CONTROL DATA
CORPORATION

CONTROL DATA®
7700 DUAL-PROCESSOR
COMPUTER SYSTEM

HARDWARE REFERENCE MANUAL

CONTROL DATA
CORPORATION

CONTROL DATA®
7700 DUAL-PROCESSOR
COMPUTER SYSTEM

HARDWARE REFERENCE MANUAL

REVISION RECORD

REVISION	DESCRIPTION
01	Preliminary edition.
(1-73)	
A	Manual released.
(11-73)	

Publication No.
60396300

Address comments concerning this manual to:

Control Data Corporation
Technical Publications Department
4201 North Lexington Avenue
Arden Hills, Minnesota 55112

or use Comment Sheet in the back of this manual.

PREFACE

This manual provides reference information for the CONTROL DATA® 7700 Dual-Processor Computer System. The first section includes a general description of the overall system. The second, third, and fourth sections describe the system hardware. The last two sections describe the instruction sets for the central processors and peripheral processor units.

The following are customer engineering manuals. Refer to the Literature Distribution Service Catalog for the latest revision of each manual.

Control Data Publications	Pub. No.
Central Computer (AA121-A) and Adjunct Processor (AD102-A)	
Theory, Diagrams, Maintenance Aids, Parts Data	60396400
Power and Reference Wire Lists	60396500
Central Processor (AA121-A)	
Chassis 1 Wire Lists	60396600
Chassis 2 Wire Lists	60396700
Chassis 3 Wire Lists	60396800
Chassis 4 Wire Lists	60396900
Chassis 5 Wire Lists	60397000
Chassis 6 Wire Lists	60397100
Chassis 7 Wire Lists	60397200
Chassis 8 Wire Lists	60397300
Chassis 9 Wire Lists	60397400
Chassis 10 Wire Lists	60397500
Chassis 11 Wire Lists	60397600
Chassis 12 Wire Lists	60397700
Chassis 13 Wire Lists	60397800
Chassis 14 Wire Lists	60397900
Chassis 15 Wire Lists	60398000
LCM Stack Cables Wire Lists	60398100
Channel Tabs Wire Lists	60398200
Adjunct Processor (AD102-A)	
Chassis 6 Wire Lists	60398300
Chassis 7 Wire Lists	60398400

Control Data Publications	Pub. No.
Chassis 8 Wire Lists	60398500
Chassis 9 Wire Lists	60398600
Chassis 10 Wire Lists	60398700
Chassis 11 Wire Lists	60398800
Chassis 12 Wire Lists	60398900
Chassis 13 Wire Lists	60399000
Chassis 14 Wire Lists	60399100
Channel Tabs Wire Lists	60399200
Peripheral Processor Unit (AB101-A, AT174-A, AB102-A/B)	60274900
Maintenance	60297900
Refrigeration System	60367300
Power Distribution and Warning System	60298600

CONTENTS

1. SYSTEM DESCRIPTION

Introduction 1-1
- Characteristics of Central Processors 1-1
- Characteristics of Large Core Memory 1-3
- Characteristics of Peripheral Processor Units (PPU) 1-4

Basic System Description 1-4
- Central Processors and Large Core Memory 1-5
- Peripheral Processor Units 1-7
- Maintenance Control Units 1-8
- Operator Stations 1-8
- Power Distribution Units 1-10
- Condensing Units 1-10

System Communication 1-10

2. CENTRAL PROCESSOR DESCRIPTION

Central Processing Unit. 2-1
- Operating Registers 2-2
- Instruction Registers 2-3
- PSD Register 2-4
- Support Registers 2-8
- BPA Register 2-9

Functional Units 2-9
- Boolean Unit 2-10
- Shift Unit 2-11
- Normalize Unit 2-11
- Floating Add Unit 2-11
- Long Add Unit 2-11
- Floating Multiply Unit 2-11
- Floating Divide Unit 2-11
- Population Count Unit 2-11
- Increment Unit 2-12

Input/Output Multiplexer 2-12
- Normal PPU to SCM Data Transfer 2-14
- Normal SCM to PPU Data Transfer 2-15
- High Speed PPU to SCM Data Transfer 2-17
- High Speed SCM to PPU Data Transfer 2-18

Small Core Memory 2-19
- Address Format 2-20
- Parity Checking 2-20
- Duty Cycle Integrator 2-20
- Memory Protection 2-21
- Memory References 2-22
- Memory Access 2-23

3. LARGE CORE MEMORY

Address Format 3-1
Parity Checking 3-2
Memory Protection 3-2
Memory References 3-3
- Block Copies 3-3
- Direct Single-Word Transfers 3-3

Access Control 3-4
- Unlocked Mode 3-4
- Locked Mode 3-4

Flag Register 3-5

4. PERIPHERAL PROCESSOR UNIT DESCRIPTION

Computation Section 4-1
- A Register 4-1
- P Register 4-2
- Q Register 4-2
- X Register 4-2

Sk Register 4-2
fd Register 4-3
k Register 4-3
Memory 4-3
Input/Output 4-3
Input Channel Control 4-4
Output Channel Control 4-5
PPU to PPU Data Transfers 4-5
PPU to Peripheral Equipment Data Transfers 4-8
Maintenance Control Unit 4-8
MCU Dead Start 4-11
PPU Dead Start 4-11
CPU Dead Start 4-11
PPU Dead Dump 4-12
Clear LCM 4-12
LCM Timeout Error 4-12
PPU and MCU Parity Errors 4-12
SCM and LCM Parity Errors 4-13
PPU Program Error 4-14

5. CENTRAL PROCESSOR INSTRUCTIONS

Instruction Formats 5-1
Instruction Issue 5-3
Program Branching 5-3
Duplicate Entries in IWS 5-3
Holes in IWS 5-4
Exchange Jump 5-4
Exchange Exit Instructions 5-6
Error Exit 5-7
Input/Output Interrupt 5-7
Real-Time Interrupt 5-8
Program Breakpoint 5-8
Step Mode 5-8
Floating Point Arithmetic 5-9
Format 5-9
Packing 5-10
Overflow 5-11
Underflow 5-12
Indefinite 5-12
Nonstandard Operands 5-13
Normalized Numbers 5-16
Rounding 5-16
Double Precision Results 5-16
Integer Arithmetic 5-17
Instruction Timing 5-18
Description of Instructions 5-23
00xxx Error Exit to EEA 5-24
010xK Return Jump to K 5-25
011jK Block Copy (Bj) + K Words from LCM to SCM 5-27
012jK Block Copy (Bj) + K Words from SCM to LCM 5-30
013jK Exchange Exit to (Bj) + K (Exit Mode Flag Set) 5-32
013xx Exchange Exit to NEA (Exit Mode Flag Not Set) 5-34
014jk Read LCM at (Xk) to Xj (Xk Bit 20 Not Set) 5-36
014jk Special LCM Functions (Xk Bit 20 Set) 5-37
015jk Write Xj into LCM at (Xk) 5-39
0160k Reset Input Channel (Bk) Buffer 5-40
016jk Read Input Channel (Bk) Status to Bj (j $\neq$ 0) 5-42
0170k Reset Output Channel (Bk) Buffer 5-43
017jk Read Output Channel (Bk) Status to Bj (j $\neq$ 0) 5-45
02ixK Jump to (Bi) + K 5-45
030jK Branch to K if (Xj) = 0 5-47
031jK Branch to K if (Xj) $\neq$ 0 5-49
032jK Branch to K if (Xj) Positive 5-49
033jK Branch to K if (Xj) Negative 5-50
034jK Branch to K if (Xj) In Range 5-50
035jK Branch to K if (Xj) Out of Range 5-51
036jK Branch to K if (Xj) Definite 5-52
037jK Branch to K if (Xj) Indefinite 5-52

04ijK Branch to K if (Bi) = (Bj) 5-53
05ijK Branch to K if (Bi) ≠ (Bj) 5-54
06ijK Branch to K if (Bi) ≥ (Bj) 5-55
07ijK Branch to K if (Bi) < (Bj) 5-56
10ijx Transmit (Xj) to Xi 5-56
11ijk Logical Product of (Xj) and (Xk) to Xi 5-57
12ijk Logical Sum of (Xj) and (Xk) to Xi 5-58
13ijk Logical Difference of (Xj) and (Xk) to Xi 5-59
14ixk Transmit Complement of (Xk) to Xi 5-60
15ijk Logical Product of (Xj) and Complement of (Xk) to Xi 5-61
16ijk Logical Sum of (Xj) and Complement of (Xk) to Xi 5-62
17ijk Logical Difference of (Xj) and Complement of (Xk) to Xi 5-63
20ijk Left Shift (Xi) by jk 5-64
21ijk Right Shift (Xi) by jk 5-65
22ijk Left Shift (Xk) Nominally (Bj) Places to Xi 5-66
23ijk Right Shift (Xk) Nominally (Bj) Places to Xi 5-68
24ijk Normalize (Xk) to Xi and Bj 5-69
25ijk Round Normalize (Xk) to Xi and Bj 5-72
26ijk Unpack (Xk) to Xi and Bj 5-74
27ijk Pack (Xk) and (Bj) to Xi 5-75
30ijk Floating Sum of (Xj) and (Xk) to Xi 5-77
31ijk Floating Difference of (Xj) and (Xk) to Xi 5-79
32ijk Floating Double Precision Sum of (Xj) and (Xk) to Xi 5-81
33ijk Floating Double Precision Difference of (Xj) and (Xk) to Xi 5-84
34ijk Round Floating Sum of (Xj) and (Xk) to Xi 5-86
35ijk Round Floating Difference of (Xj) and (Xk) to Xi 5-89
36ijk Integer Sum of (Xj) and (Xk) to Xi 5-92
37ijk Integer Difference of (Xj) and (Xk) to Xi 5-93
40ijk Floating Product of (Xj) and (Xk) to Xi 5-94
41ijk Round Floating Product of (Xj) and (Xk) to Xi 5-97
42ijk Floating Double Precision Product of (Xj) and (Xk) to Xi 5-98
43ijk Form Mask of jk Bits to Xi 5-100
44ijk Floating Divide (Xj) by (Xk) to Xi 5-101
45ijk Round Floating Divide (Xj) by (Xk) to Xi 5-104
46xxx Pass 5-105
47ixk Population Count of (Xk) to Xi 5-105
50ijK Set Ai to (Aj) + K 5-106
51ijK Set Ai to (Bj) + K 5-107
52ijK Set Ai to (Xj) + K 5-108
53ijk Set Ai to (Xj) + (Bk) 5-110
54ijk Set Ai to (Aj) + (Bk) 5-111
55ijk Set Ai to (Aj) - (Bk) 5-112
56ijk Set Ai to (Bj) + (Bk) 5-114
57ijk Set Ai to (Bj) - (Bk) 5-115
60ijK Set Bi to (Aj) + K 5-116
61ijK Set Bi to (Bj) + K 5-116
62ijK Set Bi to (Xj) + K 5-117
63ijk Set Bi to (Xj) + (Bk) 5-118
64ijk Set Bi to (Aj) + (Bk) 5-118
65ijk Set Bi to (Aj) - (Bk) 5-118
66ijk Set Bi to (Bj) + (Bk) 5-119
67ijk Set Bi to (Bj) - (Bk) 5-119
70ijK Set Xi to (Aj) + K 5-119
71ijK Set Xi to (Bj) + K 5-120
72ijK Set Xi to (Xj) + K 5-120
73ijk Set Xi to (Xj) + (Bk) 5-121
74ijk Set Xi to (Aj) + (Bk) 5-121
75ijk Set Xi to (Aj) - (Bk) 5-122
76ijk Set Xi to (Bj) + (Bk) 5-122
77ijk Set Xi to (Bj) - (Bk) 5-122

6. PERIPHERAL PROCESSOR UNIT INSTRUCTIONS

Instruction Formats 6-1
Address Modes 6-2
No Address Mode 6-3
Constant Mode 6-3
Direct Address Mode 6-3
Indexed Direct Address Mode 6-3
Indirect Address Mode 6-3
Examples of Address Modes 6-4
Restrictions on Instruction Loops 6-4
Instruction Timing 6-5
Description of Instructions 6-9
00xx Error Stop 6-10
0100m Long Jump to m 6-10
01dm Long Jump to m + (d) 6-10
0200m Return Jump to m 6-11
02dm Return Jump to m + (d) 6-11
03d Unconditional Jump d 6-12
04d Zero Jump d 6-12
05d Nonzero Jump d 6-13
06d Positive Jump d 6-13
07d Negative Jump d 6-14
10d Shift (A) by d 6-14
11d Logical Difference (A) and d 6-15
12d Logical Product (A) and d 6-15
13d Selective Clear (A) by d 6-16
14d Load d 6-16
15d Load Complement d 6-16
16d Add (A) + d 6-17
17d Subtract (A) - d 6-17
20dm Load dm 6-17
21dm Add (A) + dm 6-18
22dm Logical Product (A) and dm 6-18
23dm Logical Difference (A) and dm 6-18
24xx Pass 6-19
25xx Pass 6-19
26xx Pass 6-19
27xx Pass 6-19
30d Load (d) 6-19
31d Add (A) + (d) 6-19
32d Subtract (A) - (d) 6-20
33d Logical Difference (A) and (d) 6-20
34d Store (A) at (d) 6-20
35d Replace Add (A) + (d) 6-21
36d Replace Add One (d) 6-21
37d Replace Subtract One (d) 6-21
40d Load ((d)) 6-22
41d Add (A) + ((d)) 6-22
42d Subtract (A) - ((d)) 6-23
43d Logical Difference (A) and ((d)) 6-23
44d Store (A) at ((d)) 6-24
45d Replace Add (A) + ((d)) 6-24
46d Replace Add One ((d)) 6-25
47d Replace Subtract One ((d)) 6-25
5000m Load (m) 6-26
50dm Load (m + (d)) 6-27
5100m Add (A) + (m) 6-27
51dm Add (A) + (m + (d)) 6-28
5200m Subtract (A) - (m) 6-29
52dm Subtract (A) - (m + (d)) 6-29
5300m Logical Difference (A) and (m) 6-30
53dm Logical Difference (A) and (m + (d)) 6-30
5400m Store (A) at (m) 6-31
54dm Store (A) at (m + (d)) 6-31
5500m Replace Add (A) + (m) 6-32
55dm Replace Add (A + (m + (d)) 6-32
5600m Replace Add One (m) 6-33
56dm Replace Add One (m + (d)) 6-34
5700m Replace Subtract One (m) 6-34
57dm Replace Subtract One (m + (d)) 6-35
60dm Jump to m if Channel d Input Word Flag Set 6-36
61dm Jump to m if Channel d Input Word Flag Not Set 6-36

62dm Jump to m if Channel d Input Record Flag Set 6-36
63dm Jump to m if Channel d Input Record Flag Not Set 6-37
64dm Jump to m if Channel d Output Word Flag Set 6-37
65dm Jump to m if Channel d Output Word Flag Not Set 6-37
66dm Jump to m if Channel d Output Record Flag Set 6-38
67dm Jump to m if Channel d Output Record Flag Not Set 6-38
70d Input to A on Channel d 6-38
71dm Input (A) Words to m on Channel d 6-39
72d Output from A on Channel d 6-40
73dm Output (A) Words from m on Channel d 6-40
74d Set Output Record Flag on Channel d 6-41
75xx Pass 6-41
76xx Pass 6-41
77xx Error Stop 6-41

APPENDIX A. 6000/7000 Result Differences

APPENDIX B. Programming Considerations

FIGURES

1-1 Basic Computer System 1-2
1-2 Mainframe Chassis Configuration 1-6
1-3 Operator Station 1-9
2-1 CPU Information Flow 2-1
2-2 PSD Register Arrangement 2-5
2-3 I/O Buffer Areas in SCM 2-13
2-4 SCM Address Format 2-20
2-5 SCM Memory Map 2-22
2-6 I/O Exchange Package Areas in SCM 2-24
3-1 LCM Address Format 3-1
4-1 PPU/PPU Communications 4-6
4-2 PPU/Controller Communications 4-9
4-3 MCU Configuration 4-10
5-1 Parcel Instruction Arrangements 5-2
5-2 Exchange Package 5-5
6-1 PPU 12-bit Instruction Format 6-1
6-2 PPU 24-bit Instruction Format 6-2

TABLES

5-1 Bits 58 and 59 Configurations 5-9
5-2 Central Processor Instruction Timing 5-18
5-3 Central Processor Instruction Designators 5-23
6-1 Addressing Modes for PPU Instructions 6-2
6-2 Peripheral Processor Unit Instruction Timing 6-5
6-3 PPU Instruction Designators 6-9

SYSTEM DESCRIPTION 1

INTRODUCTION

The CONTROL DATA 7700 Dual-Processor Computer System combines two 15-million instruction per second central processors with a powerful input/output (I/O) structure. This gives a general purpose computer capable of processing concurrently a wide range of scientific and business applications. Individual users have the full power of the large central processors but the economies of a single facility shared by all departments of a corporation.

This computer system lends itself to all the common problem processing methods: batch, remote batch, and interactive. The efficiencies of multiprogramming provided by system software give unequalled throughput. Advanced concepts like distributed computing, tape staging, and parallel processing meet the challenge of a corporate data processing facility for current and future needs.

The basic computer system (Figure 1-1) comprises two central processors, a single large core memory (LCM), and up to 32 first level peripheral processor units (PPUs). Two of these PPUs are used as maintenance control units (MCUs). LCM is shared by the two central processors. A third access to LCM is provided for future expansion. Some of the PPUs are physically located with the central processors and others may be located in remote stations. Each PPU has up to six data links to peripheral equipment and/or second level PPU, one data link to a central processor, and one data link to an MCU.

CHARACTERISTICS OF CENTRAL PROCESSORS

CENTRAL PROCESSING UNIT (CPU)

- 60-bit internal word
- Synchronous internal logic with 27.5-nanosecond clock period
- 12-word instruction word stack (IWS)
- Eight 60-bit operand (X) registers
- Eight 18-bit address (A) registers
- Eight 18-bit index (B) registers

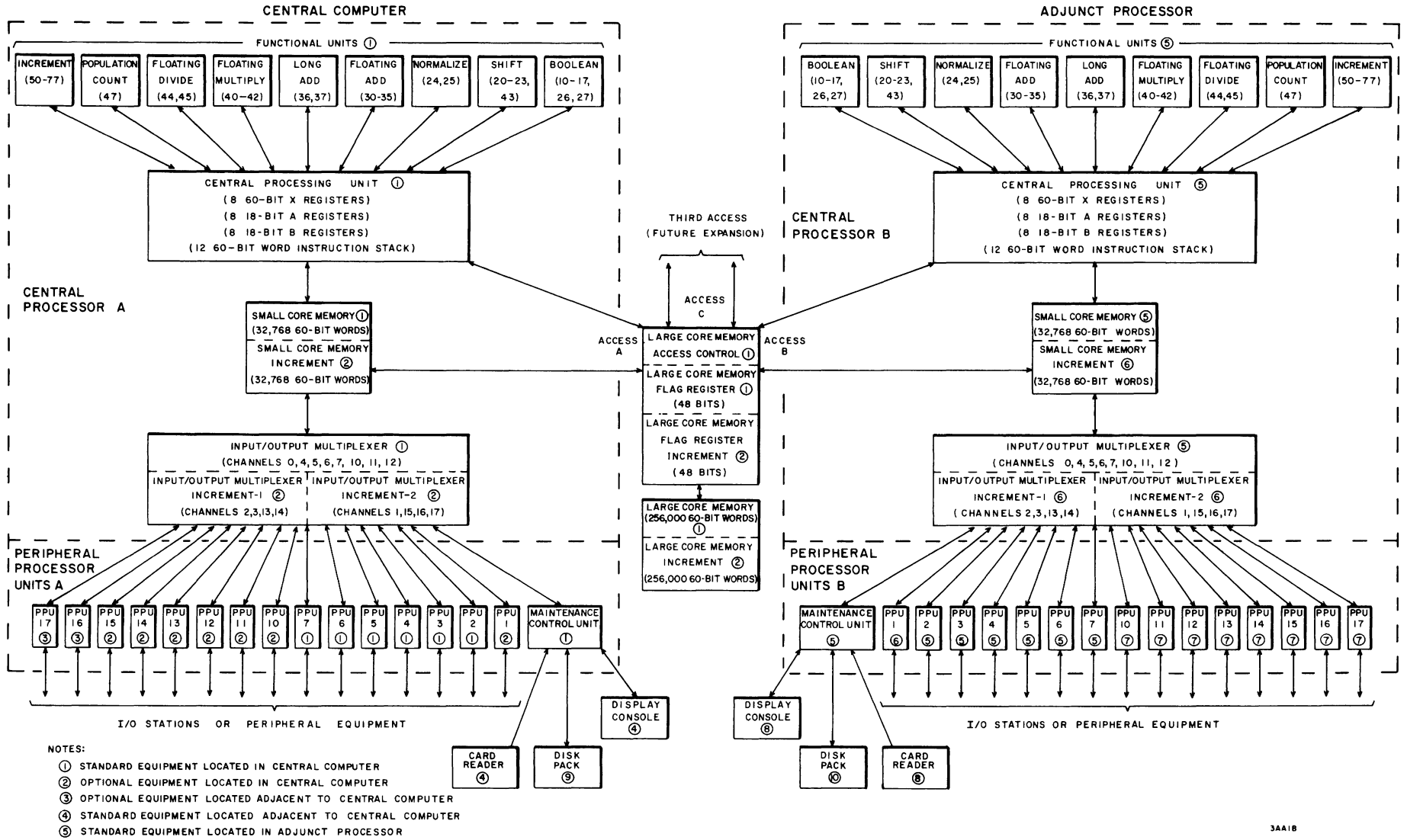


Figure 1-1. Basic Computer System

FUNCTIONAL UNITS

- Boolean unit
- Shift unit
- Normalize unit
- Floating add unit
- Long add unit
- Floating multiply unit
- Floating divide unit
- Population count unit
- Increment unit

INPUT/OUTPUT MULTIPLEXER (MUX)

- 8, 12, or 16 independent 12-bit channels
- Each channel bidirectional
- Fixed 128-word buffer areas in small core memory (SCM) for normal channels 1 and 10 through 17

  Fixed 256-word buffer areas in SCM for high speed channels 2 through 7

  Any area in SCM addressable by maintenance control unit (MCU) through channel 0

SMALL CORE MEMORY (SCM)

- 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 (2048 words per bank)
- 275-nanosecond read/write cycle time
- 27.5 nanoseconds per word maximum transfer rate

CHARACTERISTICS OF LARGE CORE MEMORY (LCM)

- 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)

- 1760-nanosecond read/write cycle time
- Eight words read simultaneously each reference
- 27.5 nanoseconds per word maximum transfer rate
- 48-bit or 96-bit flag register to permit locking of individual areas
- Accessible from either central processor
- Third access provided for future expansion

CHARACTERISTICS OF PERIPHERAL PROCESSOR UNITS (PPU)

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 (A) register
- 12-bit program address (P) register
- 13-bit memory read (X) register
- 12-bit instruction (fd) register
- 12-bit working (Q) register

MEMORY

- 4096 12-bit words of coincident current memory with a parity bit for each 12-bit word (odd parity)
- Organized into two independent banks (2048 words per bank)
- 275-nanosecond read/write cycle time

INPUT/OUTPUT SECTION

- Eight independent channels (asynchronous)
- Each channel bidirectional (12-bit)

BASIC SYSTEM DESCRIPTION

The computer system mainframe includes two central processors, a single LCM, two MCUs, and up to 20 PPUs. Also, up to 10 PPUs may be located externally. Typically, these additional PPUs are part of operator stations. Two power distribution units (PDUs) provide power for the system. The PDUs also contain warning circuits that monitor chassis and dew point temperatures. Four condensing units provide cooling for the system. These system elements are described briefly in the following paragraphs. The central processor, LCM, and PPU are described in more detail in sections 2, 3, and 4, respectively. The mainframe chassis configuration is illustrated in Figure 1-2.

