

CONTROL DATA
CORPORATION

CONTROL DATA[®]
7600 COMPUTER SYSTEM

REFERENCE MANUAL

INSTRUCTION INDEX

CENTRAL PROCESSOR

00000	Error exit to EEA	3-2
0100K	Return jump to K	3-17
011jK	Block copy (Bj) + K words from LCM to SCM	3-3
012jK	Block copy (Bj) + K words from SCM to LCM	3-6
013jk	Exchange exit to NEA (Exit Mode flag cleared)	3-8
013jK	Exchange exit to (Bj) + K (Exit Mode Flag set)	3-9
014jk	Read LCM at (Xk) to Xj	3-10
015jk	Write Xj into LCM at (Xk)	3-11
0160k	Reset Input channel (Bk) buffer if j=0	3-12
016jk	Read Input channel (Bk) status to Bj, if j≠0	3-13
0170k	Reset Output channel (Bk) buffer if j=0	3-14
017jk	Read Output channel (Bk) status to Bj if j≠0	3-16
02i0K	Jump to (Bi) + K	3-19
030jK	Branch to K if (Xj) = 0	3-20
031jK	Branch to K if (Xj) ≠ 0	3-20
032jK	Branch to K if (Xj) positive	3-20
033jK	Branch to K if (Xj) negative	3-20
034jK	Branch to K if (Xj) in range	3-20
035jK	Branch to K if (Xj) out of range	3-20
036jK	Branch to K if (Xj) definite	3-20
037jK	Branch to K if (Xj) indefinite	3-20
04ijK	Branch to K if (Bi) = (Bj)	3-21
05ijK	Branch to K if (Bi) ≠ (Bj)	3-21
06ijK	Branch to K if (Bi) > (Bj)	3-21
07ijK	Branch to K if (Bi) < (Bj)	3-21
10ij0	Transmit (Xj) to Xi	3-22
11ijk	Logical product of (Xj) and (Xk) to Xi	3-22
12ijk	Logical sum of (Xj) and (Xk) to Xi	3-23
13ijk	Logical difference of (Xj) and (Xk) to Xi	3-24
14i0k	Transmit complement of (Xk) to Xi	3-24
15ijk	Logical product of (Xj) and comp (Xk) to Xi	3-25
16ijk	Logical sum (Xj) and comp (Xk) to Xi	3-25
17ijk	Logical difference of (Xj) and comp (Xk) to Xi	3-26
20ijk	Left shift (Xi) by jk	3-29
21ijk	Right shift (Xi) by jk	3-29
22ijk	Left shift (Xk) nominally (Bj) places to Xi	3-30
23ijk	Right shift (Xk) nominally (Bj) places to Xi	3-31
24ijk	Normalize (Xk) to Xi and Bj	3-32
25ijk	Round and normalize (Xk) to Xi and Bj	3-33
26ijk	Unpack (Xk) to Xi and Bj	3-27
27ijk	Pack (Xk) and (Bj) to Xi	3-28
30ijk	Floating sum of (Xj) and (Xk) to Xi	3-34
31ijk	Floating difference of (Xj) minus (Xk) to Xi	3-36
32ijk	Floating DP sum of (Xj) and (Xk) to Xi	3-36
33ijk	Floating DP difference of (Xj) minus (Xk) to Xi	3-37
34ijk	Round floating sum of (Xj) and (Xk) to Xi	3-38
35ijk	Round floating difference of (Xj) minus (Xk) to Xi	3-39
36ijk	Integer sum of (Xj) plus (Xk) to Xi	3-39
37ijk	Integer difference of (Xj) minus (Xk) to Xi	3-40
40ijk	Floating product of (Xj) and (Xk) to Xi	3-40
41ijk	Round floating product of (Xj) and (Xk) to Xi	3-42
42ijk	Floating DP product of (Xj) and (Xk) to Xi	3-43
43ijk	Form mask of jk bits to Xi	3-32
44ijk	Floating divide (Xj) by (Xk) to Xi	3-43
45ijk	Round floating divide (Xj) by (Xk) to Xi	3-45
46000	No operation (pass)	3-46
47i0k	Population count of (Xk) to Xi	3-46
50ijk	Set Ai to (Aj) + K	3-47
51ijk	Set Ai to (Bj) + K	3-47
52ijk	Set Ai to (Xj) + K	3-47
53ijk	Set Ai to (Xj) + (Bk)	3-47
54ijk	Set Ai to (Aj) + (Bk)	3-47
55ijk	Set Ai to (Aj) - (Bk)	3-47
56ijk	Set Ai to (Bj) + (Bk)	3-47
57ijk	Set Ai to (Bj) - (Bk)	3-47
60ijk	Set Bi to (Aj) + K	3-48
61ijk	Set Bi to (Bj) + K	3-48
62ijk	Set Bi to (Xj) + K	3-48
63ijk	Set Bi to (Xj) + (Bk)	3-48
64ijk	Set Bi to (Aj) + (Bk)	3-48
65ijk	Set Bi to (Aj) - (Bk)	3-48
66ijk	Set Bi to (Bj) + (Bk)	3-48
67ijk	Set Bi to (Bj) - (Bk)	3-48

70ijK	Set Xi to (Ai) + K	3-49
71ijK	Set Xi to (Bj) + K	3-49
72ijK	Set Xi to (Xj) + K	3-49
73ijk	Set Xi to (Xj) + (Bk)	3-49
74ijk	Set Xi to (Aj) + (Bk)	3-49
75ijk	Set Xi to (Aj) - (Bk)	3-49
76ijk	Set Xi to (Bj) + (Bk)	3-49
77ijk	Set Xi to (Bj) - (Bk)	3-49

PERIPHERAL PROCESSORS

00	Error Stop	6-5
01	Long jump to m + (d)	6-6
02	Return jump to m + (d)	6-6
03	Unconditional jump d	6-6
04	Zero jump d	6-7
05	Nonzero jump d	6-7
06	Plus jump d	6-7
07	Minus jump d	6-8
10	Shift (A) by d	6-8
11	Logical difference (A) and d	6-8
12	Logical product (A) and d	6-9
13	Selective clear (A)	6-9
14	Load d	6-9
15	Load complement d	6-10
16	Add (A) + d	6-10
17	Subtract (A) - d	6-10
20	Load dm	6-10
21	Add (A) + dm	6-11
22	Logical product (A) and dm	6-11
23	Logical difference (A) and dm	6-11
24	Pass	6-5
25	Pass	6-5
26	Pass	6-5
27	Pass	6-5
30	Load (d)	6-12
31	Add (d) + (A)	6-12
32	Subtract (A) - (d)	6-12
33	Logical difference (A) and (d)	6-13
34	Store (A) at d	6-13
35	Replace add (A) + (d)	6-13
36	Replace add one (d)	6-14
37	Replace subtract one (d)	6-14
40	Load ((d))	6-15
41	Add (A) + ((d))	6-15
42	Subtract (A) - ((d))	6-15
43	Logical difference (A) and ((d))	6-16
44	Store (A) at (d)	6-16
45	Replace add (A) + ((d))	6-16
46	Replace add one ((d))	6-17
47	Replace subtract one ((d))	6-17
50	Load (m + (d))	6-17
51	Add (A) + (m + (d))	6-18
52	Subtract (A) - (m + (d))	6-18
53	Logical difference (A) and (m + (d))	6-18
54	Store (A) at m + (d)	6-19
55	Replace add (A) + (m + (d))	6-19
56	Replace add one (m + (d))	6-19
57	Replace subtract one (m + (d))	6-20
60	Jump on input word flag	6-20
61	Jump if no input word flag	6-20
62	Jump on input record flag	6-21
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	6-21
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	6-24
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

REFERENCE MANUAL

CONTENTS

<p>1. SYSTEM DESCRIPTION</p> <p>Introduction 1-1</p> <p style="padding-left: 20px;">Central Processor Unit Characteristics 1-1</p> <p style="padding-left: 20px;">Peripheral Processor Unit Characteristics 1-4</p> <p>Basic System Description 1-4</p> <p style="padding-left: 20px;">Central Processor Unit (CPU) 1-5</p> <p style="padding-left: 20px;">Peripheral Processor Unit (PPU) 1-7</p> <p style="padding-left: 20px;">Maintenance Control Unit (MCU) 1-7</p> <p style="padding-left: 20px;">Power Distribution Cabinet 1-8</p> <p style="padding-left: 20px;">Refrigeration System 1-8</p> <p style="padding-left: 20px;">Operator Station 1-8</p> <p>System Communication 1-10</p> <p>2. CENTRAL PROCESSOR UNIT</p> <p>Computation Section 2-1</p> <p style="padding-left: 20px;">Operating Registers 2-1</p> <p style="padding-left: 20px;">CPU Instruction Formats 2-4</p> <p style="padding-left: 20px;">Instruction Word Stack 2-6</p> <p style="padding-left: 20px;">Functional Units 2-9</p> <p style="padding-left: 20px;">Exchange Jump 2-10</p> <p style="padding-left: 20px;">Program Status Designators 2-15</p> <p>3. CENTRAL PROCESSOR INSTRUCTIONS</p> <p>Instruction Formats 3-1</p> <p>Monitor, LCM and I/O 3-2</p> <p>Branch 3-17</p> <p>Boolean Unit 3-22</p> <p>Shift Unit 3-29</p> <p>Normalize Unit 3-32</p>		<p>Floating Point Add Unit 3-34</p> <p>Long Add Unit 3-39</p> <p>Floating Point Multiply Unit 3-40</p> <p>Floating Point Divide Unit 3-43</p> <p>Pass Instruction 3-45</p> <p>Population Count Unit 3-46</p> <p>Increment Unit 3-47</p> <p>4. CENTRAL PROCESSOR MEMORY</p> <p>Introduction 4-1</p> <p>Memory Protection 4-1</p> <p>Small Core Memory 4-2</p> <p style="padding-left: 20px;">Address Format 4-3</p> <p style="padding-left: 20px;">Parity Conditions 4-3</p> <p style="padding-left: 20px;">Duty Cycle Integrator 4-3</p> <p style="padding-left: 20px;">Small Core Memory Access 4-4</p> <p style="padding-left: 20px;">Memory Reference 4-7</p> <p style="padding-left: 20px;">I/O Multiplexer 4-8</p> <p>Large Core Memory 4-15</p> <p style="padding-left: 20px;">Address Format 4-16</p> <p style="padding-left: 20px;">Parity Conditions 4-16</p> <p style="padding-left: 20px;">Large Core Memory Access 4-17</p> <p>5. PERIPHERAL PROCESSOR UNIT</p> <p>Organization 5-1</p> <p style="padding-left: 20px;">Computation Section 5-2</p> <p style="padding-left: 20px;">Memory 5-3</p> <p style="padding-left: 20px;">Input/Output 5-3</p> <p>6. PERIPHERAL PROCESSOR INSTRUCTIONS</p> <p>Instruction Formats 6-1</p>
--	--	---

Address Modes	6-1	Maintenance Control Unit	7-1
No Address Mode	6-3	MCU Scanner	7-1
Constant Mode	6-3	MCU Dead Start	7-3
Direct Address Mode	6-3	PPU Dead Start	7-3
Indexed Direct Address Mode	6-3	CPU Dead Start	7-4
Indirect Address Mode	6-3	PPU Dead Dump	7-4
Description of Peripheral Instructions	6-4	PPU and MCU Parity Errors	7-4
Error Stop	6-5	SCM and LCM Parity Errors	7-5
No Operation	6-5	PPU Program Error	7-6
Branch	6-6	Console	7-6
No Address	6-8	Alphanumeric Keyboard	7-6
Constant	6-10	Display	7-7
Direct Address	6-12		
Indirect Address	6-15	APPENDIXES	
Indexed Direct Address	6-17	A. Timing Notes	
Input/Output	6-20	B. Floating Point Arithmetic	
		C. Mnemonic Codes	
7. MANUAL CONTROL		D. 6000/7000 Result Differences	
Introduction	7-1		

FIGURES

1-1 Basic System Diagram	1-2	4-5 LCM Address Format	4-16
1-2 Typical Operator Station	1-9	5-1 Signals for One PPU Bi-Directional Channel	5-8
2-1 CPU Information Flow	2-2	5-2 PPU/Controller Communications	5-11
2-2 Parcel Instruction Arrangement	2-5	6-1 PPU 12-Bit Instruction Format	6-1
2-3 Exchange Package	2-11	6-2 PPU 24-Bit Instruction Format	6-1
2-4 Flag and Register Arrangement	2-16	6-3 71 Flow Chart	6-23
4-1 Memory Map	4-2	6-4 73 Flow Chart	6-25
4-2 SCM Address Format	4-3	7-1 MCU Configuration	7-2
4-3 I/O Exchange Package Areas	4-5	7-2 Display Console	7-8
4-4 Typical Buffer Area Arrangements	4-9		

TABLES

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

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 (Figure 1-1) 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.

CENTRAL PROCESSOR UNIT CHARACTERISTICS

Computation Section

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

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

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

Operating Registers

- eight 60-bit operand registers (X registers)

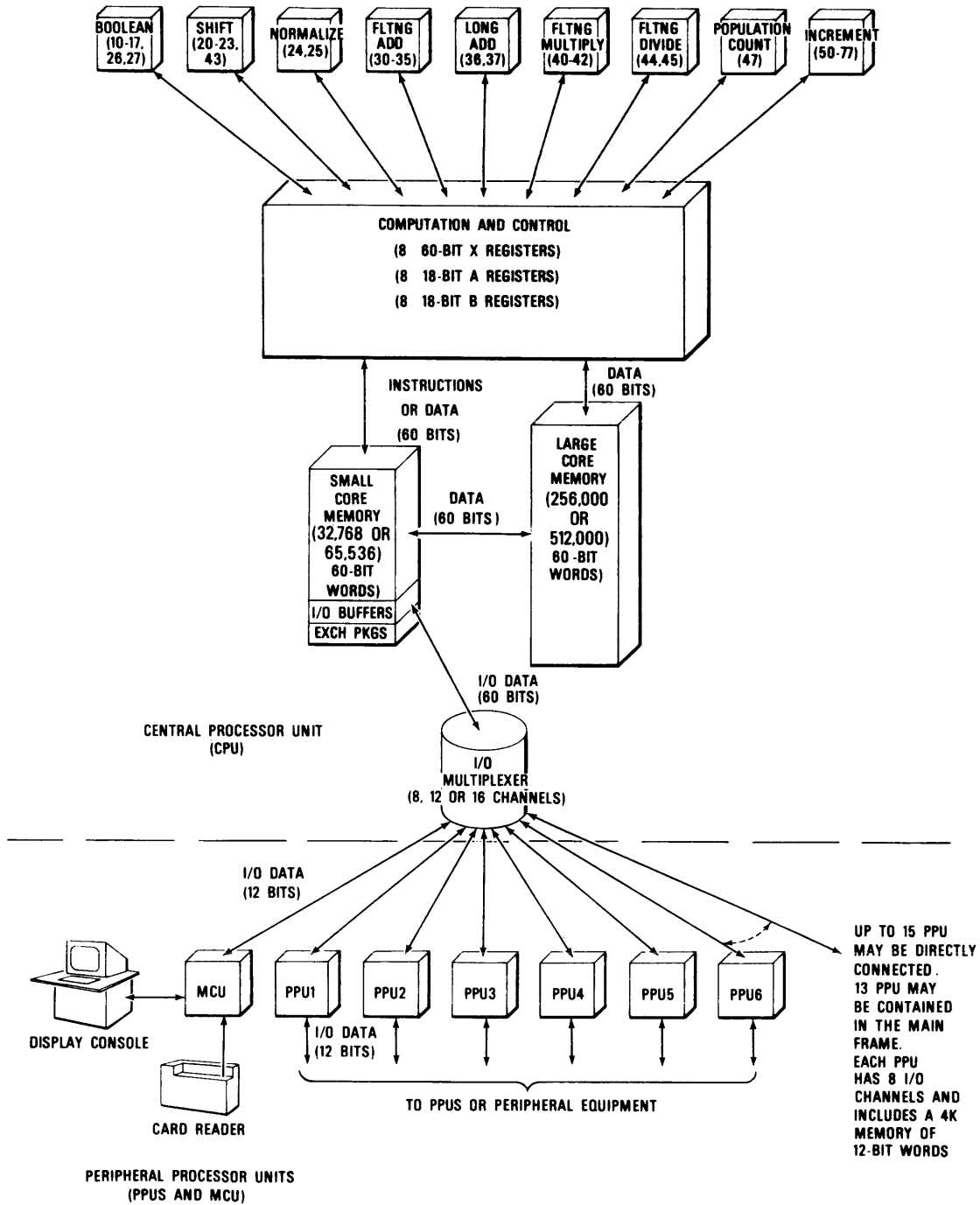


Figure 1-1. Basic System Diagram

- eight 18-bit address registers (A registers)
- eight 18-bit index registers (B registers)

Small Core Memory

- 32,768 or 65,536 60-bit words of coincident current memory with five parity bits per 60-bit word
- organized into 16 or 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

- 256,000 or 512,000 60-bit words of linear select memory with four parity bits per 60-bit word
- organized into four or eight independent banks (64,000 words per bank)
- 1,760 nanosecond read/write cycle time
- eight words read simultaneously each reference
- 27.5 nanosecond per word maximum transfer rate (512,000 memory only)

Multiplexer

- Seven independent 12-bit channels (expandable to 15 channels)
- each channel bi-directional
- fixed SCM buffer areas for each channel; 128 or 256 60-bit words
- normal channels and high speed channels

System Options

- I/O Multiplexer Channel Increments (MUXI) to increase the number of channels to 11 or 15
- Large Core Memory Increment (LCMI) to increase a basic system of 256,000 words to 512,000 60-bit words
- Small Core Memory Increment (SCMI) to increase a basic system of 32,768 words to 65,536 60-bit words

PERIPHERAL PROCESSOR UNIT CHARACTERISTICS

Computation Section

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

Operating Registers

- 18-bit Arithmetic Register (A)
- 12-bit Program Address Register (P)
- 13-bit Memory Read Register (X)
- 12-bit Instruction Register (fd)
- 12-bit Working Register (Q)

Core Memory

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

Input/Output Section

- eight independent channels (asynchronous)
- each channel bi-directional (12-Bit)

BASIC SYSTEM DESCRIPTION

The 7600 mainframe includes a Central Processor Unit (with its associated memory), a Maintenance Control Unit, and 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. Unless otherwise specified, these descriptions assume the largest system configuration of memory and I/O channels.

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 the 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 nine 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 contains 12 60-bit words for issuing of instructions. This register memory holds the latest ten instruction words and the two-instruction word look-ahead. Program loops can be held in the stack thereby avoiding memory references.
2. The Small Core Memory (SCM) contains 32,768 or 65,536 60-bit words arranged in 16 or 32 banks of 2,048 60-bit words each. Each bank is phased; that is, consecutive addresses go to different banks. This gives a marked decrease in memory conflicts and allows overlapping of memory cycles. Each bank is made up of ten memory stacks. Each stack contains 1,024 12-bit words plus one parity bit per 12-bit word (odd parity). 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 256,000 or 512,000 60-bit words arranged in four or eight phased banks. This memory is a linear select memory with one parity bit for each 15 bits (odd parity). Each LCM word contains eight 60-bit words 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 CPU 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 bi-directional channel. In the basic system, there are eight such channels in the MUX, one of which is reserved for use by the Maintenance Control unit. Each channel has assembly/disassembly registers to convert 12-bit channel data to 60-bit CPU 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 CPU program for monitor action on the buffer data.

Each channel normally 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 I/O exchange package areas and the buffer areas are permanently assigned in the lowest order addresses of SCM.

Some of the I/O channels are called high speed channels as opposed to normal channels. High speed channels transfer data at approximately twice the speed of normal channels. The maximum speeds of normal and high speed channels are listed below.

	<u>Normal Channel</u>	<u>High Speed Channel</u>
Input:	60 clock periods/60-bit word	34 clock periods/60-bit word
Output:	72 clock periods/60-bit word	35 clock periods/60-bit word

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. A PPU may be connected to SCM, another PPU, a peripheral device or a mix of these. PPU's that connect directly to the CPU, whether on the mainframe or remotely located, are termed first level PPU's. Each PPU has a computation section that performs binary computation in fixed point arithmetic. A PPU 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 memory stacks used in a bank contain 1024 12-bit words each. These same stacks are used in groups of five to form SCM. The PPU 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 PPU input/output facility provides a flexible arrangement for very high speed communication with a variety of I/O devices. The bi-directional 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 is capable of selecting any PPU that 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 the 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 directing and monitoring system maintenance activity.

POWER DISTRIBUTION CABINET

The power distribution cabinet distributes 400-Hz power to the central computer dc power supplies. It also contains a warning system that monitors logic chassis temperature, room dew point temperature, and condensing unit condition.

A warning panel contains relay circuits that activate a horn and automatically shut off computer power when cooling system malfunctions occur.

REFRIGERATION SYSTEM

Two water-cooled condensing units, each with a capacity rating of 10 tons, provide cooling for the mainframe and up to two stand-alone Operator Stations adjacent to the mainframe. Each remotely located Operator Station is cooled by a condensing unit with a capacity of two tons. The condensing units cool by pumping R12 refrigerant through cold plates in each chassis.

There are two types of two-ton condensing units, water-cooled and air-cooled. In the water-cooled condensing units, water from an external source removes the heat from the condenser. In the air-cooled condensing units, fans draw air around refrigerant tubes in the condenser. The air absorbs heat from the refrigerant and exhausts the heat into the computer room.

OPERATOR STATION

An Operator Station (Figure 1-2) is a self-contained data processing system that serves primarily as an input/output processor for the CPU. Typically, the Operator Station is composed of:

- 6 Peripheral Processor Units
- 1 Display console

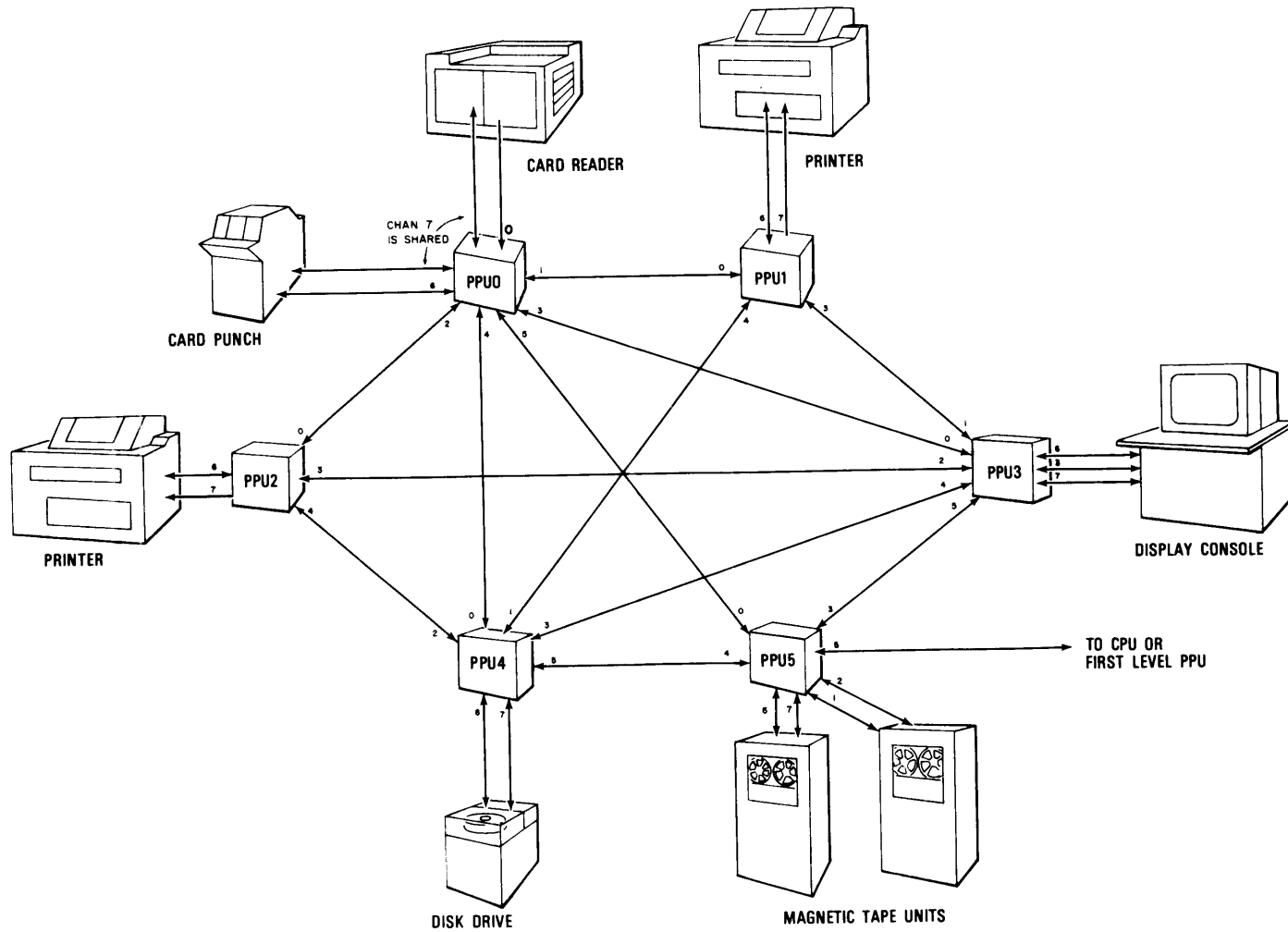


Figure 1-2. Typical Operator Station

- 1 Disk drive
- 1 Card reader with Dead Start capability
- 1 Card punch unit
- 2 Line printers
- 2 Magnetic tape drives

One of the six PPU's in the Operator Station usually connects to the CPU through an I/O Multiplexer channel. An alternate method is for the Operator Station to connect to the CPU through a first level PPU residing in the mainframe.

The maximum cable length between the CPU and a first level PPU, or between a first level PPU and the Operator Station is 200 feet. Communication is over 12-bit bi-directional channels.

The PPU connected to the card reader receives the Dead Start signal and is assigned the task of loading the resident programs and Dead Starting the other five PPU's in the station. The system also performs its own parity error detection and dead dump procedures.

SYSTEM COMMUNICATION

System communication paths are illustrated in Figure 1-1. All input data enters and leaves the system via peripheral equipment. The PPU's gather input data from the peripheral equipment 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 or controller can deliver or accept data.

