | Increment (Aj) minus (Bk) to Ai |
| SA | 56ijk | Increment (Bj) plus (Bk) to Ai |
| SA | 57ijk | Increment (Bj) minus (Bk) to Ai |
| SB | 60ijK | Increment (Aj) plus K to Bi |
| SB | 61ijK | Increment (Bj) plus K to Bi |
| SB | 62ijK | Increment (Xj) plus K to Bi |
| SB | 63ijk | Increment (Xj) plus (Bk) to Bi |
| SB | 64ijk | Increment (Aj) plus (Bk) to Bi |
| SB | 65ijk | Increment (Aj) minus (Bk) to Bi |
| SB | 66ijk | Increment (Bj) plus (Bk) to Bi |
| SB | 67ijk | Increment (Bj) minus (Bk) to Bi |
| SX | 70ijK | Increment (Aj) plus K to Xi |
| SX | 71ijK | Increment (Bj) plus K to Xi |
| SX | 72ijK | Increment (Xj) plus K to Xi |
| SX | 73ijk | Increment (Xj) plus (Bk) to Xi |
| SX | 74ijk | Increment (Aj) plus (Bk) to Xi |
| SX | 75ijk | Increment (Aj) minus (Bk) to Xi |
| SX | 76ijk | Increment (Bj) plus (Bk) to Xi |
| SX | 77ijk | Increment (Bj) minus (Bk) to Xi |

PERIPHERAL PROCESSORS

| | | |
|---|---|---|
| ESN | 00 | Error Stop |
| LJM | 01 | Long jump to m + (d) |
| RJM | 02 | Return jump to m + (d) |
| UJN | 03 | Unconditional jump d |
| ZJN | 04 | Zero jump d |
| NJN | 05 | Nonzero jump d |
| PJN | 06 | Positive jump d |
| MJN | 07 | Negative jump d |

| | | |
|---|---|---|
| SHN | 10 | Shift d |
| LMN | 11 | Logical difference d |
| LPN | 12 | Logical product d |
| SCN | 13 | Selective clear d |
| LDN | 14 | Load d |
| LCN | 15 | Load complement d |
| ADN | 16 | Add d |
| SBN | 17 | Subtract d |
| | | |
| LDC | 20 | Load dm |
| ADC | 21 | Add dm |
| LPC | 22 | Logical product dm |
| LMC | 23 | Logical difference dm |
| PSN | 24 | Pass |
| PSN | 25 | Pass |
| PSN | 26 | Pass |
| PSN | 27 | Pass |
| | | |
| LDD | 30 | Load (d) |
| ADD | 31 | Add (d) |
| SBD | 32 | Subtract (d) |
| LMD | 33 | Logical difference (d) |
| STD | 34 | Store (d) |
| RAD | 35 | Replace add (d) |
| AOD | 36 | Replace add one (d) |
| SOD | 37 | Replace subtract one (d) |
| | | |
| LDI | 40 | Load ((d)) |
| ADI | 41 | Add ((d)) |
| SBI | 42 | Subtract ((d)) |
| LMI | 43 | Logical difference ((d)) |
| STI | 44 | Store ((d)) |
| RAI | 45 | Replace add ((d)) |
| AOI | 46 | Replace add one ((d) |
| SOI | 47 | Replace subtract one ((d)) |

| | | |
|---|---|---|
| LDM | 50 | Load (m+(d)) |
| ADM | 51 | Add (m+(d)) |
| SBM | 52 | Subtract (m+(d)) |
| LMM | 53 | Logical difference (m+(d)) |
| STM | 54 | Store (m+(d)) |
| RAM | 55 | Replace add (m+(d)) |
| AOM | 56 | Replace add one (m+(d)) |
| SOM | 57 | Replace subtract one (m+(d)) |
| | | |
| FIM | 60 | Jump on input word flag |
| EIM | 61 | Jump if no input word flag |
| IRM | 62 | Jump on input record flag |
| NIM | 63 | Jump if no input record flag |
| FOM | 64 | Jump on output word flag |
| EOM | 65 | Jump if no output word flag |
| ORM | 66 | Jump on output record flag |
| NOM | 67 | Jump if no output record flag |
| | | |
| IAN | 70 | Input to A from channel d |
| IAM | 71 | Input (A) words to m from channel d |
| OAN | 72 | Output from A on channel d |
| OAM | 73 | Output (A) words from m on channel d |
| RFN | 74 | Output record flag on channel d |
| PSN | 75 | Pass |
| PSN | 76 | Pass |
| ESN | 77 | Error stop |

# COMMENT SHEET

MANUAL TITLE CONTROL DATA 7600 Computer System

Reference Manual

PUBLICATION NO. 60258200 REVISION

**FROM:** NAME:

BUSINESS ADDRESS:

## COMMENTS:

This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number references and fill in publication revision level as shown by the last entry on the Record of Revision page at the front of the manual. Customer engineers are urged to use the TAR.

CUT ALONG LINE

FORM CA231 REV.1-67 PRINTED IN U.S.A

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

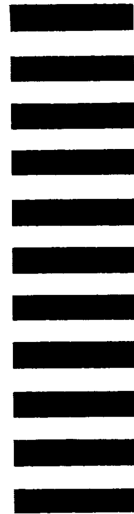
FOLD ON DOTTED LINES AND STAPLE

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.



**BUSINESS REPLY MAIL**
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

**CONTROL DATA CORPORATION**

**8100 34TH AVENUE SOUTH**

**MINNEAPOLIS, MINNESOTA 55440**

**ATTN: TECHNICAL PUBLICATIONS DEPT.**
**PLANT TWO**

CUT ALONG LINE

FOLD FOLD

60258200 LITHO USA

CONTROL DATA
CORPORATION