CENTRAL PROCESSORS AND LARGE CORE MEMORY

Each central processor consists of a CPU, nine functional units, an MUX, and an SCM. The central processors share a single LCM.

Computation is performed by the functional units. Data moves into and out of the functional units through the operating registers (A, B, and X) in the CPU.

The central processor uses three types of memory arranged in a hierarchy of speed and size.

1. The IWS contains 12 60-bit words for issuing of instructions. This register memory is part of the CPU and holds instruction words previously read from SCM. Program loops can be held in the IWS, thereby avoiding memory references.
2. The SCM contains 32,768 or 65,536 60-bit words arranged in 16 or 32 banks of 2048 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 10 memory stacks. Each stack contains 1024 12-bit words plus one parity bit per 12-bit word (odd parity). These stacks are identical to the PPU memory stacks. Either instructions or data may be held in SCM.
3. The LCM is shared by the central processors and 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 also be accessed. Instructions cannot be sent directly from LCM to the IWS.

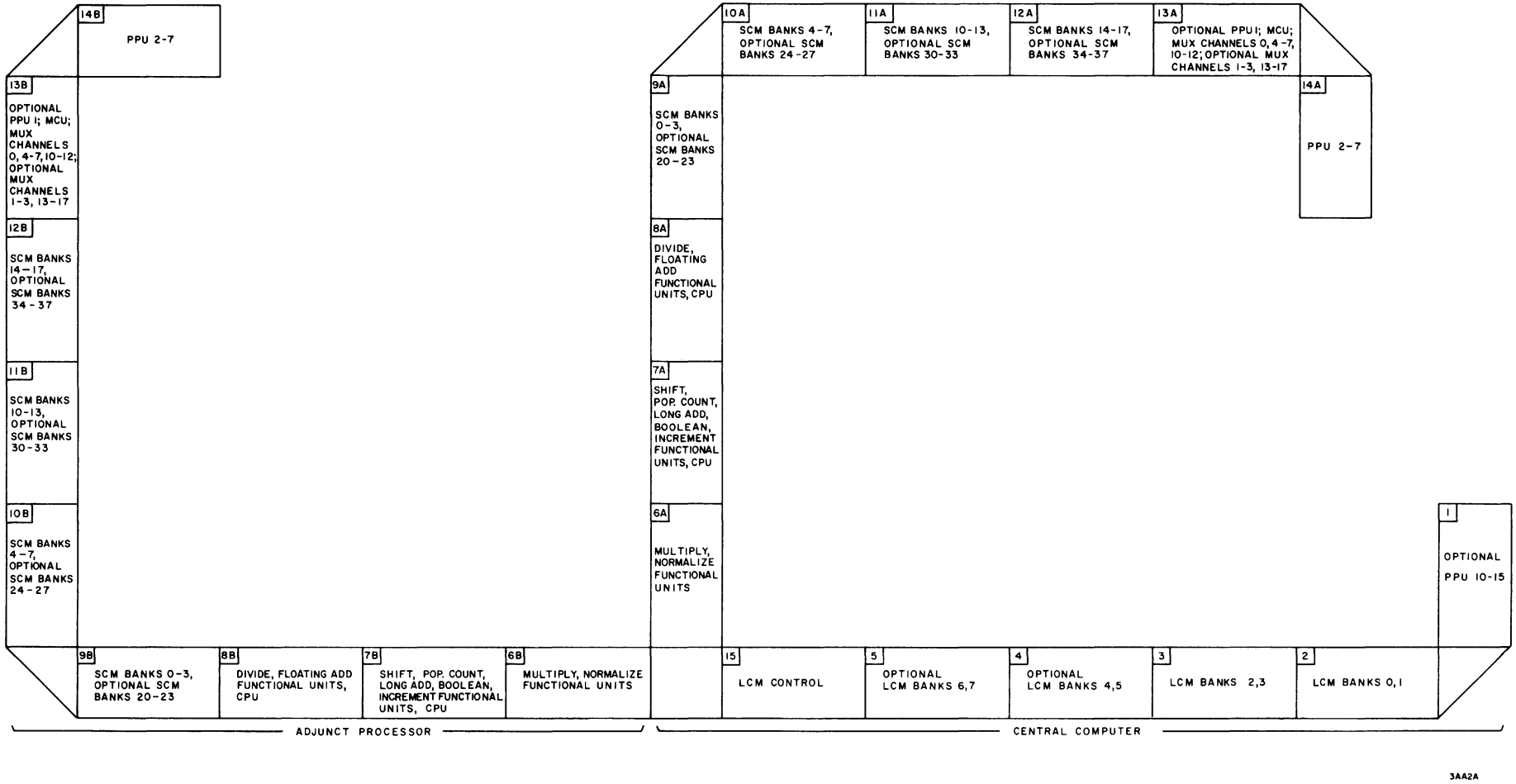


Figure 1-2. Mainframe Chassis Configuration

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 PPUs. Central processor object programs do not have access to these areas. The remainder of SCM may be divided between fields of program code and fields of data for the currently executing program. A small portion may contain a resident monitor program.

The MUX includes the mechanism to buffer data to (or from) PPUs that are directly connected to the central processor. The PPUs communicate with the central processor over 12-bit bidirectional MUX channels. In the basic system, there are eight channels, one of which is reserved for use by the MCU. Each channel has assembly/disassembly registers to convert 12-bit PPU words to 60-bit central processor 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 central processor for monitor action on the SCM buffer data. 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. Each normal channel has an SCM buffer area for incoming data and a separate buffer area for outgoing data. The high-speed channels share buffer areas. 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.

PERIPHERAL PROCESSOR UNITS

The PPUs are separate and independent computers, some of which reside in the mainframe. Others may be remotely located. A PPU may be connected to the MUX, another PPU, a peripheral device, or a combination of these. PPUs that connect directly to the MUX, whether in the mainframe or remotely located, are termed first level PPUs. Each PPU has a computation section that performs binary computation in fixed point arithmetic. A PPU memory provides storage for 4096 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. This type of stack is also used in 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 high-speed communication with a variety of I/O devices. The bidirectional channels allow additional PPUs to be added to the system by linking PPU to PPU.

MAINTENANCE CONTROL UNITS

The MCUs are mainframe PPUs with specially connected I/O channels. They are capable of dead starting the first level PPUs. An MCU can reference any part of SCM by specifying the SCM address. It can dead start the central processor 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.

OPERATOR STATIONS

The operator stations (Figure 1-3) are self-contained data processing systems that serve primarily as I/O processors for the central processors. An operator station is composed of the following elements.

PPUs (six)

Display console

Disk drive

Card reader

Card punch

Printers (two)

Magnetic tape units (two)

One of the six PPUs in an operator station usually connects to the central processor through a MUX channel. An alternate method is for an operator station to connect to the central processor through a first level PPU residing in the mainframe.

The maximum cable length between the mainframe and an external first level PPU, or between a mainframe first level PPU and an operator station, is 200 feet. Communication is over 12-bit bidirectional 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 PPUs in the station. The system also performs its own error detection and dead dump procedures.

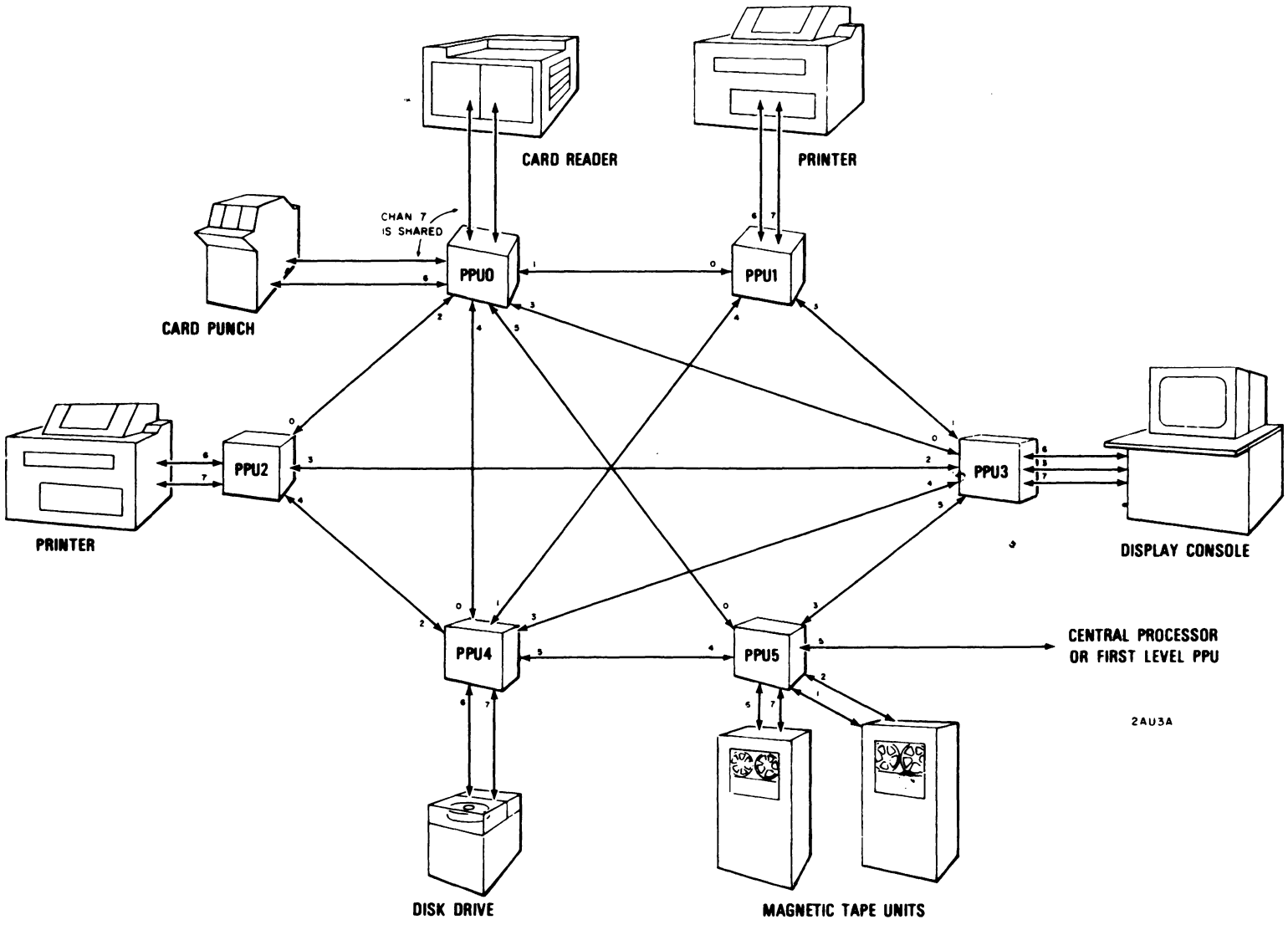


Figure 1-3. Operator Station

POWER DISTRIBUTION UNITS

Two PDUs distribute 400-Hz power to the dc power supplies located in the mainframe and in operator stations adjacent to the mainframe. Each PDU also contains a warning system that monitors logic chassis temperature, room dew point temperature, and condensing unit condition. A warning panel in the PDU contains relay circuits that activate a horn and automatically shut off computer power when cooling system malfunctions occur.

CONDENSING UNITS

Four condensing units, each with a capacity rating of 10 tons, provide cooling for the mainframe and the operator stations adjacent to the mainframe. Any remotely located operator station is cooled by a condensing unit with a capacity of 2 tons. The condensing units cool by pumping refrigerant through cold plates in each chassis.

SYSTEM COMMUNICATION

System communication paths are illustrated in Figure 1-1. All input data enters and leaves the system via peripheral equipment. The first level PPUs gather input data from the peripheral equipment for delivery to a central processor and distribute processed data to the output devices. Communication between a 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 first level PPU and a central processor is over a channel identical to that used for communication between the PPU and peripheral equipment. All first level PPUs may be in operation at the same time. Data may be sent to or from a central processor 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 program that can empty or fill the buffer area some 50 times faster than a PPU can fill or empty it. For example, a 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 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 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.

CENTRAL PROCESSING UNIT

Each CPU includes all central processor logic circuits not contained within the functional units, MUX, and SCM. It includes the registers and control logic to direct the arithmetic operations and provide interface between the functional units, SCM, and LCM. In addition to instruction execution, the CPU performs instruction fetching, address preparation, memory protection, and data fetching and storing. Figure 2-1 illustrates the general flow of information.

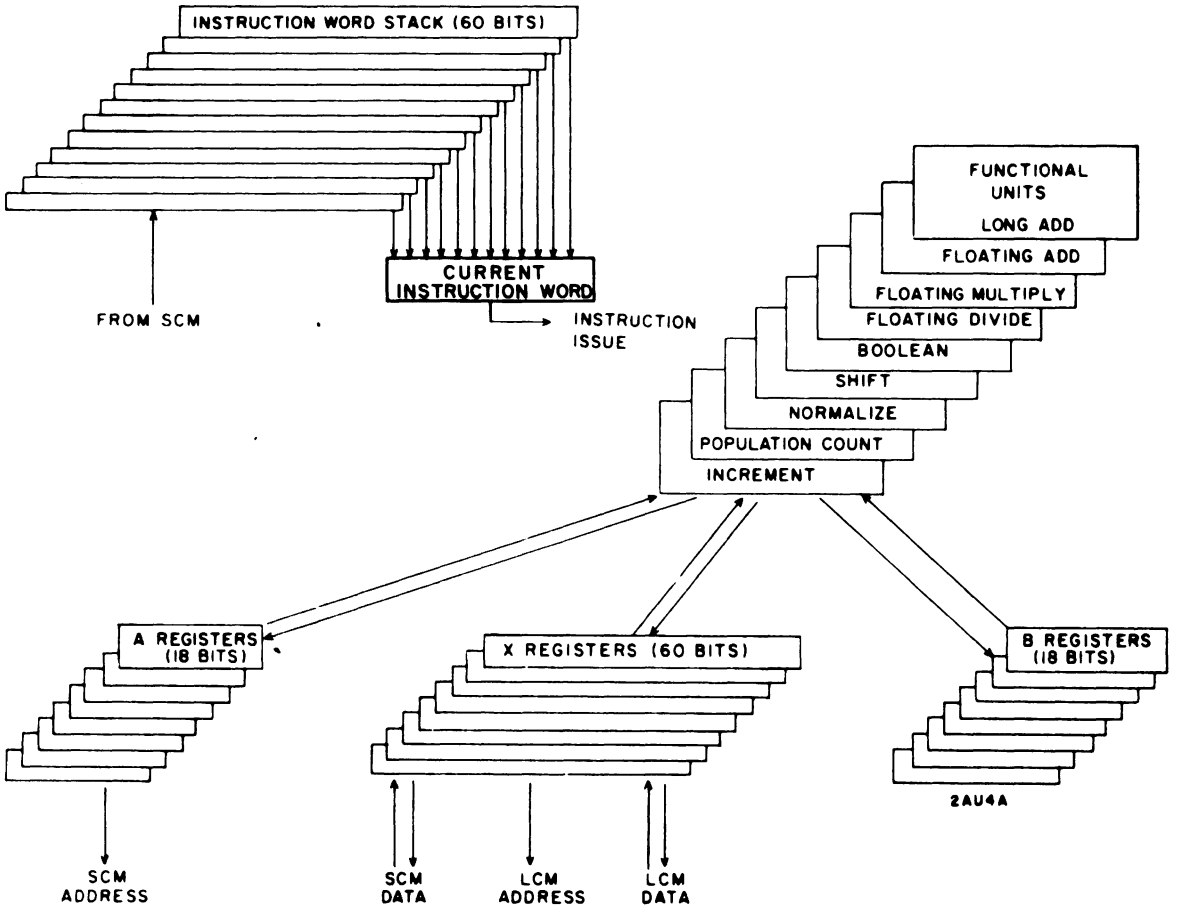


Figure 2-1. CPU Information Flow

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 reentry.

The CPU reads 60-bit words from SCM and enters them in the IWS which is capable of holding up to 12 60-bit words. Each instruction word in turn leaves the IWS and enters the current instruction word (CIW) register for interpretation and testing. The CIW 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 CIW register to one of nine functional units. The functional units obtain the instruction operands from and store results in 24 operating registers. Reservation control keeps an account of active operating registers to avoid conflicts.

OPERATING REGISTERS

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

Function	Identity	Length
Operand Registers	X0 through X7	60 Bits
Address Registers	A0 through A7	18 Bits
Index Registers	B0 through B7	18 Bits

X REGISTERS

There are eight 60-bit operand (X) registers in the CPU. These registers (X0, X1,..., X7) are the principal data handling registers for computation. Data flows from these registers to SCM and 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 address (A) registers in 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 through A5

causes an immediate SCM read reference to that relative address and sends the SCM word to the corresponding operand register X1 through 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 index (B) registers in 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 but cannot hold results from instructions.

INSTRUCTION REGISTERS

INSTRUCTION WORD STACK

The IWS is a group of twelve 60-bit registers that hold program instruction words for execution. It is essentially a moving window in the program code. The IWS 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 IWS. When this happens, the loop may be executed repeatedly without further references to SCM.

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. If the IWS contains sequential program instruction words, the contents of the rank one register corresponds with the lowest storage address in the instruction address stack.

INSTRUCTION ADDRESS STACK

A group of twelve 18-bit address registers are associated with the IWS. 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 IWS. The rank one register contains the SCM address from which the word in rank one of the IWS was read. All ranks are compared with the current program address. If coincidence occurs for any rank, the corresponding rank in the IWS is sent to the CIW register.

CIW REGISTER

The CIW register is divided into four 15-bit parcels. All four parcels are loaded when an instruction word is read from the IWS. An instruction issues from the CIW register when conditions in the functional units and operating registers are such that the instruction will be executed without conflicting with previously issued instructions. The other parcels are then shifted left in the CIW register by either 15 or 30 bits, depending upon the instruction format.

P REGISTER

The 18-bit program address (P) register contains the current program execution address. It serves as a program address counter and holds the relative SCM address for each program step. (Refer to appendix B for related information.) 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 CIW 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.

PSD REGISTER

The program status designator (PSD) register 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 illustrated in Figure 2-2.

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.

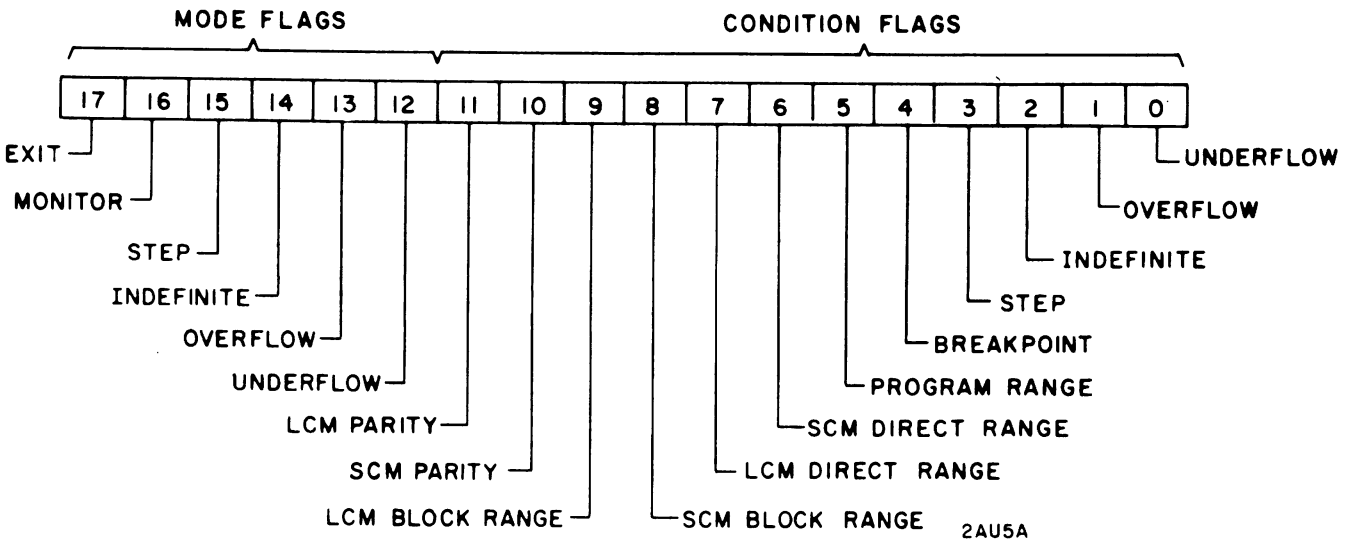


Figure 2-2. PSD Register Arrangement

MODE FLAGS

The upper six bits (12 through 17) in the PSD register are mode flags. These flags remain unaltered throughout the execution interval for the exchange package.

EXIT MODE FLAG (BIT 17)

This flag determines the initial SCM address for the exchange package during execution of an exchange exit (013) instruction. If this flag is set, the exchange package address is K + (Bj) + (RAS). If clear, the address is (NEA). This flag also controls execution of the 014 special LCM functions instruction. It can be executed if this flag or the monitor mode flag is set. If both flags are clear, the instruction is executed as a pass instruction.

MONITOR MODE FLAG (BIT 16)

This flag determines when an I/O interrupt request is honored. If this flag is set, the current program continues until execution is complete. If an I/O interrupt request occurs during this time, it is not honored until the end of the current program. If the flag is clear, an I/O interrupt is honored immediately.

The monitor mode flag also controls execution of the 0160 and 0170 reset buffer instructions and the 014 special LCM functions instruction. If this flag is set, the instructions are executed as described. If clear, the 0160 and 0170 instructions are executed as pass instructions. If both this flag and the exit mode flag are clear, the 014 special LCM functions instruction is executed as a pass instruction. **(Refer to appendix B for** related information.)

STEP MODE FLAG (BIT 15)

If this flag is set and the first instruction of the current exchange interval has issued, the step condition flag (bit 3) sets. The step condition flag then terminates the execution interval after the last instruction in the CIW register has issued.