Communication between a PPU and the CPU is over a channel identical to that used for communication between the PPU and peripheral equipment. All 15 CPU 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 I/O interrupts that initiate a CPU program that can empty or fill the buffer area some fifty times faster than a PPU can fill or empty it.

For 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 reinitiate the CPU program which has completed its task of emptying the lower half of the buffer. Meanwhile, the PPU starts to refill the buffer at the lower address.

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:

<u>Function</u>	<u>Identity</u>	<u>Length</u>	<u>Number</u>
Operand Registers	X0 - X7	60 Bits	8
Address Registers	A0 - A7	18 Bits	8
Index Registers	B0 - B7	18 Bits	8

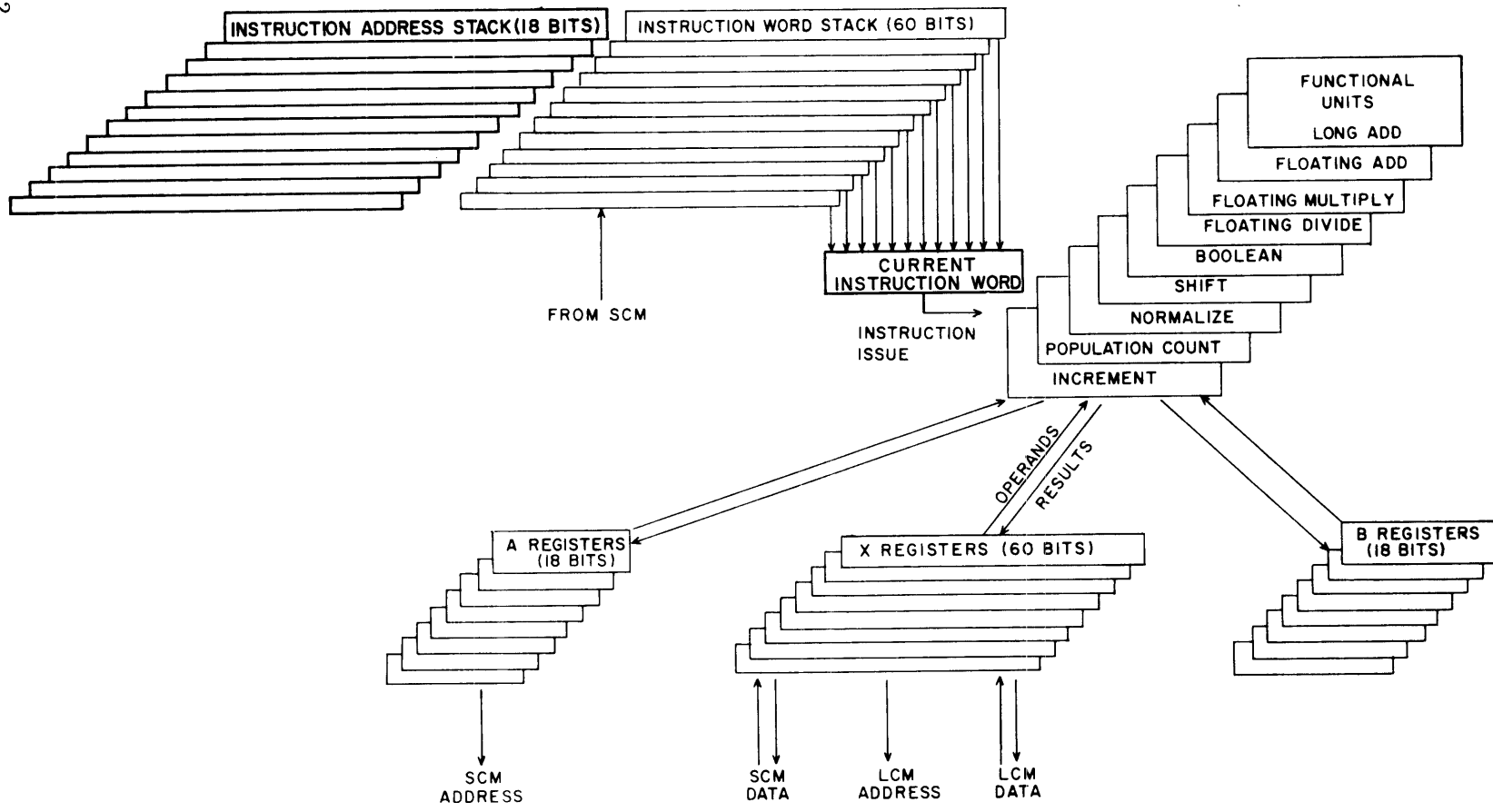


Figure 2-1. CPU Information Flow

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 SCM addresses into corresponding address registers.

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 registers also hold shift counts for pack and normalize operations and the channel number for channel status requests.

B0 always contains positive zero. It can be used as an operand (positive 0), but cannot hold results from instructions.

CPU INSTRUCTION FORMATS

Program instruction words are divided into 15-bit fields called parcels. The first parcel (parcel 0) is the highest order 15 bits of the 60-bit word. The second, third, and fourth parcels (parcels 1, 2 and 3) 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 contained all zeros; it will not obtain its second 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.

A one parcel Pass instruction 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 60, 61, 62 instructions with i equal to zero become Pass instructions, (Page 3-48). Since these are 30-bit instructions they may be used as two parcel pass instructions. Pass instructions 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 six 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.

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 of up to ten instruction words may 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 associated 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 corresponds 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 an instruction word is sent to the Current Instruction Word register.
2. P is set to the address specified by a 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 frame. 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 (parcel 0 instruction first) 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 may 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.

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.

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 an instruction word is sent to the Current Instruction Word register.
2. P is set to the address specified by a 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 frame. 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 (parcel 0 instruction first) 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.

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.

The floating divide unit is the only functional unit in which an iterative algorithm is executed. There is no segmentation possible in this unit. However, to increase execution speed, the beginning of a new divide operation can follow a previous divide operation by 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) - 18 bits
2. Reference address for Small Core Memory (RAS) - 18 bits
3. Field length of program for Small Core Memory (FLS) - 18 bits
4. Reference address for Large Core Memory (RAL) - 19 bits






SCM LOCATION	59	53	35	17	0
n		P	A0	BPA	
n + 1		RAS	A1	B1	
n + 2		FLS	A2	B2	
n + 3		PSD	A3	B3	
n + 4		64 RAL	A4	B4	
n + 5		FLL	A5	B5	
n + 6		64 NEA	A6	B6	
n + 7		EEA	A7	B7	
n + 8	X0				
n + 9	X1				
n + 10	X2				
n + 11	X3				
n + 12	X4				
n + 13	X5				
n + 14	X6				
n + 15	X7				

Figure 2-3. Exchange Package



No hardware registers exist; bits not used.



Hardware registers exist; bits not used. These bits are reserved for hardware use and are not to be used as software flags.



Hardware registers exist; system software uses bits $2^{54}-2^{59}$

5. Field length of program for Large Core Memory (FLL) - 19 bits
6. Program Status Designation register (PSD) - 18 bits
7. Normal exit address (NEA) - 16 bits
8. Error exit address (EEA) - 16 bits
9. Breakpoint address (BPA) - 18 bits
10. Current contents of the eight A registers
11. Current contents of the eight X registers
12. Current 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 Designators (page 2-15).

INPUT/OUTPUT INTERRUPT

The I/O Multiplexer section of the CPU monitors I/O activity between the PPU and SCM. The multiplexer issues an interrupt request to the CPU when the threshold of an SCM input or output buffer is reached. A Record Pulse from a PPU also causes an interrupt request. When accepted, an I/O interrupt request initiates an exchange jump to the CPU program.

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 Read Input Channel (0) Status instruction. An overflow of the highest order bit in this counter sets the Real Time Clock Interrupt flag, which can be seen as the 18th bit when the time is read.

The Real Time Clock Interrupt flag 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) are equal to (P) in the initiating exchange jump package, no instructions are executed. Normally, no instructions are executed at address BPA.

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 DESIGNATORS

The Program Status Designator 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.

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 $(B_j) + K + (RAS)$. 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.

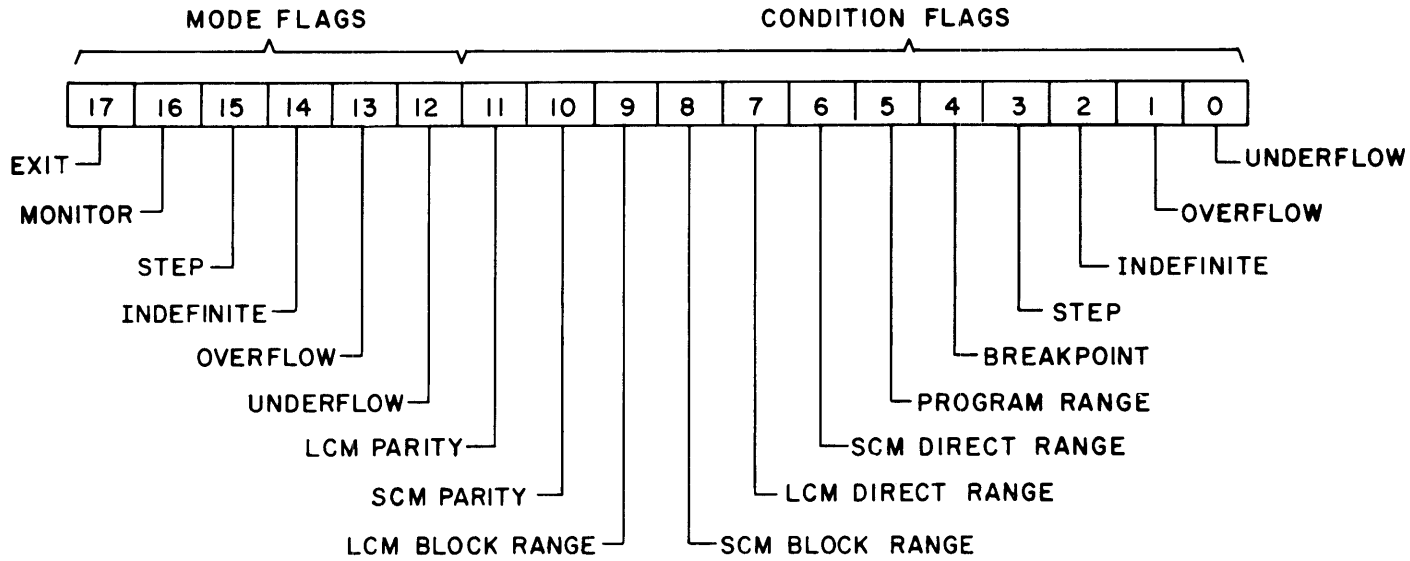


Figure 2-4. Flag and Register Arrangement

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. 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. Note, however, that this 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 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. Note, however, that this 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.

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. Note, however, that this 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.

CONDITION FLAGS

Whenever Condition flags $2^3 - 2^{11}$ enter the PSD register from an exchange package an error exit will occur and no program instructions will be executed. Condition flags $2^0 - 2^2$ require the corresponding Mode flags to be set.

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 Increment instruction (50-57) issues that causes an SCM reference to 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 (00) is issued. When this flag is set by P equal to zero, the execution interval for the exchange package terminates immediately. When this flag is set by a 00 instruction, execution 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.

INSTRUCTION FORMATS

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 those 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).

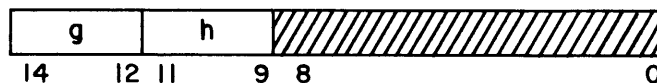
TABLE 3-1. CENTRAL PROCESSOR INSTRUCTION DESIGNATORS (Cont'd)

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.

MONITOR, LCM, AND I/O

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.

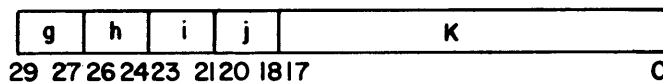
This instruction format is not intended for use in user program code. The Program Range Condition flag is set in the PSD register to indicate that the program has jumped to an area of the SCM field which may be in range but is not valid program code. This should occur when an incorrectly coded program 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.

The use of this instruction in user program code for system calls, etc., may result in erratic execution of the instruction (e.g. missed Error Exits).

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 relative address (X0) in the LCM field. The words are stored in the SCM field beginning at relative address (A0). The number of words to be copied is determined by the sum of K + (Bj). This quantity, (K + (Bj)) cannot exceed 1777₈ words. If a quantity larger than this is used LCM truncates the quantity to the 10-bit maximum. Thus a block count of 3000₈ words will transfer 1000₈ words. No error indications are given when this occurs, unless the field length is exceeded causing a Block Range Error.

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 B_j. Either quantity may be used to increment, or decrement, the other. The addition is performed in an 18-bit one's 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, B_j). 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.

LCM OUT OF RANGE

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 (B_j) 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. The exchange jump to (EEA) resulting from setting the LCM Block Range Condition flag will occur before execution of the next program 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. The exchange jump to (EEA) resulting from setting the SCM Block Range Condition flag will occur before execution of the next program 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 occur before execution of the next program 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.

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.

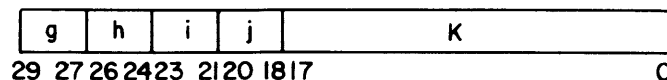
I/O INTERRUPT DURING EXECUTION

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.

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 relative address (A0) in the SCM field. The words are stored in the LCM field beginning at relative 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, unless the field length is exceeded causing a Block Range Error.

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 Storage Address Stack, 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.

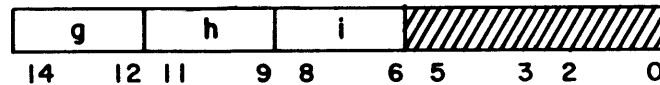
The lowest order 19 bits of (X0) are used to determine the initial address in the LCM field for the block copy. The higher order bits are ignored. If (X0) is negative the lowest order 19 bits are masked out and treated as a positive integer.

For treatment of special situations, refer to instruction 011 Block Copy LCM to SCM.

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.

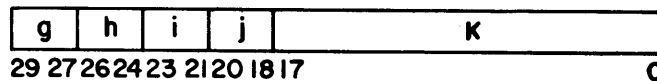
This instruction is intended for use in calling a system monitor program of input/output requests, library calls etc. All operating register values, program address, and mode selections are preserved in the exchange package for the object program in order that the object program may be continued at a 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 and the remaining instructions, if any, in the current instruction word will not be executed.

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* (30 Bits)
(Exit mode flag set)



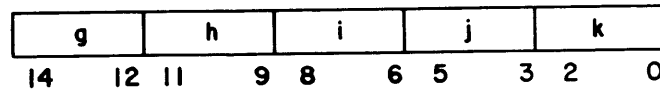
If the Exit Mode flag is set, this instruction causes the current program sequence to terminate with an exchange jump to an address in the SCM field for the current program. The exchange package is located at relative address (Bj) + K.

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, and the remaining instructions in the current program instruction word will not be executed.

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



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) and 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, with the exception of the first read.

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 which case the requested address is lost when the word arrives at the Xj register.

ADDRESS OUT OF RANGE

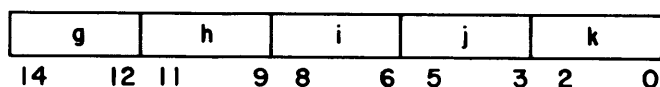
The lowest order 19 bits of (Xk) are compared with (FLL) to determine if the requested address is in the assigned LCM field. If the requested address is greater than, or equal to, (FLL) the LCM Direct Range Condition flag is set in the PSD register. This flag will cause an error exit request to interrupt 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 generally will not occur before one or more subsequent instructions have been executed.

015

Write Xj into LCM at (Xk)

(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, with the exception of the first read.

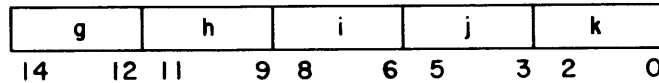
The lowest order 19 bits of (Xk) are used to determine the address in the LCM field. The higher order bits are ignored. If (Xk) is negative the lowest order 19 bits are masked out and treated as a positive integer. No error flags are set for these conditions unless the resulting address is out of range.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

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.

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 generally will not occur before one or more subsequent instructions have been executed.

0160 *Reset Input Channel (Bk) Buffer:* *(15 Bits)*
j = 0



This instruction prepares the (Bk) input channel buffer for a new record transmission from a PPU to SCM. The instruction clears the input channel buffer address and resets the input channel assembly counter to the first 12-bit position in the SCM word.

This instruction is intended to be privileged to an input routine; that is, one that terminates a record of incoming data and prepares for the next record.

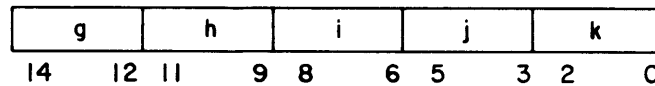
The input routine removes the data in the input channel buffer and then executes this instruction to prepare the buffer for the next incoming record. This instruction is effective only if the Monitor Mode flag is set in the Program Status register. If the Monitor Mode flag is cleared this instruction 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 input channel buffer.

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 or is for a nonexistent channel, this instruction becomes a Pass instruction.

Two or more Reset Input Channel 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 Channel 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 Input Channel (Bk) status
to Bj: j ≠ 0 (Read Real time clock:
(Bk) = 0)* (15 Bits)



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

This instruction is intended for use in monitoring the progress of input to the input channel buffer in SCM. The input channel buffer area is divided into 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 multiplexer interrupt request is generated by the threshold testing mechanism whenever the input channel 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 multiplexer interrupt request was generated by a buffer threshold test or by a Record flag. The program must retain the input channel 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.

For systems using less than the full complement of I/O channels, this instruction can be used to determine whether an input channel exists. Execution of this instruction to a nonexistent input channel causes a status word of 400000 to be entered into Bj.

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 Input Channel 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.

If correct results are to be obtained, a Read Input Channel Status instruction must not immediately follow a Reset Input Channel Buffer instruction. A delay of one clock period is sufficient.

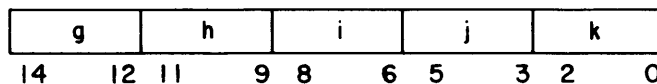
REAL TIME CLOCK

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 multiplexer 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 for a channel input buffer address.

0170

Reset Output Channel (Bk) Buffer: j = 0

(15 Bits)



This instruction initiates a new record transmission from SCM to a PPU. It clears the output channel (Bk) buffer address and disassembly counter, initiates a SCM reference for the first word to be output, and transmits a Record Pulse over the output channel data path to the PPU.

This instruction is intended for execution in an output routine to initiate a new record transmission over an output channel data path. The output channel buffer is normally inactive when this instruction is executed. The output channel buffer is loaded with the data for the next record, and this instruction is executed to initiate the transmission. A Record pulse and a Word pulse are transmitted as soon as the SCM word is entered in the output channel 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 output channel.

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 or is for a nonexistent channel, this instruction becomes a Pass instruction.

The output program should check for completion of the previous record before executing this instruction. There are two methods that a program can use to detect end of record. One method is to read the output channel buffer address and compare it with a known record length. The other is to receive a positive response from the peripheral unit over the corresponding input channel data path. If for some reason the output channel buffer is actively moving data over the output channel data path at the time this instruction is executed, conflicting commands may be sent to the multiplexer. 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 Channel 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.

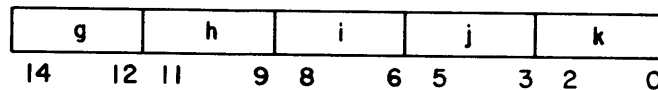
Two or more Reset Output Channel Buffer instructions may occur in consecutive program instruction locations referencing the same channel. These instructions will issue in consecutive clock periods and repeatedly perform the same functions. A Record pulse will be transmitted over the channel output data path for each instruction

execution. The output channel buffer will be repeatedly restarted, and a data word may, or may not, be transmitted over the output channel data path depending on the timing of the instructions and the conflicts that occur.

017

Read Output channel (Bk) status to Bj:
 $j \neq 0$

(15 Bits)



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

This instruction is intended for use in monitoring the progress of output from the output channel buffer. The output channel 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 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.

For systems using less than the full complement of I/O channels, this instruction can be used to determine whether an output channel exists. Execution of this instruction to a nonexistent output channel causes a status word of 000000 to be entered into Bj.

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.

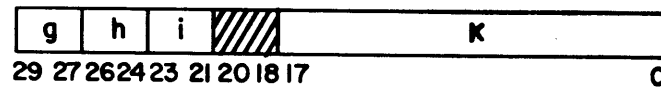
Two or more Read Output Channel Status instructions may occur in consecutive program instruction locations referencing the same or different channels. These instructions may issue in consecutive clock periods providing the Bj register reservations do not cause a delay. No interference will result in the multiplexer in these situations. If correct results are to be obtained, a Read Output Channel Status instruction must not immediately follow a Reset Output Channel Buffer instruction. A delay of one clock period is sufficient.

BRANCH

010

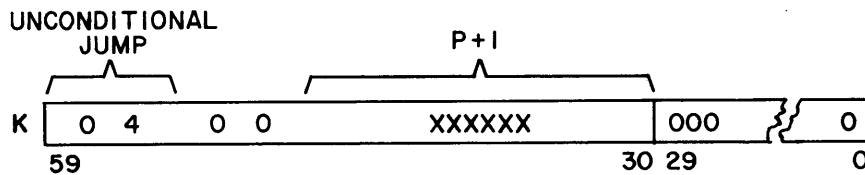
Return jump to K

(30 Bits)



This instruction stores an 04 unconditional jump and the current address plus one (P) + 1 in the upper half of relative 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.

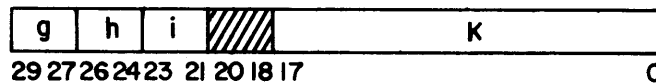
SPECIAL SITUATIONS

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 SCM Direct 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 Program 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 multiplexer 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.



This instruction adds the contents of index register B_i 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

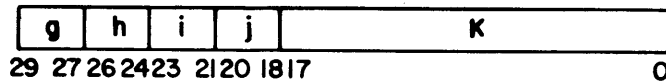
If an I/O interrupt request or an error exit request exists at the time this instruction is executed, the instruction is executed to completion before the interrupt occurs.

If the branch point destination address is greater than the SCM field length, the SCM Direct 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 Direct 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	Branch to K if $(X_j) = 0$	(30 Bits)
031	Branch to K if $(X_j) \neq 0$	(30 Bits)
032	Branch to K if (X_j) positive	(30 Bits)
033	Branch to K if (X_j) negative	(30 Bits)
034	Branch to K if (X_j) in range	(30 Bits)
035	Branch to K if (X_j) out of range	(30 Bits)
036	Branch to K if (X_j) definite	(30 Bits)
037	Branch to K if (X_j) 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 X_j . The decision will not be made until the X_j 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 X_j . 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 X_j . If the sign bit 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 words or for coefficients in floating point words.

3. The 034 and 035 operations examine the upper-order 12 bits of X_j . 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.

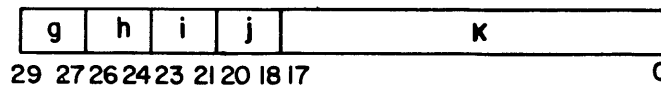
4. The 036 and 037 operations examine the upper-order 12 bits of X_j . Both positive and negative indefinite forms are detected:

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	Branch to K if $(B_i) = (B_j)$	(30 Bits)
05	Branch to K if $(B_i) \neq (B_j)$	(30 Bits)
06	Branch to K if $(B_i) \geq (B_j)$	(30 Bits)
07	Branch to K if $(B_i) < (B_j)$	(30 Bits)



These instructions test an 18-bit word from register B_i against an 18-bit word from register B_j 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 $B_j = B_0$. 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.

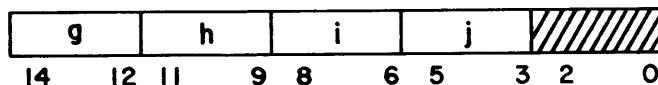
For special situations, refer to the 02 Jump instruction.

BOOLEAN UNIT

10

Transmit (Xj) to Xi

(15 Bits)

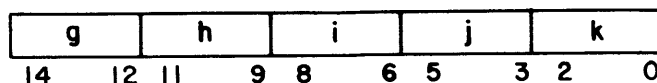


This instruction transfers a 60-bit word from operand register Xj to operand register Xi. It is intended for moving data from one X register to another X register as rapidly as possible. No logical function is performed on the data.

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:

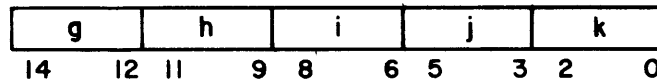
$$\begin{array}{r}
 (X_j) = 0101 \\
 (X_k) = 1100 \\
 \hline
 (X_i) = 0100
 \end{array}$$

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 Transmit instruction.

.12

Logical sum of (Xj) and (Xk) to Xi

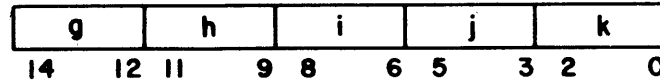
(15 Bits)



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:

$$\begin{array}{r}
 (X_j) = 0101 \\
 (X_k) = 1100 \\
 \hline
 (X_i) = 1101
 \end{array}$$

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 Transmit instruction.



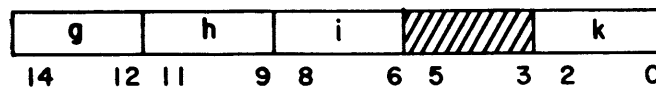
This instruction forms the logical difference (exclusive OR) of 60-bit words from operand registers Xj and Xk and places the difference in operand register Xi. Bits of register Xi are set to "1" if the corresponding bits in the Xj and Xk registers are unlike as in the following example:

(Xj) = 0101

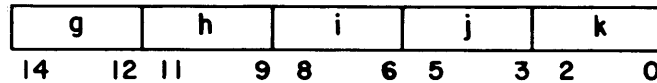
(Xk) = 1100

(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.



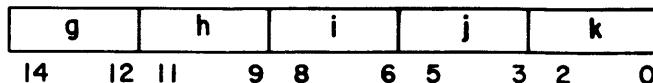
This instruction extracts the 60-bit word from operand register Xk, complements it, and transmits this complemented quantity to operand register Xi. It is intended for changing the sign of a fixed point or floating point quantity as quickly as possible.



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" where the corresponding bits of the Xj register and the complement of the Xk register are "1" as in the following example:

$$\begin{array}{r}
 (X_j) = 0101 \\
 \text{Complemented } (\overline{X_k}) = 0011 \\
 \hline
 (X_i) = 0001
 \end{array}$$

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.