INDEFINITE MODE FLAG (BIT 14)

This flag enables interruption of the current exchange interval when an indefinite floating point result occurs. If this flag and the indefinite condition flag (bit 2) are both set, the execution interval is terminated after the last instruction in the CIW register has issued.

OVERFLOW MODE FLAG (BIT 13)

This flag enables interruption of the current exchange interval when an overflow of the floating point range occurs. If this flag and the overflow condition flag (bit 1) are both set, the execution interval is terminated after the last instruction in the CIW register has issued.

UNDERFLOW MODE FLAG (BIT 12)

This flag enables interruption of the current exchange interval when an underflow of the floating point range occurs. If this flag and the underflow condition flag (bit 0) are both set, the execution interval is terminated after the last instruction in the CIW register has issued.

CONDITION FLAGS

The lower 12 bits (0 through 11) in the PSD register are condition flags. These flags may be set from the exchange package or from conditions that may occur during the execution interval. When this occurs, the execution interval for the exchange package is terminated after the last instruction in the CIW register has issued. (Refer to appendix B for related information.)

LCM PARITY CONDITION FLAG (BIT 11)

This flag sets when a parity error exists in a word read from LCM.

SCM PARITY CONDITION FLAG (BIT 10)

This flag sets when a parity error exists in a word read from SCM.

LCM BLOCK RANGE CONDITION FLAG (BIT 9)

This flag sets when a block copy instruction is issued which would cause an LCM reference to an address which equals or exceeds (FLL). If an LCM timeout error occurs, this flag sets in the CPU which caused the error.

SCM BLOCK RANGE CONDITION FLAG (BIT 8)

This flag sets when a block copy instruction is issued which would cause an SCM reference to an address which equals or exceeds (FLS).

LCM DIRECT RANGE CONDITION FLAG (BIT 7)

This flag sets when a direct read or write LCM instruction is issued which would cause an LCM reference to an address which equals or exceeds (FLL). If an LCM timeout error occurs, this flag sets in the CPU which caused the error.

SCM DIRECT RANGE CONDITION FLAG (BIT 6)

This flag sets whenever an increment instruction (50 through 57) is issued which causes an SCM reference to an address equal to or greater than (FLS), or whenever (P) is equal to or greater than (FLS).

PROGRAM RANGE CONDITION FLAG (BIT 5)

This flag sets whenever (P) equals zero or an error exit instruction (00) is issued. When this flag is set by a 00 instruction, execution terminates immediately.

BREAKPOINT CONDITION FLAG (BIT 4)

This flag sets whenever (P) equals (BPA).

STEP CONDITION FLAG (BIT 3)

This flag sets whenever the step mode flag (bit 15) is set and a new word has entered the CIW register.

INDEFINITE CONDITION FLAG (BIT 2)

This flag sets when one of the floating point functional units generates an indefinite result. The execution interval is not terminated unless the indefinite mode flag (bit 14) is also set.

OVERFLOW CONDITION FLAG (BIT 1)

This flag sets when an overflow of the floating point range occurs in the result from a functional unit. The execution interval is not terminated unless the overflow mode flag (bit 13) is also set.

UNDERFLOW CONDITION FLAG (BIT 0)

This flag sets when an underflow of the floating point range occurs in the result from a functional unit. The execution interval is not terminated unless the underflow mode flag (bit 12) is also set.

SUPPORT REGISTERS

The support registers are used to assist the operating registers during the execution of programs. These registers are loaded from SCM during the execution of an exchange sequence. The information is not altered during the execution interval for an exchange package. When the execution interval has been completed, the data in these registers is sent back to SCM.

RAS REGISTER

The 18-bit reference address-SCM (RAS) register is loaded from SCM during the second word of an exchange sequence. An absolute SCM address is formed by adding (RAS) to the relative address which is determined by the instruction. SCM references from the MUX are absolute addresses. Therefore, they are not added to (RAS).

FLS REGISTER

The 18-bit field length-SCM (FLS) register is loaded from SCM during the third word of an exchange sequence. Relative SCM addresses are compared with (FLS). If a relative SCM address equals or exceeds (FLS), the SCM block range or SCM direct range condition flag sets in the PSD register.

RAL REGISTER

The 24-bit reference address-LCM (RAL) register is loaded from SCM during the fifth word of an exchange sequence. An absolute LCM address is formed by adding (RAL) to the relative address which is determined by the instruction.

FLL REGISTER

The 24-bit field length-LCM (FLL) register is loaded from SCM during the sixth word of an exchange sequence. Relative LCM addresses are compared with (FLL). If a relative LCM address equals or exceeds (FLL), the LCM block range or LCM direct range condition flag sets in the PSD register.

NEA REGISTER

The 24-bit normal exit address (NEA) register is loaded from SCM during the seventh word of an exchange sequence. This register is used during an exchange exit (013) instruction when the exit mode flag in the PSD register is clear. When this occurs, the current program is terminated with an exchange sequence. The absolute SCM address for the new exchange package is contained in the NEA register.

EEA REGISTER

The 24-bit error exit address (EEA) register is loaded from SCM during the eighth word of an exchange sequence. This register is used whenever an error exit occurs during the execution interval for an exchange package. When this occurs, (EEA) comprises the absolute address in SCM for the exchange package that terminates the program.

BPA REGISTER

The 18-bit breakpoint address (BPA) register is loaded from SCM during the first word of an exchange sequence. This register allows a program to be run in small sections. When (BPA) and (P) are equal, the breakpoint condition flag is set in the PSD register. This terminates the execution interval for the exchange package after the last instruction in the CIW register has issued.

FUNCTIONAL UNITS

There are nine functional units in the central processor. Each is a specialized arithmetic unit with algorithms for a portion of the central processor instructions. Each unit is independent of the other units, and a number of functional units may be in operation

at the same time. There are no visible registers in the functional units from a programming standpoint. A functional unit receives one or two operands from operating registers at the beginning of instruction execution and delivers the result to the operating registers when the function has been performed. There is no information retained in a functional unit for reference in subsequent instructions. These units operate essentially in a three-address mode, with limited source and destination addressing.

All functional units, with the exception of the floating multiply and divide units, have 1-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 every clock period. Therefore, it is 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 1 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 unit.

The floating multiply unit has 2-clock-period segmentation. Operands may enter the multiply unit in any clock period providing there was no operation initiated in the preceding clock period. There is a 1-clock-period delay in initiating a multiply instruction if another multiply instruction has just been started.

The floating divide unit is the only functional unit in which an iterative algorithm is executed. There is essentially no segmentation possible in this unit although the beginning of a new operation can overlap the completion of the previous operation by 2 clock periods.

A brief description of the operations performed by the functional units is provided in the following paragraphs. Refer to the description of central processor instructions for details.

BOOLEAN UNIT

The boolean unit executes instructions 10 through 17, 26, and 27. These instructions require bit-by-bit data manipulation. These include both the logical operations (11 through 13 and 15 through 17) and the transmissive operations (10, 14, 26, and 27).

SHIFT UNIT

The shift unit executes instructions 20 through 23 and 43. These instructions require shifting the entire 60-bit field of data within the operand word.

NORMALIZE UNIT

The normalize unit executes instructions 24 and 25. These instructions increase the coefficient portion of a floating point operand to the largest possible value while reducing the exponent by a corresponding amount.

FLOATING ADD UNIT

The floating add unit executes instructions 30 through 35. These instructions require single or double precision addition or subtraction of operands in floating point format.

LONG ADD UNIT

The long add unit executes instructions 36 and 37. These instructions require integer addition or subtraction of operands in fixed point format.

FLOATING MULTIPLY UNIT

The floating multiply unit executes instructions 40 through 42. These instructions require single or double precision multiplication of operands in floating point format.

FLOATING DIVIDE UNIT

The floating divide unit executes instructions 44 and 45. These instructions require single precision division of operands in floating point format.

POPULATION COUNT UNIT

The population count unit executes instruction 47. This instruction counts the number of bits in a 60-bit operand which have a value of one.

INCREMENT UNIT

The increment unit executes instructions 50 through 77. These instructions require arithmetic operations on two selected 18-bit operands. During 50 through 57 instructions, the result is transmitted to an A register. Also, this result plus (RAS) is sent to SCM when the A1 through A7 registers are used. During 60 through 67 instructions, the result is transmitted to a B register. During 70 through 77 instructions, the result is transmitted to an X register.

INPUT/OUTPUT MULTIPLEXER

The MUX supervises the transfer of data to or from the SCM and the directly connected PPUs. The PPUs communicate with the SCM over 12-bit bidirectional channels. The MUX has a channel control for each channel. It translates I/O control signals going to or coming from a PPU. The channel control unit also includes assembly and disassembly registers for converting 12-bit PPU words to 60-bit central processor words and vice versa.

There are 8, 12, or 16 channels in the MUX. Fifteen channels, numbered 1 through 17 (octal), can connect to PPUs. A sixteenth channel, channel 0, connects to the MCU but operates differently. It is not included in the following description, but is included in the PPU description.

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 MUX channel has an SCM buffer area which is reserved for I/O data. Each buffer is divided into two fields, a lower field and an upper field. Data is entered (or removed) in the buffer area in a circular mode. That is, 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 program to process the buffer data. The PPU continues to fill (or empty) the second buffer field while the data in the first buffer field is being processed. The buffer areas in SCM are illustrated in Figure 2-3. These locations are fixed and cannot be changed unless a wiring modification is made. Note that the high-speed channels share common buffer areas.

The basic channel configuration includes high-speed channels 4 through 7 and normal channels 10 through 12. If both optional MUX increments are installed, the system includes high-speed channels 2 through 7 and normal channels 1 and 10 through 17. High-speed channels transfer data at approximately twice the speed of normal channels. Also, the high-speed channel buffer areas are twice as large as the normal channel buffer areas.

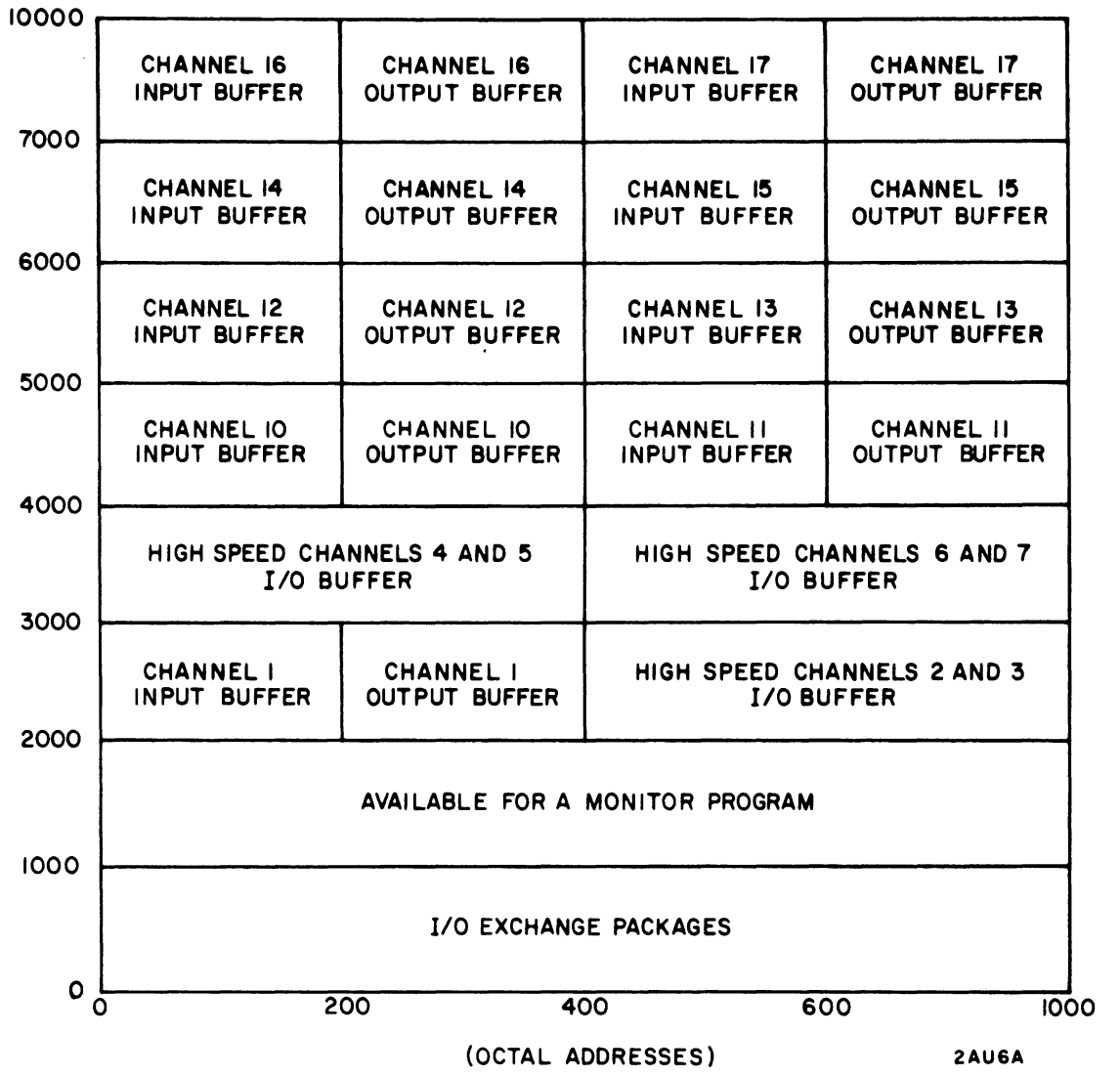


Figure 2-3. I/O Buffer Areas in SCM

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 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 SCM is ready to receive data. It does this by transmitting a message to the PPU over the associated MUX 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 input channel control for this channel in the MUX. The MUX enters the first word in the upper 12 bits of the 60-bit assembly register. Then the MUX 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 MUX does not accept the next 12-bit word from the PPU 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 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 fill half of its assigned SCM buffer area, the MUX sends an interrupt request to the CPU. When this is accepted by the CPU, an exchange jump is initiated to a program that processes the data in the first half of the buffer. Meanwhile, the PPU continues to transmit 12-bit words, which the MUX assembles into 60-bit words and stores in the upper half of the buffer. When the upper half of the buffer becomes full, the MUX sends another interrupt request to the CPU, provided that the program from the first interrupt has completed processing the lower half of the buffer and has performed an exchange exit. Otherwise, the interrupt request is not sent to the CPU and further input from the PPU is locked out until the exchange exit is executed.

NOTE

If an error condition occurs which causes the I/O program to exit to an error handling routine at EEA, the error routine may, in returning to the I/O program, inadvertently release the interrupt lockout condition prematurely by performing an exchange exit instruction (013). To prevent this situation from occurring, bit 17 of EEA can be set in the incoming exchange package. This bit is sent to exchange jump control and blocks an 013 instruction from releasing any I/O interrupt request flags that might be set.

When the interrupt request has been sent, the PPU begins to enter data into the lower half of the buffer while the data in the upper half is being processed. Thus, the buffer operates in a circular mode with interrupts at the center and 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 MUX. Before sending the record pulse, the PPU ensures that the last 12-bit word was accepted by the MUX. (If the PPU output word flag is clear, the MUX has accepted the last word.) Upon receipt of the record pulse, the MUX sends an 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 handles this request the same as an interrupt request caused by a threshold condition. The resulting 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 I/O program processes the input data according to the situation sensed.

The PPU must not begin transmitting a new record of input data until the data in the buffer has been processed. 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. If the PPU proceeds before the CPU has reset the input buffer, the incoming data for the new record may be partially lost. The incoming record continues to be input with no indication of error except that the record is 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 I/O program has already loaded the output buffer with some data. The output sequence begins with a reset output buffer instruction that sets the output buffer

address to zero and sends a word request to SCM to read the first word from the output buffer to the 60-bit disassembly register in the MUX. When SCM delivers the 60-bit word to the disassembly register, the output channel control for this channel clears the disassembly counter and outputs a record pulse and a word pulse to the PPU to indicate that transmission of a new record is starting.

The upper 12 bits of the data in the disassembly register are placed on the input 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 MUX. The MUX output data remains on the PPU input channel until the PPU accepts it.

When the resume pulse arrives from the PPU, the 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 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 MUX. At this time, the MUX sends another 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 emptied half of its assigned buffer area, the MUX sends an interrupt request to the CPU. When this is accepted by the CPU, an exchange jump is initiated to the program that refills the portion of the buffer that has just been emptied. This operation is similar to that performed for a PPU to SCM transfer. 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 becomes empty, the MUX sends another interrupt request to the CPU provided that the program from the first interrupt has completed processing the lower half of the buffer and has performed an exchange exit. Otherwise, the interrupt request is not sent to the CPU and further output to the PPU is locked out until the exchange exit is executed.

NOTE

If an error condition occurs which causes the I/O program to exit to an error handling routine at EEA, the error routine may, in returning to the I/O program, inadvertently release the interrupt lockout condition prematurely by performing an exchange exit instruction (013). To prevent this situation from occurring, bit 17 of EEA can be set on the incoming exchange package. This bit is sent to exchange jump control and blocks an 013 instruction from releasing any I/O interrupt request flags that might be set.

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 on its input channel. This stops further transmission by the 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 MUX.

The sequence begins with a 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 input channel control for this channel in the MUX. The MUX enters the 12-bit word in the upper 12 bits of the 60-bit assembly register.

A static high-speed resume signal is sent to the PPU during this time. This clears the output 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 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 if one arrives before the request for 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.

Interrupt requests resulting from reaching a buffer threshold or receiving a 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 MUX. Also, the output data path includes a series of three output data buffer registers. The output channel control also controls the flow of data from the disassembly register through these buffer registers to the PPU. The three buffer registers are designated ranks A, B, and C.

The output sequence begins with a reset output buffer instruction that sets the output buffer address to the starting address of the buffer. At this time, the MUX also sends a word request to SCM to read the first word from the buffer to the 60-bit disassembly register. When the 60-bit word has been delivered to the disassembly register, the MUX clears the disassembly counter and sends a word pulse and a record pulse to the high-speed control. Concurrently, the upper 12-bit word in the disassembly register is transmitted to rank A of the buffer registers.

Upon receipt of the record pulse from the output channel control, the high-speed buffer control transmits a record pulse to the PPU. This sets the input record flag in the PPU.

Upon receipt of the word pulse from the output channel control, the high-speed buffer control enters the 12-bit word from the disassembly register into rank A. Then it outputs the word pulse to the PPU where it sets the input word flag. The word pulse is not sent to the PPU if an interrupt lockout condition exists in the output channel control.

In consecutive clock periods, the data moves from rank A to rank B to rank C of the buffer registers. The data in rank C is transmitted to the PPU and remains in the data path until the PPU transmits a resume pulse to the high-speed control.

The high-speed control does not wait for the resume pulse from the PPU before sending a resume pulse to the output channel control. The output channel control increments the disassembly count and transmits the second 12-bit word to rank A. At this time, the output channel control advances the address register to the next address in the SCM buffer and sends a word pulse to the high-speed control. Upon receipt of this second word pulse, rank A is entered with the second 12-bit word and the resume pulse is again sent to the output channel control. In the following clock period, the data in rank A moves into rank B.

The process is repeated for the third 12-bit word. However, when the output channel control sends the third word to rank A, the resume pulse is not sent back to the output channel control.

At this point all action stops until the PPU accepts the first 12-bit word and transmits a resume pulse to the high-speed buffer control. When a resume pulse arrives from the PPU, rank C is cleared and entered with the second 12-bit word in rank B. A resume pulse is then sent to the output channel control.

The sequence continues until the fifth 12-bit word has been sent to rank A by the output channel control. At this time, the output channel control sends another word request to SCM for the next 60-bit word in the buffer.

At the time the output word request flag sets, the last two 12-bit words are in ranks B and C. The PPU accepts the fourth word and transmits a resume pulse to the high-speed buffer control. Rank C is then cleared and entered with the fifth word from rank B. When this data has been delivered to the PPU, action halts until the requested word is delivered to the disassembly register from SCM.

Some 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.

Interrupt requests resulting from reaching a buffer threshold are the same as for the normal SCM to PPU data transfer.

SMALL CORE MEMORY

SCM is a 16- or 32-bank coincident current type memory with a capacity of 32,768 or 65,536 65-bit words (including five odd parity bits per word). Each bank is independent of the other banks. Sequentially addressed words reside in sequential banks. This allows a maximum data transfer rate of one word each clock period. Each bank has a 4-clock-period access time from arrival of the address to readout of the word. The total read/write cycle time for a bank is 10 clock periods. Therefore, it is possible for a maximum of 10 banks to be in operation at one time. This occurs during block copy instructions between SCM and LCM in which sequentially addressed words cannot cause SCM conflicts. In random addressing of SCM, an average of four banks in operation at one time is normal.

ADDRESS FORMAT

The location of each word in SCM is identified by a 16-bit address. The address format is illustrated in Figure 2-4. 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. Numerically consecutive addresses reference consecutive banks. This is the most efficient use of bank phasing.

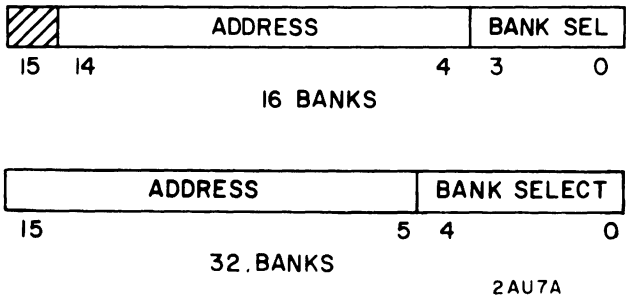


Figure 2-4. SCM Address Format

PARITY CHECKING

Parity is checked during each SCM read/write cycle. When a parity error is detected, the SCM parity condition flag is set in the PSD register. Also, the 16-bit address of the failing word and five section parity error bits are sent to the MCU. (Refer to appendix B for related information.)
If a parity error occurs during a block copy operation, the block copy is executed to completion. 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. This prevents stack

damage due to overheating. The duty cycle integrator remains active for 1 millisecond after all references to a bank cease.

The duty cycle integrator is activated by approximately 800 repeated references to a stack more frequently than once every 20 clock periods. 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. The following are examples of such programs.

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 IWS.

   A suggested way of doing this type of program (which 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, 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 IWS using the X and B registers. This prevents a situation in which one memory bank receives an extraordinary number of consecutive references from the CPU.

MEMORY PROTECTION

All central processor references to SCM are made relative to a reference address (Figure 2-5). The reference address in SCM (RAS) defines the lower limit of the program and/or data. The upper limit is defined by the program field length added to the RAS. The field length in SCM (FLS) is the number of 60-bit words comprising the field. It is established by the operating system prior to program execution. All references to SCM from the current program must lie within this field.

During an exchange jump, the RAS and FLS are loaded from the exchange package into the RAS and FLS registers to define the limits of the field.

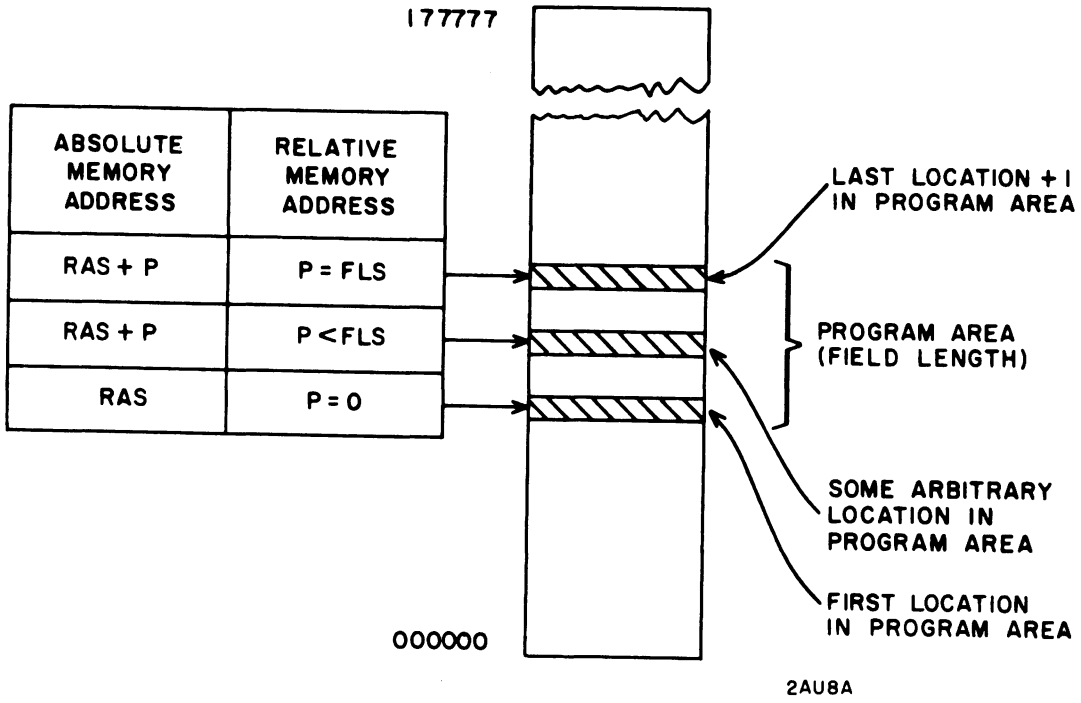


Figure 2-5. SCM Memory Map

When the program specifies a read or write address, it is automatically checked to see if it lies within the limits of the field. If it does, the program proceeds normally; if it does not, an unconditional exit is made and the program is terminated.

The PSD register contains two flags which indicate SCM status. The direct range condition flag sets if a single requested address is outside the limits of the field. A block range condition flag sets if a block transfer between SCM and LCM causes a reference to an address in SCM which is outside the limits of the field.

MEMORY REFERENCES

When an SCM storage reference is initiated, the address is sent to all banks in the memory. Only the selected 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 is also held in the SAS. Thus, requests for three

addresses may be waiting 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).

NOTE

It is possible to abort a valid SCM memory write when it is followed by an SCM out-of-range write. The following sequence of instructions could produce this situation: write SCM bank X, write SCM bank X, and write SCM out of range. The first valid write is 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 causes the SCM direct range condition flag to be set in the PSD register before the second write to bank X can be initiated. Since this flag stops any write into SCM, both the second write to bank X (which is valid) and the out-of-range write are aborted.

All addresses presented to SCM are processed in the order in which they are received. SCM requests received simultaneously are given a priority that determines which address will be allowed access first. These priorities are:

1. Exchange sequence request
2. Increment unit request
3. Return jump exit request
4. I/O request
5. Instruction fetch 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.

MEMORY ACCESS

SCM transmits data to and receives data from the PPU, CPU, LCM, and MCU.

PPU ACCESS

PPU access is limited to certain buffer areas in the lower order addresses of SCM (Figure 2-3). These areas are used for data transfers and for PPU-CPU communication. The PPUs can reference the I/O buffer areas at any time, but to avoid loss of data, they should do so only when directed by the CPU program.

As the PPUs 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 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 illustrated in Figure 2-6.

The MUX establishes I/O priorities for the PPUs, assembles incoming 12-bit PPU words into 60-bit SCM words, and disassembles the 60-bit outgoing SCM words into 12-bit PPU words.

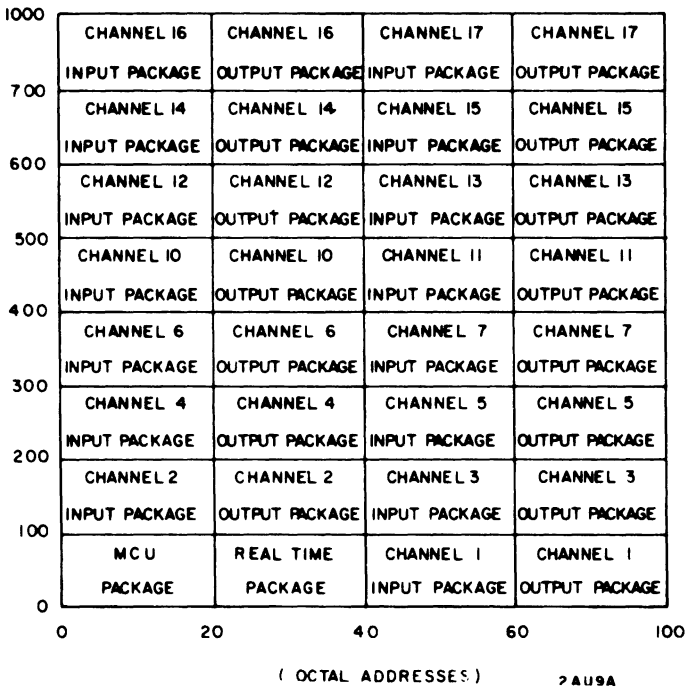


Figure 2-6. I/O Exchange Package Areas in SCM

CPU ACCESS

Increment unit instructions 51 through 57 are used by a CPU program for referencing single SCM words. The A registers used by these instructions are divided into five read address registers (A1 through A5) and two write address registers (A6 through 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 is stored in SCM. If the A register is A1 through A5, the corresponding X register is loaded with the contents of that SCM location. If the referenced SCM address is outside the field limits of the currently executing CPU program, the SCM direct range condition flag is set in the PSD register.

The CPU also references SCM when the IWS requires another SCM word because the stack advanced or because a branch was out of the stack. The word is read from SCM to the IWS even though it might be outside the field limits of the currently executing program. However, it sets the SCM direct range condition 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 limits of the currently executing program, no 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 limits of the CPU program initiating the transfer. If not, the SCM block range condition flag is set in the PSD register.

MCU ACCESS

The 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 the MUX.

LARGE CORE MEMORY 3

Large core memory (LCM) is a four- or eight-bank linear select type memory with a capacity of 256,000 or 512,000 64-bit words (including four odd parity bits per word). Each bank is independent of the other banks. A storage reference to a bank results in a read/write cycle which requires 64 clock periods to complete. Each bank has a 20-clock period read access time (measured from the time a processor has gained LCM access to the arrival of data at its destination). Eight 64-bit words are simultaneously read from or written into a bank memory stack. These words are held in a 512-bit bank operand register. A subsequent reference to one of these words can be made without the delay of another read/write cycle. Maximum data transfer rate is one word each clock period. This occurs during block copies between SCM and LCM. Two central processors share LCM. A third access is provided for future expansion.

ADDRESS FORMAT

The location of each word in LCM is identified by a 19-bit address. The address format is illustrated in Figure 3-1. 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.

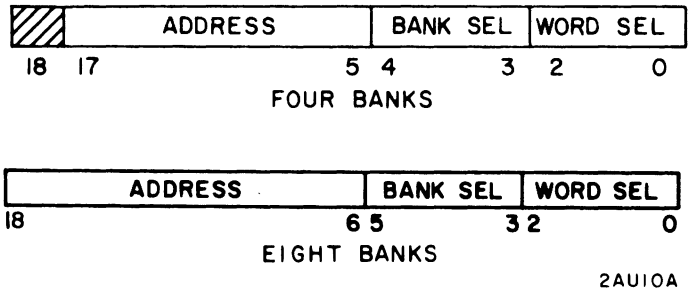


Figure 3-1. LCM Address Format

PARITY CHECKING

Parity is checked every time a 60-bit word is read from LCM. When an LCM parity error is detected, the LCM parity condition flag is set in the PSD register. The 19-bit address of the failing word and four section parity error bits are sent to the MCU.

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

MEMORY PROTECTION

All references to LCM are made relative to a reference address. The reference address in LCM (RAL) defines the lower limit of the data. The upper limit is defined by the data field length added to the RAL. The field length in LCM (FLL) is the number of 60-bit words comprising the field. It is established by the operating system prior to program execution. All references to LCM from the current program must lie within this field.

During an exchange jump, the RAL and FLL are loaded from the exchange package into the RAL and FLL 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 limits of the field. If it does, the program proceeds normally; if it does not, an unconditional exit is made and the program is terminated.

The PSD register contains two flags which indicate LCM status. The LCM direct range condition flag sets if a single requested address is outside the limits of the field. The LCM block range condition flag sets if a block transfer between SCM and LCM causes a reference to an address in LCM which is outside the limits of the field. If a timeout error occurs, both flags are set in the CPU which caused the error.

MEMORY REFERENCES

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 word request for SCM occurs during this transfer, the data flow is interrupted for 1 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 1-clock-period delay. An additional delay occurs 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 upon 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 requires 20 clock periods to deliver a 60-bit word to the designated X register. A read LCM instruction for a word already residing in a bank operand register as a result of a previous instruction requires 6 clock periods to deliver the requested word to the designated X register. Thus, although the first 60-bit word requires 20 clock periods, the second through eighth words in the same LCM word require 6 clock periods each. This means that consecutive LCM operands are available, on an average, every 7.75 clock periods. All of these times increase if bank conflicts occur.

The execution time for writing a word into LCM from an X register normally requires 5 clock periods. A delay occurs 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 is held in the LCM write register until the LCM bank is free.

ACCESS CONTROL

Three accesses into LCM are provided. Two accesses are for the central processors and a third access is for future expansion. Only one of the accesses may be in operation at any one time. The access control ensures that this is always the case. The access control may be operated in either an unlocked mode or a locked mode.

UNLOCKED MODE

The unlocked mode is the normal mode of operation. In this mode, requests from the three accesses are handled on a first-come, first-served basis if LCM is idle. A priority network handles simultaneous requests. If a request is made while LCM is busy, the requesting central processor waits until LCM is free before the request is honored. If more than one access has requested LCM when LCM becomes free, the access with the highest priority is honored regardless of the order in which the requests are received.

LOCKED MODE

The locked mode is used by executive programs during periods of pointer manipulations where several separate pieces of LCM data must be read, changed, and rewritten. The other central processor(s) is prevented from accessing this data. A 1-bit locking register is associated with each access to provide a long-term locking mechanism. If one central processor executes an instruction (014) to set its locking register, the other central processor(s) is prevented from accessing LCM. No more than one locking register may be in the set state at any one time. If two central processors attempt to set their locking registers at the same time, the central processor with the highest priority sets its register. If a central processor that is locked out requests LCM access, it waits until the LCM access control is unlocked before the request is honored.

An 18-bit timeout counter ensures that no access can lock up LCM indefinitely. The counter resets to zero whenever one of the locking registers sets. It is then incremented every clock period that the locking register is set. If the count reaches 40,000 (octal) (450.56 microseconds), the locking register is forced clear. At this time, the LCM block range error and LCM direct range error condition flags in the PSD register are set in the CPU whose locking register caused the error. This causes the erring CPU to make an error exit. In addition, a timeout error signal is sent to the associated MCU.

FLAG REGISTER

A second locking facility provides very long-term locking of individual small areas in LCM. This is a 48-bit flag register, expandable to 96 bits, which can be read or modified from any of the three accesses. The setting of a bit in the flag register does not cause any physical change in LCM nor does it cause a hardware lock to occur. The significance of each bit is determined by software convention. Individual bits may be set or cleared using the 014 instruction.

PERIPHERAL PROCESSOR UNIT DESCRIPTION 4

Each PPU is a completely independent and self-contained computer. Therefore, each PPU may be executing a different program at the same time. The primary function of a PPU is to perform I/O tasks at the request of a central processor. Two of the PPUs are used as MCUs. They are identical to the other PPUs except they have specific, invariant channel connections. The MCUs are described later in this section.

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 A register. Arithmetic is binary in a ones complement mode. The program instructions make use of specially assigned locations in the lowest order 64 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 a central processor. In addition, the PPU may be programmed to perform the synchronizing function required in interfacing an electro-mechanical device to a central processor. 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, and so on.

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

The 18-bit 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

The 12-bit P register holds the address of the current instruction. During the execution of the current instruction, the contents of P is advanced by one or two 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

The 12-bit 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 six bits of an 18-bit operand in the lower six bits of the register during operand arithmetic.

X REGISTER

The 13-bit X 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

The 6-bit 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 contents 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

The 12-bit fd register holds the current instruction word for translation. The upper six bits are the f designator and the lower six bits are the d designator from the instruction.

k REGISTER

The 3-bit k register is the instruction cycle counter and is used to count the number of memory references required during execution.

MEMORY

Each processor has its own 12-bit, 4096-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. Consecutive addresses alternate between these banks to increase processing speed. The memory consists of four 12-bit memory modules, each 1024 words. Two of these modules form one memory bank. Associated with each bank is an S register which holds the address of the operand in storage, a Z register which holds operands to be stored, and the X register which receives operands read from either bank. There are, therefore, two Z and two S registers for each PPU. Associated with each Z register is a parity generating circuit that generates an odd parity bit that is stored in the memory with the operand. Parity is checked when operands are read from memory. In the event of a parity error, the PPU sends a parity error signal to the MCU.

INPUT/OUTPUT

A PPU communicates with a central processor and with other devices over bidirectional channels.

There are provisions for eight input and eight output cables in each PPU. Each cable provides 12 bits of incoming or outgoing data and the associated control lines for that data. The PPU may enter the data on any one of these eight input or output cables at any one time. Each path has two associated control lines carrying control information in the direction of data flow. These lines carry a word pulse to indicate passage of each 12-bit word of data and a record pulse to indicate the completion of a record of

data. Each path has one associated control line carrying control information against the direction of the data flow. This line carries a resume pulse to indicate receipt of a data word.

INPUT CHANNEL CONTROL

There are provisions for eight input cables in each PPU. Each input cable provides a path for 12 bits of incoming data and the associated control lines for that data. The PPU may accept the data on any one of these eight input cables at any one time. The Channel selection of the input channels, which are numbered 0 through 7, is determined by the lowest order three bits 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. (Refer to appendix B for related information.)

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 accepted 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 the PPU. The flag is cleared when the PPU has accepted the next following input data word and sends a resume pulse to the data transmitter at the other end of the cable. The 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 1 clock period when the PPU has accepted 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 the PPU over the input cable during the time in which this flag is set.

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. Channel selection of the output channels, which are numbered 0 through 7, is determined the lowest order three bits in the d portion of the fd register. Data is transmitted over the output cables 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. (Refer to appendix B for related information.)

OUTPUT WORD FLAG

The flag is set when the PPU transmits a 1-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 clear during a dead start condition. A PPU senses the status of this flag by executing I/O jump instruction 64 or 65.

OUTPUT RECORD FLAG

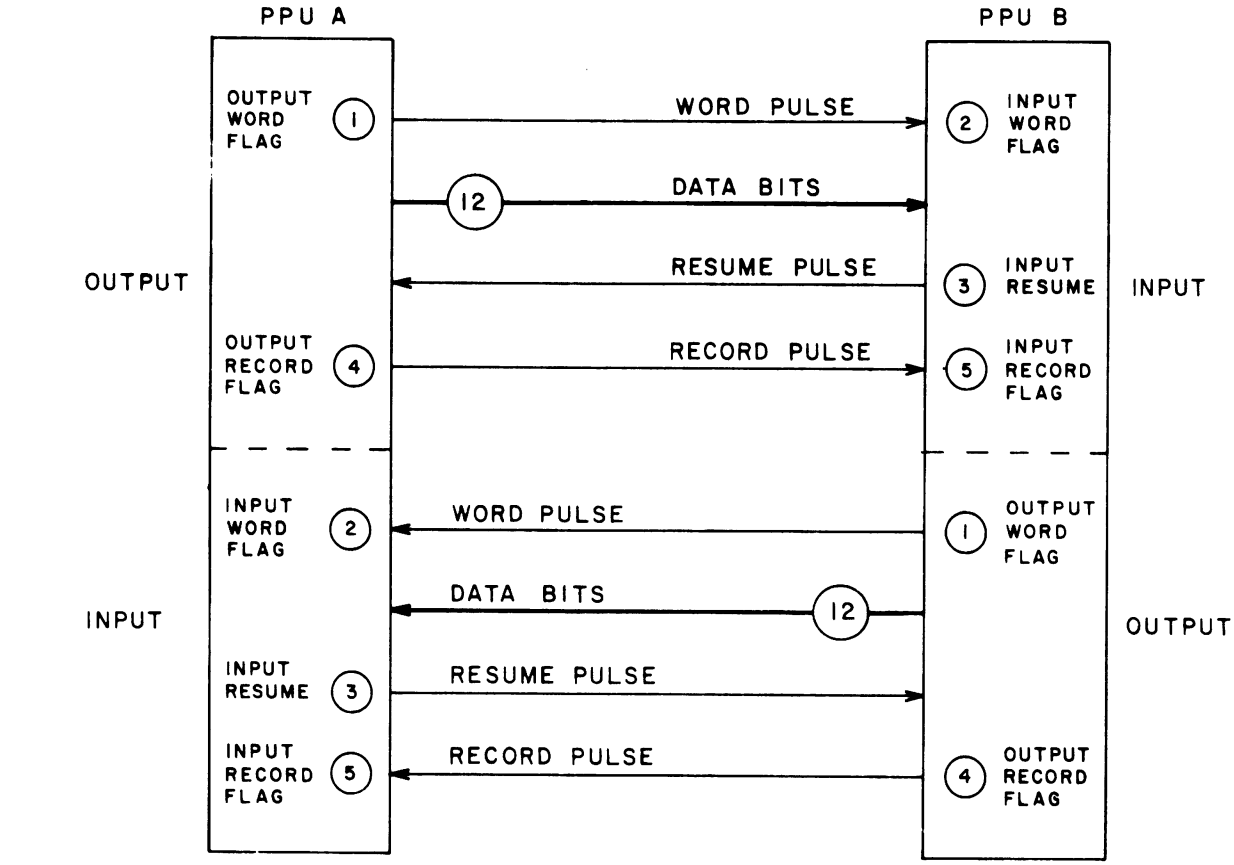
This flag is set when the PPU transmits a 1-clock-period-wide record pulse over the associated output cable. The flag is cleared when a resume pulse is received over this output cable. This flag is forced clear during a dead start condition. A PPU senses the status of this flag by executing I/O jump instruction 66 or 67.

PPU TO PPU DATA TRANSFERS

Figure 4-1 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 sequence 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 enters 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 word 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 flag is still 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.



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 2AU11A

Figure 4-1. PPU/PPU Communications

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 enters 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 enters 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 hangs 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 sends a record pulse to PPU B, which terminates input to PPU B. If PPU B terminates first, PPU A hangs up and remains 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 EQUIPMENT DATA TRANSFERS

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 1 or 0, or assigned functions. Figure 4-2 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.

MAINTENANCE CONTROL UNIT

Two of the mainframe PPUs are used as MCUs. They are internally identical to any other PPU. However, the fixed external connections as illustrated in Figure 4-3 are quite different. Both MCUs are capable of communicating with each other, with an SCM, and with up to 13 first level PPUs. Each MCU has eight bidirectional channels. They gain additional capability through scanners, which expand the number of channels to allow communication with the PPUs.

The MCU scanner selects incoming data for the MCU from one of 17 (octal) scanner input channels and distributes MCU output to one of 16 (octal) scanner output channels. Scanner channels 1 through 15 connect to channel 0 of PPU 1 through 15. Scanner input channel 16 carries parity error address bits, and scanner input channel 17 carries LCM parity error address bits to the MCU. Scanner output channel 16 connects the MCU to a reference voltage scanner for testing marginal operation of mainframe modules. The scanner also connects directly to the PPU for transmission of dead start, dead dump, and clear parity error signals and for receipt of program error and stack parity error bits.

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 through 17) as bits 8 through 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.