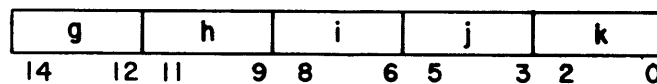


This instruction forms the logical sum (inclusive OR) of the 60-bit quantity from operand register Xj and the complement of the 60-bit word from operand register Xk, and places the result in operand register Xi. Thus, bits of Xi are set to "1" if the corresponding bit of the Xj register or complement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

$$\begin{array}{r}
 (X_j) = 0101 \\
 \text{Complemented } \underline{(X_k) = 0011} \\
 (X_j) = 0111
 \end{array}$$

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.

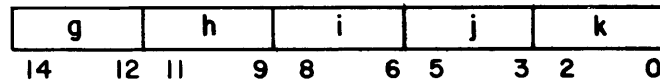
17 *Logical difference of (Xj) and complement of (Xk) to Xi* (15 Bits)



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:

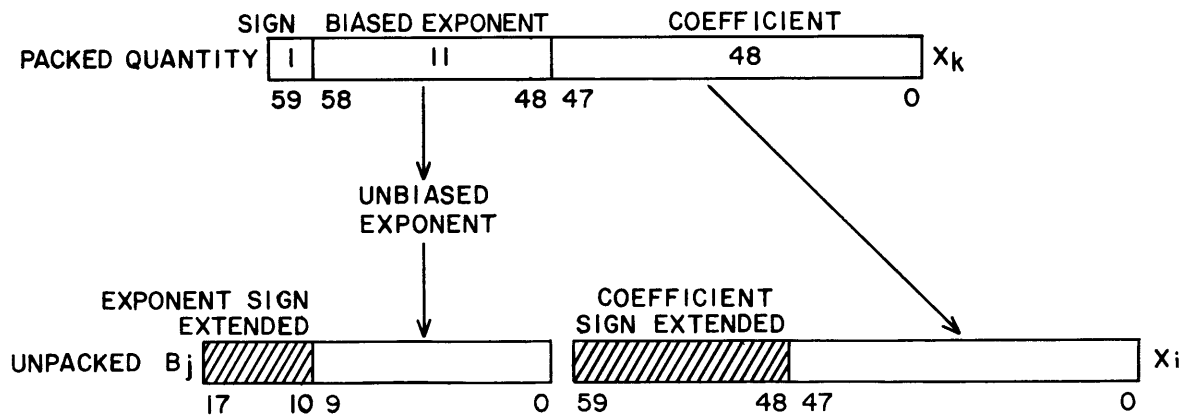
$$\begin{array}{r}
 (X_j) = 0101 \\
 \text{Complemented } \underline{(X_k) = 0011} \\
 (X_i) = 0110
 \end{array}$$

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.

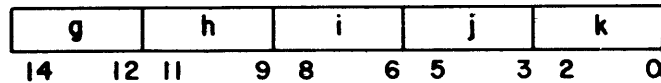


This instruction unpacks the floating point quantity from operand register X_k and sends the 48-bit coefficient to operand register X_i and the 11-bit exponent to index register B_j . The exponent bias is removed during Unpack so that the quantity in B_j is the true one's complement representation of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



This instruction packs a floating point number in operand register X_i . The coefficient of the number is obtained from operand register X_k and the exponent from index register B_j . Bias is added to the exponent during the Pack operation. The instruction does not normalize the coefficient.

This instruction obtains the exponent and coefficient from the proper low-order bits of the respective registers and packs them as shown for the packed quantity in the illustration for the Unpack (26) instruction. Thus, bits 48 to 58 of X_k and bits 11 to 17 of B_j are ignored. There is no test for overflow or underflow. No flags are set in the PSD register by this instruction.

Note that if (X_k) is positive, the packed exponent occupying positions 48 to 58 of X_i is obtained from bits 0 to 10 of B_j by complementing bit 10; if X_k is negative, bit 10 is not complemented but bits 0 to 9 are.

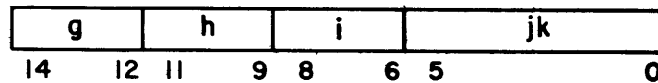
The j designator may be set to zero in this instruction to pack a fixed point integer into floating point format (exponent = 0) without using one of the active B registers.

SHIFT UNIT

20

Left shift (Xi) by jk

(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.

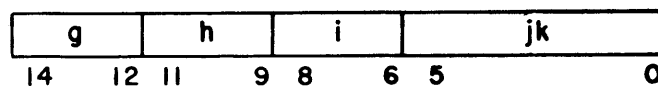
Example: Initial (Xi) = 2323 6600 0000 0000 0111
 jk = 12
 Final (Xi) = 7540 0000 0000 0022 2464

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

Right shift (Xi) by jk

(15 Bits)



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.

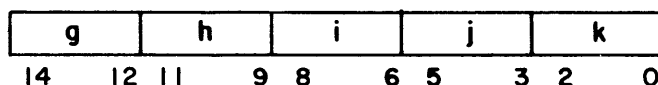
Example: Initial (Xi) = 2004 7655 0002 3400 0004
 jk = 30
 Final (Xi) = 0000 0000 2004 7655 0002

If the shift count is greater than 60-bit register length the result will contain 60 copies of the sign bit. If the operand were 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

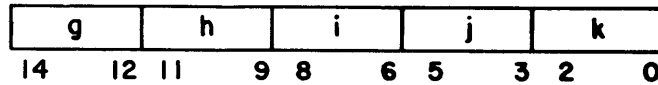
(15 Bits)



This instruction shifts the 60-bit quantity from operand register Xk the number of places specified by the quantity in index 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.

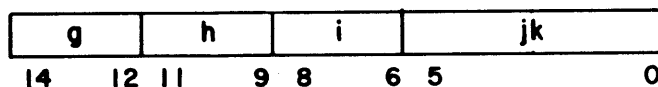
The contents of Bj might be the result of an unpack operation; in which case it is the unbiased exponent and (Xi) is the coefficient. This instruction is used for shifting a coefficient from a floating point number to the integer position after an unpack operation.



This instruction shifts the 60-bit quantity from operand register X_k the number of places specified by the quantity in index register B_j and places the result in operand register X_i .

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



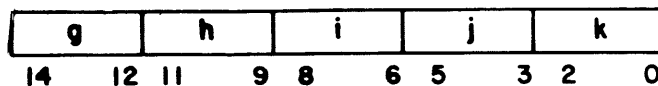
The instruction forms a mask in operand register *Xi*. The 6-bit quantity *jk* defines the number of "1's" in the mask as counted in octal from the highest order bit in *Xi*. The completed masking word consists of "1's" in the high order bit positions of the word and "0's" in the remainder of the word.

Example: *j* = 2
 k = 4
 (*Xi*) = 7777 7760 0000 0000 0000

The contents of operand register *Xi* = 0 when *jk* = 0. The contents of operand register *Xi* 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 may create an arbitrary field mask faster than by reading a pre-generated mask from storage.

NORMALIZE UNIT



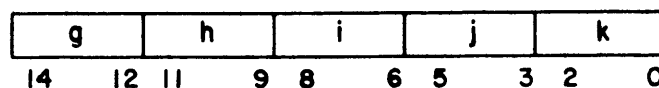
This instruction normalizes the floating point quantity from operand register Xk and places it in operand register Xi. Normalizing consists of left shifting the coefficient the minimum number of positions required to make bit 47 different from bit 59. This places the most significant bit of the coefficient 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 index 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 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.



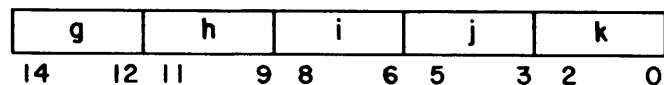
This instruction performs the same operation as instruction 24 except that the quantity from the operand register Xk is rounded before it is normalized. Rounding is accomplished by placing a "1" round bit immediately to the right of the least significant coefficient bit. The resulting coefficient is 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, infinite and indefinite results as for instruction 24.

FLOATING POINT ADD UNIT

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.

SPECIAL SITUATIONS

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 the exponents of both operands are zero and no overflow occurs, the instruction effects an ordinary integer addition of the coefficients.

If the exponents of the two operands differ by 48, or more, the result of the floating add operation will be a copy of the operand with the larger exponent.

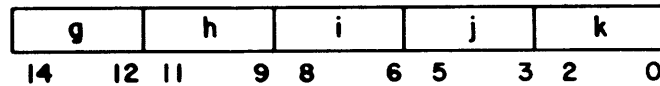
If either or both operands are indefinite, the result will be indefinite. The operand coefficients are ignored and the resulting word delivered to the Xi register is a positive indefinite exponent with a zero coefficient. The Indefinite Condition flag is set in the PSD register for this case.

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 both operands have overflow exponents and the operand coefficients have different signs, the resulting word delivered to the Xi register is a positive indefinite exponent with a zero coefficient. The Indefinite Condition flag is set in the PSD register for this case.

If either or both operands have an overflow exponent and the coefficient signs agree, the result is an overflow word. In this case, the operand coefficients are ignored, and the word delivered to the Xi register is a complete overflow exponent with a zero coefficient. The sign of the resulting word is the same as the sign of the operand with the overflow exponent. The Overflow Condition flag is set in the PSD register for this case.

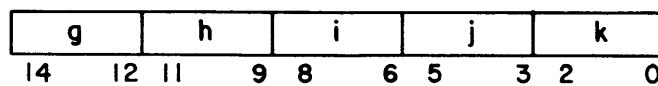
An operand with an underflow exponent is treated as a normal operand in this instruction. No special condition flags are set in the PSD register.



This instruction forms the difference of the floating point quantities from operand registers X_j and X_k and packs the result in operand register X_i . 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 X_i register will be the same as the exponent for the operands. The sign of the result will be positive.

When the exponents of both operands are zero and no overflow occurs, an ordinary integer subtraction of the coefficients is performed.



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 that for the upper sum. The result is not necessarily normalized. Except for the conditions noted, special situations are the same as for instruction 30, Floating Sum.

SPECIAL SITUATIONS

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.

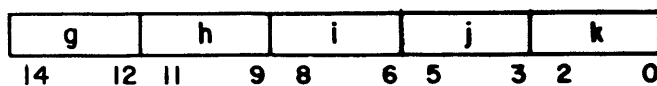
If the two operands are near the lower limit of the floating point range, the exponent for the lower half of the double precision sum may be exactly -1777 unbiased. This result is processed as a normal floating point number and no error condition flags are set in the PSD register. Subsequent use of this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

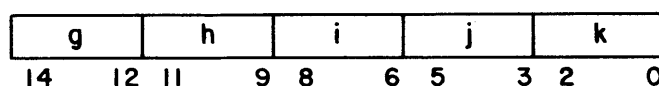
If the exponent for the lower half of the double precision sum is less than -1777 unbiased, the result delivered to the Xi register is a complete underflow word with a zero coefficient. The sign of the result will be the same as the sign of the (Xj) if Xj has the larger exponent. The sign of the result will be the complement of the coefficient of the sign of (Xk) if Xk has the larger exponent or if the exponents are identical.

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

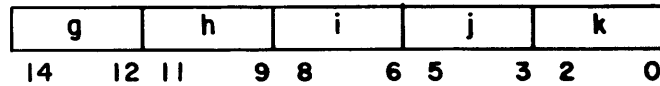
(15 Bits,



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.

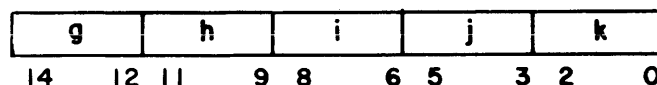


This instruction forms the round difference of the floating point quantities from operand registers Xj and Xk and packs the upper difference of the double precision result in operand register Xi. This instruction is intended for use in floating point calculations involving single precision accuracy. 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.

LONG ADD UNIT



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.

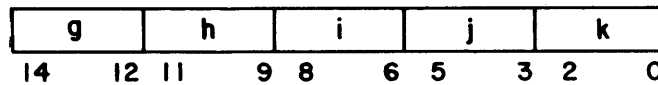
This instruction is intended for addition of integers too large for handling in the increment unit. It is also useful for merging and comparing data fields during data processing.

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) minus (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.

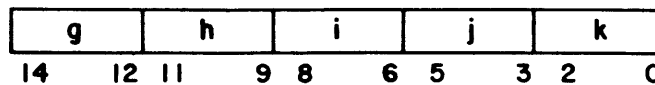
If (Xj) is a negative zero quantity, and (Xk) is a positive zero quantity, the result is a negative zero quantity. The other three combinations of positive and negative zero operands result in a positive zero quantity.

FLOATING POINT MULTIPLY UNIT

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 of the result.

The result is a normalized quantity only when both operands were normalized; the exponent in this case is the sum of the exponents plus 47 (or 48). The result is not normalized when either or both operands are not normalized.

SPECIAL SITUATIONS

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 the upper half of the 96-bit double precision product. 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 regardless of whether a subsequent left shift causes the result to be only partial overflow.

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 it. 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.

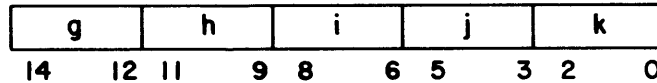
A complete underflow occurs for this instruction whenever the exponent computation results in less than -1776 octal. This situation is sensed as a special case, and a complete zero word with proper sign is delivered to the Xi register. The coefficient calculation is ignored in this case, and the Underflow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

If either operand is zero, the Underflow Condition flag will set in the PSD register.

41

Round floating product of (Xj) and (Xk) to Xi

(15 Bits)

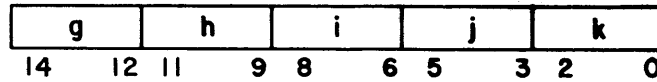


This instruction multiplies the floating point number from operand register Xk (multiplier), by the floating point number from operand register Xj. The upper product result is packed in operand register Xi. (No lower product is available.) The multiply operation is identical to that of instruction 40 except that a rounding bit is then extracted to form the coefficient for the result. An alternate output path is provided with a left shift of one-bit position 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 decremented 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 not normalized when either or both operands are not normalized. For treatment of special situations and operands, refer to instruction 40, Floating Product.

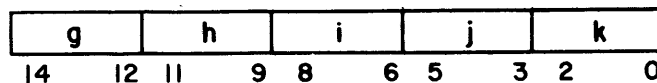


This instruction multiplies two floating point quantities obtained from operand registers Xj and Xk and packs the lower product in operand register Xi. The two 48-bit coefficients are multiplied together to form a 96-bit product. The lower-order 48 bits of this product (bits 47-00) are then packed together 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.

FLOATING POINT DIVIDE UNIT



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 adding one to 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.

SPECIAL SITUATIONS

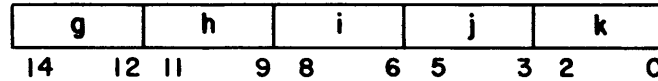
If the divisor is not normalized and the dividend coefficient is larger than the divisor by a factor of two or more, the quotient coefficient will be incorrect. The quotient is disregarded in this case, and the word delivered to the Xi register is positive indefinite with a zero coefficient if an Overflow or Underflow Condition does not exist. 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 may, 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.



This instruction divides the floating quantity from operand register X_j (dividend) by the floating point quantity from operand registers X_k (divisor) and packs the rounded quotient in operand register X_i . 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 X_j . This round bit has the effect of increasing the magnitude of the dividend by one-half the value of the least significant bit.

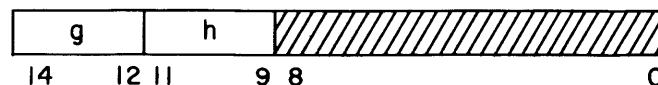
The result is a normalized quantity when both the dividend and the divisor are normalized.

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.

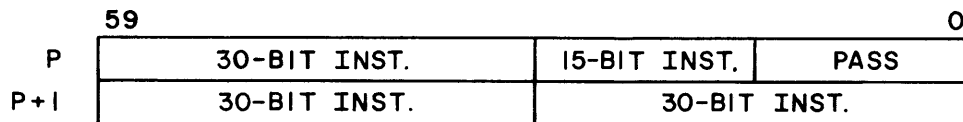
For treatment of special situations and operands, refer to instruction 44 Floating Divide.

PASS INSTRUCTION



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

EXAMPLE:



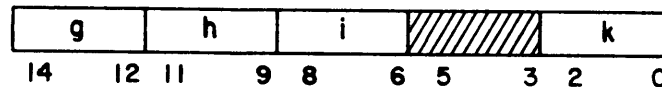
In this example, a Pass instruction is used to pad the remainder of the word at P. Since the next instruction is 30 bits, it cannot fit in P and must be placed in P + 1.

POPULATION COUNT UNIT

47

Population count of (Xk) to Xi

(15 Bits)



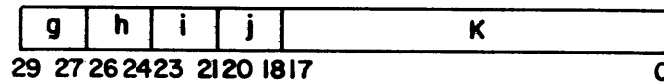
This instruction counts the number of "1 bits" in operand register Xk and stores the count in the lower order 6 bits of operand register Xi. Bits 6 through 59 are cleared to zero.

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

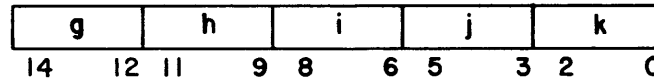
If Xk is a word of all zeros, a zero word is delivered to the Xi register.

INCREMENT UNIT

50	Set A_i to $(A_j) + K$	(30 Bits)
51	Set A_i to $(B_j) + K$	(30 Bits)
52	Set A_i to $(X_j) + K$	(30 Bits)



53	Set A_i to $(X_j) + (B_k)$	(15 Bits)
54	Set A_i to $(A_j) + (B_k)$	(15 Bits)
55	Set A_i to $(A_j) - (B_k)$	(15 Bits)
56	Set A_i to $(B_j) + (B_k)$	(15 Bits)
57	Set A_i to $(B_j) - (B_k)$	(15 Bits)



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

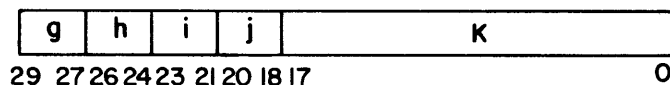
These instructions are intended for fetching operands from storage for computation and for delivering results back into storage. If the i designator is non-zero, a storage reference is made to SCM using the lower 16 bits of the resulting sum or difference as the relative storage address. The upper two bits are ignored. 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 X_i
- $i = 6, 7$; write into SCM from register X_i

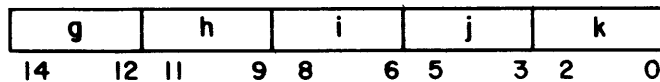
If this instruction makes a storage reference to SCM the address is compared with (FLS) to determine if the reference is within the assigned SCM field. If the address

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 reading to an X register, the out of range word addressed will be read to 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)



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)

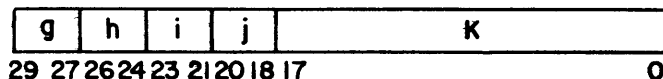


These instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in index register Bi. An overflow condition is ignored.

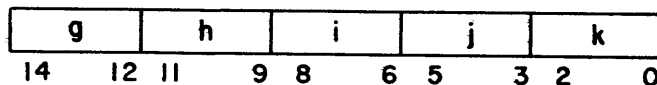
Operands are obtained from address (A), index (B), and operand (X) registers as well as the instruction itself (K = 18-bit signed constant). Operands obtained from an Xj operand register are the truncated lower 18 bits of the 60-bit word. The highest order bits are ignored. If the designator i is a zero, these instructions becomes Pass instructions.

* See Note 8 of Appendix D, 6000/7000 Result Differences.

70	Set X_i to $(A_i) + K$	(30 Bits)
71	Set X_i to $(B_j) + K$	(30 Bits)
72	Set X_i to $(X_j) + K$	(30 Bits)



73	Set X_i to $(X_j) + (B_k)$	(15 Bits)
74	Set X_i to $(A_j) + (B_k)$	(15 Bits)
75	Set X_i to $(A_j) - (B_k)$	(15 Bits)
76	Set X_i to $(B_j) + (B_k)$	(15 Bits)
77	Set X_i to $(B_j) - (B_k)$	(15 Bits)



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 X_i . The sign of the result is extended to the upper 42 bits of operand register X_i . An overflow condition is ignored.

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

INTRODUCTION

The Central Processor Unit contains two memories: Large Core Memory (LCM) and Small Core Memory (SCM). In the following descriptions the term Central Processor Unit (CPU) refers to 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. The upper limit is defined by the program field length added to the reference address. The field length is the number of 60-bit words comprising the program and it is established by the operating system prior to program execution. All references to memory from the program must lie within this field length.

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 a read or write address, it is automatically checked to see if it lies within the field length of the program. If it does, the program proceeds normally; if it does not, an unconditional exit is made and the program is terminated. These constraints are applied to both SCM and LCM addresses. Therefore, two reference address (RAS and RAL) and two field lengths (FLS and FLL) are required; one for each memory.

Two error conditions are noted in the Program Status Designator register for each memory; Direct Range Condition and Block Range Condition. A Direct Range Condition flag will be set if a single requested address is outside the limits of the program. A Block Range Condition flag will be set if a block transfer between SCM and LCM will cause a reference to an address outside the limits of the program.

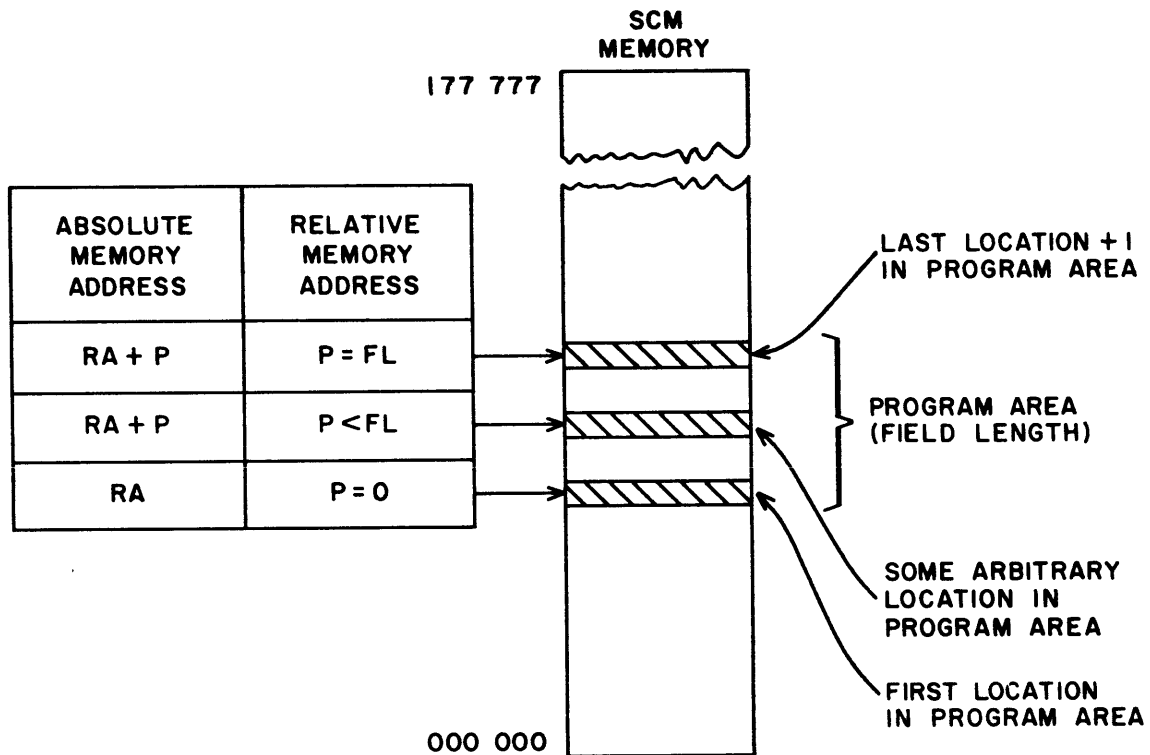


Figure 4-1. Memory Map

SMALL CORE MEMORY

Small Core Memory (SCM) is organized into 32,768 or 65,536 60-bit words, (plus 5 parity bits, odd parity) in 16 or 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. Each bank is divided into an odd and an even stack, each containing 1,024 words. 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 a 16-bit address. The address format is shown in Figure 4-2. Within the address format, the lowest four or five bits specify one of 16 or 32 banks. The next bit of this address specifies which stack in a bank (odd or even) is to be referenced. 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 efficient use of the bank phasing.

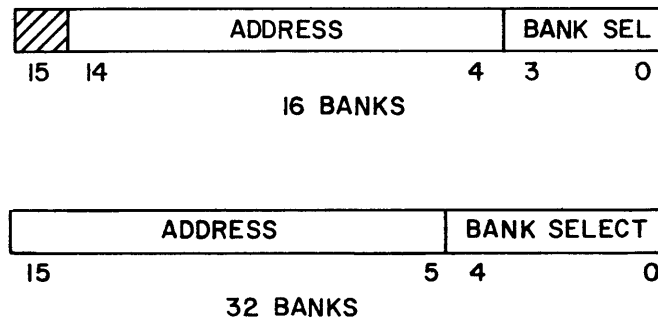


Figure 4-2. SCM Address Format

PARITY CONDITIONS

SCM parity is checked during each SCM read/write cycle. When a SCM parity error is detected, the SCM Parity Condition flag is set in the PSD register and the 16-bit address of the failing word and five SCM Section Parity Error bits are sent to the Maintenance Control Unit.

If a parity error occurs during a block copy operation, the block copy is executed to completion and the execution interval for the exchange package is terminated at the end of the current program instruction word.

DUTY CYCLE INTEGRATOR

Each odd and even stack in a bank of SCM includes a duty cycle integrator circuit that, when activated, prevents continued repetitive referencing of a stack at the maximum rate of once every 10 clock periods.

When active, the duty cycle integrator circuit slows references for the entire bank to a read/write cycle time of 880 nanoseconds or 32 clock periods. The duty cycle integrator will remain active for 1 millisecond after all references to a bank cease.

The duty cycle integrator is activated by repeated references to a stack oftener than once every 20 clock periods more than approximately 800 times. To prevent activating the duty cycle integrator, programmers should not use SCM locations for counters or tables requiring repeated references that exceed this rate.

It would be extremely unusual for normal user's programs to activate the duty cycle integrator.

It is not possible to activate the duty cycle integrator from the I/O channels. Only very unusual programs written for the CPU can cause the integrator to activate.

Examples of such programs are:

1. Counting in a tight loop by loading the count in an X register, adding to the count, storing the new count, and then branching back to reload the count. These instructions execute entirely within the instruction stack.

A suggested way of doing this type of program (that also decreases the program execution time) is to load the count in an X register and keep it there until just prior to exiting from the routine. Then and only then, store the final updated count into SCM. Certain benchmark programs have used this type of routine.

2. Idle loops, which are merely waiting for an I/O interrupt, should attempt to do all activities within the instruction stack using the X and B registers. This will prevent a situation in which one memory bank will receive an extraordinary number of consecutive references from the CPU.

SMALL CORE MEMORY ACCESS

Small Core Memory transmits data to and receives data from the PPU, CPU, LCM, and the MCU. Thus, SCM can be considered to be the center of the Computer System.

As the PPU's write or read the buffer data, a CPU I/O program empties the input buffers or fills the output buffers. An I/O interrupt to this CPU program occurs at

threshold and upon receipt of a Record Pulse from a PPU. A separate I/O exchange package for the I/O program exists for each input and output channel. The I/O exchange packages are permanently assigned in the lower order addresses of SCM. These areas are arranged as shown in Figure 4-3.

PPU ACCESS

PPU access is limited to certain buffer areas in the low order addresses of SCM. These areas are used for data transfers and for PPU-CPU communication. The PPU can normally read the output data buffer areas at any time but to avoid loss of data should do so only when directed to by the CPU. The PPU can write into the input buffer areas at any time, but to avoid loss of data should do so only when directed to by the CPU program.

1000	CHANNEL 16 INPUT PACKAGE	CHANNEL 16 OUTPUT PACKAGE	CHANNEL 17 INPUT PACKAGE	CHANNEL 17 OUTPUT PACKAGE
700	CHANNEL 14 INPUT PACKAGE	CHANNEL 14 OUTPUT PACKAGE	CHANNEL 15 INPUT PACKAGE	CHANNEL 15 OUTPUT PACKAGE
600	CHANNEL 12 INPUT PACKAGE	CHANNEL 12 OUTPUT PACKAGE	CHANNEL 13 INPUT PACKAGE	CHANNEL 13 OUTPUT PACKAGE
500	CHANNEL 10 INPUT PACKAGE	CHANNEL 10 OUTPUT PACKAGE	CHANNEL 11 INPUT PACKAGE	CHANNEL 11 OUTPUT PACKAGE
400	CHANNEL 6 INPUT PACKAGE	CHANNEL 6 OUTPUT PACKAGE	CHANNEL 7 INPUT PACKAGE	CHANNEL 7 OUTPUT PACKAGE
300	CHANNEL 4 INPUT PACKAGE	CHANNEL 4 OUTPUT PACKAGE	CHANNEL 5 INPUT PACKAGE	CHANNEL 5 OUTPUT PACKAGE
200	CHANNEL 2 INPUT PACKAGE	CHANNEL 2 OUTPUT PACKAGE	CHANNEL 3 INPUT PACKAGE	CHANNEL 3 OUTPUT PACKAGE
100	MCU PACKAGE	REAL TIME PACKAGE	CHANNEL 1 INPUT PACKAGE	CHANNEL 1 OUTPUT PACKAGE
0				
	0	20	40	60
	(OCTAL ADDRESSES)			

Figure 4-3. I/O Exchange Package Areas

An I/O Multiplexer (MUX) establishes I/O priorities for the PPU's, assembles incoming 12-bit PPU words into 60-bit SCM words, and disassembles the 60-bit outgoing SCM words into 12-bit PPU words. The I/O Multiplexer is described in more detail later in this section.

CPU ACCESS

Increment instructions (51-57) are used by a CPU program for referencing single SCM words. The A registers used by the increment instructions are divided into five read address registers (A1 - A5) and two write address registers (A6, A7). Placing a quantity into an A register causes a reference to that SCM location. If the A register is A6 or A7, the contents of the corresponding X register are stored in SCM. If the A register is A1 - A5, the corresponding X register is loaded with the contents of that memory location. If the referenced SCM address is outside the field length of the currently executing CPU program, the SCM Direct Range flag is set in the PSD register.

The CPU also references SCM when the Instruction Word Stack (IWS) requires another SCM word because the stack advanced or because of a branch out of the stack. The word will be read from SCM to the IWS even though it might be outside the field length of the currently executing program. However, it will produce a SCM Direct Range flag in the PSD register when the P register advances to or is set to the out-of-range address.

The CPU accesses SCM in a third way when it executes an exchange sequence. An exchange sequence involves reading the exchange package from SCM for the initiating program and storing the exchange package into SCM for the terminating program. Since the exchange package is not usually within the field length of the currently executing program, no field length checks are made.

LCM ACCESS

The Block Copy instructions (011 and 012) transfer large blocks of LCM data to or from SCM. The portions of SCM used for block copies must lie within the SCM field length of the CPU program initiating the transfer.

MCU ACCESS

The Maintenance Control Unit (MCU) has access to any part of SCM. Each SCM word is referenced separately by a 16-bit absolute address. The MCU accesses SCM through a special control unit in the I/O Multiplexer.

MEMORY REFERENCE

When a SCM 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 in a Storage Address Stack (SAS) until that bank is free. Instruction issue stops when a second address is sent to SCM and the previous address has not been accepted. At this time there may be a third address in process in the increment unit that cannot be stopped. This address will also be held in the SAS. Thus, requests for three addresses may be waiting for SCM in the SAS at the same time. Instruction issue does not start again until all unaccepted addresses have been accepted by SCM (up to three addresses).

It is possible to abort a valid SCM memory write when it is followed by an SCM write out of range. The following sequence of instructions could produce this situation: write SCM bank X, write SCM bank X, write SCM out of range. The first valid write will be accepted by the bank. The second write to bank X is held up in the SAS because it is going to the same bank. While the second write is waiting, the range check for the out-of-range write is being performed. This will cause the SCM Direct Range flag to be set in the PSD register before the second write to bank X can be initiated. Since the SCM Direct Range flag 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 received simultaneously from various parts of the computer are given a priority that determines which address shall be allowed access first. These priorities are:

1. Exchange Sequence Request
2. Increment Unit Request
3. Return Jump Exit Request
4. I/O Multiplexer Request
5. Read Next Instruction Request
6. LCM Block Copy Request

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

I/O MULTIPLEXER

The I/O Multiplexer (MUX) supervises the transfer of data to or from the SCM and the directly connected PPU's. The PPU's connected with the CPU communicate over 12-bit bi-directional channels. In the I/O Multiplexer, each channel has a channel control unit that translates I/O control signals going to or coming from the PPU. The channel control unit includes assembly and disassembly registers for converting the 12-bit channel data to 60-bit CPU words, and vice versa.

There may be a total of 16 channels in the Multiplexer. Fifteen channels, numbered 1 to 17_8 can connect to PPU's. A sixteenth channel, channel 0, connects to the MCU but operates differently. It is not included in the following description of the I/O Multiplexer.

Priority for SCM access and I/O interrupts is assigned in order by channel number with the lowest order channels having the highest priority; input has priority over output.

Each channel normally has a SCM buffer area for incoming data and a separate SCM buffer for outgoing data. Each buffer is divided into two fields, a lower field and an upper field. Data is entered (or removed) for 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 PPU fills or empties a buffer area to the point where a field boundary is crossed, the CPU is interrupted and an exchange sequence is performed to initiate a CPU I/O program to process the buffer data. The PPU continues to fill (or empty) the other buffer field while the CPU is processing buffer data in the first buffer field.

The I/O Multiplexer buffer areas are assigned positions in SCM. A typical arrangement for the buffer areas is shown in Figure 4-4. I/O Multiplexer space in SCM cannot exceed absolute address $10,000_8$. Note that Channel 2 input and output share a buffer area with Channel 3 input and output. This is also true of Channels 4 and 5, and Channels 6 and 7. These six channels are High Speed Channels as opposed to Normal Channels, which use the smaller buffers. The maximum speeds of Normal and High Speed Channels are listed on page 1-7.

The basic 7-channel configuration includes High Speed Channels 4, 5, 6 and 7 with the larger buffers and Normal Channels 10, 11, and 12 with the small buffers. Two I/O Multiplexer Channel Increments of four channels each would upgrade the system to include the full complement of 15 I/O MUX channels.

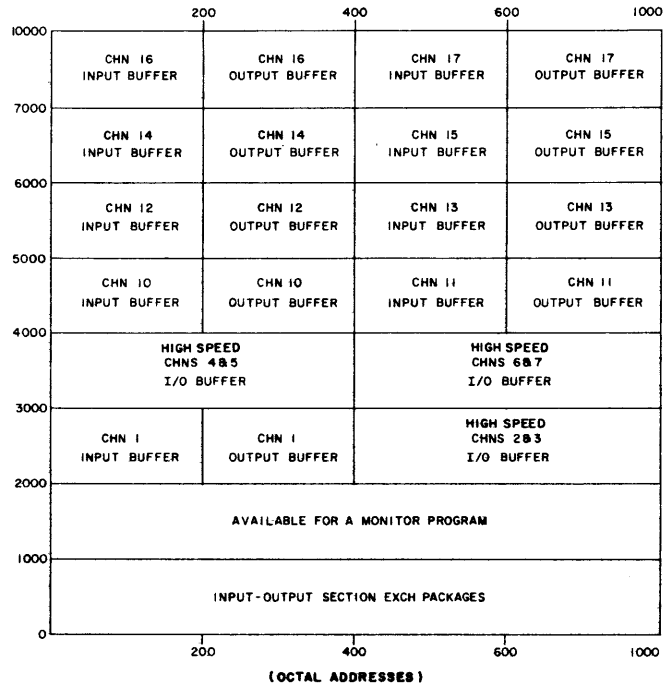


Figure 4-4. Typical Buffer Area Arrangements

NORMAL PPU TO SCM DATA TRANSFER

The following description lists the events in a normal PPU to SCM input record sequence. The sequence begins with a CPU Reset Input Channel Buffer instruction that resets the input channel buffer for receipt of a new record. This sets the input assembly counter and the input buffer address to zero. The CPU then notifies the PPU that it is ready to receive data. It does this by transmitting a message to the PPU over the associated output channel. The content and format of the message depend upon the communication scheme, which is determined by the software.

Upon receipt of this message, the PPU enters the first 12-bit word into its output register. This entry causes the transmission of a Word Pulse and 12 data bits to the channel control unit for this channel in the I/O Multiplexer. The I/O MUX samples the 12-bit word and assembles it in the upper 12 bits of the 60-bit assembly register.

Then it sends a Resume Pulse to the PPU and advances the assembly counter. The Resume Pulse clears the Output Word Flag at the PPU and the second 12-bit word enters the PPU output register. The sequence of Word Pulse, input assembly, and Resume Pulse is repeated for each 12-bit word transmitted over the data path. When five 12-bit words have been assembled into a 60-bit word, a Resume Pulse is sent to the PPU and a Word Request is made for SCM access. The I/O MUX will not sample the next 12-bit word, however, until the request for SCM access has been accepted by the SCM. This may be only a few clock periods, or many clock periods, depending upon the SCM bank conflicts. Once the Word Request has been accepted by SCM, the buffer address is advanced, the assembly counter is reset to zero, and the transmit and assembly procedure is repeated for the next 60-bit word.

When the PPU has transmitted enough words to half fill its assigned buffer area, the I/O MUX sends an I/O Interrupt Request to the CPU. The I/O Interrupt Request, when accepted by the CPU, causes a CPU exchange sequence to the CPU I/O program that processes the data in the first half of the input buffer. Meanwhile, the PPU continues to transmit 12-bit words, which the I/O MUX assembles into 60-bit words and stores in the upper half of the input buffer. When the upper half of the buffer area becomes full, the I/O MUX will send another I/O Interrupt Request to the CPU provided that the I/O program from the first interrupt has completed processing the lower half of the input buffer and has performed an exchange exit. Otherwise, if the I/O program is still processing the lower half, the I/O Interrupt Request will not be sent to the CPU and further input from the PPU will be locked out until the exchange exit is executed.

NOTE

Should an error condition occur which causes the I/O program to exit to EEA, the EEA program could in returning to the I/O program, inadvertently release the Interrupt Lockout condition prematurely by performing an Exchange Exit instruction. There are no hardware checks to prevent this situation from occurring.

When the I/O Interrupt Request has been sent, the PPU begins to enter data into the lower half of the buffer while the CPU I/O program processes data in the upper half. Thus, the buffer in SCM operates in a circular mode with I/O interrupts at the center and at the end of the buffer area.

An input record may contain any amount of data. The transmitting PPU terminates the record by sending a Record Pulse to the I/O MUX. Before sending the Record Pulse, the PPU should first check to see that the last 12-bit word was accepted by MUX. (If the PPU Output Word Flag is clear, MUX has accepted the last word.)

Upon receipt of the Record Pulse, the I/O MUX sends an I/O Interrupt Request to the CPU. If the PPU has not transmitted enough 12-bit words to form a complete 60-bit word, the remainder of the word is filled with zeros. Other than this, the CPU hardware handles this I/O Interrupt Request the same as an interrupt request caused by a threshold condition and initiates an exchange sequence to the CPU I/O program. The CPU I/O program determines whether the interrupt was caused by a buffer threshold or a Record Pulse. It does this by reading the SCM address (Read Input Channel Status instruction) to determine whether a threshold has been crossed since the last interrupt. The CPU I/O program processes the input data according to the situation sensed.

The PPU must not begin transmitting a new record of input data before the CPU has completed processing the data in the input buffer. There is no hardware provision to prevent the PPU from doing this. Therefore, the PPU program must not enter new data until directed to do so by the CPU program. Should the PPU proceed before the CPU has reset the input buffer, the incoming data for the new record may be partially lost. The incoming record will continue to be input with no indication of error except that the record will be shortened by the lost data.

NORMAL SCM TO PPU DATA TRANSFER

The following description lists the events in a normal SCM to PPU output record sequence. The CPU I/O program has already loaded the output buffer with some data. The output sequence begins with a CPU Reset Output Buffer instruction that sets the output buffer address to zero and sends an I/O Word Request to SCM to read the first word from the output buffer to the 60-bit disassembly register. When SCM delivers the 60-bit word to the output disassembly register, channel control for this output channel clears the disassembly counter and outputs a Record Pulse and a Word Pulse to the PPU to indicate transmission of a new record is starting.

The output channel control also places the upper 12 bits of the data in the disassembly register on the output channel for the PPU. When the PPU program senses the Record Pulse on its input channel, it reads the 12 bits of data, and sends a Resume Pulse to the I/O MUX. The output data remains static on the output channel until the PPU samples it.

When the Resume Pulse arrives from the PPU, the I/O MUX advances the disassembly register to the next 12 bits of the 60-bit word and sends another Word Pulse to the PPU. The output buffer address also advances to the next address at this time so that a CPU program monitoring this channel could determine that the PPU has accepted the first 12-bits of a new 60-bit word. The sequence of output disassembly, Word Pulse, PPU input, and Resume Pulse continues until the entire 60-bit word has been sent by the I/O MUX. At this time I/O MUX sends another I/O Word Request to SCM for the next word in the output buffer. When this word arrives in the disassembly register, the upper 12 bits and a Word Pulse are sent to the PPU and the process of delivering a new 60-bit word is repeated.

When the PPU has half emptied its assigned buffer area, the I/O MUX sends an I/O Interrupt Request to the CPU. The I/O Interrupt Request, when accepted by the CPU, causes a CPU exchange sequence to the CPU I/O program that refills the portion of the output buffer that has just been emptied. This operation is similar to that performed for a PPU to SCM transfer and output to the PPU continues from the upper half of the buffer while the lower half is being refilled.

When the upper half of the buffer area becomes empty, the I/O MUX will send another I/O Interrupt Request to the CPU provided that the I/O program from the first interrupt has completed processing the lower half of the input buffer and has performed an exchange exit. Otherwise, if the I/O program is still processing the lower half, the I/O Interrupt Request will not be sent to the CPU and further output to the PPU will be locked out until the exchange exit is executed.

NOTE

Should an error condition occur which causes the I/O program to exit to EEA, the EEA program could in returning to the I/O program, inadvertently release the Interrupt Lockout condition prematurely by performing an Exchange Exit instruction. There are no hardware checks to prevent this situation from occurring.

Using a software determined communication scheme, the CPU has notified the PPU of the length of the record. When the PPU has received the expected amount of data, it simply stops reading data from the PPU input channel. This stops further transmission on the part of the I/O MUX.

HIGH SPEED PPU TO SCM DATA TRANSFER

The following description lists the events in a high speed input record sequence. The sequence for a high speed channel is basically the same as for a normal channel except that the Word and Record pulses from the PPU are not synchronized by the I/O Multiplexer.

The sequence begins with a CPU Reset Input Channel instruction that resets the input channel buffer for receipt of a new record. This sets the input assembly counter to zero and the input buffer address to the starting address of the buffer for the selected channel.

Next, the PPU enters the first 12-bit word into its output register. This causes the transmission of a Word Pulse and 12 data bits to the channel control unit for this channel in the I/O Multiplexer. The I/O MUX samples the 12-bit word and assembles it in the upper 12 bits of the 60-bit assembly register.

A static High Speed Resume is sent to the PPU during this time. The High Speed Resume clears the Word flag in the PPU immediately after it is set. The second 12-bit word may now be entered in the PPU output register. This sequence continues as each 12-bit word is transmitted over the data path.

When five 12-bit words have been assembled into a 60-bit word, the I/O MUX sets the Input Word Request flag for SCM access. This blocks the High Speed Resume signal to the PPU, and clears the input assembly counter in preparation for the arrival of the next PPU word. It also blocks the processing of a new 12-bit word should one arrive before the request for SCM access has been accepted by SCM. This may be only a few clock periods, or many clock periods, depending upon SCM bank conflicts and channel priority. Once the Word Request has been accepted by SCM, the buffer address is advanced, the Input Word Request flag is cleared, and the High Speed Resume signal is again sent to the PPU. The transmit and assembly procedure is then repeated for the next 60-bit word.

The descriptions of the Input Interrupt Request and the Record Pulse from the PPU are the same as for the normal PPU to SCM data transfer.

HIGH SPEED SCM TO PPU DATA TRANSFER

The following description lists the events in a high speed output record sequence. The sequence for a high speed channel is basically the same as for a normal channel except that the Resume pulse is not resynchronized by the I/O Multiplexer and the output data path includes a series of three output data buffer registers. The High Speed channel output control also includes additional circuitry for controlling the flow of data from the disassembly register through these output data buffers to the PPU.

The output sequence begins with a CPU Reset Output Buffer instruction that sets the output buffer address to the starting address of the buffer. At this time, the I/O MUX also sends an Output Word Request to SCM to read the first word from the output buffer to the 60-bit disassembly register. When the 60-bit word has been delivered to the output disassembly register, the I/O MUX clears the disassembly counter and sends a Word Pulse and a Record Pulse to the high speed control circuitry. Concurrently, the upper byte of the disassembly register is transmitted to the Rank A of the high speed output buffer.

Upon receipt of the Record Pulse from the output channel control unit, the high speed buffer control transmits a Record Pulse to the PPU. This sets the Input Record flag at the PPU.

Upon receipt of the Word Pulse from the output channel control unit, the high speed buffer control enters the 12 bits of data from the output disassembly register into the Rank A of the high speed output buffer. Then it outputs the Word Pulse to the PPU, thereby setting the PPU Input Word flag at the PPU. The Word Pulse will not be sent to the PPU if an Interrupt Lockout condition exists in the output channel control unit.

In consecutive clock periods the data will move from the Rank A to Rank B and then to the Rank C of the high speed buffer registers. The data in the Rank C is transmitted to the PPU and remains static on the lines until the PPU transmits a Resume Pulse to the high speed control circuitry.

The High speed control circuitry does not wait for the Resume from the PPU, however, before sending a Resume Pulse to the output channel control unit. The channel control unit increments the output disassembly count and transmits the second 12-bit byte to the high speed output buffer over the output data path. At this time, channel control advances the content of the output address register to the next address in the SCM buffer, and sends a one clock period wide Word Pulse to the high speed control circuitry. Upon receipt of this second Word Pulse, the Rank A of the output buffer is cleared and entered with the second 12-bit byte and the Resume Pulse is again sent to the output channel control unit. In the following clock period, the data in the Rank A moves into the Rank B of the high speed output buffer.

The process is repeated for the third 12-bit byte. When the output channel control sends the third byte to the Rank A of the high speed output buffer, however, it does not send the Resume Pulse to the output channel control unit.

At this point all action stops until the PPU samples the 12 data lines and transmits a Resume Pulse to the high speed buffer control circuitry. When a Resume Pulse arrives from the PPU, the Rank C is cleared and entered with 12 bits from the Rank B, and a Resume Pulse is again sent to the output channel control unit.

The sequence continues until the entire 60-bit word has been sent to the high speed output buffer by the output channel control unit. At this time, the channel control unit sends another Output Word Request for the next word buffer.

At the time the Output Word Request flag sets, the last two bytes of data are in the Ranks B and C of the high speed output buffer. The PPU samples the data and transmits a Resume Pulse to the high speed buffer control. The Rank C is then cleared and entered with data from the Rank B. When this data has been delivered to the PPU, action halts until the requested word is delivered to the disassembly register from the SCM.

Some number of clock periods later, the word is delivered to the disassembly register and the process of delivering a new 60-bit word to the PPU begins.

The descriptions of the Output Interrupt Request and the Record Pulse from the PPU are the same as for the normal PPU to SCM data transfer.

LARGE CORE MEMORY

Large Core Memory is a 2-wire, word organized memory with a 1.76 μ sec cycle time. Parity checking is provided with one parity bit for each 15 data bits (odd parity). LCM is designed to provide a large amount of storage that emphasizes rapid block transfer of data.

LCM has a capacity of 256,000 or 512,000 60-bit words arranged in four or eight banks of 64,000 words each. Within a bank, eight consecutive 60-bit words are grouped into one LCM word with a parity bit for each 15 bits. The next consecutive

words are grouped into an LCM word that is stored in the next bank, etc. A memory reference to a location not previously read into a register reads all eight 60-bit words and parity bits from one LCM word simultaneously and stores them into a 512 bit bank operand register. As a 60-bit word enters or leaves the bank operand register for various destinations, parity is checked. A parity error will set a bit in the PSD register.

ADDRESS FORMAT

The location of each word in LCM is identified by a 19-bit address. The address format is shown in Figure 4-5. Within the address format, the lowest three bits specify one of eight 60-bit words within an LCM word. The next lower two or three bits specify one of four or eight LCM banks. The 13-bit address defines the location within the specified bank. For numerically consecutive addresses, consecutive banks are referenced at every eighth address for systems using all eight banks.

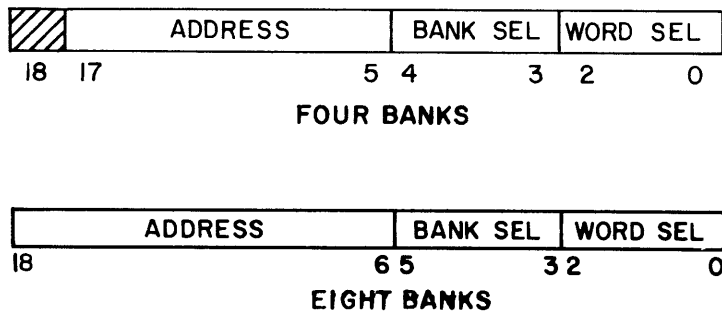


Figure 4-5. LCM Address Format

PARITY CONDITIONS

LCM parity is checked during every time a 60-bit word is read from LCM. When a LCM parity error is detected, the LCM Parity Condition flag is set in the PSD register and the 19-bit address of the failing word and four LCM Section Parity Error bits are sent to the Maintenance Control Unit.

If a parity error occurs during a block copy operation, the block copy completes execution and the execution interval for the exchange package terminates at the end of the current program instruction word.

LARGE CORE MEMORY ACCESS

LCM can be accessed in two ways: by block copies between LCM and SCM and by single word transfers between LCM and the X registers.

BLOCK COPIES

Block copy instructions move quantities of data between LCM and SCM as quickly as possible. All other activity in the CPU, except for I/O Word Requests, is stopped during a block copy operation. All instructions issued prior to this instruction are executed to completion and no further instructions issue until the block copy is nearly completed. As a result of these restrictions the data flow between LCM and 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 copy are resumed with a minimum of a one clock period delay. An additional delay will occur if the I/O reference causes a bank conflict in SCM.

DIRECT SINGLE-WORD TRANSFERS

A direct single-word transfer either reads one 60-bit word from LCM and enters this word into an X register or writes one 60-bit word directly into LCM from an X register.

The execution time for transferring a word from LCM to an X register depends on whether the requested word already resides in one of the bank operand registers. A Read LCM instruction for a word not currently residing in a bank operand register will require 17 clock periods for delivering a field of eight 60-bit words to the designated X register. A Read LCM instruction for a word already residing in a LCM bank operand register as a result of a previous instruction will require three clock periods to deliver the requested word to the designated X register. Thus, although the first 60-bit word will require 16 clock periods, the second through eighth words in the same LCM word require three clock periods each. This means that consecutive LCM operands are available, on an average, every five clock periods as opposed to SCM operands at eight clock periods.

The execution time for writing a word into LCM from an X register normally requires three clock periods. A delay will occur if the required LCM bank is busy completing a bank read/write cycle for a different block of eight words than that required for this instruction. In this case the word will be held in the LCM write register until the LCM bank is free.

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 seven peripheral processor units. One of these seven is designated the Maintenance Control Unit (MCU). It is identical to the other PPUs except it has specific, invariant channel connections. These channel connections may be made to other PPU's and to the CPU to dead start them or to monitor error conditions. (see MCU, Manual Control, page 7-1)

Operation of the PPU is controlled by a stored program that is sequentially executed in a one-address mode. All manipulative operations are performed in an 18-bit arithmetic (A) register. Arithmetic is binary in a ones complement mode. The program instructions make use of specially assigned locations in the lowest order 64_{10} words of PPU memory. Address arithmetic involving these words is performed in a separate address arithmetic unit that adds two addresses in a 12-bit ones complement mode.