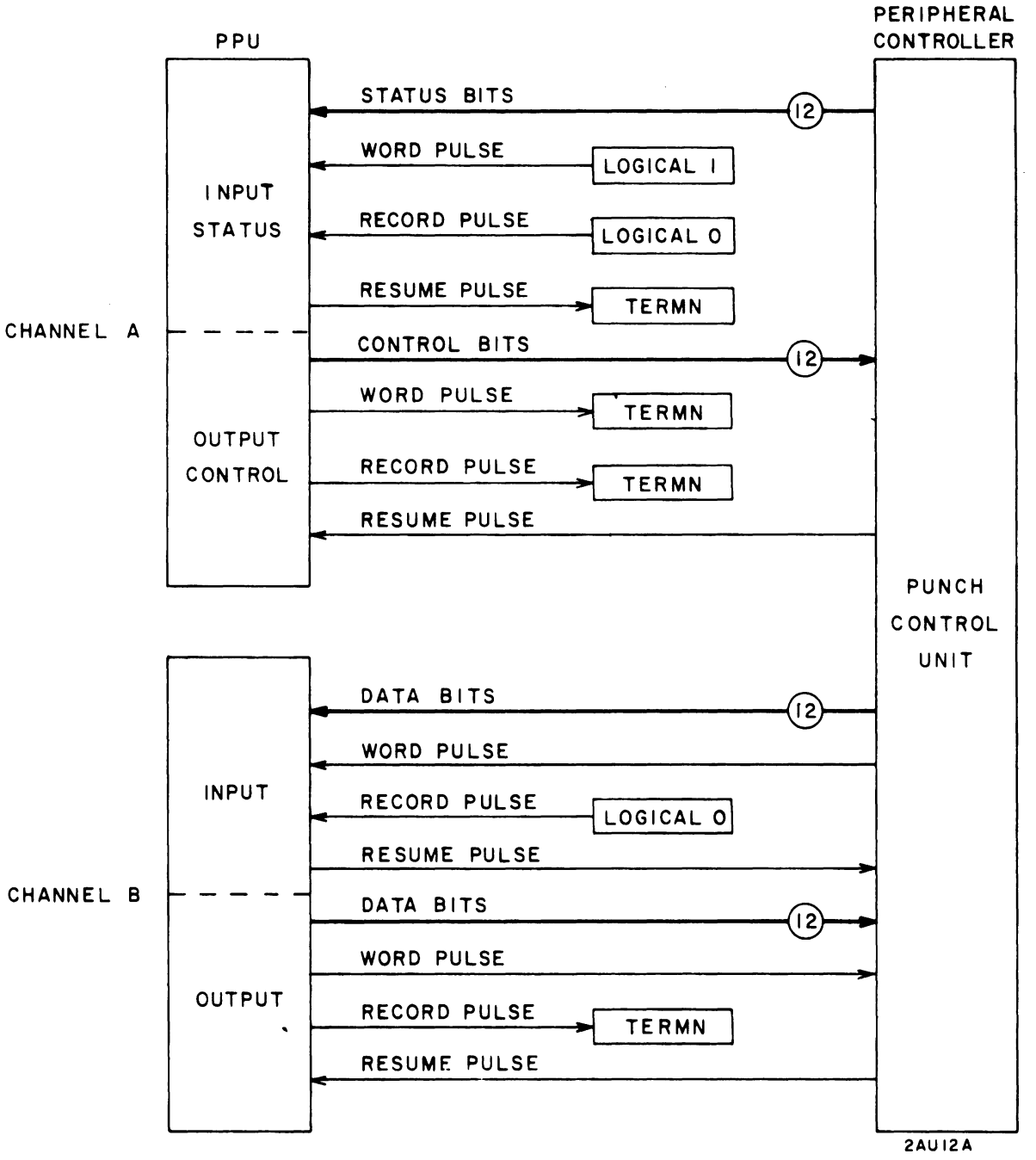


Figure 4-2. PPU/Controller Communications

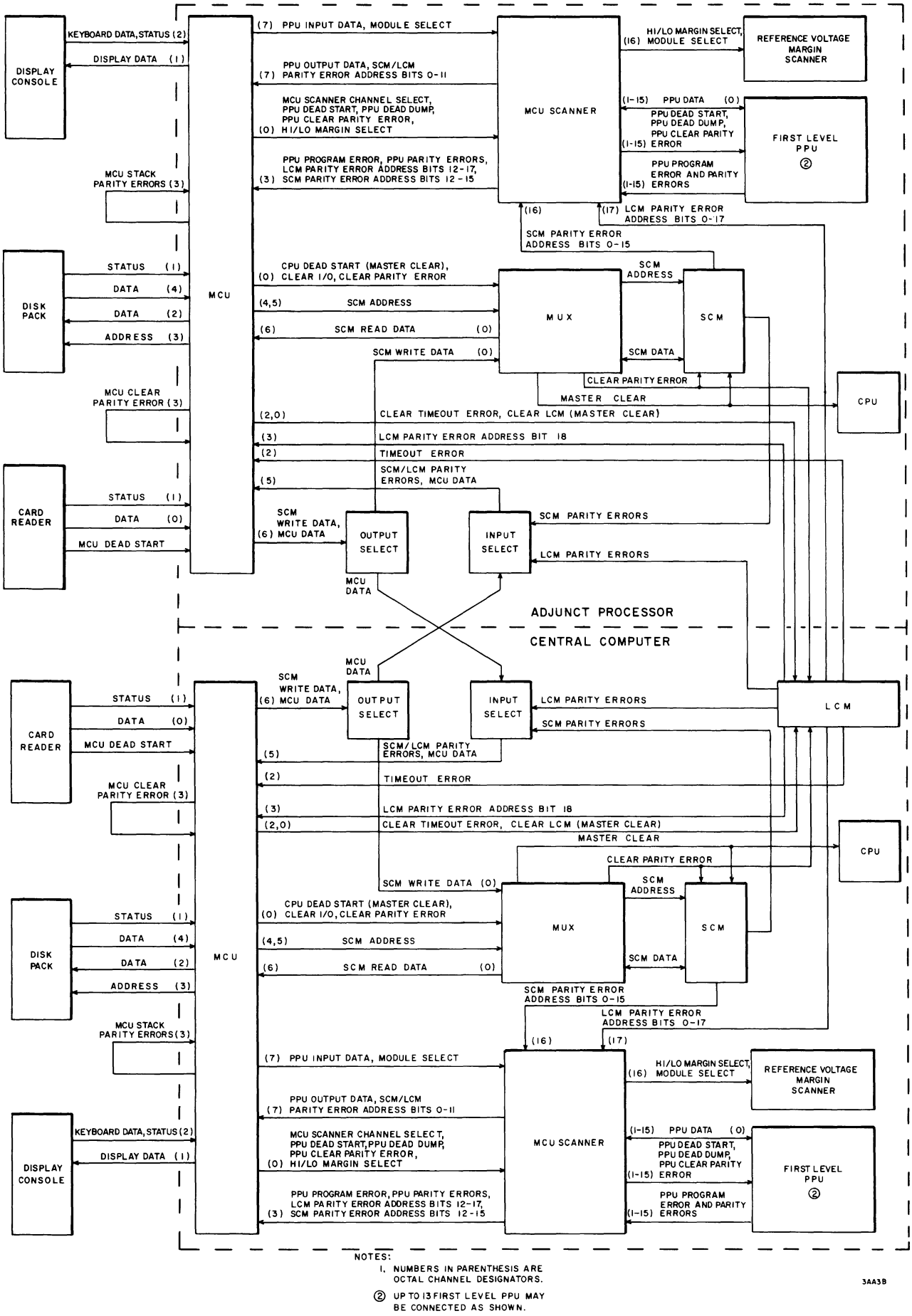


Figure 4-3. MCU Configuration

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 then inputs 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. (Refer to appendix B for related information.)

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. Software determines how the CPU loads the system programs.

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 memory over PPU output channel 0. The transmission starts at PPU address 0 and terminates at PPU address 7776. Input of this data at the MCU is under control of the MCU program. Refer to appendix B for related information.)

CLEAR LCM

Either MCU can output a clear LCM signal as bit 6 on MCU output channel 0. This signal clears the locking registers and control flags in LCM.

LCM TIMEOUT ERROR

If one of the central processors accesses LCM for longer than 450.56 microseconds while in the locked mode, a timeout error signal is sent to the associated MCU as bit 8 on MCU input channel 2. The MCU clears the timeout error flag in LCM by outputting a clear timeout error signal as a record pulse on MCU output channel 2.

PPU AND MCU PARITY ERRORS

The MCU and each PPU contains hardware for detecting parity errors during memory read cycles. The memory stack that has failed is indicated by setting a stack parity error bit in a four-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 through 3 parity error bits as bits 0 through 3 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 7 through 10 of MCU input channel 3. The MCU clears its parity errors by outputting the MCU clear parity error signal as a record pulse on MCU output channel 3.

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 through 4 parity error bits as bits 0 through 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	SCM Section
4	4 (bits 48 through 59)
3	3 (bits 36 through 47)
2	2 (bits 24 through 35)
1	1 (bits 12 through 23)
0	0 (bits 0 through 11)

The MCU program reads the SCM parity error address by selecting scanner channel 16 and inputting address bits 0 through 11 on MCU input channel 7 and address bits 12 through 15 on MCU input channel 3 (bits 0 through 3).

Under MCU program control, the MCU inputs the LCM section 0 through 3 parity error bits as bits 5 through 8 on MCU input channel 5. When set, the LCM section parity error bits indicate bad parity for the following portions of the LCM word.

Error Bit	LCM Section
3	3 (bits 45 through 59)
2	2 (bits 30 through 44)
1	1 (bits 15 through 29)
0	0 (bits 0 through 14)

The MCU program reads bits 0 through 17 of the LCM parity error address by selecting scanner channel 17 and inputting address bits 0 through 11 on MCU input channel 7 and address bits 12 through 17 on MCU input channel 3 (bits 0 through 5). LCM address bit 18 bypasses the scanner and is received as bit 6 on input channel 3.

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 PROGAM 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 as bit 4 on MCU input channel 3. The PPU can be restarted only when the dead start signal is received from the MCU.

CENTRAL PROCESSOR INSTRUCTIONS 5

This section describes the central processor instructions. The instructions tend to fall into two distinct categories, those causing computation and those causing storage references or program branching. The instructions causing computation are generally executed in a fixed amount of time after they have issued from the CIW 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 this section to allow a complete analysis of those situations warranting the programming effort.

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. An instruction may occupy either one or two parcels, depending upon the type of instruction. The possible arrangements of one- and two-parcel instructions are illustrated in Figure 5-1. 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 is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros; it does 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, and 62 instructions with i equal to zero become pass instructions. 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.

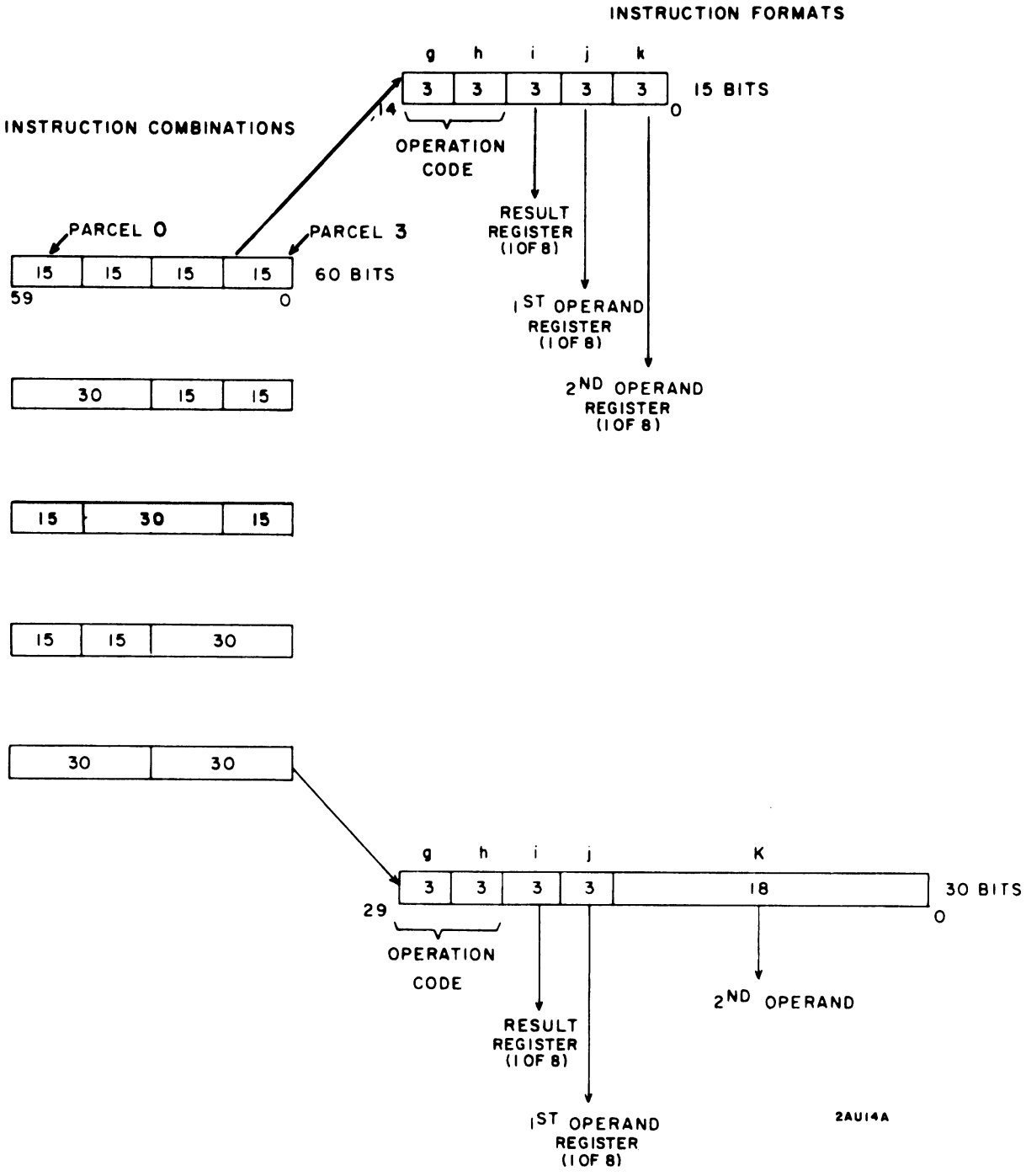


Figure 5-1. Parcel Instruction Arrangements

INSTRUCTION ISSUE

Program instruction words are read one at a time from the IWS into the 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). The proper allowance must be made for the mixture of one- and two-parcel instruction formats. The five possible instruction arrangements are illustrated in Figure 5-1.

PROGRAM BRANCHING

When program execution reaches a branch instruction, the action taken depends upon whether the destination address is already in the IAS. If the destination address is in the IAS, the P register is altered to the new program address and the corresponding word is read from the IWS 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 IAS, 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 IWS

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 IWS.

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 IWS

Several small program sequences may reside in the IWS 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 IWS may branch in such a way that sequential execution is continued into a program area not loaded into the IWS on the initial pass. When this happens, the next sequential instruction word may be missing in the IWS and no request has been made for it.

This situation is equivalent to a branch out of IWS with no branch instruction involved. As soon as the missing word is detected, the hardware initiates a branch out of stack sequence which allows the program to continue with no loss of information.

EXCHANGE JUMP

The exchange jump is a mechanism for switching 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 5-2) provides the following information on a program to be executed.

Program address (P) - 18 bits

Reference address for SCM (RAS) - 18 bits

Field length of program for SCM (FLS) - 18 bits

Reference address for LCM (RAL) - 19 bits

Field length of program for LCM (FLL) - 19 bits

Program status designation register (PSD) - 18 bits

Normal exit address (NEA) - 16 bits

Error exit address (EEA) - 17 bits

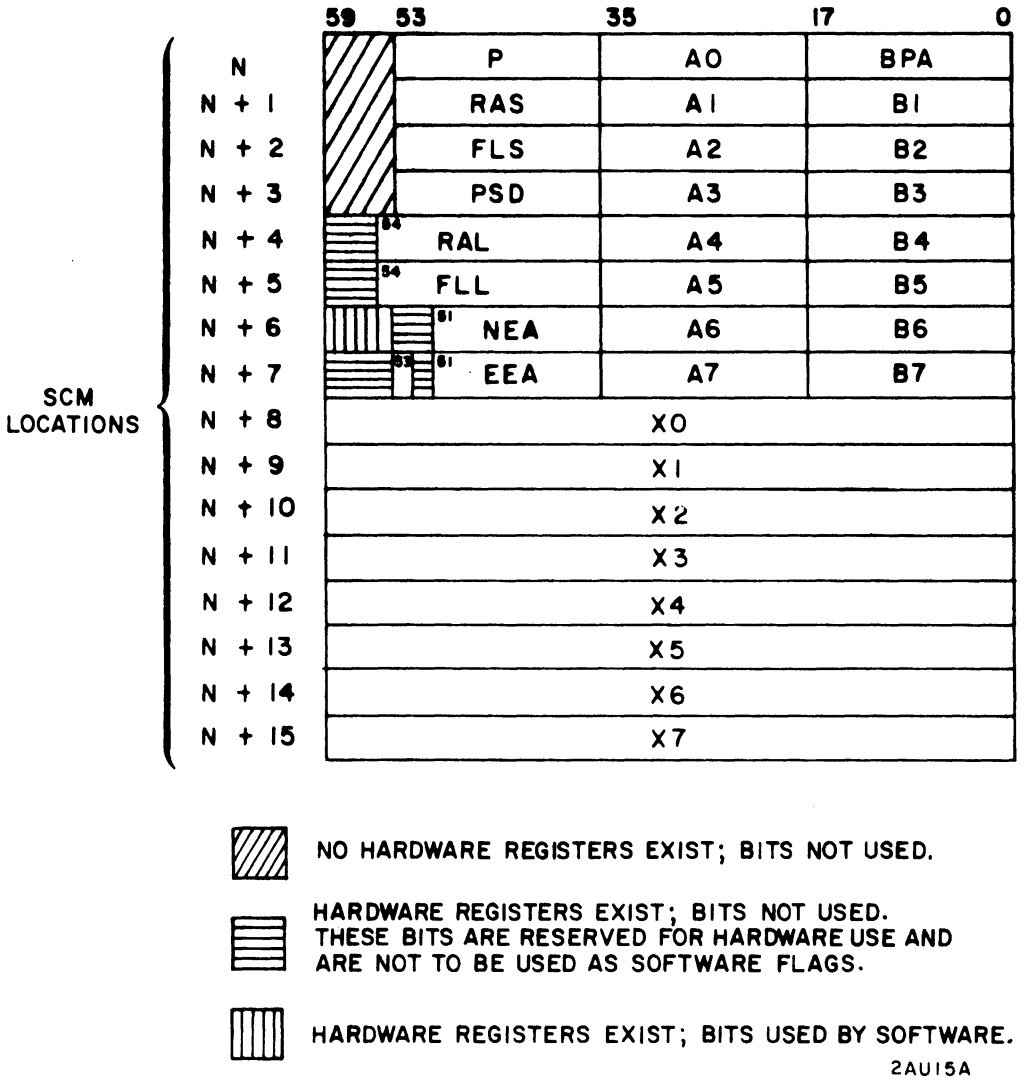


Figure 5-2. Exchange Package

NOTE

Bit 53 is used as a flag and is not part of the address. For a description of its use, refer to the description of the MUX.

Breakpoint address (BPA) - 18 bits

Current contents of the eight A registers

Current contents of the eight X registers

Current contents of 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. (Refer to appendix B for related information.)

Exchange exit instructions (013xx and 013jK)

Error exit

I/O interrupt

Real-time interrupt

Program breakpoint

Step mode

EXCHANGE EXIT INSTRUCTIONS

The normal termination for an exchange package execution interval is caused by an exchange exit instruction (013xx 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 013xx instruction, the 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. If a program erroneously executes an exchange exit instruction with an overflow condition, the exchange jump sequence begins 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 EEA upon encountering an error exit instruction (00) or under certain conditions defined by the PSD register. 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 upon 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 PSD register, 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.

INPUT/OUTPUT INTERRUPT

The MUX section of the central processor monitors I/O activity between the PPU and SCM. The MUX 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

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 clock periods.

The CPU clock period counter contains a 17-bit register that can be sensed 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 is actually the 18th bit of the register.

The real-time clock interrupt flag attempts an interrupt of the program to absolute address 0020 in SCM every 3.6 milliseconds (approximate). The real-time exchange package at this SCM address executes a 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 BPA register. This is a hardware register in the CPU that is loaded from the program exchange package. A coincidence test is made between (BPA) and (P) as each program instruction word is read from the IWS. If coincidence occurs and no other exchange sequence is in process, the program execution terminates with an exchange jump to EEA. If (BPA) is equal to (P) in the initiating exchange jump package, no instructions are executed. Normally, no instructions are executed at BPA.

STEP MODE

A program may be executed in step mode by setting the step mode flag in the PSD 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 EEA.

FLOATING POINT ARITHMETIC

FORMAT

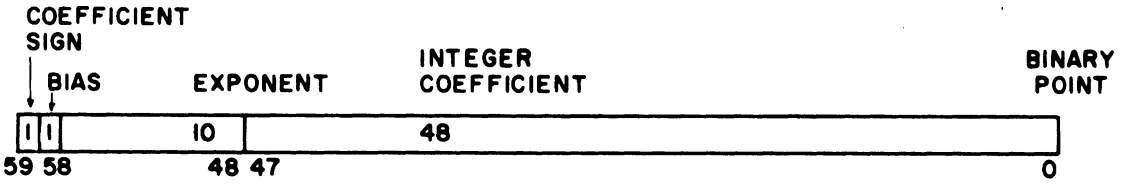
Floating point arithmetic expresses a number with the general expression kB^n, where:

k = coefficient

B = base number

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

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



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

Table 5-1 summarizes the configurations of bits 58 and 59 and the implications, regarding signs, of the possible combinations.

TABLE 5-1. BITS 58 AND 59 CONFIGURATIONS

Bit 59	Bit 58	Coefficient Sign	Exponent Sign
0	1	Positive	Positive
0	0	Positive	Negative
1	0	Negative	Positive
1	1	Negative	Negative

PACKING

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

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

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

Since positive exponents are expressed in true form, begin with a bias of 2000 (positive zero) and add the magnitude of the exponent. The range of positive exponents is 0000 through 1777. In packed form, the range is 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, begin with a bias of 1777 (negative zero) and subtract the magnitude of the exponent. The range of negative exponents is -0000 through -1777. In packed form, the range is 1777 through 0000.

When the coefficient is negative, the packed negative exponent is complemented to become 6000 through 7777.

Examples of packed and unpacked floating point numbers are shown in octal notation to illustrate the packing process. Examples 1 and 2 are different forms of the integer +1. Example 3 is +100 (decimal) and example 4 is -100 (decimal). Examples 5 and 6 are 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 (3777 or 4000 in packed form). This is the largest 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 overflow. 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 generates an overflow.

A complete overflow occurs whenever a floating point functional unit computes a result that requires an exponent larger than +1777. 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 is the same as that which would have been generated if the result had not overflowed the floating point range. The packed floating point result is shown in the paragraph on Nonstandard Operands. The coefficient portion consists of all zeros regardless of the sign of the coefficient.

UNDERFLOW

Underflow of the floating point range is indicated by an exponent value of -1777 (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. 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. 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 packed floating point result is shown in the paragraph on Nonstandard Operands. This result consists of all zeros regardless of the sign which would have been generated if underflow had not occurred.

INDEFINITE

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 0 and the dividend is also 0. 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 -0 exponent and a 0 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 generates an indefinite result no matter what the other operand value. Although indefinite indicators are always generated with a positive sign by the floating point units, they may occur as operands with negative sign because of inversion in the boolean unit.

NONSTANDARD OPERANDS

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 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 PSD register.

Positive overflow (+ ∞)	= 37770------0	Overflow condition flag
Negative overflow (- ∞)	= 40000------0	Overflow condition flag
Positive indefinite (+IND)	= 17770------0	Indefinite condition flag
Positive underflow (+0)	= 00000------0	Underflow condition flag
Negative underflow (-0)	= 00000------0	Underflow condition flag

The following tabulations indicate the resulting forms when various combinations of underflow, overflow, and indefinite forms are used in floating point operations. The designations W and N are defined as follows:

W = Any word except ± ∞ and ±IND

N = Any word except ± ∞ , ±IND, and ±0

Xj PLUS Xk

(Instructions 30, 32, 34)

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

Xj MINUS Xk

(Instructions 31, 33, 35)

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

Xj MULTIPLIED BY Xk

(Instructions 40, 41, 42)

		Xk						
		+N	-N	+0	-0	+∞	-∞	±IND
Xj	+N	-	-	+0	-0	+∞	-∞	IND
	-N	-	-	-0	+0	-∞	+∞	IND
	+0	+0	-0	INTEGER* MULTIPLY		IND	IND	IND
	-0	-0	+0			IND	IND	IND
	+∞	+∞	-∞	IND	IND	+∞	-∞	IND
	-∞	-∞	+∞	IND	IND	-∞	+∞	IND
	± IND	IND	IND	IND	IND	IND	IND	IND

* If both operands used in the integer multiply are normalized, the integer multiply hardware will not be enabled and the underflow condition flag in the PSD register will be set.