The PPU may also directly control peripheral equipment devices with a minimum of intervening circuits. A modest amount of character conversion and formatting of data may be performed in the PPU before data is transmitted to the CPU. In addition, the PPU may be programmed to perform the synchronizing function required in interfacing an electro-mechanical device to the CPU. In this mode the PPU is generally dedicated to one or a small number of specific devices such as printers, card readers, tape units, disk files, etc.

The computation, memory, and I/O sections of the PPU are described in greater detail in the following paragraphs.

COMPUTATION SECTION

The computation section of a PPU performs the arithmetic operations associated with manipulating operands and with indirect addressing. These arithmetic operations involve seven registers: A, P, Q, Z, Sk, fd, and k. Only the A register is used directly by a programmer.

A REGISTER (18 BITS)

The arithmetic or A register is the principal operand register. In an arithmetic operation the A register always holds one of the operands and always receives the arithmetic result. 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 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.

The A register counts the length of the block for block input or output instructions. As each word is transmitted, the A register is entered with the new count.

The A register receives the input data word (12 bits) for the Input to A instruction and holds the output data word (12 bits) for the Output from A instruction.

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 (13 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 execution of shift instructions. The lowest order five bits contain the number of bit positions by which the content of the A register is to be shifted. The highest order bit determines whether the shift is left circular or right open-ended.

fd REGISTER (12 BITS)

The fd register holds the current instruction word for translation. The upper six bits are the f and the lower six bits the d designator from the instruction.

k REGISTER (3 BITS)

The k register is the instruction cycle counter and is used to count the number of memory references required during execution.

MEMORY

NOTE

Program loops in the PPU should be four executable words or longer to avoid reducing memory margins. A Pass instruction is an executable instruction.

Each processor has its own 12-bit, 4,096-word, magnetic core, random access memory with a read/write 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 memory modules, each of 1,024 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 bi-directional channels. Information may be transmitted from the PPU to a particular device at the same time that information is being sent to the PPU from that device. Each bi-directional channel consists of an input data path and an output data path plus the associated control lines for each path. The 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.

CONTROL SIGNALS

The three control lines associated with each path carry control signals for directing data flow between a PPU and another device. These signals are: the Word Pulse, Record Pulse, and Resume Pulse.

WORD PULSE

A Word Pulse is normally a one clock period pulse transmitted over a cable to notify the receiving device that the 12 data lines contain new information that is ready to be sampled. In special situations the Word Pulse 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 the transmitter and receiver in this case and the receiver must interpret the data to extract information on time changes.

RECORD PULSE

A Record Pulse 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 an SCM to PPU data transfer, however, (see page 4-11) the Record Pulse precedes the data and indicates to the PPU that transmission of a new record is about to start. In cases where the Word Pulse is continuously set the Record Pulse is not used.

RESUME PULSE

A Resume Pulse 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 a 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 CHANNEL CONTROL

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 channel selection 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 and in the *d* portion of the *fd* register.

An input channel is controlled by the setting and clearing of control flags within the PPU. The flags are directly associated with the control signals transmitted or received over the input channel.

INPUT WORD FLAG

This flag is set when a Word Pulse is received over the input cable by 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 forced to a cleared state during a Dead Start condition. A PPU senses the status of this flag by executing I/O jump instruction 60 or 61.

INPUT RECORD FLAG

This flag is set when a Record Pulse is received over the input cable by this PPU. The flag is cleared when the PPU has sampled the next following input data word and sends a Resume Pulse to the data transmitter at the other end of the cable. This flag is forced to a cleared state during a Dead Start condition. A PPU senses the status of this flag by executing I/O jump instruction 62 or 63.

INPUT RESUME FLAG

This 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 by this PPU over the input cable during the time in which this flag is set. The Resume Flag cannot be sensed directly by the PPU.

OUTPUT CHANNEL CONTROL

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 of the lowest order three bits in the d register. Data is transmitted statically over the output and remains on the cable until changed by the transmitting PPU.

An output channel is controlled by the setting and clearing of control flags within the PPU. The flags are directly associated with the control signals transmitted or received over the output channel.

OUTPUT WORD FLAG

The flag is set when the PPU transmits a one clock period wide Word Pulse over the associated output cable. The flag is cleared when a Resume Pulse is received by the PPU over this output cable. This flag is forced to a cleared position during a Dead Start condition. A one clock period wide Word Pulse is formed for transmission over an input cable. A PPU senses the status of this flag by executing I/O jump instruction 64 or 65.

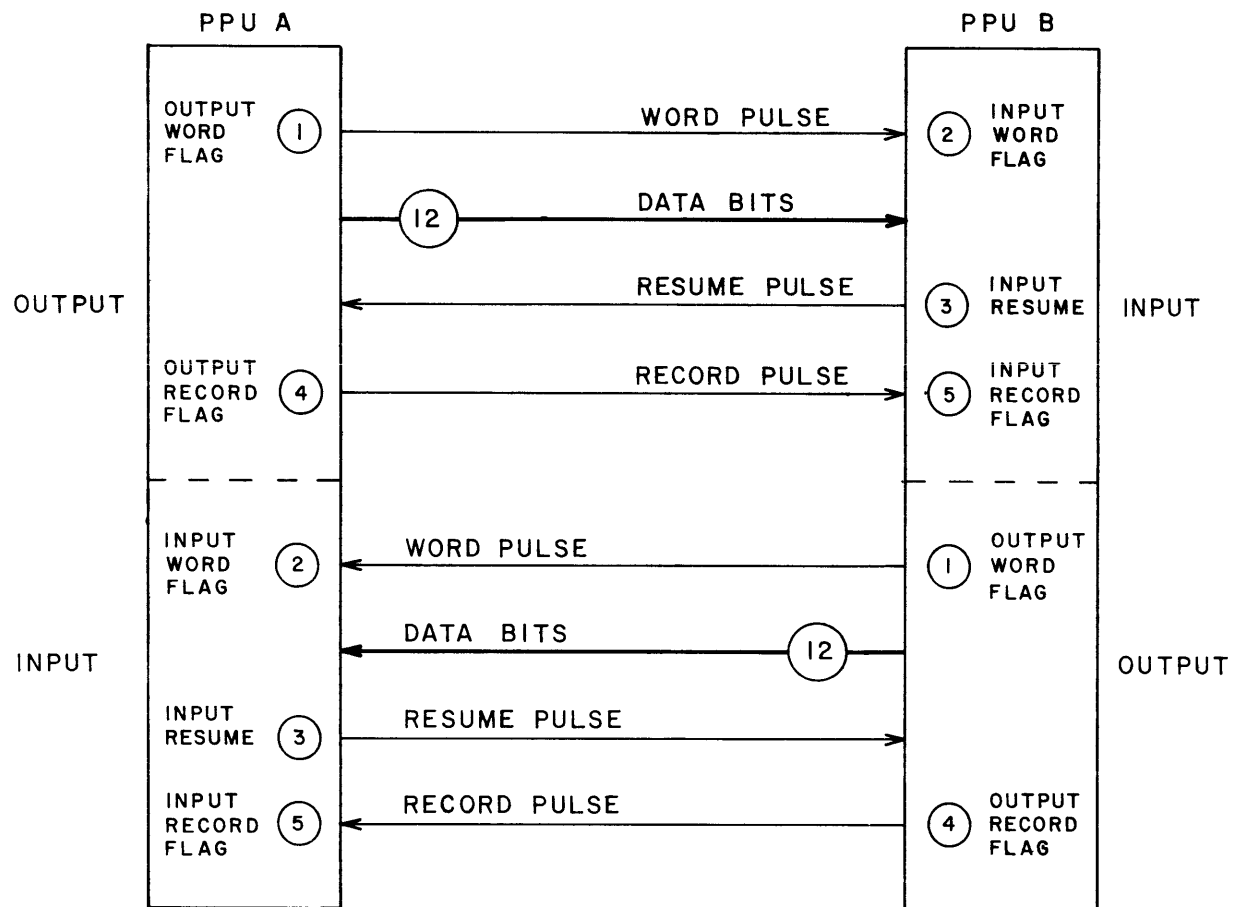
OUTPUT RECORD FLAG

This flag is set when the PPU transmits a one clock period wide Record Pulse over the associated output cable. The flag is cleared when a Resume Pulse is received over this output cable. This flag is also forced to a cleared position during a Dead Start condition. A PPU senses the status of this flag by executing I/O jump instruction 66 or 67.

PPU TO PPU DATA TRANSFERS

Figure 5-1 illustrates two PPUs with an interconnecting channel. Assume that a series of one-word transfers is required and that PPU A is the output and PPU B is the input PPU. The following sequences describes one method by which a one-word data transfer between the two PPUs can be accomplished:

1. PPU A executes an Output From A instruction (72). The instruction places 12 bits of data from the A register on the output channel, sets the Output Word Flag, and sends a Word Pulse to PPU B.
2. PPU B is periodically executing a Jump on Input Word Flag instruction (60). Upon receipt of the Word Pulse from PPU A, the Input Word Flag sets and PPU B jumps to an input program and executes an Input to A instruction (70). This instruction samples the 12 bits on the input channel, clears the Input Word Flag, and sends a Resume Pulse to PPU A.
3. At PPU A, the Resume Pulse clears the Output Data Flag and the Output Record Flag, if it is set. After executing the Output From A instruction (Step 1) PPU A repeatedly executes a Jump on No Output Word Flag instruction (65). If PPU B has not yet accepted the output word, the Output Word



- | | |
|---|--------------------------------|
| ① SET BY ANY OUTPUT DATA INSTRUCTION (72, 73) | CLEARED BY A RESUME PULSE |
| ② SET BY A WORD PULSE | CLEARED BY RESUME PULSE |
| ③ SET BY ANY INPUT DATA INSTRUCTION (70, 71) | CLEARED AFTER ONE CLOCK PERIOD |
| ④ SET BY OUTPUT RECORD FLAG INSTRUCTION (74) | CLEARED BY A RESUME PULSE |
| ⑤ SET BY OUTPUT RECORD PULSE | CLEARED BY RESUME PULSE |

Figure 5-1. Signals for One PPU Bi-Directional Channel

Flag will still be set. Otherwise, when PPU B has accepted the word, the Resume Pulse has cleared the Output Word Flag, and PPU A proceeds to the next instruction.

Note that in this example, PPU A notified PPU B that it was transmitting a word by sending a Word Pulse. PPU A could also have accomplished this by executing an Output Record Flag instruction that sends a Record Pulse to PPU B. In this case, PPU B would periodically monitor the status of the Record Flag instead of the Word Flag. Then, when the Record Flag sets upon receipt of the Record Pulse, PPU B would go to a data transfer sequence.

For block transfers, some of the flag monitoring functions are performed automatically by the block output and block input hardware. The following sequence illustrates one method by which a block transfer between two PPUs can be accomplished:

1. PPU A prepares for the block transfer by placing the length of the block to be transferred in the A register. It then executes a block output instruction (73). This instruction sets the Output Word Flag and sends 12 bits and a Word Pulse to PPU B.
2. Assuming that PPU B has been notified of the length of the block through a software determined communication scheme, PPU B prepares for an input by placing the length of the expected block in its A register. Then it repeatedly executes a Jump on Input Word Flag instruction (60).
3. The Word Pulse from PPU A sets the Input Word Flag at PPU B and PPU B proceeds to execute the Block Input instruction (71). This instruction samples the 12 bits, clears the Input Word Flag and Input Record Flag, if set, and sends a Resume Pulse to PPU A.
4. At PPU A, the Resume Pulse clears the Output Word Flag and the Output Record Flag, if set. The block output hardware automatically decrements the output count in the A register and sends the next 12 bits and another Word Pulse to PPU B.
5. Similarly, at PPU B, the block input hardware decrements the input count in the A register and samples the next 12 bits. The sequence repeats until the A register in PPU A becomes zero and PPU A sends a Record Pulse to PPU B.

Note that if the two counts in the A registers are unequal, the PPU with the larger count will hang up waiting for the proper response from the other PPU, which has already terminated its block transfer operation. Normally, however, if PPU A terminates first it would send a Record Pulse to PPU B, which will terminate input to PPU B. If PPU B terminates first, PPU A will hang up and remain hung up until PPU B inputs enough additional words to decrease the output count in PPU A to zero, or until PPU A is Dead Started.

PPU TO PERIPHERAL DEVICE

A direct-driven peripheral device requires two PPU channels. One channel is assigned to control and status and the other is used for data transfers. Depending upon the peripheral device, the associated control signals are terminated, set to logical one or zero, or assigned functions. Figure 5-2 shows one configuration that might be used for a card punch controller.

For detailed information on data transfers between a PPU and a peripheral device, refer to the documentation on the specific peripheral device.

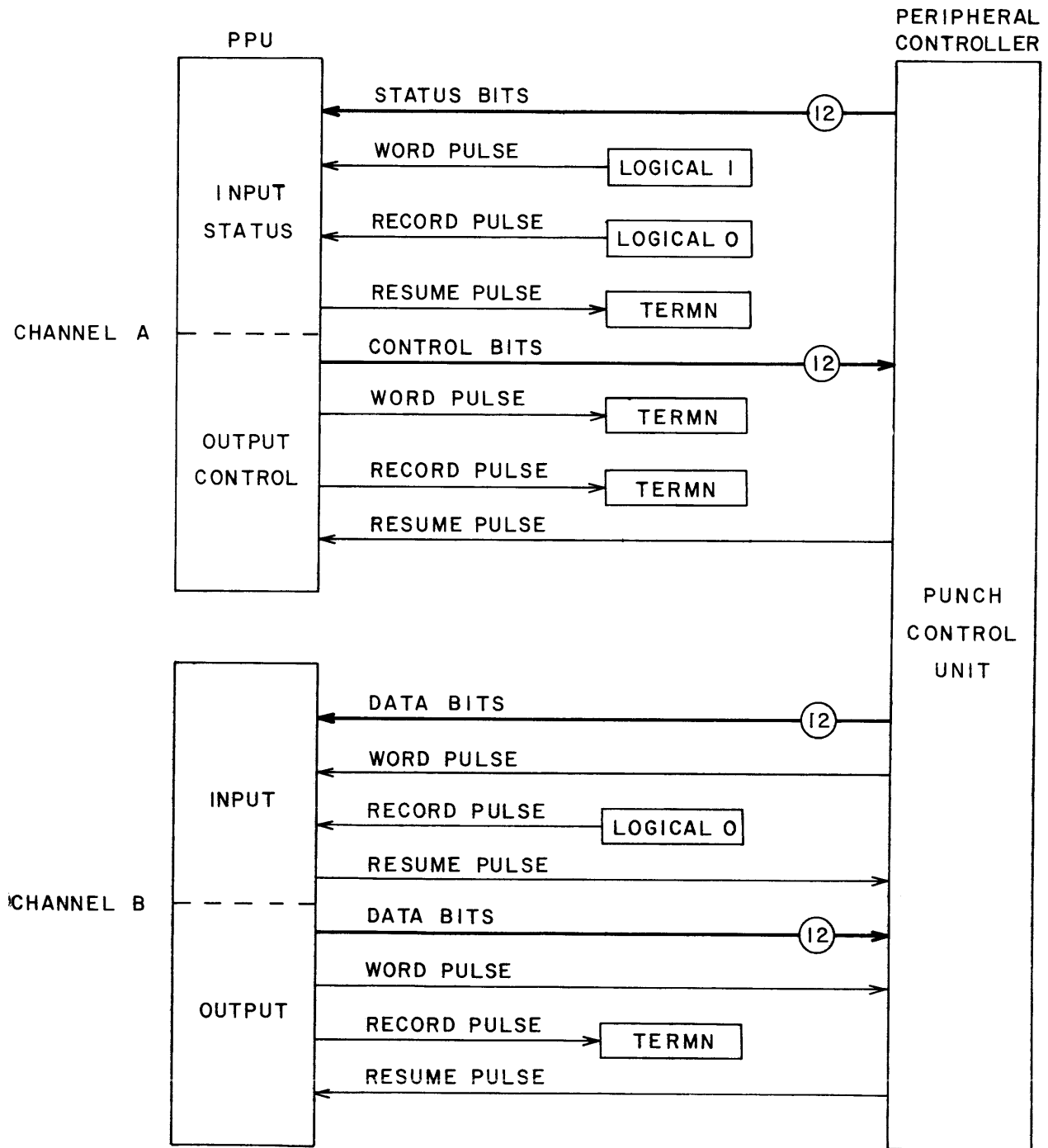


Figure 5-2. PPU/Controller Communications

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*.

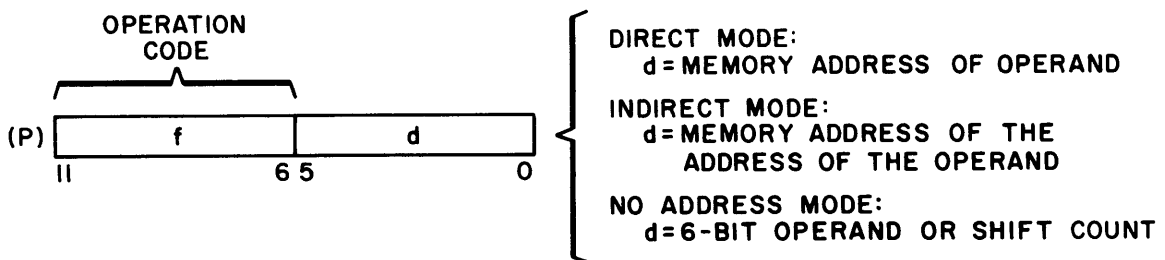


Figure 6-1. PPU 12-bit Instruction Format

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

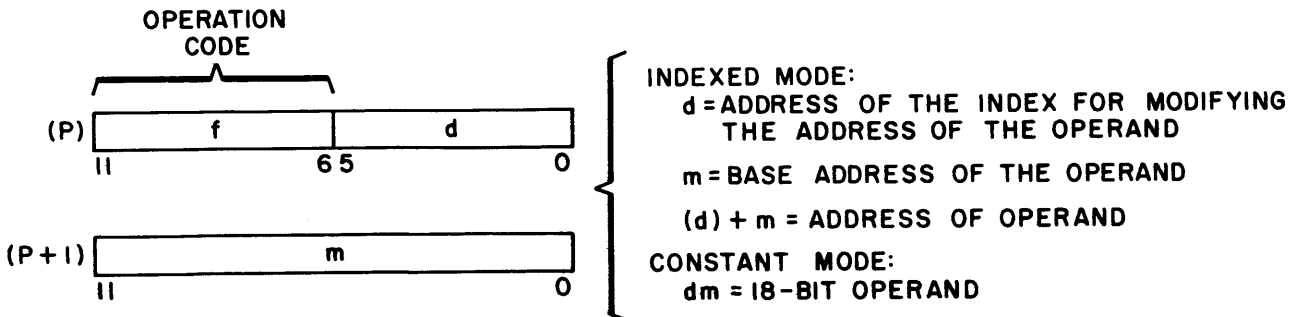


Figure 6-2. PPU 24-bit Instruction Format

ADDRESS MODES

Program indexing is accomplished and operands manipulated in several modes. The 12-bit and 24-bit instruction formats provide for 6-bit, 12-bit or 18-bit operands and 6-bit or 12-bit addresses. 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	Indexed	No Address	Constant
Load	30	40	50	14	20
Add	31	41	51	16	21
Subtract	32	42	52	17	
Logical Difference	33	43	53	11	23
Store	34	44	54		
Replace Add	35	45	55		
Replace Add One	36	46	56		
Replace Subtract One	37	47	57		
Long Jump			01		
Return Jump			02		
Unconditional Jump				03	
Zero Jump				04	
Non-Zero Jump				05	
Plus Jump				06	
Minus Jump				07	
Shift				10	
Logical Product				12	22
Selective Clear				13	
Load Complement				15	

NO ADDRESS MODE

In this mode d is taken directly as an operand. This mode eliminates the need for storing many constants in storage. The d quantity is considered as a 6-bit operand or shift count.

CONSTANT MODE

In this mode dm is taken directly as an operand. This mode also eliminates the need for storing many constants in storage. The dm quantity uses d as the upper six bits and m as the lower 12 bits of an 18-bit constant.

DIRECT ADDRESS MODE

In this mode, d is used as the address of the operand. The d quantity specifies one of the first 64 addresses in memory ($0000 - 0077_8$).

INDEXED DIRECT ADDRESS MODE

In this mode, $m + (d)$ is used as the address of the operand. The d quantity specifies the contents of one of the first 64 addresses in memory ($0000 - 0077_8$). The m quantity is a base address that is added to the contents of d to form a 12-bit address for referencing all possible memory locations but one ($0000 - 7776_8$). It is not possible to reference address 7777_8 . If d is nonzero, the content of address d is added to m to produce a 12-bit operand address (indexed addressing). If d is zero, m is taken as the operand address.

INDIRECT ADDRESS MODE

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.

Examples of Address Modes

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

Then:

<u>MODE</u>	<u>INSTRUCTION</u>	<u>A REGISTER</u>
No Address (6-bit Operand)	1425	000025
Constant (18-bit Operand)	2025 0100	250100
Direct Address	3025	000150
Indexed Direct Address	5025 0100	001234
Indirect Address	4025	007776

DESCRIPTION OF PERIPHERAL INSTRUCTIONS

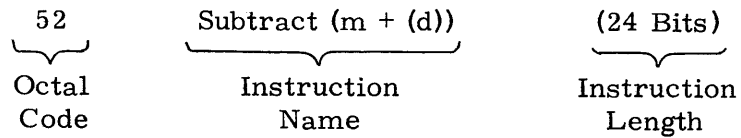
This section describes the Peripheral Processor instructions. Table 6-2 lists designators used throughout the section.

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 or (d) to form an 18-bit operand or 12-bit 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.

NOTE

Program loops in the PPU should be four executable instruction words or longer to avoid reducing memory margins. A Pass instruction is an executable instruction.

ERROR STOP

00	<i>Error Stop</i>	<i>(12 Bits)</i>
77	<i>Error Stop</i>	<i>(12 Bits)</i>

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

NO OPERATION

24	<i>Pass</i>	<i>(12 Bits)</i>
25	<i>Pass</i>	<i>(12 Bits)</i>
26	<i>Pass</i>	<i>(12 Bits)</i>
27	<i>Pass</i>	<i>(12 Bits)</i>
75	<i>Pass</i>	<i>(12 Bits)</i>
76	<i>Pass</i>	<i>(12 Bits)</i>



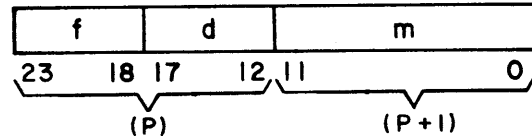
These instructions specify that no operation be performed. They provide a means of padding out a program.

BRANCH

01

Long jump to $m + (d)$

(24 Bits)

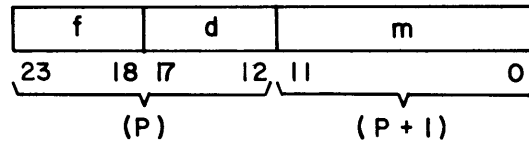


This instruction jumps to the sequence beginning at the address given by $m + (d)$. If $d = 0$, then m is not modified.

02

Return jump to $m + (d)$

(24 Bits)

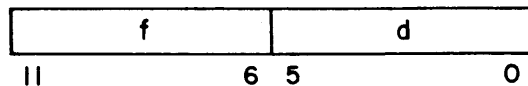


This instruction jumps to the sequence beginning at the address given by $m + (d)$. If $d = 0$ then m is not modified. The current program address (P) plus two is stored at the jump address. The new program commences at the jump address plus one. This program should end with a long jump to, or normal 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.

03

Unconditional jump d

(12 Bits)



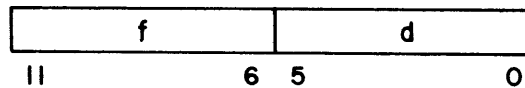
This instruction provides an unconditional jump to any instruction up to 31 steps forward or backward from the current program address. The value of d is added to the current program address. If d is positive ($01 - 37_8$), then $0001 (+1) - 0037_8 (+31)$ is added and the jump is forward. If d is negative ($40_8 - 76_8$) then $7740_8 (-31) - 7776_8 (-1)$ is added and the jump is backward. The value of d must not equal 00 or 77_8 . Either of these values cause the PPU to hang in a loop on the 03 instruction.

This will violate the restriction on program loops. 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)

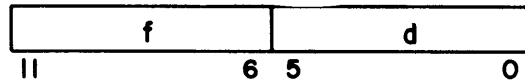


This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is zero, the jump is taken. If the content of A is nonzero, the next instruction is executed. Negative zero (777777_g) is treated as nonzero.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

05

Nonzero jump d

(12 Bits)

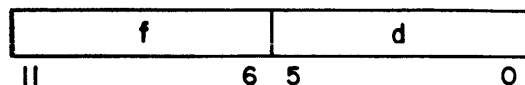


This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is nonzero, the jump is taken. If A is zero, the next instruction is executed. Negative zero (777777_g) is treated as nonzero. For interpretation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

06

Plus jump d

(12 Bits)



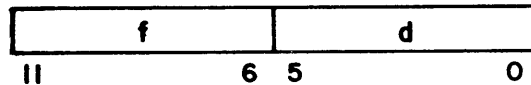
This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is positive, the jump is taken. If A is negative, the next instruction is executed.

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)



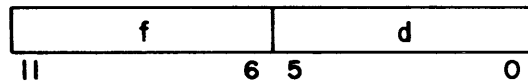
This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is negative, the jump is taken. If A is positive, the next instruction is executed. Positive zero is treated as a positive quantity; negative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

NO ADDRESS

10

Shift (A) by d

(12 Bits)

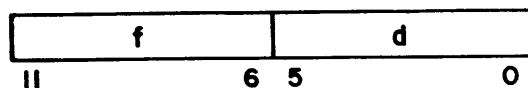


This instruction shifts the contents of A right or left d places. If d is positive (00-37₈) the shift is left circular; if d is negative (40-77₈) A is shifted right (end off with no sign extension) by the complemented d amount. Thus, d = 06 requires a left shift of six places. A right shift of six places results when d = 71₈.

11

Logical difference (A) and d

(12 Bits)

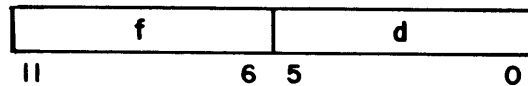


This instruction forms in A the bit-by-bit logical difference of d and the lower six bits of A. This is equivalent to complementing individual bits of A that correspond to bits of d that are one. The upper 12 bits of A are not altered.