Xj DIVIDED BY Xk

(Instructions 44, 45)

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

NORMALIZED NUMBERS

A floating point number in packed format is normalized if the coefficient sign bit is different from bit 47. This condition indicates that the coefficient has been shifted to the left as far as possible; 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 unnormalized results even when both operands are normalized. Therefore, it is 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.

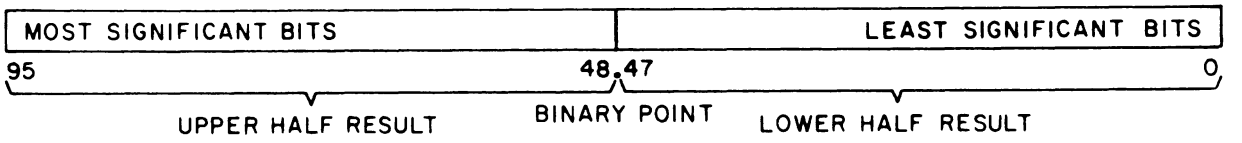
ROUNDING

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 define the effects of rounding in detail.

DOUBLE PRECISION RESULTS

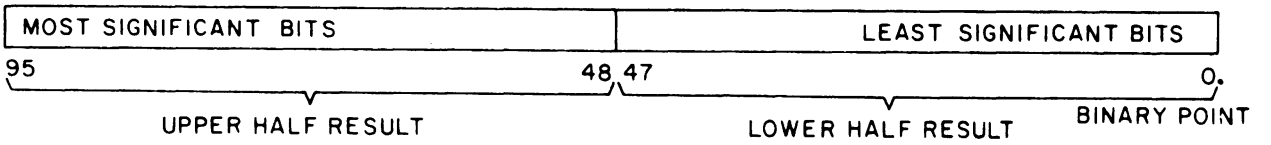
The floating point arithmetic instructions generate double precision results. Use of unrounded instructions allows separate recovery of upper and lower half results with proper exponents; rounded instructions allow only upper half results to be obtained. The position of the binary point and the exponent 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 are returned as the result. Selecting double precision causes only the lower 48 bits of the 96-bit result and the larger exponent minus 60 (octal) to be returned as the result. The subtraction of 60 (octal) is necessary because the binary point is effectively moved from the right of bit 48 to the right of bit 0.

The multiply unit generates a 96-bit product 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 (octal) are returned as the result. The addition of 60 (octal) 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 are 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 48-bit product can be formed by using the double-precision multiply instruction. Both operands must have an exponent value of ±0 and the coefficients both cannot be normalized. The result is sign-extended to 60 bits and sent to an X register.

In integer division, the divisor must be normalized 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 (22) instruction to obtain the integer quotient.

INSTRUCTION TIMING

Execution times for the central processor instruction are given in Table 5-2. The times listed in the Execution Time column assume that no conflicts will occur. However, a delay results unless all of the conditions listed in the Timing Notes column exist for that particular instruction. The Timing Notes are at the end of the table.

TABLE 5-2. CENTRAL PROCESSOR INSTRUCTION TIMING

Mnemonic Code	Instruction Code	Description	Functional Unit	Execution Time (CP)	Timing Notes
ES	00xxx	Error Exit to EAA	-	-	-
RJ	010xK	Return Jump to K	-	13	1, 2, 3
RL	011jK	Block Copy (Bj) + K Words from LCM to SCM	-	(Bj) + K + 18	2, 3, 4, 5, 6, 18, 20, 22
WL	012jK	Block Copy (Bj) + K Words from SCM to LCM	-	(Bj) + K + 13	2, 3, 4, 5, 6, 18, 21, 22
MJ	013jK	Exchange Exit to (Bj) + K (Exit Mode Flag Set)	-	28	1, 2, 3, 4
MJ	013xx	Exchange Exit to NEA (Exit Mode Flag Not Set)	-	28	1, 2, 3, 4
RX	014jk	Read LCM at (Xk) to Xj (Xk Bit 20 Not Set)	-	20	5, 7, 8, 9, 18
RX	014jk	Special LCM Functions (Xk Bit 20 Set)	-	5	8, 19
WX	015jk	Write Xj into LCM at (Xk)	-	5	5, 7, 8, 18
RI	0160k	Reset Input Channel (Bk) Buffer	-	4	8
IB	016jk	Read Input Channel (Bk) Status to Bj (j $\neq$ 0)	-	3	8
TB	016j0	Read Real-Time Clock to Bj	-	3	8
RO	0170k	Reset Output Channel (Bk) Buffer	-	16	8

TABLE 5-2. CENTRAL PROCESSOR INSTRUCTION TIMING (Cont'd)

Mnemonic Code	Instruction Code	Description	Functional Unit	Execution Time (CP)	Timing Notes
OB	017jk	Read Output Channel (Bk) Status to Bj (j ≠ 0)	-	3	8
JP	02ixK	Jump to (Bi) + K	-	11	1, 2, 8, 10
ZR	030jK	Branch to K if (Xj) = 0	-	11	1, 2, 10, 11
NZ	031jK	Branch to K if (Xj) ≠ 0	-	11	1, 2, 10, 11
PL	032jK	Branch to K if (Xj) Positive	-	11	1, 2, 10, 11
NG	033jK	Branch to K if (Xj) Negative	-	11	1, 2, 10, 11
IR	034jK	Branch to K if (Xj) In Range	-	11	1, 2, 10, 11
OR	035jK	Branch to K if (Xj) Out of Range	-	11	1, 2, 10, 11
DF	036jK	Branch to K if (Xj) Definite	-	11	1, 2, 10, 11
ID	037jK	Branch to K if (Xj) Indefinite	-	11	1, 2, 10, 11
EQ	04ijK	Branch to K if (Bi) = (Bj)	-	11	1, 2, 10, 11
NE	05ijK	Branch to K if (Bi) ≠ (Bj)	-	11	1, 2, 10, 11
GE	06ijK	Branch to K if (Bi) ≥ (Bj)	-	11	1, 2, 10, 11
LT	07ijK	Branch to K if (Bi) < (Bj)	-	11	1, 2, 10, 11
BX	10ijx	Transmit (Xj) to Xi	Boolean	2	2, 8, 12, 13
BX	11ijk	Logical Product of (Xj) and (Xk) to Xi	Boolean	2	2, 8, 12, 13
BX	12ijk	Logical Sum of (Xj) and (Xk) to Xi	Boolean	2	2, 8, 12, 13
BX	13ijk	Logical Difference of (Xj) and (Xk) to Xi	Boolean	2	2, 8, 12, 13
BX	14ixk	Transmit Complement of (Xk) to Xi	Boolean	2	2, 8, 12, 13
BX	15ijk	Logical Product of (Xj) and Complement of (Xk) to Xi	Boolean	2	2, 8, 12, 13

TABLE 5-2. CENTRAL PROCESSOR INSTRUCTION TIMING (Cont'd)

Mnemonic Code	Instruction Code	Description	Functional Unit	Execution Time (CP)	Timing Notes
BX	16ijk	Logical Sum of (Xj) and Complement of (Xk) to Xi	Boolean	2	2,8,12,13
BX	17ijk	Logical Difference of (Xj) and Complement of (Xk) to Xi	Boolean	2	2,8,12,13
LX	20ijk	Left Shift (Xi) by jk	Shift	2	2,8,12,13
AX	21ijk	Right Shift (Xi) by jk	Shift	2	2,8,12,13
LX	22ijk	Left Shift (Xk) Nominally (Bj) Places to Xi	Shift	2	2,8,12,13
AX	23ijk	Right Shift (Xk) Nominally (Bj) Places to Xi	Shift	2	2,8,12,13
NX	24ijk	Normalize (Xk) to Xi and Bj	Normalize	3	2,8,12,13
ZX	25ijk	Round Normalize (Xk) to Xi and Bj	Normalize	3	2,8,12,13
UX	26ijk	Unpack (Xk) to Xi and Bj	Boolean	2	2,8,12,13
PX	27ijk	Pack (Xk) and (Bj) to Xi	Boolean	2	2,8,12,13
FX	30ijk	Floating Sum of (Xj) and (Xk) to Xi	Floating Add	4	2,8,12,13
FX	31ijk	Floating Difference of (Xj) and (Xk) to Xi	Floating Add	4	2,8,12,13
DX	32ijk	Floating Double Precision Sum of (Xj) and (Xk) to Xi	Floating Add	4	2,8,12,13
DX	33ijk	Floating Double Precision Difference of (Xj) and (Xk) to Xi	Floating Add	4	2,8,12,13
RX	34ijk	Round Floating Sum of (Xj) and (Xk) to Xi	Floating Add	4	2,8,12,13
RX	35ijk	Round Floating Difference of (Xj) and (Xk) to Xi	Floating Add	4	2,8,12,13
IX	36ijk	Integer Sum of (Xj) and (Xk) to Xi	Long Add	2	2,8,12,13

TABLE 5-2. CENTRAL PROCESSOR INSTRUCTION TIMING (Cont'd)

Mnemonic Code	Instruction Code	Description	Functional Unit	Execution Time (CP)	Timing Notes
IX	37ijk	Integer Difference of (Xj) and (Xk) to Xi	Long Add	2	2, 8, 12, 13
FX	40ijk	Floating Product of (Xj) and (Xk) to Xi	Multiply	5	2, 8, 12, 13, 14
RX	41ijk	Round Floating Product of (Xj) and (Xk) to Xi	Multiply	5	2, 8, 12, 13, 14
DX	42ijk	Floating Double Precision Product of (Xj) and (Xk) to Xi	Multiply	5	2, 8, 12, 13, 14
MX	43ijk	Form Mask of jk Bits to Xi	Shift	2	2, 8, 12, 13
FX	44ijk	Floating Divide (Xj) by (Xk) to Xi	Divide	20	2, 8, 12, 13, 15
RX	45ijk	Round Floating Divide (Xj) by (Xk) to Xi	Divide	20	2, 8, 12, 13, 15
NO	46xxx	Pass	-	1	-
CX	47ixk	Population Count of (Xk) to Xi	Pop. Count	2	2, 8, 12, 13
SA	50ijK	Set Ai to (Aj) + K	Increment	8	2, 8, 16, 17
SA	51ijK	Set Ai to (Bj) + K	Increment	8	2, 8, 16, 17
SA	52ijK	Set Ai to (Xj) + K	Increment	8	2, 8, 16, 17
SA	53ijk	Set Ai to (Xj) + (Bk)	Increment	8	2, 8, 16, 17
SA	54ijk	Set Ai to (Aj) + (Bk)	Increment	8	2, 8, 16, 17
SA	55ijk	Set Ai to (Aj) - (Bk)	Increment	8	2, 8, 16, 17
SA	56ijk	Set Ai to (Bj) + (Bk)	Increment	8	2, 8, 16, 17
SA	57ijk	Set Ai to (Bj) - (Bk)	Increment	8	2, 8, 16, 17
SB	60ijK	Set Bi to (Aj) + K	Increment	2	2, 8, 12, 13
SB	61ijK	Set Bi to (Bj) + K	Increment	2	2, 8, 12, 13
SB	62ijK	Set Bi to (Xj) + K	Increment	2	2, 8, 12, 13
SB	63ijk	Set Bi to (Xj) + (Bk)	Increment	2	2, 8, 12, 13

TABLE 5-2. CENTRAL PROCESSOR INSTRUCTION TIMING (Cont'd)

Mnemonic Code	Instruction Code	Description	Functional Unit	Execution Time (CP)	Timing Notes
SB	64ijk	Set Bi to (Aj) + (Bk)	Increment	2	2, 8, 12, 13
SB	65ijk	Set Bi to (Aj) - (Bk)	Increment	2	2, 8, 12, 13
SB	66ijk	Set Bi to (Bj) + (Bk)	Increment	2	2, 8, 12, 13
SB	67ijk	Set Bi to (Bj) - (Bk)	Increment	2	2, 8, 12, 13
SX	70ijK	Set Xi to (Aj) + K	Increment	2	2, 8, 12, 13
SX	71ijK	Set Xi to (Bj) + K	Increment	2	2, 8, 12, 13
SX	72ijK	Set Xi to (Xj) + K	Increment	2	2, 8, 12, 13
SX	73ijk	Set Xi to (Xj) + (Bk)	Increment	2	2, 8, 12, 13
SX	74ijk	Set Xi to (Aj) + (Bk)	Increment	2	2, 8, 12, 13
SX	75ijk	Set Xi to (Aj) - (Bk)	Increment	2	2, 8, 12, 13
SX	76ijk	Set Xi to (Bj) + (Bk)	Increment	2	2, 8, 12, 13
SX	77ijk	Set Xi to (Bj) - (Bk)	Increment	2	2, 8, 12, 13

TIMING NOTES

1. All previous instruction fetches are completed.
2. No SCM conflicts or SAS backup caused by SCM conflicts exist.
3. No I/O word request occurs.
4. All operating registers are free.
5. LCM is not busy.
6. All LCM banks have completed previously initiated read/write cycles.
7. The requested LCM bank has completed a previously initiated read/write cycle.
8. The requested operating register(s) is free.
9. If the requested word is in an LCM bank operand register because of a previous reference, the minimum execution time is 6 clock periods.
10. If the address is in the IAS, the execution time is 3 clock periods.
11. If the branch conditions are not met, the execution time is 2 clock periods.
12. The requested destination register(s) input data path is free during the required clock period.
13. After the instruction has issued to the functional unit, no further delay is possible.

14. The multiply unit is free.

15. The divide unit is free.

16. If no storage reference is required (i=0), the execution time is 2 clock periods.

17. After the instruction has issued to the increment unit, no further delays are possible in the delivery of data to the Ai register. However, SCM conflicts may delay the resulting storage reference.

18. LCM is not being used by another processor.

19. LCM flag register or locking registers are not being used by another processor.

20. For 256K LCM, maximum transfer rate is 32 words per 64 clock periods.

21. For 256K LCM, maximum transfer rate is 32 words per 65 clock periods.

22. If (Bj)+K=0 or an SCM or LCM range error occurs, the execution time is 7 clock periods.

DESCRIPTION OF INSTRUCTIONS

This portion of the manual describes the central processor instructions in detail. Each instruction is described separately as to what it does, briefly how it does it, and what happens if unusual or special situations arise. Instruction designators are listed and defined in Table 5-3.

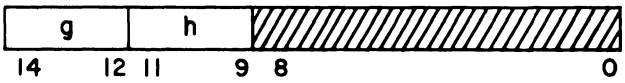
TABLE 5-3. CENTRAL PROCESSOR INSTRUCTION DESIGNATORS

Designator	Use
gh	6-bit instruction code
ghi	9-bit instruction code
i	3-bit code specifying one of eight registers
j	3-bit code specifying one of eight registers
jk	6-bit code specifying amount of shift
k	3-bit code specifying one of eight registers
K	18-bit operand or address

TABLE 5-3. CENTRAL PROCESSOR INSTRUCTION DESIGNATORS (Cont'd)

Designator	Use
x	Unused designator
A	One of eight 18-bit address registers
B	One of eight 18-bit index registers, B0 is fixed and equal to zero
X	One of eight 60-bit operand registers
( )	Contents of a register

00xxx ERROR EXIT TO EEA



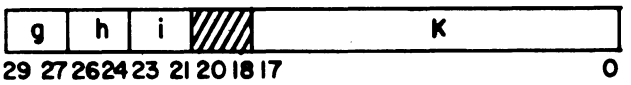
This instruction is treated as an error condition and sets the program range condition flag in the PSD register. This condition flag then generates an error exit request which causes an exchange jump to address (EEA). All instructions which have issued prior to this instruction are run to completion. Any instructions following this instruction in the CIW are not executed. When all operands have arrived at the operating registers as a result of previously issued instructions, an exchange jump occurs to the exchange package designated by (EEA).

The i, j, and k designators in this instruction are ignored. The program address stored in the exchange package on the terminating exchange jump is advanced one count from the address of the CIW. This is true regardless of which parcel of the CIW contains the error exit instruction.

This instruction is not intended for use in normal program code. The program range condition flag is set in the PSD register to indicate that the program has jumped to an area of SCM which may be in range but is not a valid program code. This should occur when an incorrectly coded program jumps into an unused area of SCM or into a data field. The program range condition flag is also set on the condition of a jump to address zero or a jump beyond the SCM field length. These conditions can be determined by the system monitor program on the basis of the register contents in exchange package. The existence of an error exit condition resulting from execution of this instruction may thus be deduced by the monitor program. The program range condition flag may not be set if another interrupt condition occurs at the same time.

A special situation may occur when a program is terminated with an error exit instruction and a previously issued instruction stores a result operand in SCM. The error exit is treated as an SCM range error which blocks a write operation in SCM as soon as the error is detected. A legitimate SCM write operation may be blocked by the error condition even though the instruction causing the write issues substantially before the error exit. The timing depends upon the SCM bank conflicts which may have occurred.

010xK RETURN JUMP TO K



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction writes a special word into SCM at relative address K. The IWS is cleared, and the current program sequence is then terminated by a jump to address K + 1 in SCM. The word stored in SCM contains a jump instruction which causes an unconditional jump to the address of this return jump instruction plus one.

This instruction is intended to call a subroutine and insert execution of this subroutine between execution of the CIW and the following instruction word. Instructions appearing after the return jump instruction in the CIW are not executed. The called subroutine exit must be at address K in SCM. The called subroutine entrance address must be K + 1 in SCM.

This instruction stores a full 60-bit word at address K in SCM. The upper half of this word contains an unconditional jump instruction (0400) with an address which is equal to the current program address plus one. The lower half of the stored word is all zeros. The octal digits in the stored word then appear as illustrated with the xxx field indicating the location of the current program address plus one.

K	0400x xxxxx 00000 00000	subroutine exit
K + 1	yyyyy yyyyy yyyyy yyyyy	subroutine entrance

DESIGNATOR j NOT ZERO

The j designator is normally zero. However, a nonzero value has no effect on the results.

LAST PARCEL

This 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 this instruction begins in the last parcel of an instruction word, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

JUMP OUT OF RANGE

If the value of K is greater than the SCM field length, the instruction is executed with the store of the exit word in SCM inhibited. The program address is altered to the value K and advanced by one count in a normal manner. The program range condition flag is set in the PSD register to indicate the jump out of range. The program sequence is then terminated with an exchange jump to (EEA). The resulting exchange package contains a program address equal to K + 1 and the program range condition flag set in the PSD register.

JUMP TO ZERO

If the value of K is zero, the instruction is executed in a normal manner. The exit word is stored at address zero in SCM. 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 is terminated at the completion of this instruction with an exchange jump to (EEA). The instruction will have advanced the program address one count so that the exchange package indicates a program address of one rather than zero.

JUMP TO BREAKPOINT ADDRESS

If the value of K is equal to (BPA), (P) is momentarily set equal to (BPA). This is detected as a breakpoint condition, and the breakpoint condition flag sets in the PSD register. This instruction advances (P) one count. This final value of (P) appears in the exchange package when the breakpoint interrupt occurs.

ERROR CONDITION DURING EXECUTION

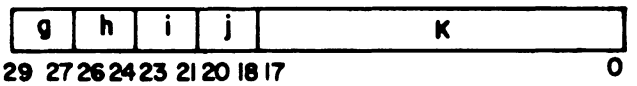
A number of error conditions may occur during the execution of this instruction. Some possible conditions are arithmetic errors due to previously issued instructions and parity errors in SCM or LCM. If any error conditions occur, the proper flags are set in the

PSD register and the instruction is executed to completion in a normal manner. The program sequence is then terminated with an exchange jump to (EEA). The resulting exchange package contains a program address equal to K + 1 and one or more error flags set in the PSD register.

I/O INTERRUPT DURING EXECUTION

An I/O interrupt request may occur during the execution of this instruction. In this 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.

011jK BLOCK COPY (Bj) + K WORDS FROM LCM TO SCM



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction reads a sequence of 60-bit words from consecutive addresses in LCM and copies them into a block of consecutive addresses in SCM. The block of words begins at address (X0) in LCM. The words are stored in SCM beginning at address (A0). The number of words to be copied is the sum of (Bj) + K. This quantity cannot exceed 1777 (octal) words. If a larger quantity is used, LCM truncates the quantity to the 10-bit maximum. Thus, a block count of 3000 (octal) words transfers 1000 (octal) 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 LCM into SCM as quickly as possible. All other activity, 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 word request for SCM occurs during this transfer, the data flow is interrupted for 1 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 minimum of a 1-clock-period delay. An additional delay occurs if the I/O reference causes a bank conflict in SCM.

The length of the block is determined by adding K to (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 10 bits by LCM. A zero result causes this instruction to be executed as a pass instruction.

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

(X0) NEGATIVE OR GREATER THAN 19 SIGNIFICANT BITS

The lowest order 19 bits of (X0) are used to determine the initial address in LCM for the block copy. The highest 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 (Bj) in an 18-bit ones complement mode. The resulting sum is treated as an 18-bit positive integer. This integer is added to the lowest order 19 bits of (X0), also treated as a positive integer. The resulting sum is compared with (FLL). If the resulting sum is greater than (FLL), the block copy is not executed, the LCM block range condition flag is set in the PSD register, and the instruction is issued as a pass. The exchange jump to (EEA) resulting from setting the LCM block range condition flag does not occur before execution of the next program instruction word unless a delay is introduced by subsequent instructions in the CIW.

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), the block copy is not executed, the SCM block range condition flag is set in the PSD register, and the instruction is issued as a pass. The exchange jump to (EEA) resulting from setting the SCM block range condition flag does not occur before execution of the next program instruction word unless a delay is introduced by subsequent instructions in the CIW.

BLOCK LENGTH NEGATIVE

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. A

negative result therefore appears as a large positive integer. The SCM block range condition flag, and possibly the LCM block range condition flag, sets in the PSD register. The instruction issues as a pass. The exchange jump to (EEA) resulting from setting the SCM block range condition flag does not occur before execution of the next program instruction word unless a delay is introduced by subsequent instructions in the CIW.

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 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 are discarded and are reread from the LCM bank.

LAST PARCEL

This instruction requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If the instruction begins in the last parcel, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

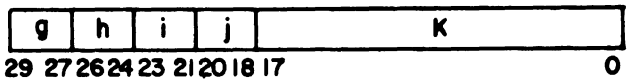
ERROR CONDITION DURING EXECUTION

An LCM or SCM parity error may occur during the execution of this 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 instruction is executed to completion. There are no error conditions which interrupt the instruction before completion.

I/O INTERRUPT DURING EXECUTION

An I/O interrupt request may occur during the execution of this instruction. The interrupt request is not honored until the block copy instruction has been completed and any subsequent instructions in the CIW have been completed.