12

Logical product (A) and d

(12 Bits)

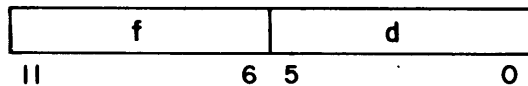


This instruction forms the bit-by-bit logical product of d and the lower six bits of the A register, and leaves this quantity in the lower 6 bits of A. The upper 12 bits of A are zero.

13

Selective clear (A)

(12 Bits)

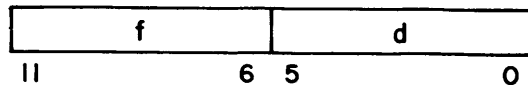


This instruction clears any of the lower six bits of the A register where there are corresponding bits of d that are one. The upper 12 bits of A are not altered.

14

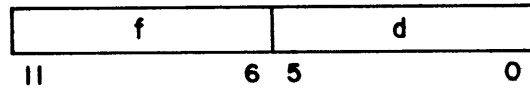
Load d

(12 Bits)



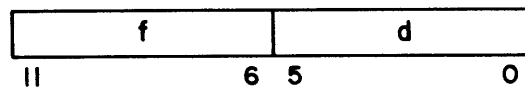
This instruction clears the A register and then loads it with d. The upper 12 bits of A are zero.

15

*Load Complement d**(12 Bits)*

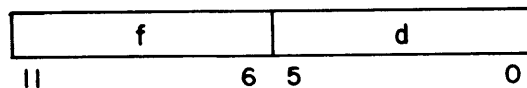
This instruction clears the A register and loads the complement of d. The upper 12 bits of A are set to one.

16

*Add (A) + d**(12 Bits)*

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

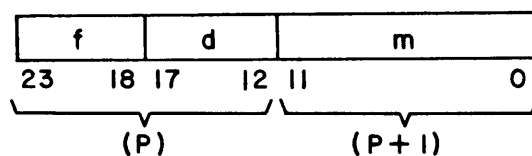
17

*Subtract (A) - d**(12 Bits)*

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

CONSTANT

20

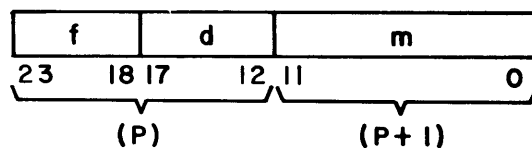
*Load dm**(24 Bits)*

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.

21

Add (A) + dm

(24 Bits)

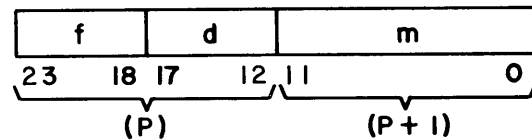


This instruction adds to the A register the 18-bit quantity consisting of d as the higher six bits and m as the lower 12 bits. The contents of the location following the present program address are read out to provide m.

22

Logical product (A) and dm

(24 Bits)

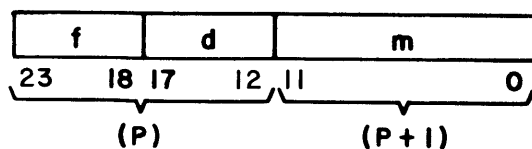


This instruction forms in the A register the bit-by-bit logical product of the contents of A and the 18-bit quantity dm. The upper six bits of this quantity consist of d and the lower 12 bits are the content of the location following the present program address.

23

Logical difference (A) and dm

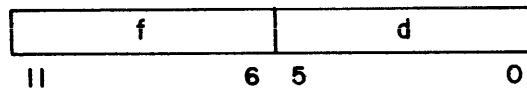
(24 Bits)



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.

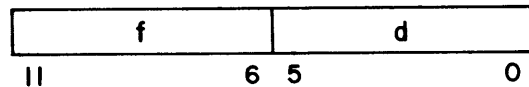
DIRECT ADDRESS

30 *Load (d)* *(12 Bits)*



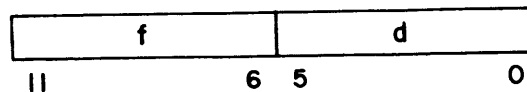
This instruction clears the A register and loads the contents of location d. The upper six bits of A are zero.

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



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

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

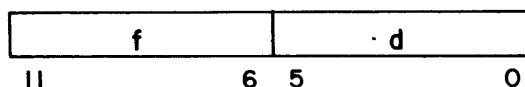


This instruction subtracts from the A register the contents of location d (treated as a 12-bit positive quantity).

33

Logical difference (A) and (d)

(12 Bits)

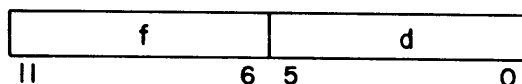


This instruction forms in A the bit-by-bit logical difference of the lower 12 bits of A and the contents of location d. This is equivalent to complementing individual bits of A which correspond to bits of (d) that are one. The upper six bits of A are not altered.

34

Store (A) at d

(12 Bits)

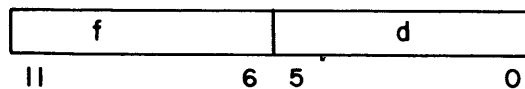


This instruction stores the lower 12 bits of A in location d.

35

Replace Add (A) + (d)

(12 Bits)

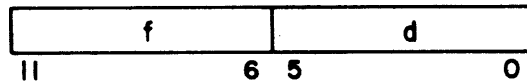


This instruction adds the quantity in location d to the contents of A and stores the lower 12 bits of the result at location d. The resultant sum is left in A at the end of the operation and the original contents of A are destroyed.

36

Replace Add one (d)

(12 Bits)

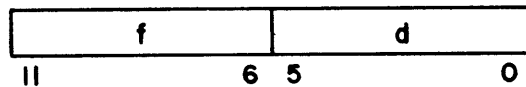


The quantity in location d is replaced by its original value plus one. The resultant sum is left in A at the end of the operation, and the original contents of A are destroyed.

37

Replace subtract one (d)

(12 Bits)



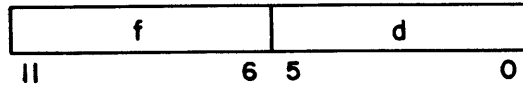
The quantity in location d is replaced by its original value minus one. The resultant difference is left in A at the end of the operation, and the original contents of A are destroyed.

INDIRECT ADDRESS

40

Load ((d))

(12 Bits)

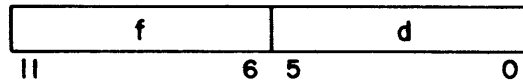


This instruction clears the A register and loads a 12-bit quantity that is obtained by indirect addressing. The upper six bits of A are zero. Location d is read out of memory, and the word obtained is used as the operand address.

41

Add (A) + ((d))

(12 Bits)

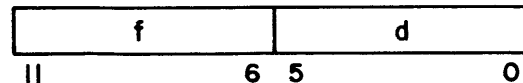


This instruction adds to the content of A a 12-bit operand (treated as a positive quantity) obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address.

42

Subtract (A) - ((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.

43

Logical difference (A) and ((d))

(12 Bits)

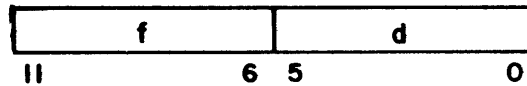


This instruction forms in A the bit-by-bit logical difference of the lower 12 bits of A and the 12-bit operand obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address. The upper six bits of A are not altered.

44

Store (A) at (d)

(12 Bits)

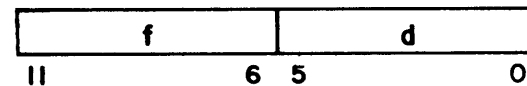


This instruction stores the lower 12 bits of A in the location specified by the contents of location d.

45

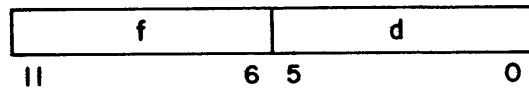
Replace add (A) + ((d))

(12 Bits)



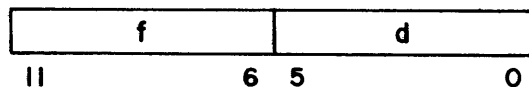
The operand which is obtained from the location specified by the contents of location d, is added to the contents of A, and the lower 12 bits of the sum replace the original operand. The resultant sum is also left in A at the end of the operation.

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.

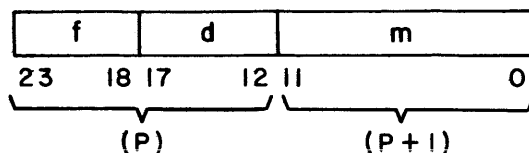
47

*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.

INDEXED DIRECT ADDRESS

50

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

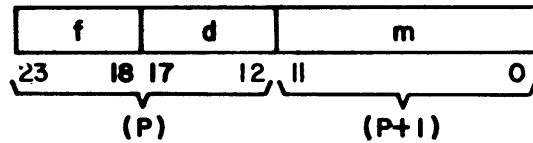
This instruction clears the A register and loads a 12-bit quantity. The upper six bits of A are zero. The 12-bit operand is obtained by indexed direct addressing. The quantity "m", read out of memory location P + 1 serves as the base operand address to which (d) is added. If d = 0, the operand address 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.

51

Add $(A) + (m + (d))$

(24 Bits)

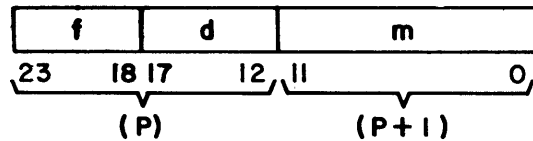


This instruction adds to the content of A a 12-bit operand (treated as a positive quantity) obtained by indexed direct addressing (see instruction 50).

52

Subtract $(A) - (m + (d))$

(24 Bits)

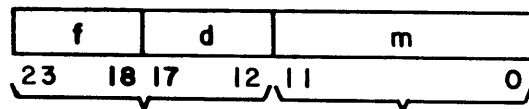


This instruction subtracts from the A register a 12-bit operand (treated as a positive quantity) obtained by indexed direct addressing (see instruction 50).

53

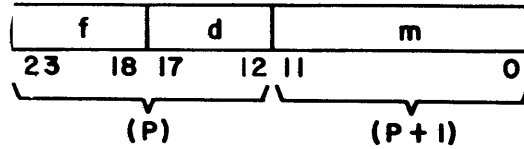
Logical difference (A) and $(m + (d))$

(24 Bits)



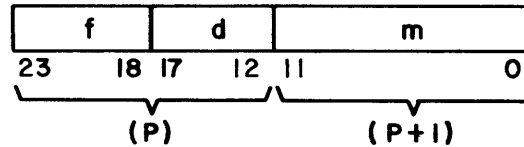
This instruction forms in A the bit-by-bit logical difference of the lower 12 bits of A and a 12-bit operand obtained by indexed direct addressing. The upper six bits of A are not altered.

54

Store (A) at $m + (d)$ *(24 Bits)*

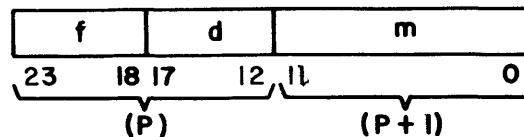
This instruction stores the lower 12 bits of A in the location determined by indexed addressing (see instruction 50).

55

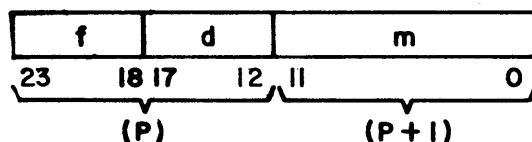
*Replace add (A) + (m + (d))**(24 Bits)*

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.

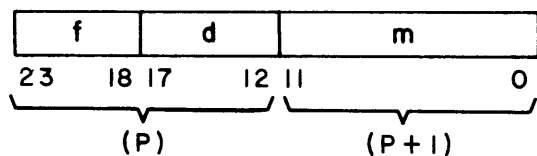


The operand, which is obtained from the location determined by indexed direct addressing, is replaced by its original value minus one (see instruction 50, for explanation of addressing). The resultant difference is also left in A at the end of the operation, and the original contents of A are destroyed.

INPUT/OUTPUT

60 *Jump to m; input word flag on channel d* (24 Bits)

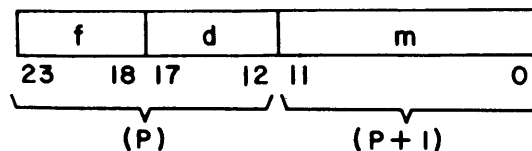
64 *Jump to m; output word flag on channel d* (24 Bits)



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.

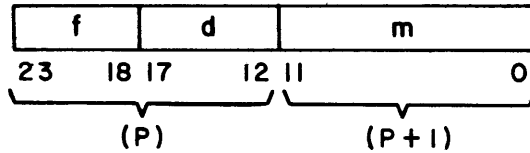
61 *Jump to m; input word flag on channel d* (24 Bits)

65 *Jump to m; no output word flag on channel d* (24 Bits)



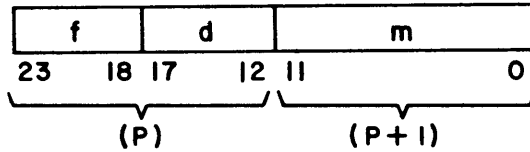
These instructions are identical to the 60 and 64 instructions except that the jump occurs if the flag is clear.

62 *Jump to m; input record flag on channel d* (24 Bits)
 66 *Jump to m; output record flag on channel d* (24 Bits)



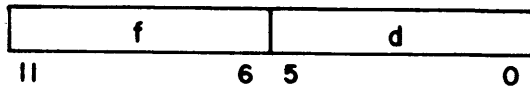
These instructions are identical to the 60 and 64 instructions except that the jump is conditioned by the status of the Record flag.

63 *Jump to m; no input record flag on channel d* (24 Bits)
 67 *Jump to m; no output record flag on channel d* (24 Bits)



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)



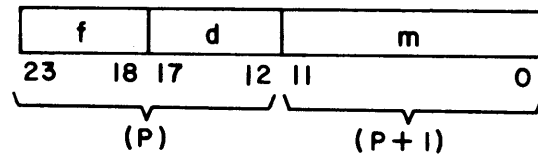
This instruction reads one word from input channel d and enters the word in the A register. This instruction will not be executed until the Input Channel d Word flag is set. If the flag is not set at the time the instruction is read from storage the PPU program will stop with the instruction in the f 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.

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, m, is obtained from address P + 1 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 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.

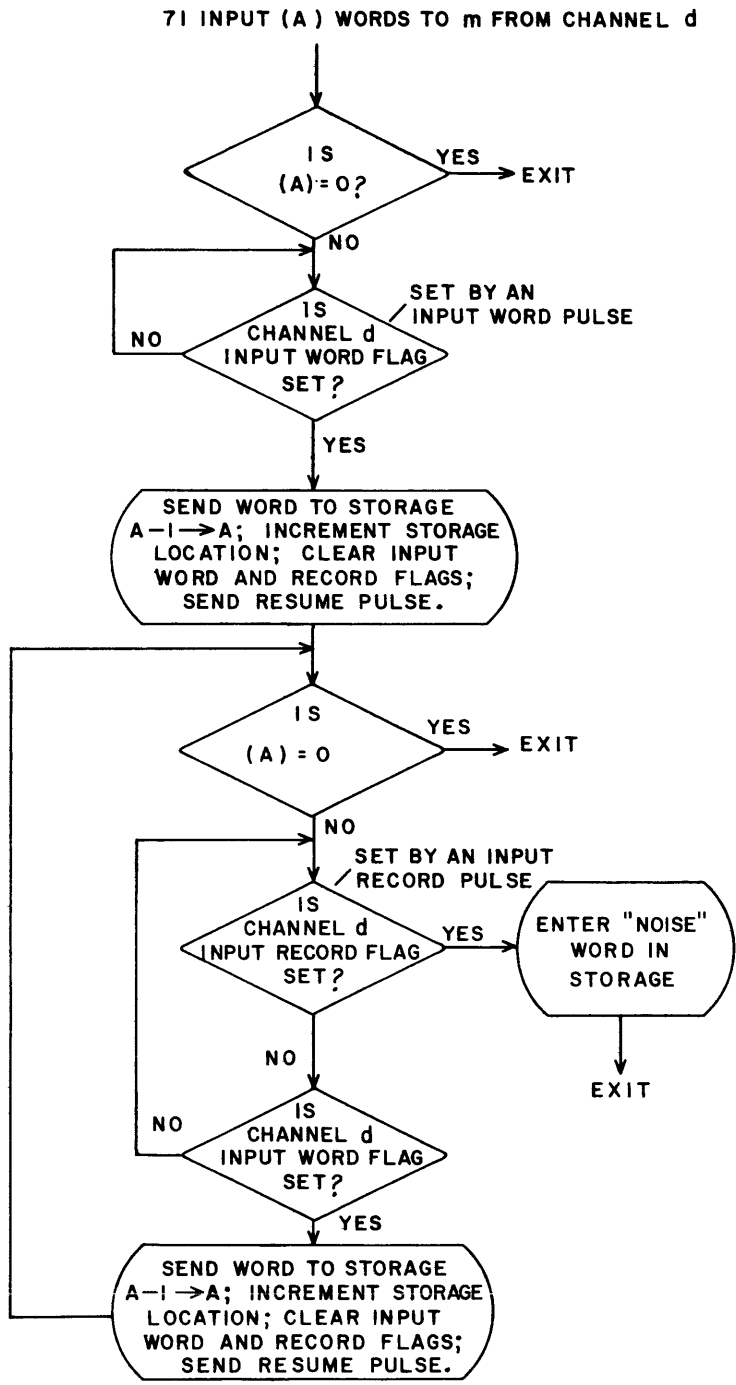


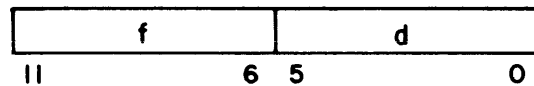
Figure 6-3. 71 Flow Chart

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 7776₈ to 0000₈, which may destroy existing data or a program.

72

Output from A on channel d

(12 Bits)

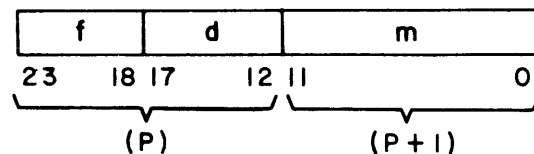


This instruction transmits one word over output channel d from the lower 12 bits of A. The A register content is not altered. This instruction will not be executed while the Output Channel d Word flag is set. If the flag is set from a previous output instruction the PPU program will stop with this instruction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

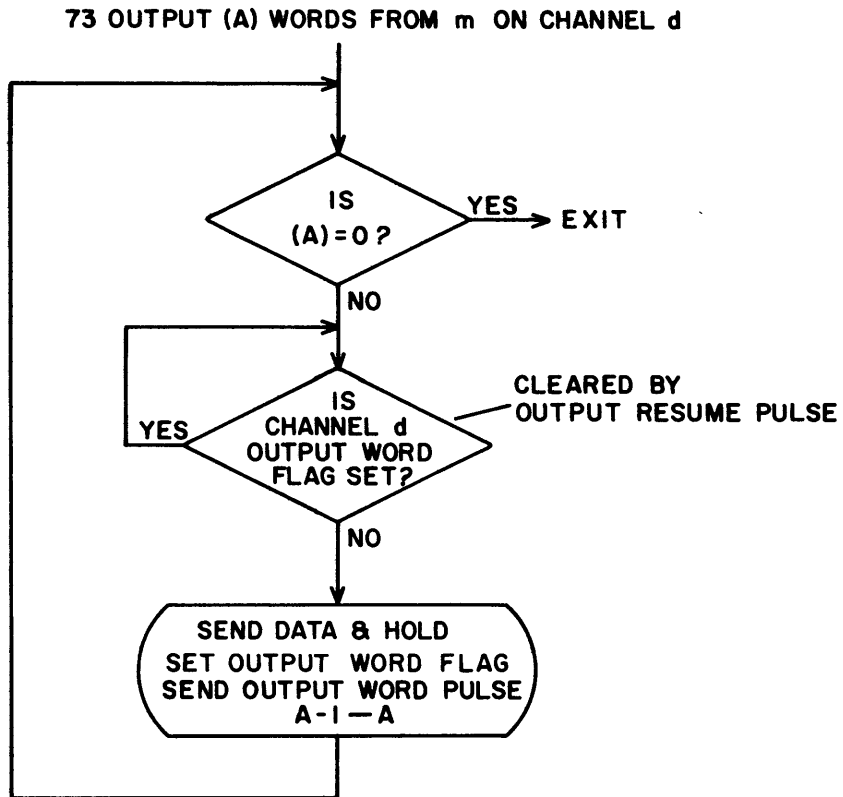
73

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.



NOTE THAT THE OUTPUT CHANNEL d RECORD FLAG IS IGNORED BY THIS INSTRUCTION.

Figure 6-4. 73 Flow Chart

The starting address, m , is obtained from the location $P + 1$ 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.

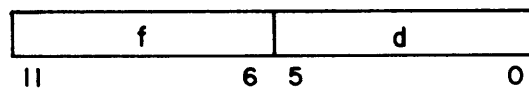
The Output Channel d Word flag must be cleared before the first word of the block can be transmitted over the channel. If this flag is set when the instruction is initiated the PPU program will stop with the instruction in the f and d registers and wait until the flag is cleared by a Resume pulse over 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

Send 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 flag 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.

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 of generating dead dumps.

MAINTENANCE CONTROL UNIT

The MCU is a stored program computer for monitoring mainframe hardware performance. Although internally it is identical to other PPU's in the system, the external connections to I/O channels differentiate it from other PPU's. Because of these special data channel connections, the MCU is capable of reading or writing any word in SCM, recording SCM and LCM parity errors, dead starting the CPU and all first level PPU's, dead dumping all first level PPU's, and loading initial programs from a card reader. Figure 7-1 illustrates the MCU channel configuration.

The MCU, like the other PPU's, possesses eight bi-directional data channels. It gains its additional capability through a device called the scanner, which connects the MCU with up to 13 PPU's.

MCU SCANNER

The MCU scanner selects incoming data for the MCU from one of 17₈ scanner input channels and distributes MCU output to one of 16₈ scanner output channels. Scanner channels 1 through 15₈ connect to channel 0 of PPU's 1 through 15₈. Scanner input channel 16₈ carries parity error address bits and scanner input channel 17₈ carries LCM parity error address bits to the MCU. Scanner output channel 16₈ connects the MCU to a Reference Voltage Scanner for testing marginal operation of mainframe modules. The scanner also connects directly to the PPU's for transmission of PPU Dead Start, Dead Dump, and Clear Parity Error signals and for receipt of PPU program error and PPU stack parity error bits.

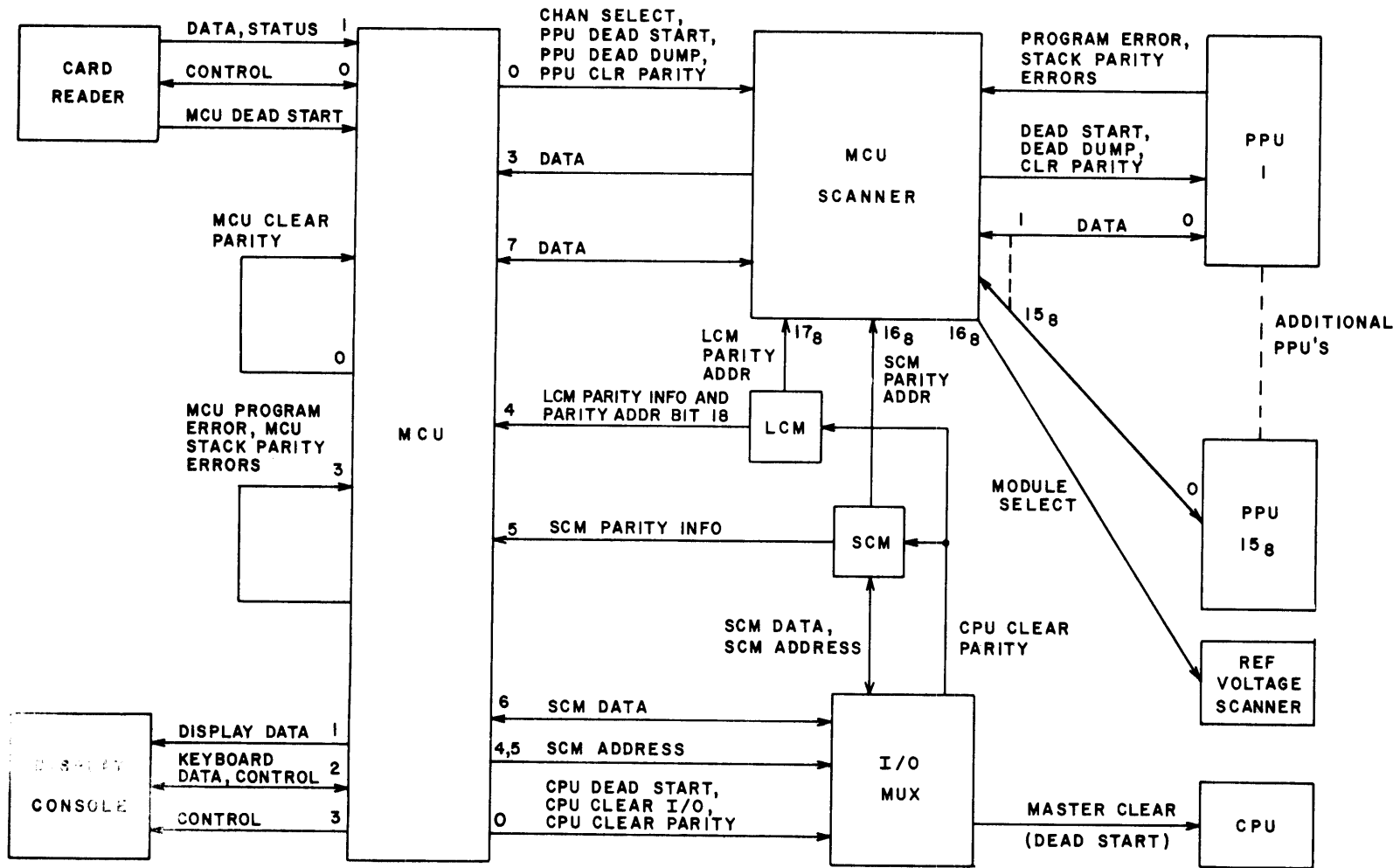


Figure 7-1. MCU Configuration

In operation, the MCU selects one of the scanner channels and communicates through that channel as if it were its own. The MCU program selects the desired channel by outputting the channel select code (1 to 17_g) as bits 8-11 on MCU channel 0. Then, when the MCU outputs data on channel 7, the data goes to the destination PPU or to the Reference Voltage Scanner via the selected scanner channel. Input to the MCU comes from the selected scanner channel and is received as data through MCU channels 3 and 7.