012jK BLOCK COPY (Bj) + K WORDS FROM SCM TO LCM



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction reads a sequence of 60-bit words from consecutive addresses in SCM and copies them into a block of consecutive addresses in LCM. The block of words begins at address (A0) in SCM. The words are stored in LCM beginning at address (X0). The number of words to be copied is the sum of (Bj) + K. This quantity cannot exceed 1777 (octal) words. If a larger quantity is used, LCM truncates the quantity to the 10-bit maximum. Thus, a block count of 3000 (octal) words transfers 1000 (octal) 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 LCM as quickly as possible. All other activity, 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 word request for SCM occurs during this transfer, the data flow is interrupted for 1 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 minimum of a 1-clock-period delay. An additional delay occurs if the I/O reference causes a bank conflict in SCM.

The length of the block is determined by adding K to (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 bits by LCM. A zero result causes this instruction to be executed as a pass instruction.

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

(X0) NEGATIVE OR GREATER THAN 19 SIGNIFICANT BITS

The lowest order 19 bits of (X0) are used to determine the initial address in LCM for the block copy. The highest 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 (Bj) in an 18-bit ones complement mode. The resulting sum is treated as an 18-bit positive integer. This integer is added to the lowest order 19 bits of (X0), also treated as a positive integer. The resulting sum is compared with (FLL). If the resulting sum is greater than (FLL), the block copy is not executed, the LCM block range condition flag is set in the PSD register, and the instruction is issued as a pass. The exchange jump to (EEA) resulting from setting the LCM block range condition flag does not occur before execution of the next program instruction word unless a delay is introduced by subsequent instructions in the CIW.

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), the block copy is not executed, 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 does not occur before execution of the next program instruction word unless a delay is introduced by subsequent instructions in the CIW.

BLOCK LENGTH NEGATIVE

The length of the block is determined by adding the quantity K to (Bj). The addition is performed in an 18-bit ones complement mode. The resultant sum is treated as an 18-bit positive integer. Therefore, a negative result appears as a large positive integer. The SCM block range condition flag, and possibly the LCM block range condition flag, set in the PSD register. The instruction issues as a pass. The exchange jump to (EEA), resulting from setting the SCM block range condition flag, does not occur before execution of the next program instruction word unless a delay is introduced by subsequent instructions in the CIW.

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.

LAST PARCEL

This instruction requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If the instruction begins in the last parcel, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

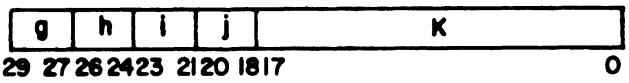
ERROR CONDITION DURING EXECUTION

An SCM parity error may occur during the execution of this 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 instruction is executed to completion. There are no error conditions which interrupt the instruction before completion.

I/O INTERRUPT DURING EXECUTION

An I/O interrupt request may occur during the execution of this instruction. The interrupt request is not honored until the block copy instruction has been completed and any subsequent instructions in the CIW have been completed.

013jK EXCHANGE EXIT TO (Bj) + K (EXIT MODE FLAG SET)



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction causes the current program sequence to terminate with an exchange jump to an address in SCM. The exchange package is located at relative address (Bj) + K in SCM. The two quantities are added in an 18-bit ones complement mode. The result is treated as an 18-bit positive integer. This integer is added to (RAS), also treated as an 18-bit positive integer, to form the absolute address of the exchange package in SCM.

This form of the 013 instruction is used by the monitor program only. The exit mode flag in the PSD register is cleared during execution of object programs. The monitor program uses this instruction to exchange jump to one of a number of possible object program exchange packages. Each of these exchange packages normally specifies a cleared exit mode flag. A selected object program exchange package then returns to this same area of SCM and resumes the monitor program when its execution interval has been completed (refer to alternate form of 013 instruction).

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, the request is denied. The rejected interrupt request is not lost since the conditions which caused it are reinstated when the exchange package enters its next execution interval.

Any remaining instructions in the CIW are not executed. The program address stored in the exchange package is advanced one count from the address of the CIW. Therefore, the program continues at the first parcel of the following instruction word during the next execution interval for this exchange package.

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

EXCHANGE ADDRESS OUT OF RANGE

There is no protection for addressing out of the SCM field on this instruction. Any error in the calculation of the exchange package address, either in range or out of range, almost certainly results in complete system failure. The exchange package address is determined by adding K to (Bj) in a ones complement mode. The result is treated as an 18-bit positive integer and is added to (RAS), also treated as an 18-bit positive integer. The lowest order 16 bits of this last addition are used as the absolute address in SCM for the exchange package.

LAST PARCEL

This instruction normally requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If this instruction begins in the last parcel of an instruction word, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

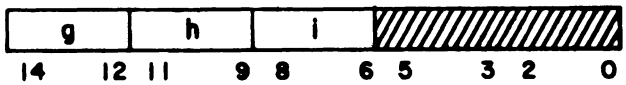
ERROR CONDITION

This instruction takes priority over an error exit request. The flag or flags associated with the error exit request are preserved in the exchange package. The error exit request is regenerated at the beginning of the next execution interval for the exchange package. This causes an error exit in the next execution interval for the exchange package before the execution of the first instruction.

I/O INTERRUPT

This instruction takes priority over an I/O interrupt request. The I/O request is not honored until the exchange jump has been completed and a new exchange package has been generated.

013xx EXCHANGE EXIT TO NEA (EXIT MODE FLAG NOT SET)



An exchange exit instruction executed in this mode 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. This mode makes no use of the j or k designators in the instruction.

This instruction is the vehicle for switching rapidly from an object program to a monitor program. All operating register values, program addresses, and mode selections are preserved in this process in order that the object program may be continued at a later time. The program address in the object program exchange package is advanced one count from the address of the instruction word containing the exchange exit instruction. The monitor program normally resumes the object program at this address.

This instruction is intended for use in calling the system monitor program for I/O requests, library calls, storage assignments, and so on. The operating register values at the time of execution of this instruction are intended as the vehicle for parameter interchange between the object program and the 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, the request is denied. The rejected interrupt request is not lost since the conditions which caused it are reinstated when the exchange package enters its next execution interval.

Any remaining instructions in the CIW are not executed. The program address stored in the exchange package is advanced one count from the address of the CIW. Therefore, the program continues at the first parcel of the following instruction word during the next execution interval for this exchange package unless the monitor program alters the exchange package.

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

DESIGNATOR j, k, NOT ZERO

A nonzero j or k designator has no effect on the results of this instruction. If the j designator is nonzero, a test is made for register Bj free. This may delay execution of the instruction but does not affect the results.

(NEA) OUT OF RANGE

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. Normally the SCM field for an object program does not include the address (NEA). If (NEA) has more than 16 bits of significance, considered as a positive integer, the upper bits are discarded and the lower 16 bits used as the absolute address in SCM for the exchange jump.

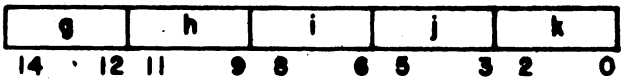
ERROR CONDITION

This instruction takes priority over an error exit request. The flag or flags associated with the error exit request are preserved in the exchange package. The error exit request is regenerated at the beginning of the next execution interval for the exchange package. This causes an error exit in the next interval for the exchange package before the execution of the first instruction.

I/O INTERRUPT

This instruction takes priority over an I/O interrupt request. The I/O request is not honored until the exchange jump has been completed and a new exchange package has been generated. The I/O request is then honored before execution of the first instruction in the new program.

014jk READ LCM AT (Xk) TO Xj (Xk BIT 20 NOT SET)



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

This instruction is intended for direct addressing of LCM of 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 proceeds without an LCM read/write cycle delay until the addressing crosses an LCM bank boundary. If another processor references the same bank between requests, the data is lost and another read/write cycle is required.

This instruction is buffered to the extent that it issues in 3 clock periods unless a previous LCM reference is in process. When this instruction issues (receives LCM priority), the LCM busy flag is set and remains set until the requested word has been delivered to the designated X register. This process differs from an SCM read reference in that only one LCM read or write may be in process at one time.

(Xk) NEGATIVE OR GREATER THAN 19 SIGNIFICANT BITS

The lowest order 19 bits of (Xk) are used to determine the address in LCM. The highest 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.

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 causes an error exit request to interrupt the program with an exchange jump to address (EEA). The instruction is executed with an LCM read reference beyond the assigned field, and a zero word is entered in the Xj register. The absolute address in LCM is 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 does not occur before one or more subsequent instructions has been executed.

USE OF THE X0 REGISTER

The X0 register may be used for either Xj or Xk in this instruction.

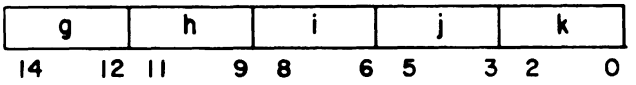
Xj AND Xk REGISTER

If the j and k designators have the same value, the requested address is lost when the word arrives at the Xj register.

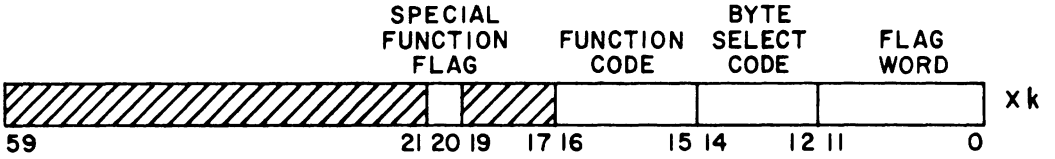
READ FROM BLOCK COPY FIELD

The requested word may reside in an LCM bank operand register as a result of a previous block copy instruction. This condition is sensed, and the word is read directly from the LCM bank operand register.

014jk SPECIAL LCM FUNCTIONS (Xk BIT 20 SET)



This instruction is used to execute two special LCM functions. One of these functions involves setting or clearing one of the three 1-bit locking registers in LCM access control. (Each access has a locking register.) The other function involves reading or modifying the common 48- or 96-bit LCM flag register. Special functions are selected by setting bit 20 of the Xk register. Also, either the exit mode flag or the monitor mode flag must be set in the PSD register. If selected, this instruction issues even through LCM may be busy performing a memory reference. This instruction may be executed in either the locked or unlocked mode. When this instruction is executed, the LCM access control decodes the contents of the Xk register as follows:



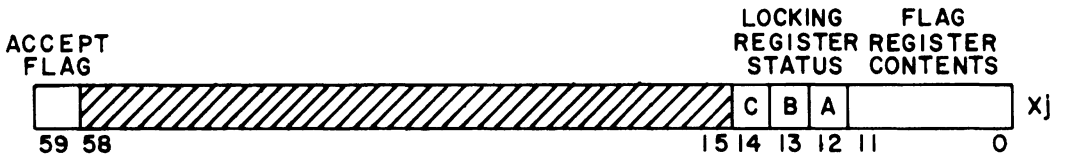
Bits 0 through 11 contain information to be placed in a selected 12-bit byte of the LCM flag register. This byte is determined by the byte select code in bit positions 12 through 14 as follows:

Byte Select Code	LCM Flag Register Bytes	
	48-Bit Register	96-Bit Register
0	0-11	0-11
1	12-23	12-23
2	24-35	24-35
3	36-47	36-47
4	0-11	48-59
5	12-23	60-71
6	24-35	72-83
7	36-47	84-95

Bits 15 and 16 contain the following special function codes.

Function Select Code	Function
0	Clear those bits in the selected LCM flag register byte in which the corresponding bits in the Xk register flag word are set.
1	Set those bits in the selected LCM flag register byte in which the corresponding bits in the Xk register flag word are set.
2	Clear the locking register. (A CPU connected to one of the LCM accesses may clear only its own locking register.)
3	Set the locking register. (A CPU connected to one of the LCM accesses may set only its own locking register.)

Upon completion of this instruction, the LCM access control returns a status word to the Xj register. The following is the Xj register format.



Bits 0 through 11 represent the final contents of the selected LCM flag register word. Bits 12, 13, and 14 represent the state of the locking registers for LCM accesses A, B, and C, respectively. Bit 59 is an accept flag. If this flag is set, the requested action was performed. If clear, the request was aborted and neither the LCM flag register nor locking registers were altered.

ABORTED FUNCTIONS

If function 0 is selected, all of the bits in the selected LCM flag register byte which are to be cleared must be in a set condition. If any of the bits are already cleared, the function is aborted.

If function 1 is selected, all of the bits in the selected LCM flag register byte which are to be set must be in a cleared condition. If any of the bits are already set, the function is aborted.

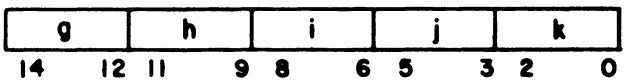
If function 2 is selected, the locking register which is to be cleared must be in a set condition. If it is already cleared, the function is aborted.

If function 3 is selected, all three of the locking registers must be in a cleared condition. If any of them are already set, the function is aborted.

BOTH EXIT AND MONITOR MODE FLAGS CLEAR

If the exit mode flag and the monitor mode flag in the PSD register are both clear, this instruction is issued as a pass.

015jk WRITE Xj INTO LCM AT (Xk)



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

This instruction is intended for direct addressing of 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 proceeds without an LCM bank read/write cycle delay until the addressing crosses an LCM bank boundary. If another processor references the same bank between requests, the data is lost and another read/write cycle is required.

This instruction is buffered to the extent that it issues in 3 clock periods unless a previous LCM reference is in process. When this instruction issues, the LCM busy flag is set and remains set until the word has been delivered to the proper LCM bank operand register. The following instruction may use either of the X registers designated in

this instruction without causing a register conflict. If the word cannot be entered immediately in the proper LCM bank operand register, it is held in the LCM write register until the LCM bank operand register is free. This process differs from an SCM write reference in that only one LCM read or write may be in process at one time.

(Xk) NEGATIVE OR GREATER THAN 19 SIGNIFICANT BITS

The lowest order 19 bits of (Xk) are used to determine the address in LCM. The highest 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.

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 causes an error exit request to interrupt the program with an exchange jump to address (EEA). The word is not written into LCM. The exchange jump resulting from the error exit condition generally does not occur before one or more subsequent instructions has been executed.

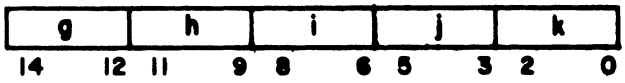
USE OF THE X0 REGISTER

The X0 register may be used for either Xj or Xk in this instruction.

Xj AND Xk SAME REGISTER

The j and k designators may have the same value in this instruction. In this case, the requested address is also the operand.

0160k RESET INPUT CHANNEL (Bk) BUFFER



This instruction resets the input channel (Bk) buffer in preparation for the next incoming record. The input channel (Bk) buffer address register is cleared to zero and the assembly register is reset to the first position.

This instruction is intended for execution in the monitor program input routine which terminates a record of incoming data and prepares for the next record. The monitor

input routine is called by an I/O interrupt request when the input record flag is set. The data in the buffer is then normally transferred to LCM, and this instruction is executed to clear the buffer for the next incoming record.

This instruction is effective only if the monitor mode flag is set in the PSD register. If the monitor mode flag is cleared, this instruction becomes a pass instruction. There are no interlocks for this instruction other than the monitor mode flag. When this instruction issues, it executes the required channel functions without regard to the current status or activity of the channel.

This instruction is normally not executed except in response to an I/O interrupt request resulting from the setting of the input record flag. This flag is cleared when the interrupt request is generated. Further entries to the buffer are not locked out by the interrupt request flag in the channel access control during the execution interval for the interrupt exchange package. The PPU must wait for a positive response from the monitor program over the output channel before beginning the next record.

(Bk) NOT A VALID CHANNEL NUMBER

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

MONITOR MODE FLAG NOT SET

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

CHANNEL ACTIVE

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

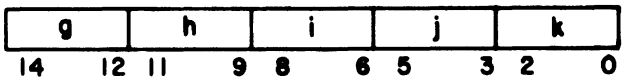
CONSECUTIVE RESETS FOR DIFFERENT CHANNELS

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

CONSECUTIVE RESETS FOR SAME CHANNEL

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

016jk READ INPUT CHANNEL (Bk) STATUS TO Bj (j ≠ 0)



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

This instruction is intended for use in monitoring the progress of the input channel buffer. The 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 interrupt request is generated by the threshold testing mechanism whenever the input channel buffer address is advanced across a field boundary. This occurs at the center and at the end of the buffer area.

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

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 channel zero access position. In this case, the current contents of the CPU clock period counter is read into the Bj register. This is a 17-bit counter which is advanced one count in a twos complement mode each clock period. This count is intended for timing measurements of programs. Timing considerations for this special use are the same as the normal timing for an input channel buffer address register.

(Bk) NOT A VALID CHANNEL NUMBER

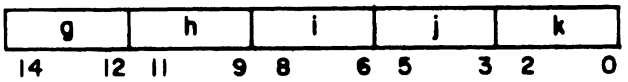
The lowest order four bits of (Bk) are used in this instruction. The highest order bits are ignored. If highest order bits are set in (Bk), the lowest order four bits are masked out and used to determine the channel number. If (Bk) = 0, this instruction reads the contents of the CPU clock period counter.

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.

CONSECUTIVE EXECUTIONS

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 results in the I/O access control.

0170k RESET OUTPUT CHANNEL (Bk) BUFFER



This instruction resets the output channel (Bk) buffer in preparation for the next record transmission. The output buffer address register is cleared to zero. A record pulse is transmitted over the output channel data path. The output word request flag is then set to read the first word from the buffer.

This instruction is intended for execution in the monitor program output routine to initiate a new record transmission over a channel output data path. The buffer is normally inactive when this instruction is executed. The buffer is loaded with the data for the next record, and then this instruction is executed to initiate the transmission. A record pulse is transmitted to indicate the beginning of a new record. The first word of data follows as soon as the output word request flag has caused the first word to be read from the output buffer to the disassembly register.

This instruction is effective only if the monitor mode flag is set in the PSD register. If the monitor mode flag is cleared, this instruction becomes a pass instruction. There are no interlocks for this instruction other than the monitor mode flag. When this instruction issues, it executes the required channel functions without regard to the current status or activity of the channel. The disassembly register is reset by the output word request flag.

(Bk) NOT A VALID CHANNEL NUMBER

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

MONITOR MODE FLAG NOT SET

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

CHANNEL ACTIVE

The output channel buffer is normally inactive when this instruction is executed because the monitor program has detected completion of the previous record before beginning this routine. There are two methods that the monitor program can use to detect end of record. One method is to read the buffer address and compare it with a known record length. The other is a positive response from the PPU over the corresponding channel input data path. If the buffer is actively moving data over the channel output data path at the time this instruction is executed, conflicting commands may be sent to the channel registers. The commands associated with this instruction have priority. The result is a loss of data in the previous record.

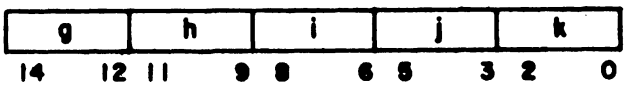
CONSECUTIVE RESETS FOR DIFFERENT CHANNELS

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

CONSECUTIVE RESETS FOR SAME CHANNEL

Two or more reset output buffer instructions may occur in consecutive program instruction locations referencing the same channel. These instructions issue in consecutive clock periods and repeatedly perform the same functions. A record pulse is transmitted over the channel output data dath for each instruction execution. The buffer is repeatedly restarted, and a data word may be transmitted over the channel output data path depending upon the timing of the instructions and the conflicts that occur.

017jk READ OUTPUT CHANNEL (Bk) STATUS TO Bj (j ≠ 0)



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

This instruction is intended for use in monitoring the progress of the output channel buffer. The 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 interrupt request is generated by the threshold testing mechanism whenever the buffer address is advanced across a field boundary. This occurs at the center of the buffer area and at the end of the buffer area.

(Bk) NOT A VALID CHANNEL NUMBER

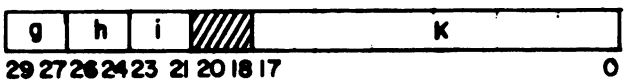
The lowest order four bits of (Bk) are used in this instruction. The highest order bits are ignored. If highest order bits are set in (Bk), the lowest order four bits are masked out and used to determine the channel number. If (Bk) = 0, this instruction reads all zeros into Bj.

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.

CONSECUTIVE EXECUTIONS

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 if the Bj register reservations do not cause a delay. No interference results in the I/O access control.

02ixK JUMP TO (Bi) + K



This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction causes the current program sequence to terminate with a jump to address (Bi) + K in SCM.

This instruction is intended as a vehicle to allow computed branch point destinations. This is the only 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 the complications associated with the IWS.

The quantities (Bi) and K are added in an 18-bit ones complement mode. The result is treated as an 18-bit positive integer. This sum specifies the beginning address in SCM for the new program sequence. The remaining instructions, if any, in the CIW are not executed. The IWS is not altered by this instruction.

DESIGNATOR j NOT ZERO

The j designator in this instruction is normally zero. However, a nonzero value has no effect on the results.

LAST PARCEL

This 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 this instruction begins in the last parcel of an instruction word, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

I/O INTERRUPT OR ERROR CONDITION

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

PREVIOUS FETCH IS DESTINATION ADDRESS

If the branch point destination address is not in the IAS at the beginning of this instruction, but is in process to the IAS as an instruction fetch address, this word arrives at the IWS and enters the CIW register in less than the minimum 11 clock periods normally required for a jump out of stack. The remainder of the instruction sequence is completed and a duplicate word is read to the IWS as a normal initial instruction fetch. This does not cause any special problems in the IWS.

JUMP OUT OF RANGE

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

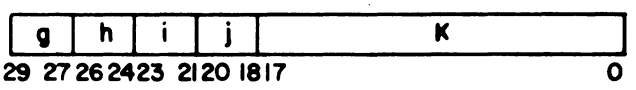
JUMP TO ZERO

A jump to relative address zero in SCM is treated in the same manner as a jump out of range. The program range condition flag is set in the PSD register, and the program is terminated with an error exit to address (EEA). The terminating exchange package contains a zero quantity in the program address field.

JUMP TO BREAKPOINT ADDRESS

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

030jK BRANCH TO K IF (Xj) = 0



This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon the contents of the Xj register. The decision is not made and the instruction does not issue from the CIW register until the Xj register is free. The branch to address K occurs only on the following conditions. The current program sequence is continued for all other cases.

Jump to K if: (Xj) = 0000 0000 0000 0000 0000 (plus zero)

(Xj) = 7777 7777 7777 7777 7777 (minus zero)

This instruction is intended for branching on a zero result from either a fixed point or a floating point operation.

LAST PARCEL

This 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 this instruction begins in the last parcel of an instruction word, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

PREVIOUS FETCH IS DESTINATION ADDRESS

If the branch point destination address is not in the IAS at the beginning of this instruction but is in process to the IAS as an instruction fetch address, this word arrives at the IWS and enters the CIW register in less than the minimum 11 clock periods normally required for a jump out of stack. The remainder of the instruction sequence is completed and a duplicate word is read to the IWS as a normal initial instruction fetch. This does not cause any special problems in the IWS.