MCU DEAD START

The MCU is dead started by the card reader. A switch on the card reader generates the Dead Start signal that sets the MCU registers to the following values:

(A) = 007777

(P) = 0000

(X) = 0000

(f) = 71

(d) = 00

(k) = 1

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. After reading the card deck, the MCU initiates execution of the program that was input at address 0001.

PPU DEAD START

To Dead Start a PPU, the MCU first selects the Scanner channel connected to the PPU and then outputs the Dead Start Signal as bit 2 on MCU output channel 0. At the selected PPU, this signal sets the PPU registers to the values described for the MCU Dead Start. Also, 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 the Dead Start signal allows a program to be loaded from the MCU into the PPU over input channel 0. PPU loading, if not terminated at some point by a Record flag, terminates when the A

register has decremented to zero (7777 octal words). The PPU then begins execution of the program at address 0001.

CPU DEAD START

Under control of the MCU program, the MCU outputs the CPU Dead Start and Clear I/O signals. (Bits 5 and 4, respectively, of MCU output channel 0.) The Clear I/O signal clears the I/O MUX of all current I/O requests. The MCU then drops the Clear I/O signal and holds the Dead Start signal up while it writes into SCM. Since the MCU has access to any part of SCM, each word sent to SCM is given a specific address. When the MCU drops the Dead Start signal, 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.

PPU DEAD DUMP

One of the control signals sent by the MCU to a PPU (through Scanner selection) is the Dead Dump signal. The MCU sends this signal, under control of the MCU program, when the PPU program has failed and a dump of PPU memory is desired to analyze the cause of failure.

To Dead Dump a PPU, the MCU program first outputs a Dead Start signal to set up the PPU registers. Then it outputs the Dead Dump signal as bit 1 on MCU output channel 0. At the selected PPU, the Dead Dump signal changes the 71 input code in the f register (set by the Dead Start signal) to a 73 output code. This output code causes the PPU to begin transmission of the entire PPU storage over PPU output channel 0. The transmission starts at PPU address 0 and terminates at PPU address 7776_8 . Input of this data at the MCU is under control of the MCU program.

PPU AND MCU PARITY ERRORS

The MCU and each PPU contains hardware for detecting parity errors during memory read cycles. The storage stack that has failed is indicated by setting a stack parity error bit in a 4-bit register.

The MCU is able to sense the contents of a PPU parity error register by selecting the PPU via the Scanner and inputting the PPU Stack 0 - 3 Parity Error bits as bits 0 - 4 on MCU input channel 3. The MCU clears the PPU Parity Error register by selecting the scanner channel for that PPU and outputting the PPU Clear Parity Error signal as bit 0 on MCU output channel 0.

The MCU senses the contents of its own parity error register by reading bits 6-9 of MCU input channel 3. The MCU clears its parity errors by outputting the MCU Clear Parity Error signal as bit 6 on MCU output channel 0.

SCM AND LCM PARITY ERRORS

The MCU is also able to monitor the SCM and LCM Section Parity Errors and read the SCM or LCM address of the failing word.

Under MCU program control, the MCU inputs the SCM Section 0 - 4 Parity Error bits as bits 0 - 4 on MCU input channel 5. When set, the SCM Parity Error bits indicate bad parity for the following portions of the SCM word:

Error Bit 4	SCM Section 4 (bits 48 - 59)
3	SCM Section 3 (bits 36 - 47)
2	SCM Section 2 (bits 24 - 35)
1	SCM Section 1 (bits 12 - 23)
0	SCM Section 0 (bits 0 - 11)

The MCU program reads the SCM parity error address by selecting Scanner channel 16₈ and inputting address bits 0 - 11 on MCU input channel 7 and address bits 12 - 16 on MCU input channel 3 (bits 0 - 4).

Under MCU program control, the MCU inputs the LCM Section 0 - 3 Parity Error bits as bits 1 - 4 on MCU input channel 4. When set, the LCM Section Parity Error bits indicate bad parity for the following portions of the LCM word:

Error bit 3	LCM Section 3 (bits 45 - 59)
2	LCM Section 2 (bits 30 - 44)

Error bit 1	LCM Section 1 (bits 15 - 29)
0	LCM Section 0 (bits 0 - 14)

The MCU program reads bits 0 - 17 of the LCM parity error address by selecting Scanner channel 17_g and inputting address bits 0 - 11 on MCU input channel 7 and address bits 12 - 17 on MCU input channel 3 (bits 0 - 5). LCM address bit 18 bypasses the Scanner and is input as bit 0 on input channel 4.

The MCU clears both the SCM and the LCM parity error bits by outputting the CPU Clear Parity Error signal as bit 3 on output channel 0.

PPU PROGRAM ERROR

PPU instructions 00 and 77 cause the PPU program to stop and to indicate a program error condition. The MCU is able to sense this PPU program error by selecting the PPU via the Scanner and reading the PPU Program Error bit as bit 4 on MCU input channel 3. The PPU can be restarted only by Dead Start signal from the MCU.

CONSOLE

The display console (Figure 7-2) consists of a cathode ray tube display and an alphanumeric keyboard. A system may include several display consoles for monitoring or controlling various areas of system activity.

ALPHANUMERIC KEYBOARD

The alphanumeric keyboard provides coded binary signals to the computer. Depression of each key generates a unique 6-bit octal code. Assembly and display of keyboard entries is a function of the system software.

DISPLAY

Control signals from the computer generate alphanumeric display images within a 10-inch by 10-inch display area on the 21-inch crt.

An electron beam at the crt produces a visible display when it strikes the phosphor coated crt screen, causing that portion of the phosphor to glow briefly. Normally, the glow fades within a fraction of a second, too soon for the human eye to perceive and identify the image. For this reason the display image must be refreshed at a rate that makes the display appear steady and of uniform intensity to the observer. Display refresh is under computer program control. To prevent damaging the crt phosphor, the program must not refresh the display image more than once every 20 milliseconds.

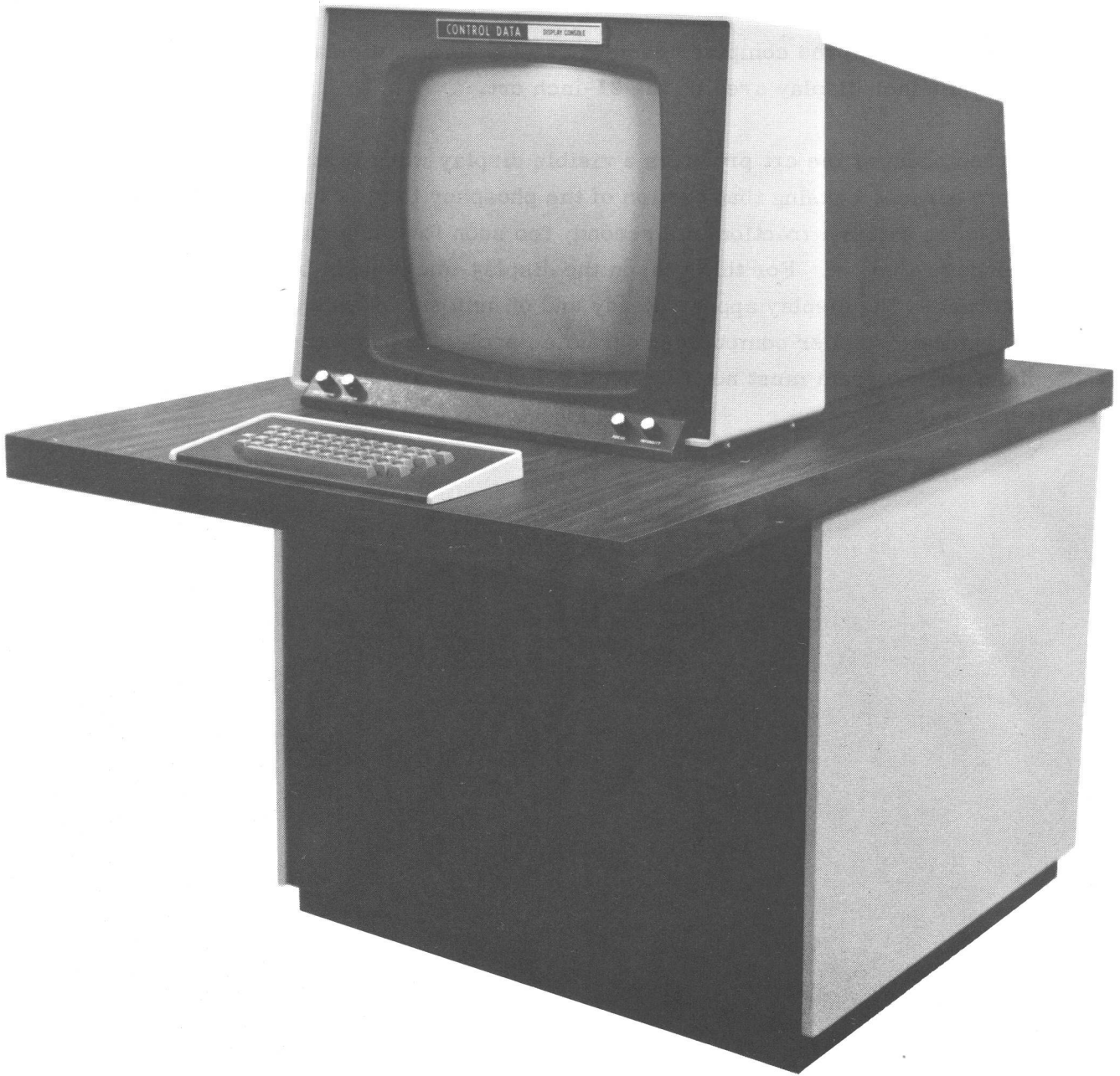


Figure 7-2. Display Console

APPENDIX A

TIMING NOTES

TIMING NOTES
CENTRAL PROCESSOR UNIT

1. Times given include clock periods 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 multiply 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 previous initiated read/write cycles.
3. A delay will occur during instructions 011 and 012 when an I/O multiplexer 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 three conditions are satisfied:
 - a. All operating registers are free.
 - b. No SCM bank conflicts exist.
 - c. All previous instruction fetches completed.
 - d. LCM is not busy.

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. X_j register is free.
 - c. X_k 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 17 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 B_i 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) may be 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.

13. A register is reserved if it is the destination of an instruction that has been initiated but has not been completed. A register is free in the clock period following the store into it.

CENTRAL PROCESSOR INSTRUCTIONS

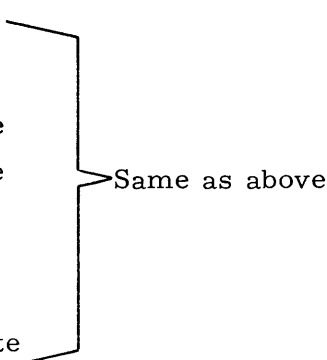
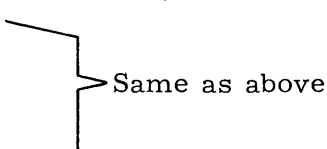
INSTRUCTION CODE	NAME	EXECUTION TIME (CLOCK PERIODS)	FUNCTIONAL UNIT
00000	Error exit to EEA	-	-
0100K	Return jump to K	Min 13*	-
011jK	Block copy K + (Bj) words from LCM to SCM	Min = N + 16**	-
012jK	Block copy K + (Bj) words from SCM to LCM	Min = N + 12**	-
01300	Exchange exit to NEA if exit flag clear	Min = 28	-
013jK†	Exchange exit to K + (Bj) if exit flag set	Min = 28	-
014jk	Read LCM at (Xk) to Xj	3, 17	-
015jk	Write (Xj) into LCM at (Xk)	3	-
0160k†	Reset input channel (Bk) buffer if j = 0	4	-
016jk	Read input channel (Bk) status to Bj if j ≠ 0	3	-
0170k†	Reset output channel (Bk) buffer if j = 0	16	-
017jk	Read output channel (Bk) status to Bj if j ≠ 0	3	-
02i0K	Jump to K + (Bi)	Min 3 (in stack jump) Min 11 (out of stack jump)	-
030jK	Branch to K if (Xj) = 0	Min 2 (branch fall through) Min 3 (branch in stack) Min 11 (branch out of stack)	-

* Refer to Timing Notes.

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

† Privileged to programs in monitor mode.

CENTRAL PROCESSOR INSTRUCTIONS (Cont'd)

INSTRUCTION CODE	NAME	EXECUTION TIME (CLOCK PERIODS)	FUNCTIONAL UNIT
031jK	Branch to K if (Xj) \neq 0		-
032jK	Branch to K if (Xj) positive		-
033jK	Branch to K if (Xj) negative		-
034jK	Branch to K if (Xj) in range		-
035jK	Branch to K if (Xj) out of range		-
036jK	Branch to K if (Xj) definite		-
037jK	Branch to K if (Xj) indefinite		-
04ijK	Branch to K if (Bi) = (Bj)	Min 2 (branch fall through) Min 3 (branch in stack) Min 11 (branch out of stack)	-
05ijK	Branch to K if (Bi) \neq (Bj)		-
06ijK	Branch to K if (Bi) \geq (Bj)		-
07ijK	Branch to K if (Bi) < (Bj)		-
10ij0	Transmit (Xj) to Xi	2	Boolean
11ijk	Logical product of (Xj) and (Xk) to Xi	2	Boolean
12ijk	Logical sum of (Xj) and (Xk) to Xi	2	Boolean
13ijk	Logical difference of (Xj) and (Xk) to Xi	2	Boolean
14i0k	Transmit complement of (Xk) to Xi	2	Boolean
15ijk	Logical product of (Xj) and comp (Xk) to Xi	2	Boolean
16ijk	Logical sum (Xj) and comp (Xk) to Xi	2	Boolean
17ijk	Logical difference of (Xj) and comp (Xk) to Xi	2	Boolean
20ijk	Left shift (Xi) by jk	2	Shift
21ijk	Right shift (Xi) by jk	2	Shift

CENTRAL PROCESSOR INSTRUCTIONS (Cont'd)

INSTRUCTION CODE	NAME	EXECUTION TIME (CLOCK PERIODS)	FUNCTIONAL UNIT
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) and (Xk) to Xi	4	Floating Add
31ijk	Floating difference of (Xj) and (Xk) to Xi	4	Floating Add
32ijk	Floating DP sum of (Xj) and (Xk) to Xi	4	Floating Add
33ijk	Floating DP difference of (Xj) and (Xk) to Xi	4	Floating Add
34ijk	Round floating sum of (Xj) and (Xk) to Xi	4	Floating Add
35ijk	Round floating difference of (Xj) and (Xk) to Xi	4	Floating Add
36ijk	Integer sum of (Xj) and (Xk) to Xi	2	Long Add
37ijk	Integer difference of (Xj) and (Xk) to Xi	2	Long Add
40ijk	Floating product of (Xj) times (Xk) to Xi	5	Floating Multiply
41ijk	Round floating produce of (Xj) times (Xk) to Xi	5	Floating Multiply
42ijk	Floating DP produce 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

CENTRAL PROCESSOR INSTRUCTIONS (Cont'd)

INSTRUCTION CODE	NAME	EXECUTION TIME (CLOCK PERIODS)	FUNCTIONAL UNIT
45ijk	Round floating divide (Xj) by (Xk) to Xi	20	Floating Divide
46000	Pass	1	
47i0k	Population count of (Xk) to Xi	2	Population Count
50ijK	Increment (Aj) plus K to Ai	2 (Set Aj) Min 8 (Read to Xi) 2 (Store from Xi)	Increment
51ijK	Increment (Bj) plus K to Ai	} Same as above	Increment
52ijK	Increment (Xj) plus K to Ai		Increment
53ijk	Increment (Xj) plus (Bk) to Ai		Increment
54ijk	Increment (Aj) plus (Bk) to Ai	2 Set (Aj) Min 8 (Read to Xi) 2 (Store from Xi)	Increment
55ijk	Increment (Aj) minus (Bk) to Ai	} Same as above	Increment
56ijk	Increment (Bj) plus (Bk) to Ai		Increment
57ijk	Increment (Bj) minus (Bk) to Ai		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

CENTRAL PROCESSOR INSTRUCTIONS (Cont'd)

INSTRUCTION CODE	NAME	EXECUTION TIME (CLOCK PERIODS)	FUNCTIONAL UNIT
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

<u>INSTRUCTION CODE (OCTAL)</u>	<u>NAME</u>	<u>EXECUTION TIME (CLOCK PERIODS)</u>
00	Error stop	7
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	7, 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

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 2 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)

<u>INSTRUCTION CODE (OCTAL)</u>	<u>NAME</u>	<u>EXECUTION TIME (CLOCK PERIODS)</u>
23	Logical difference dm	10
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

PERIPHERAL PROCESSOR INSTRUCTIONS (Cont'd)

INSTRUCTION CODE (OCTAL)	NAME	EXECUTION TIME (CLOCK PERIODS)
5400	Store (m)	20
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	+

* 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 for each case are stated on the following page.

PERIPHERAL PROCESSOR INSTRUCTIONS (Cont'd)

<u>INSTRUCTION CODE (OCTAL)</u>	<u>NAME</u>	<u>EXECUTION TIME (CLOCK PERIODS)</u>
74	Output record flag on channel d	5
75	Pass	5
76	Pass	5
77	Error Stop	- (restart only by a Dead Start)

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.

Execution Time = 10 Clock Periods

APPENDIX B

FLOATING POINT ARITHMETIC

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.

When the coefficient is negative, the packed positive exponent is complemented to become:

5777 through 4000.

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.

When the coefficient is negative, the packed negative exponent is complemented to become:

6000 through 7777.

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.

Note that the packed negative zero exponent is not used for normal operation in the machine. Instead, the value 1777 is used to indicate the special error condition of indefinite.

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 = 3777 0-----0 Partial Overflow = 3777 X-----X	Complete Overflow = 4000 7----7 Partial Overflow = 4000 X---X
INTEGERS	Largest: $7\text{-----}7. \times 2^{+1776} = 3776\ 7\text{-----}7$ Smallest: $1. \times 2^0 = 2000\ 0\text{-----}01$	*Largest: $-7\text{-----}7. \times 2^{+1776} = 4001\ 0\text{-----}0$ *Smallest: $-1. \times 2^0 = 5777\ 7\text{----}76$
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: $7\text{-----}7. \times 2^{-60} = 1717\ 7\text{-----}7$ Smallest: $1. \times 2^{-1777} = 0000\ 0\text{-----}01$	*Largest: $-7\text{-----}7. \times 2^{-60} = 6060\ 0\text{-----}0$ *Smallest: $-1. \times 2^{-1777} = 7777\ 7\text{----}76$
UNDER-FLOW	Complete Underflow = 0000 0-----0 Partial Underflow = 0000 X-----X	Complete Underflow = 7777 7----7 Partial Underflow = 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 (177770-----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 overflow (+ ∞)	= 3777X-----X
negative overflow (- ∞)	= 4000X-----X
positive indefinite (+IND)	= 1777X-----X
negative indefinite (-IND)	= 6000X-----X
positive underflow (+0)	= 0000X-----X
negative underflow (-0)	= 7777X-----X

When a floating point arithmetic unit uses one of these six special forms as an operand 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

$$X_i = X_j + X_k$$

(Instructions 30, 32, 34)

		X _k			
		W	+∞	-∞	±IND
X _j	W	-	+∞	-∞	IND
	+∞	+∞	+∞	IND	IND
	-∞	-∞	IND	-∞	IND
	±IND	IND	IND	IND	IND

SUBTRACT

$$X_i = X_j - X_k$$

(Instructions 31, 33, 35)

		X _k			
		W	+∞	-∞	±IND
X _j	W	-	-∞	+∞	IND
	+∞	+∞	IND	+∞	IND
	-∞	-∞	-∞	IND	IND
	±IND	IND	IND	IND	IND

MULTIPLY

$$X_i = X_j * X_k$$

(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

DIVIDE

$$X_i = X_j / X_k$$

(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

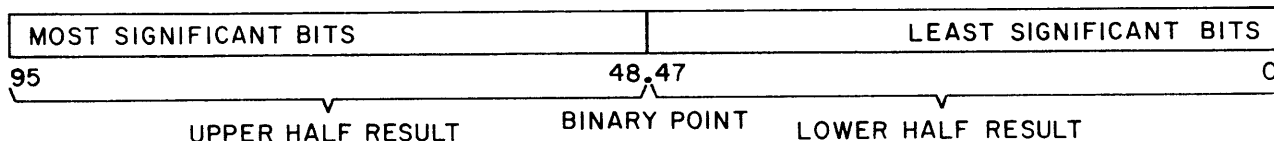
Optional floating point instructions are provided to round the results in single precision computation. These instructions are executed in the same amount of time as the unrounded versions. The operands are modified in the functional units to accomplish the rounding function. The amount of bias introduced 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. Two instructions, one single precision and one double precision, are required to retrieve an entire double precision result.

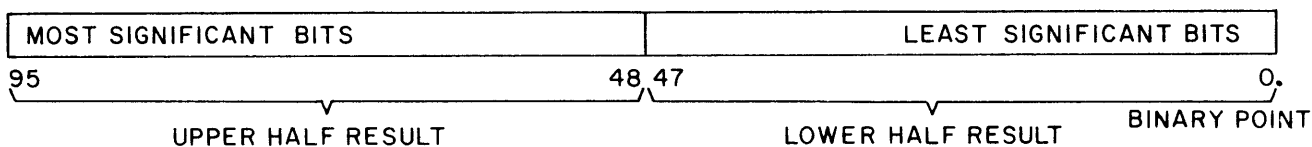
To add or subtract two floating point numbers, the floating point Add unit enters the coefficient having the smaller exponent into the upper half of an accumulator and shifts it right by the difference of the exponents. Then it adds the other coefficient

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 only 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, only 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

COMPASS
MNEMONIC CODES

CENTRAL PROCESSOR INSTRUCTIONS

ES	00000	Error exit to EEA	15 bits
RJ	0100K	Return jump to K	30 bits
RL	011jK	Block copy K + (Bj) words from LCM to SCM	30 bits
WL	012jK	Block copy K + (Bj) words from SCM to LCM	30 bits
MJ	01300	Exchange exit to NEA if exit flag clear	15 bits
MJ	013jK	Exchange exit to K + (Bj) if exit flag set	30 bits
RX	014jk	Read LCM at (Xk) to Xj	15 bits
WX	015jk	Write (Xj) into LCM at Xk	15 bits
RI	0160k	Reset input channel (Bk) buffer if j=0	15 bits
IB	016jk	Read input channel (Bk) status to Bj if j≠0	15 bits
TB	016j0	Read real time clock to Bj if (Bk)=0	15 bits
RO	0170k	Reset output channel (Bk) buffer if j=0	15 bits
OB	017jk	Read output channel (Bk) status to Bj if j≠0	15 bits
JP	02i0K	Jump to K + (Bi)	30 bits
ZR	030jK	Branch to K if (Xj)=0	30 bits
NZ	031jK	Branch to K if (Xj)≠0	30 bits
PL	032jK	Branch to K if (Xj) positive	30 bits
NG	033jK	Branch to K if (Xj) negative	30 bits
IR	034jK	Branch to K if (Xj) in range	30 bits
OR	035jK	Branch to K if (Xj) out of range	30 bits
DF	036jK	Branch to K if (Xj) definite	30 bits
ID	037jK	Branch to K if (Xj) indefinite	30 bits

EQ	04ijk	Branch to K if $(B_i)=(B_j)$	30 bits
NE	05ijk	Branch to K if $(B_i)\neq(B_j)$	30 bits
GE	06ijk	Branch to K if $(B_i)\geq(B_j)$	30 bits
LT	07ijk	Branch to K if $(B_i)<(B_j)$	30 bits
BX	10ij0	Transmit (X_j) to X_i	15 bits
BX	11ijk	Logical product of (X_j) and (X_k) to X_i	15 bits
BX	12ijk	Logical sum of (X_j) and (X_k) to X_i	15 bits
BX	13ijk	Logical difference of (X_j) and (X_k) to X_i	15 bits
BX	14i0k	Transmit complement of (X_k) to X_i	15 bits
BX	15ijk	Logical product of (X_j) and comp (X_k) to X_i	15 bits
BX	16ijk	Logical sum (X_j) and comp (X_k) to X_i	15 bits
BX	17ijk	Logical difference of (X_j) and comp (X_k) to X_i	15 bits
LX	20ijk	Left shift (X_i) by jk	15 bits
AX	21ijk	Right shift (X_i) by jk	15 bits
LX	22ijk	Left shift (X_k) by (B_j) to X_i	15 bits
AX	23ijk	Right shift (X_k) by (B_j) to X_i	15 bits
NX	24ijk	Normalize (X_k) to X_i and B_j	15 bits
ZX	25ijk	Round and normalize (X_k) to X_i and B_j	15 bits
UX	26ijk	Unpack (X_k) to X_i and B_j	15 bits
PX	27ijk	Pack (X_k) and (B_j) to X_i	15 bits
FX	30ijk	Floating sum of (X_j) and (X_k) to X_i	15 bits
FX	31ijk	Floating difference of (X_j) minus (X_k) to X_i	15 bits
DX	32ijk	Floating DP sum of (X_j) and (X_k) to X_i	15 bits
DX	33ijk	Floating DP difference of (X_j) minus (X_k) to X_i	15 bits
RX	34ijk	Round floating sum of (X_j) and (X_k) to X_i	15 bits
RX	35ijk	Round floating difference of (X_j) minus (X_k) to X_i	15 bits
IX	36ijk	Integer sum of (X_j) and (X_k) to X_i	15 bits
IX	37ijk	Integer difference of (X_j) minus (X_k) to X_i	15 bits

FX	40ijk	Floating product of (Xj) and (Xk) to Xi	15 bits
RX	41ijk	Round floating product of (Xj) and (Xk) to Xi	15 bits
DX	42ijk	Floating DP product of (Xj) and (Xk) to Xi	15 bits
MX	43ijk	Form mask of jk bits to Xi	15 bits
FX	44ijk	Floating divide (Xj) by (Xk) to Xi	15 bits
RX	45ijk	Round floating divide (Xj) by (Xk) to Xi	15 bits
NO	46000	Pass	15 bits
CX	47i0k	Population count of (Xk) to Xi	15 bits
SA	50ijK	Set Ai to (Aj) plus K	30 bits
SA	51ijK	Set Ai to (Bj) plus K	30 bits
SA	52ijK	Set Ai to (Xj) plus K	30 bits
SA	53ijk	Set Ai to (Xj) plus (Bk)	15 bits
SA	54ijk	Set Ai to (Aj) plus (Bk)	15 bits
SA	55ijk	Set Ai to (Aj) minus (Bk)	15 bits
SA	56ijk	Set Ai to (Bj) plus (Bk)	15 bits
SA	57ijk	Set Ai to (Bj) minus (Bk)	15 bits
SB	60ijK	Set Bi to (Aj) plus K	30 bits
SB	61ijK	Set Bi to (Bj) plus K	30 bits
SB	62ijK	Set Bi to (Xj) plus K	30 bits
SB	63ijk	Set Bi to (Xj) plus (Bk)	15 bits
SB	64ijk	Set Bi to (Aj) plus (Bk)	15 bits
SB	65ijk	Set Bi to (Aj) minus (Bk)	15 bits
SB	66ijk	Set Bi to (Bj) plus (Bk)	15 bits
SB	67ijk	Set Bi to (Bj) minus (Bk)	15 bits

SX	70ijk	Set X_i to (A_j) plus K	30 bits
SX	71ijk	Set X_i to (B_j) plus K	30 bits
SX	72ijk	Set X_i to (X_j) plus K	30 bits
SX	73ijk	Set X_i to (X_j) plus (B_k)	15 bits
SX	74ijk	Set X_i to (A_j) plus (B_k)	15 bits
SX	75ijk	Set X_i to (A_j) minus (B_k)	15 bits
SX	76ijk	Set X_i to (B_j) plus (B_k)	15 bits
SX	77ijk	Set X_i to (B_j) minus (B_k)	15 bits

PERIPHERAL PROCESSOR INSTRUCTIONS

ESN	00	Error Stop	12 bits
LJM	01	Long jump to $m + (d)$	24 bits
RJM	02	Return jump to $m + (d)$	24 bits
UJN	03	Unconditional jump d	12 bits
ZJN	04	Zero jump d	12 bits
NJN	05	Nonzero jump d	12 bits
PJN	06	Plus jump d	12 bits
MJN	07	Minus jump d	12 bits
SHN	10	Shift (A) by d	12 bits
LMN	11	Logical difference (A) and d	12 bits
LPN	12	Logical product (A) and d	12 bits
SCN	13	Selective clear (A)	12 bits
LDN	14	Load d	12 bits
LCN	15	Load complement d	12 bits
ADN	16	Add $d + (A)$	12 bits
SBN	17	Subtract (A) - d	12 bits
LDC	20	Load dm	24 bits
ADC	21	Add $dm + (A)$	24 bits
LPC	22	Logical product dm and (A)	24 bits
LMC	23	Logical difference dm and (A)	24 bits
PSN	24	Pass	12 bits
	25	Pass	12 bits
	26	Pass	12 bits
	27	Pass	12 bits

LDD	30	Load (d)	12 bits
ADD	31	Add (d) + (A)	12 bits
SBD	32	Subtract (A) - (d)	12 bits
LMD	33	Logical difference (A) and (d)	12 bits
STD	34	Store (A) at d	12 bits
RAD	35	Replace add (d) + (A)	12 bits
AOD	36	Replace add one (d)	12 bits
SOD	37	Replace subtract one (d)	12 bits
LDI	40	Load ((d))	12 bits
ADI	41	Add ((d)) + (A)	12 bits
SBI	42	Subtract (A) - ((d))	12 bits
LMI	43	Logical difference (A) - ((d))	12 bits
STI	44	Store (A) at (d)	12 bits
RAI	45	Replace add ((d)) + (A)	12 bits
AOI	46	Replace add one ((d))	12 bits
SOI	47	Replace subtract one ((d))	12 bits
LDM	50	Load (m+(d))	24 bits
ADM	51	Add (m+(d)) + (A)	24 bits
SBM	52	Subtract (A) - (m+(d))	24 bits
LMM	53	Logical difference (A) and (m+(d))	24 bits
STM	54	Store (A) at m+(d)	24 bits
RAM	55	Replace add (A) + (m+(d)) to m+(d)	24 bits
AOM	56	Replace add one (m+(d)) + 1 to m+(d)	24 bits
SOM	57	Replace subtract one (m+(d)) -1 to m+(d)	24 bits

FIM	60	Jump to m; Input Word flag on channel d	24 bits
EIM	61	Jump to m; no Input Word flag on channel d	24 bits
IRM	62	Jump to m; Input Record flag on channel d	24 bits
NIM	63	Jump to m; no Input Record flag on channel d	24 bits
FOM	64	Jump to m; Output Word flag on channel d	24 bits
EOM	65	Jump to m; no Output Word flag on channel d	24 bits
ORM	66	Jump to m; Output Record flag on channel d	24 bits
NOM	67	Jump to m; no Output Record flag on channel d	24 bits
IAN	70	Input to A from channel d	12 bits
IAM	71	Input (A) words to m from channel d	24 bits
OAN	72	Output from A on channel d	12 bits
OAM	73	Output (A) words from m on channel d	24 bits
RFN	74	Send Record flag on channel d	12 bits
	75	Pass	12 bits
	76	Pass	12 bits
ESN	77	Error stop	12 bits

APPENDIX D

6000 / 7000 RESULT DIFFERENCES

APPENDIX D
6000/7000 RESULT DIFFERENCES

- Whenever infinite, indefinite, or zero exponent results are generated by a floating point unit, only the following octal words can occur as results:

	<u>6000</u>	<u>7000</u>
positive overflow	37770-----0	37770-----0
negative overflow	40000-----0	40007-----7
positive indefinite	17770-----0	17770-----0
positive underflow	00000-----0	00000-----0
negative underflow	00000-----0	77777-----7

Example: 24012 or
 25012

where: X2 = 7753 7777 7773 0000 0000

7000 Result: X0 = 7777 7777 7777 7777 7777

6000 Result: X0 = 0000 0000 0000 0000 0000

- A difference exists in the way in which a round divide is handled on a 6000 machine and on a 7000 machine. The 6000 performs a 1/3 round on the divide and the 7000 machine performs a 1/2 round. The 7000, therefore, can give a different answer from the 6000 when using certain operands.

Example: 45012

Where: X1 = 2027 7223 2220 7175 5360

 X2 = 1347 4255 6115 0364 7225

7000 Result: X0 = 2400 6557 3505 0613 2701

6000 Result: X0 = 2400 6557 3505 0613 2700

- An Error Exit instruction (00XXX) in the 7000 machine causes an Exchange Jump to the Error Exit Address (EEA) and does not halt the CPU. An Error Exit instruction in the 6000 machine causes the CPU to stop executing until a PPU Exchange Jump causes the CPU to reinitiate.

4. A difference exists when an exponent overflow of a floating product occurs and the coefficient result requires a left shift of one to give a normalized answer. The 7000 tests for the overflow condition by checking for the exponent being greater than +1777 before correction, if any, is made for a left shift of one. Thus, even though the left shift of one may cause the exponent to equal exactly +1777 (partial overflow), this condition is treated as a complete overflow and the result is the overflow exponent with a zero coefficient. The 6000 machine tests for the overflow condition by checking for the exponent greater than +1777 after correction, if any, is made for a left shift of one. In this case, if the resulting exponent is equal to exactly +1777 (partial overflow), the result is the overflow exponent with the computed coefficient.

Example: 40012

Where: X1 = 3700 4000 0000 0000 0000
 X2 = 2020 4000 0000 0000 0000
 7000 Result: X0 = 3777 0000 0000 0000 0000
 6000 Result: X0 = 3777 4000 0000 0000 0000

A similar situation exists when an exponent underflow of a floating product occurs and the coefficient result does not require a left shift of one to give a normalized answer. The 7000 tests for the underflow condition by checking for the exponent being less than -1776 before correction, if any is made for a left shift of one. Thus, although no left shift of one is performed, an exponent of -1777 (partial underflow) is treated as a complete underflow and the result is the underflow exponent with a zero coefficient. The 6000 machine tests for the underflow condition by checking for the exponent less than -1777 after correction, if any, is made for a left shift of one. In this case, if the resulting exponent is equal to exactly -1777 (partial underflow), the result is the underflow exponent with the computed coefficient.

Example: 40012

Where: X1 = 0647 7777 7777 7777 7776
 X2 = 1050 4444 4444 4444 4444
 7000 Result: X0 = 0000 0000 0000 0000 0000
 6000 Result: X0 = 0000 4444 4444 4444 4442

5. A difference exists when an exponent underflow of a floating double precision sum occurs and the coefficient result requires a right shift of one because coefficient overflow occurred. The 7000 tests for the underflow condition by checking for the exponent being less than -1777 before correction, if any, is made for a right shift of one. Thus, even though the right shift of one may cause the exponent to equal exactly -1777 (partial underflow), this condition is treated as a complete underflow and the result is the underflow exponent with a zero coefficient. The 6000 machine tests for the exponent underflow condition by checking for the exponent less than -1777 after correction, if any, is made for a right shift of one. In this case, if the resulting exponent is equal to exactly -1777 (partial underflow) the result is the underflow exponent with the computed coefficient.

Example: 32012

Where: X1 = 0057 4000 0000 0000 0001

X2 = 0057 4000 0000 0000 0000

7000 Result: X0 = 0000 0000 0000 0000 0000

6000 Result: X0 = 0000 4000 0000 0000 0000

6. When instruction 22 or 23 is used for a right shift, the 7000 checks bits $2^6 - 2^{11}$ for a shift greater than or equal to 64_{10} and ignores bits $2^{12} - 2^{16}$. For these instructions, the 6000 checks bits $2^6 - 2^{10}$ and ignores bits $2^{11} - 2^{16}$.
7. A difference exists between the 6000 and 7000 in signaling a Divide Fault condition on a floating divide instruction. If a Divide Fault is sensed in the 7000, an Indefinite Condition is indicated only if no Overflow or Underflow Condition also exists. If an Overflow or Underflow Condition exists, the Divide Fault situation is ignored. If a Divide Fault is sensed in the 6000, it is always identified as an Indefinite Condition.

Example: 44012

Where: X1 = 3700 0222 0000 0000 0000

X2 = 1600 0022 0000 0000 0000

7000 Result: X0 = 3777 0000 0000 0000 0000 (Overflow Condition)

6000 Result: X0 = 1777 0000 0000 0000 0000 (Indefinite Condition)

8. A difference exists between the 6000 and 7000 when an increment instruction makes a read reference to SCM and the address is out of range. In the 7000, when the reference involves reading to an X register, the out of range word addressed is read to the X register. The 6000 machine aborts the read and gates zeros to the X register. In both cases, an address out of range indication is given and the error exit sequence is performed.

Example: 51100 00100

Where: FL is set at 50

RA = 0

(100) = 5252 5252 5252 5252 5252

7000 Result: A1 = 100
X1 = 5252 5252 5252 5252 5252

6000 Result: A1 = 100
X1 = 0000 0000 0000 0000 0000

Future plans are to FCO the 7000 to make it agree with the 6000 by late 1971.

9. The 7000 Floating Add Unit may generate a different result from the 6000 Floating Add Unit when at least one operand has a zero coefficient and the difference between the exponents is greater than or equal to 128_{10} .

Example: 30012 Floating Add

Where: X1 = 4277 7777 7777 7777 7777

X2 = 5277 5555 5555 5555 5555

7000 Result: X0 = 3500 0000 0000 0000 0000

6000 Result: X0 = 4277 7777 7777 7777 7777

Reversing the operands (30021) gives the same results as shown above.

Example: 31012 Floating Difference

Where: X1 = 4277 7777 7777 7777 7777

X2 = 2500 2222 2222 2222 2222

7000 Result: X0 = 3500 0000 0000 0000 0000

6000 Result: X0 = 4277 7777 7777 7777 7777

Example: 31012 Floating Difference

Where: X1 = 5277 5555 5555 5555 5555

X2 = 3500 0000 0000 0000 0000

7000 Result: X0 = 3500 0000 0000 0000 0000

6000 Result: X0 = 4277 7777 7777 7777 7777

Reversing the operands (31021) on either of the examples for a Floating Difference gives compatible results on the 6000 and 7000 machines. The result on both machines is 3500 0000 0000 0000 0000.

GLOSSARY

A0-A7	Address Registers-CPU
B0-B7	Index Registers-CPU
BPA	Breakpoint Address
CIW	Current Instruction Word Register
Clock Period	27.5 nanoseconds
CPU	Central Processing Unit
EEA	Error Exit Address
FLL	Field Length-LCM
FLS	Field Length-SCM
IAS	Instruction Address Stack
IWS	Instruction Word Stack
LCM	Large Core Memory
LCMI	Large Core Memory Increment
MCU	Maintenance Control Unit
MUX	I/O Multiplexer
MUXI	I/O Multiplexer Increment
NEA	Normal Exit Address
P	Program Address Register
PPU	Peripheral Processing Unit
PSD	Program Status Designator
RAL	Reference Address-LCM
RAS	Reference Address-SCM
SAS	Storage Address Stack
SCM	Small Core Memory
SCMI	Small Core Memory Increment
SWS	Storage Word Stack
X0-X7	Operand Registers-CPU

INDEX

- A register,
 - Central Processor, 2-3
 - Peripheral Processor, 5-2
- Absolute memory address, 4-2
- Access,
 - Large Core Memory, 4-17
 - Small Core Memory, 4-4
- Address,
 - Absolute, 4-2
 - Error Exit, 2-12, 2-13
 - Large Core Memory, 4-16
 - Modes, 6-1
 - Reference, 4-1
 - Small Core Memory, 4-3
- Address Out of Range, 3-10, 3-12
- Arithmetic,
 - Floating point, B-1
 - Integer, B-11
- Assembly counter, 3-12
- Assembly register 1-6, 4-8
- B register, Central Processor, 2-3
- Block copy, 3-3, 3-6, 4-17
- Block Range condition, 4-1
- Boolean Unit, 3-22
- Breakpoint Address register, 2-12, 2-14
- Breakpoint Condition flag, 2-16, 2-19
- Buffer,
 - Input/output, 4-5, 4-7, 4-8
 - Small Core Memory, 1-5, 1-10
 - Threshold, 3-13, 3-16
- Central Processor, 1-1, 1-5, 2-1
 - A register, 2-3
 - B register, 2-3
 - Boolean Unit, 3-22
 - Branch instructions, 3-17
 - Characteristics, 1-1
 - Computation section, 1-1, 1-5, 2-1
 - Core memory, 1-5
 - Divide instructions, 3-43
 - Floating Point Add instructions 3-34
 - Functional units, 2-9
 - Increment instructions, 3-47
 - Index registers, 2-3
 - Input/Output instructions, 3-2
 - Instruction designators, 2-4, 3-1
 - Instruction formats, 2-4, 2-5
 - Instruction stack, 1-5, 2-6
 - Instructions, 3-1
 - Large Core Memory, 1-2
 - LCM instructions, 3-2
 - Long Add instructions, 3-39
 - Mask instructions, 3-32
 - Memory, 4-1
 - Mnemonic codes, COMPASS, C-1
 - Monitor instructions, 3-2
 - Multiplexer, 1-2
 - Multiply instructions, 3-40
 - No Operation instruction, 3-45
 - Normalize instructions, 3-32
 - Operating registers, 1-1, 2-1
 - Operation code, 2-5
 - Pack instruction, 3-28
 - Pass instruction, 3-45
 - Population Count, 3-46
 - P register, 2-7
 - Shift instructions, 3-29
 - Small Core Memory, 1-2
 - Time, instruction execution, A-4
 - Unpack instruction, 3-27
 - X register, 2-1
- Channel,
 - Bi-directional, 5-4
 - High Speed, 1-3, 1-6, 1-7, 4-8
 - Input, 3-13, 3-14
 - Normal, 1-3, 1-6, 1-7, 4-8
 - Output, 3-14, 3-15, 3-16
- Clear Parity Error, 7-1
- Clock period, 1-1
- COMPASS, mnemonic codes
 - Central Processor, C-1
 - Peripheral Processor, C-5
- Condensing Units, 1-8
- Console, display, 7-6
- Constant mode, 6-1, 6-3
- Core Memory, 1-5

- Counter
 - Assembly, 3-12
 - Clock Period, 2-14
 - Disassembly, 3-14
- Current Instruction Word Register, 2-1, 2-7
- Data Channel Unit, 1-3, 4-9
- Data Transfer
 - PPU to Peripheral Device, 5-10
 - PPU to PPU, 5-7
 - PPU to SCM, 4-9, 4-12
 - SCM to PPU, 4-11, 4-13
- Dead Dump, Peripheral Processor, 7-1, 7-4
- Dead Start, 7-1
 - CPU, 7-1, 7-4
 - MCU, 7-3
 - Operator station, 1-10
 - PPU, 7-3
 - System, 1-7
- Delay, execution, A-1
- Direct Address mode, 6-2, 6-3
- Direct Range condition, 4-1
- Disassembly Counter, 3-14
- Disassembly register, 1-6, 4-8
- Display Console, 7-6, 7-8
- Duty Cycle Integrator, 4-3
- Error Exit, 2-12, 2-13, 3-2
- Error Exit address, 2-12, 2-13, 2-14
- Exchange Exit, 2-12, 3-8, 3-9
- Exchange Jump, 2-10
- Exchange package, 1-6, 2-10, 2-11
 - Input/Output, 4-5
 - MCU, 7-4
 - Real Time, 2-14
- Execution interval, 2-12, 2-13
- Exit Mode flag, 2-12, 2-13, 3-8
- Field length, 4-1
- First Level PPU, 1-7
- Floating point,
 - Arithmetic, B-1
 - Double precision, B-9
 - Format, B-1
 - Indefinite, 3-21, B-5
 - Indefinite Condition flag, 2-16, 2-20
 - Indefinite Mode flag, 2-16, 2-17
 - Normalize, 3-33, B-9
 - Overflow, 3-21, B-3
 - Overflow Condition flag, 2-16, 2-20
 - Overflow Mode flag, 2-16, 2-17
 - Packing, 3-28, B-2
 - Single precision, B-10
 - Underflow, B-5
 - Underflow Condition flag, 2-16, 2-20
 - Underflow Mode flag, 2-16, 2-17
 - Unpacking, 3-22
- Floating Point Add Unit, 3-34
- Floating Point Divide Unit, 3-43
- Floating Point Multiply Unit, 3-40
- Format,
 - Address, LCM, 4-16
 - Address, SCM, 4-3
 - Floating point, B-1
 - Instruction, Central Processor, 2-3
 - Instruction, Peripheral Processor, 6-1
- Functional Units, Central Processor, 1-1, 2-9
 - Boolean, 3-22
 - Floating Point Add, 3-34
 - Floating Point Divide, 3-44
 - Floating Point Multiply, 3-40
 - Increment, 3-47
 - Long Add, 3-39
 - Normalize, 3-32
 - Population Count, 3-46
 - Segmentation, 2-10
 - Shift, 3-29
- High Speed Channel, 1-3, 1-6, 1-7, 4-8
- Increment Unit, 3-47
- Indefinite Condition flag, 2-16, 2-20
- Indefinite Mode flag, 2-16, 2-17
- Indefinite result, B-5
- Indexed Direct Address mode, 6-2, 6-3
- Indirect Address mode, 6-2, 6-3
- Input/Output,
 - Buffers, 4-4, 4-9
 - Instructions, 3-2
 - Interrupt, 2-14
 - Peripheral Processor, 5-4

- Input Record flag, 5-6
- Input Resume flag, 5-6
- Input Word flag, 5-5
- Instruction Address Stack, 2-6
- Instruction designators,
 - Central Processor, 3-1
 - Peripheral Processor, 6-4
- Instruction formats,
 - Central Processor, 2-5
 - Peripheral Processor, 6-1
- Instruction issue, Central Processor, 2-7
- Instruction Word Stack, 1-5, 2-6, 4-7
- Instructions,
 - Central Processor, 3-1
 - Peripheral Processor, 6-1, 6-4
- Integer arithmetic, B-11
- Interrupt,
 - Input/output, 2-14, 3-8, 3-9, 3-13, 4-4
 - I/O multiplexer, 3-6
 - Real Time, 2-14
- Large Core Memory, 1-4, 4-1, 4-15
 - Access, 4-17
 - Address format, 4-16
 - Bank, 4-15
 - Bank Operand register, 4-16
 - Block copy, 4-17
 - Block Range Condition flag, 2-16, 2-18
 - Direct Range Condition flag, 2-16, 2-18
 - Instructions, 3-2
 - Out of Range, 3-4
 - Parity, 4-16
 - Parity Condition flag, 2-16, 2-18
 - Parity Error, 4-16, 7-4
 - Read, 3-10
 - Single word transfer, 4-17
 - Write, 3-11
- Long Add Unit, 3-39
- Maintenance Control Unit, 1-7, 4-6, 5-1, 7-1
 - Clear Parity Error, 7-1, 7-4
 - Dead Dump, 7-1, 7-4
 - Dead Start, 7-1, 7-3
 - Scanner, 7-1
- Manual Control, 7-1
- Memory,
 - Central Processor, 4-1
 - Large Core Memory, 1-6, 4-1, 4-15
 - Peripheral Processor, 5-3
 - Small Core Memory, 1-5, 4-1, 4-2
- Modes, Peripheral Processor, 6-1
- Monitor instructions, 3-2
- Monitor Mode flag, 2-15, 2-16
- Multiplexer, 1-6, 4-5, 4-8
- No Address mode, 6-1, 6-3
- Normal Channel, 1-3, 1-6, 1-7, 4-8
- Normalize Unit, 3-32
- Operator Station, 1-8
- Output Record flag, 5-7
- Output Word flag, 5-3
- Overflow, B-3
- Overflow Condition flag, 2-16, 2-20
- Overflow Mode flag, 2-16, 2-17
- P register,
 - Central Processor, 2-7
 - Peripheral Processor, 5-2
- Parcel, 2-4, 2-5
- Parity,
 - LCM Parity Condition flag, 2-16, 2-18
 - Peripheral Processor Memory, 5-3
 - SCM Parity Condition flag, 2-16, 2-18
- Peripheral Processor, 1-1, 1-7, 5-1
 - A register, 5-2
 - Address modes, 6-1, 6-2
 - Branch instructions, 6-6
 - Characteristics, 1-4
 - Computation section, 1-3, 5-1
 - Constant instructions, 6-10
 - Direct Address instructions, 6-12
 - Error Stop, 6-5
 - First Level PPU, 1-7
 - Indexed Direct Address instruction, 6-17
 - Indirect Address instructions, 6-15
 - I/O control, 5-3
 - I/O instructions, 6-20
 - Instruction designators, 6-4
 - Instructions, 6-4

- Memory, 1-3, 1-7
- Mnemonic codes, COMPASS, C-5
- Operating registers, 1-3
- No Address instructions, 6-8
- No Operation instructions, 6-5
- P register, 5-2
- Q register, 5-2
- S register, 5-4
- Sk register, 5-3
- Time, instruction execution, A-9
- X register, 5-3, 5-4
- Z register, 5-4
- Population Count Unit, 3-46
- Power Distribution, 1-8
- Priority, SCM, 4-7
- Program Breakpoint, 2-14
- Program Error, Peripheral Processor, 7-5
- Program Range Condition flag, 2-16, 2-19
- Program Status Designators, 2-15, 2-16
 - Condition flags, 2-18
 - Mode flags, 2-15
- Q register, Peripheral Processor, 5-2
- Real Time Clock, 3-14
- Real Time Interrupt, 2-14
- Record Pulse, 5-5
- Reference Address, 4-1
- Reference Voltage Scanner, 7-1
- Refrigeration System, 1-6
- Registers
 - Assembly, 1-6, 4-8
 - Breakpoint Address, 2-14
 - Current Instruction Word, 2-1, 2-7
 - Disassembly, 1-6, 4-8
 - Operating, Central Processor, 1-1, 2-1
 - Operating, Peripheral Processor, 1-3, 5-2
 - Program Status Designator, 2-15
- Result difference, 6000/7000, D-1
- Resume Pulse, 5-5
- Rounded computation, B-9
- S register, Peripheral Processor, 5-4
- Scanner,
 - Maintenance Control Unit, 7-1
 - Reference Voltage, 7-1
- Shift Unit, 3-29
- Sk register, Peripheral Processor, 5-3
- Small Core Memory, 1-4, 4-1, 4-2
 - Access, 4-3
 - Address format, 4-3
 - Bank, 4-2
 - Block Range Condition flag, 2-16, 2-18
 - Buffer, 1-6, 1-10
 - Direct Range Condition flag, 2-16, 2-19
 - Memory reference, 4-7
 - Out of Range, 3-4
 - Parity, 4-3
 - Parity Condition flag, 2-16, 2-18
 - Parity Error, 7-4
 - Priority, 4-7
 - Stack, 4-2
- Step Condition flag, 2-16, 2-19
- Step mode, Central Processor, 2-16, 2-17
- Storage Address Stack, 4-7
- Storage Module,
 - Large Core Memory, 1-3
 - Small Core Memory, 1-3
- System communication, 1-10
- System, computer,
 - Characteristics, 1-1, 1-3
 - Description, 1-3
- Threshold, buffer, 3-13, 3-15
- Time, instruction execution, A-1
- Underflow, B-5
- Underflow Condition flag, 2-16, 2-20
- Underflow Mode flag, 2-16, 2-17
- Warning System, 1-8
- Word Pulse, 5-4
- X register,
 - Central Processor, 2-1
 - Peripheral Processor, 5-2, 5-4
- Z register, Peripheral Processor, 5-4

COMMENT SHEET

MANUAL TITLE CONTROL DATA 7600 COMPUTER SYSTEM

Reference Manual

PUBLICATION NO. 60258200 REVISION C

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

PRINTED IN U.S.A.

AA3419 REV. 11/69

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.



CUT ALONG LINE

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Technical Publications Department
4201 North Lexington Avenue
Arden Hills, Minnesota 55112

FOLD

FOLD



**CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD**

LITHO IN U.S.A.