JUMP OUT OF RANGE

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

JUMP TO ZERO

A jump to relative address zero in SCM is treated in the same manner as a jump out of range. The program range condition flag is set in the PSD register, and the program is terminated with an error exit to address (EEA). The terminating exchange package contains a zero quantity in the program address field.

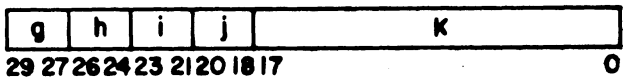
JUMP TO BREAKPOINT ADDRESS

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

I/O INTERRUPT OR ERROR CONDITION

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

031jK BRANCH TO K IF (Xj) ≠ 0



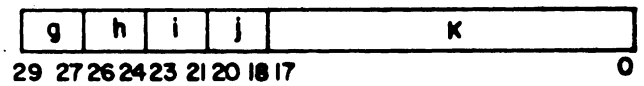
This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon the contents of the Xj register. This decision is not made and the instruction does not issue from the CIW register until the Xj register is free. The program sequence is continued only on the following conditions. The branch to address K occurs for all other cases.

Continue if:	(Xj) = 0000 0000 0000 0000 0000	(plus zero)
	(Xj) = 7777 7777 7777 7777 7777	(minus zero)

This instruction is intended for branching on a nonzero result from either a fixed or a floating point operation.

The special situations for this instruction are the same as those listed for the 030 instruction.

032jK BRANCH TO K IF (Xj) POSITIVE



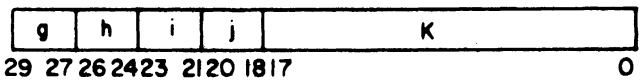
This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon the contents of the Xj register. This decision is not made and the instruction does not issue from the CIW register until the Xj register is free. The branch decision for this instruction is based on the value of the sign bit in (Xj).

Jump to K if:	Bit 59 of (Xj) = 0	(positive)
Continue if:	Bit 59 of (Xj) = 1	(negative)

This instruction is intended for branching on a positive result from either a fixed point or a floating point operation.

The special situations for this instruction are the same as those listed for the 030 instruction.

033jK BRANCH TO K IF (Xj) NEGATIVE



This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon the contents of the Xj register. This decision is not made and the instruction does not issue from the CIW register until the Xj register is free. The branch decision for this instruction is based on the value of the sign bit in (Xj).

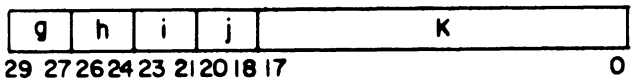
Jump to K if: Bit 59 of (Xj) = 1 (negative)

Continue if: Bit 59 of (Xj) = 0 (positive)

This instruction is intended for branching on a negative result from either a fixed point or a floating point operation.

The special situations for this instruction are the same as those listed for the 030 instruction.

034jK BRANCH TO K IF (Xj) IN RANGE



This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon the contents of the Xj register. This decision is not made and the instruction does not issue from the CIW register until the Xj register is free. The program sequence is continued only on the following conditions. The branch to address K occurs for all other cases.

Continue if: (Xj) = 3777 xxxx xxxx xxxx xxxx (positive overflow)

(Xj) = 4000 xxxx xxxx xxxx xxxx (negative overflow)

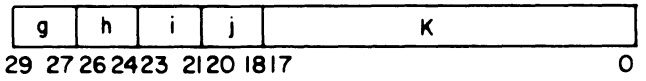
(Xj) = 1777 xxxx xxxx xxxx xxxx (positive indefinite)

(Xj) = 6000 xxxx xxxx xxxx xxxx (negative indefinite)

This instruction is intended for branching on a floating point quantity within the floating point range. The value of the coefficient is ignored in making this branch test. An underflow quantity is considered in range for purposes of this branch test.

The special situations for this instruction are the same as those listed for the 030 instruction.

035jK BRANCH TO K IF (Xj) OUT OF RANGE



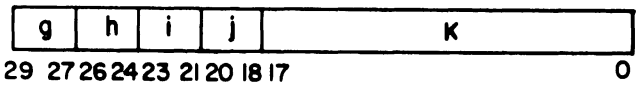
This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon the contents of the Xj register. This decision is not made and the instruction does not issue from the CIW register until the Xj register is free. The branch to address K occurs only on the following conditions. The current program sequence is continued for all other cases.

Jump to K if:	(Xj) = 3777 xxxx xxxx xxxx xxxx	(positive overflow)
	(Xj) = 4000 xxxx xxxx xxxx xxxx	(negative overflow)
	(Xj) = 1777 xxxx xxxx xxxx xxxx	(positive indefinite)
	(Xj) = 6000 xxxx xxxx xxxx xxxx	(negative indefinite)

This instruction is intended for branching on a floating point quantity which is not in the floating point range. The value of the coefficient is ignored in making this branch test. An underflow quantity is considered in range for purposes of this branch test.

The special situations for this instruction are the same as those listed for the 030 instruction.

036jK BRANCH TO K IF (Xj) DEFINITE



This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon the contents of the Xj register. This decision is not made and the instruction does not issue from the CIW register until the Xj register is free. The program sequence is continued only on the following conditions. The branch to address K occurs for all other cases.

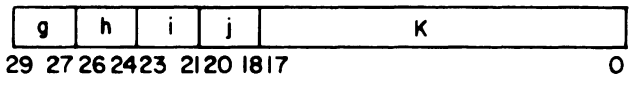
Continue if: (Xj) = 1777 xxxx xxxx xxxx xxxx (positive indefinite)

(Xj) = 6000 xxxx xxxx xxxx xxxx (negative indefinite)

This instruction is intended for branching on a floating point quantity which may be out of range but is still defined. The value of the coefficient is ignored in making this branch test. An overflow quantity or an underflow quantity is considered defined for purposes of this branch test.

The special situations for this instruction are the same as those listed for the 030 instruction.

037jK BRANCH TO K IF (Xj) INDEFINITE



This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon the contents of the Xj register. This decision is not made and the instruction does not issue from the CIW register until the Xj register is free. The branch to address K occurs only on the following conditions. The current program sequence is continued for all other cases.

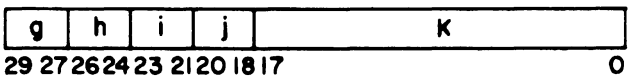
Jump to K if: (Xj) = 1777 xxxx xxxx xxxx xxxx (positive indefinite)

(Xj) = 6000 xxxx xxxx xxxx xxxx (negative indefinite)

This instruction is intended for branching on a floating point quantity which is not defined. The value of the coefficient is ignored in making this branch test. An overflow quantity or an underflow quantity is considered defined for purposes of this branch test.

The special situations for this instruction are the same as those listed for the 030 instruction.

04ijK BRANCH TO K IF (Bi)=(Bj)



This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon a comparison of the contents of the Bi and Bj registers. This decision is not made and the instruction does not issue from the CIW register until the Bi and Bj registers are free. The branch to address K occurs only if the two quantities are identical on a bit-by-bit comparison basis. The current program sequence is continued for all other cases.

This instruction is intended for branching on an index equality test. A quantity consisting of all zeros and a quantity consisting of all ones are not equal for this test.

DESIGNATORS i AND j HAVE THE SAME VALUE

If the i and j designators have the same value, the designated B register is compared against itself. The branch condition test is made as if two different B registers were designated and the contents of the two B registers were identical.

LAST PARCEL

This 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 this instruction begins in the last parcel of an instruction word, it is not executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

PREVIOUS FETCH IS DESTINATION ADDRESS

If the branch point destination address is not in the IAS at the beginning of this instruction, but is in process to the IAS as an instruction fetch address, this word arrives at the IWS and enters the CIW register in less than the minimum 11 clock periods normally required for a jump out of stack. The remainder of the instruction sequence is completed and a duplicate word is read to the IWS as a normal initial instruction fetch. This does not cause any special problems in the IWS.

JUMP OUT OF RANGE

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

JUMP TO ZERO

A jump to relative address zero in SCM is treated in the same manner as a jump out of range. The program range condition flag is set in the PSD register, and the program is terminated with an error exit to address (EEA). The terminating exchange package contains a zero quantity in the program address field.

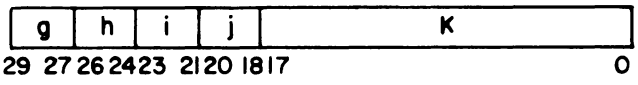
JUMP TO BREAKPOINT ADDRESS

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

I/O INTERRUPT OR ERROR CONDITION

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

05ijK BRANCH TO K IF (Bi) ≠ (Bj)



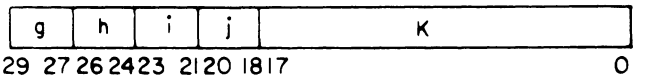
This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon a comparison of the contents of the Bi and Bj registers. This decision is not made and the instruction does not issue from the CIW register until the Bi and

Bj registers are free. The program sequence is continued only if the two quantities are identical on a bit-by-bit comparison basis. The branch to address K occurs for all other cases.

This instruction is intended for branching on an index inequality test. A quantity consisting of all zeros and a quantity consisting of all ones are not equal for this test.

The special situations for this instruction are the same as those listed for the 04 instruction.

06ijK BRANCH TO K IF (Bi) ≥ (Bj)

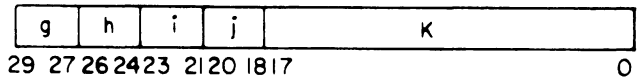


This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon a comparison of the contents of the Bi and Bj registers. Both quantities are treated as signed integers. This decision is not made and the instruction does not issue from the CIW register until the Bi and Bj registers are free. The branch to address K occurs if the contents of register Bi is greater than or equal to the contents of register Bj. The current program sequence is continued if the contents of register Bi is less than the contents of register Bj.

This instruction is intended for branching on an index threshold test. The test is 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. A branch to address K occurs if the sign of the result is positive. The current sequence is continued if the sign of the result is negative. A positive zero quantity and a negative zero quantity are not treated as equal in this test.

The special situations for this instruction are the same as those listed for the 04 instruction.

07ijK BRANCH TO K IF (Bi<(Bj)

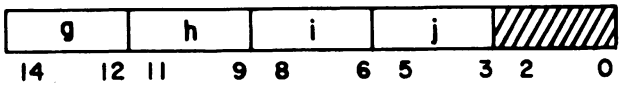


This instruction is a two-parcel instruction in which the lower order 18 bits are used as an operand K. Execution of this instruction causes the program sequence to terminate with a jump to address K in SCM or to continue with the current program sequence, depending upon a comparison of the contents of the Bi and Bj registers. Both quantities are treated as signed integers. This decision is not made and the instruction does not issue from the CIW register until the Bi and Bj registers are free. The branch to address K occurs if the contents of register Bi is less than the contents of register Bj. The current program sequence is continued if the contents of register Bi is greater than or equal to the contents of register Bj.

This instruction is intended for branching on an index threshold test. The test is 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. A branch to address K occurs if the sign of the result is negative. The current sequence is continued if the sign of the result is positive. A positive zero quantity and a negative zero quantity are not treated as equal in this test.

The special situations for this instruction are the same as those listed for the 04 instruction.

10ijx TRANSMIT (Xj) TO Xi



This instruction causes the boolean unit to read a 60-bit word from the Xi register and copy this word into the Xj register.

This instruction 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.

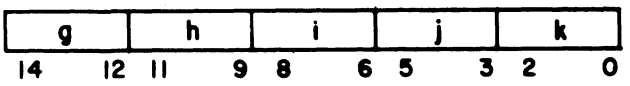
DESIGNATOR k NOT ZERO

The k designator in this instruction is normally zero. However, a nonzero value has no effect on the results.

DESIGNATORS i AND j HAVE THE SAME VALUE

If the i and j designators have the same value, this instruction reads a 60-bit word from the designated X register and then writes the same information back into that X register. The timing is the same as for the normal case, and no special conflicts occur.

11ijk LOGICAL PRODUCT OF (Xj) AND (Xk) TO Xi



This instruction causes the boolean unit to read operands from two X registers, operate upon them to form a result, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). The resultant word delivered to the Xi register is the bit-by-bit logical product of the two operands. Each of the 60 bits in (Xj) is compared with the corresponding bit of (Xk) to form a single bit in (Xi). A sample computation is listed in octal notation to illustrate the operation performed and includes the four possible bit combinations that may occur.

(Xj) = 7777 7000 0123 4567 1010

(Xk) = 0123 4567 0077 7700 1100

(Xi) = 0123 4000 0023 4500 1000

This instruction is intended for extracting portions of a 60-bit word during data processing as distinguished from numerical computation. This instruction, together with the other boolean and shift instructions, may be used to manipulate alphanumeric or other coded data not related to the 60-bit machine word length.

DESIGNATORS j AND k HAVE THE SAME VALUE

If the j and k designators have the same value, the designated X register contents is operated upon by a copy of this same quantity. The instruction degenerates into a copy instruction. The timing is the same as the timing for the normal case, and no special conflicts occur.

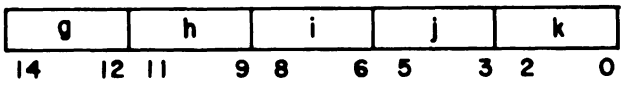
DESIGNATORS i AND j HAVE THE SAME VALUE

If the i and j designators have the same value, the quantity (Xj) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

DESIGNATORS i AND k HAVE THE SAME VALUE

If the i and k designators have the same value, the quantity (Xk) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

12ijk LOGICAL SUM OF (Xj) AND (Xk) TO Xi



This instruction causes the boolean unit to read operands from two X registers, operate upon them to form a result, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). The resultant word delivered to the Xi register is the bit-by-bit logical sum of the two operands. Each of the 60 bits in (Xj) is compared with the corresponding bit of (Xk) to form a single bit in (Xi). A sample computation is listed in octal notation to illustrate the operation performed and includes the four possible bit combinations that may occur.

(Xj) = 0000 7777 0123 4567 1010

(Xk) = 0123 4567 7777 0000 1100

(Xi) = 0123 7777 7777 4567 1110

This instruction is intended for merging portions of a 60-bit word into a composite word during data processing as distinguished from numerical computation. This instruction, together with the other boolean and shift instructions, may be used to manipulate alphanumeric or other coded data not related to the 60-bit machine word length.

DESIGNATORS j AND k HAVE THE SAME VALUE

If the j and k designators have the same value, the designated X register contents is merged with another copy of the same quantity. The instruction degenerates into a copy instruction. The timing is the same as the timing for the normal case, and no special conflicts occur.

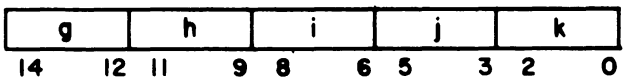
DESIGNATORS i AND j HAVE THE SAME VALUE

If the i and j designators have the same value, the quantity (Xj) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

DESIGNATORS i AND k HAVE THE SAME VALUE

If the i and k designators have the same value, the quantity (Xk) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

13ijk LOGICAL DIFFERENCE OF (Xj) AND (Xk) TO Xi



This instruction causes the boolean unit to read operands from two X registers, operate upon them to form a result, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). The resultant word delivered to the Xi register is the bit-by-bit logical difference of the two operands. Each of the 60 bits in (Xj) is compared with the corresponding bit of (Xk) to form a single bit in (Xi). A sample computation is listed in octal notation to illustrate the operation performed and includes the four possible bit combinations that may occur.

(Xj) = 0123 7777 0123 4567 1010

(Xk) = 0123 4567 7777 3210 1100

(Xi) = 0000 3210 7654 7777 0110

This instruction is intended for comparing bit patterns or for complementing bit patterns during data processing as distinguished from numerical computation. This instruction, together with the other boolean and shift instructions, may be used to manipulate alphanumeric or other coded data not related to the 60-bit machine word length.

DESIGNATORS j AND k HAVE THE SAME VALUE

If the j and k designators have the same value, a logical difference is formed between two identical quantities. The result is a word of all zeros written into the Xi register. The timing is the same as for the normal case.

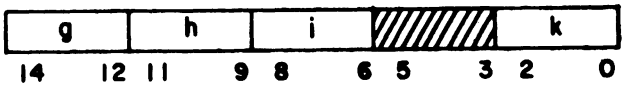
DESIGNATORS i AND j HAVE THE SAME VALUE

If the i and j designators have the same value, the quantity (Xj) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

DESIGNATORS i AND k HAVE THE SAME VALUE

If the i and k designators have the same value, the quantity (Xk) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

14ixk TRANSMIT COMPLEMENT OF (Xk) TO Xi



This instruction causes the boolean unit to read a 60-bit word from the Xk register, complement the word, and write the result into the Xi register.

This instruction is intended for changing the sign of a fixed point or floating point quantity as quickly as possible. This instruction is also useful in data processing for inverting an entire 60-bit field. The result is usually returned to the same X register.

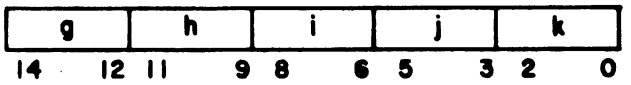
DESIGNATOR j NOT ZERO

The j designator in this instruction is normally zero. However, a nonzero value has no effect on the results.

DESIGNATORS i AND k HAVE THE SAME VALUE

The i and k designators frequently have the same value in this instruction. The quantity read from the designated X register is complemented and returned to the same X register. The timing is the same.

15ijk LOGICAL PRODUCT OF (Xj) AND COMPLEMENT OF (Xk) TO Xi



This instruction causes the boolean unit to read operands from two X registers, operate upon them to form a result, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). The resultant word delivered to the Xi register is the bit-by-bit logical product of (Xj) and the complement of (Xk). Each of the 60 bits in (Xj) is compared with the corresponding bit of (Xk) to form a single bit in (Xi). A sample computation is listed in octal notation to illustrate the operation performed and includes the four possible bit combinations that may occur.

(Xj) = 7777 7000 0123 4567 1010

(Xk) = 0123 4567 0007 7700 1100

(Xi) = 7654 3000 0120 0067 0010

This instruction is intended for extracting portions of a 60-bit word during data processing as distinguished from numerical computation. This instruction, together with the other boolean and shift instructions, may be used to manipulate alphanumeric or other coded data not related to the 60-bit machine word length.

DESIGNATORS j AND k HAVE THE SAME VALUE

If the j and k designators have the same value, a logical product is formed between two complementary quantities. The result is a word of all zeros written into the Xi register. The timing is the same as for the normal case.

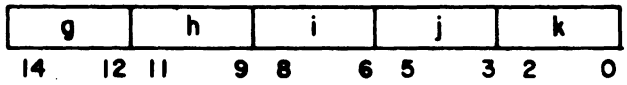
DESIGNATORS i AND j HAVE THE SAME VALUE

If the i and j designators have the same value, the quantity (Xj) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

DESIGNATORS i AND k HAVE THE SAME VALUE

If the i and k designators have the same value, the quantity (Xk) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

16ijk LOGICAL SUM OF (Xj) AND COMPLEMENT OF (Xk) TO Xi



This instruction causes the boolean unit to read operands from two X registers, operate upon them to form a result, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). The resultant word delivered to the Xi register is the bit-by-bit logical sum of (Xj) and the complement of (Xk). Each of the 60 bits in (Xj) is compared with the corresponding bit of (Xk) to form a single bit in (Xi). A sample computation is listed in octal notation to illustrate the operation performed and includes the four possible bit combinations that may occur.

(Xj) = 0000 7777 0123 4567 1010

(Xk) = 0123 4567 7777 0000 1100

(Xi) = 7654 7777 0123 7777 1011

This instruction is intended for merging portions of a 60-bit word into a composite word during data processing as distinguished from numerical computation. This instruction, together with the other boolean and shift instructions, may be used to manipulate alphanumeric or other coded data not related to the 60-bit machine word length.

DESIGNATORS j AND k HAVE THE SAME VALUE

If the j and k designators have the same value, a logical sum is formed from two complementary quantities. The result is a word of all ones written into the Xi register. The timing is the same as for the normal case.

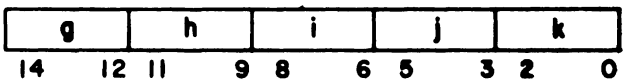
DESIGNATORS i AND j HAVE THE SAME VALUE

If the i and j designators have the same value, the quantity (Xj) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

DESIGNATORS i AND k HAVE THE SAME VALUE

If the i and k designators have the same value, the quantity (Xk) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

17ijk LOGICAL DIFFERENCE OF (Xj) AND COMPLEMENT OF (Xk) TO Xi



This instruction causes the boolean unit to read operands from two X registers, operate upon them to form a result, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). The resultant word delivered to the Xi register is the bit-by-bit logical difference of (Xj) and the complement of (Xk). Each of the 60 bits in (Xj) is compared with the corresponding bit of (Xk) to form a single bit in (Xi). A sample computation is listed in octal notation to illustrate the operation performed and includes the four possible combinations that may occur.

(Xj) = 0123 7777 0123 4567 1010

(Xk) = 0123 4567 7777 3210 1100

(Xi) = 7777 4567 0123 0000 1001

This instruction is intended for comparing bit patterns or for complementing bit patterns during data processing as distinguished from numerical computation. This instruction, together with the other boolean and shift instructions, may be used to manipulate alphanumeric or other coded data not related to the 60-bit machine word length.

DESIGNATORS j AND k HAVE THE SAME VALUE

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 written into the Xi register. The timing is the same as for the normal case.

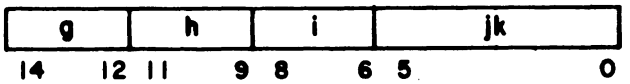
DESIGNATORS i AND j HAVE THE SAME VALUE

If the i and j designators have the same value, the quantity (Xj) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

DESIGNATORS i AND k HAVE THE SAME VALUE

If the i and k designators have the same value, the quantity (Xk) is replaced by the resultant quantity (Xi) at the end of the operation. No special conflicts occur as a result of this combination.

20ijk LEFT SHIFT (Xi) BY jk



This instruction causes the shift unit to read one operand from the Xi register, shift the 60-bit word left circularly by jk bit positions, and write the resulting 60-bit word back into the same Xi register. The designators j and k are treated as a single 6-bit positive integer operand in this instruction.

A left circular shift implies that the bit pattern in the 60-bit word is displaced toward the highest order bit positions. The bits which are shifted off the upper end of the 60-bit word are inserted in the lowest order bit positions in the same sequence. The resulting 60-bit word has the same quantity of bits with values of one and zero as in the original operand.

A sample computation is listed in octal notation to illustrate the operation performed.

Initial (Xi) = 2323 6600 0000 0000 0111

jk = 12 (octal)

Final (Xi) = 7540 0000 0000 0022 2464

This instruction is intended for use in data processing as distinguished from numerical computation. This instruction, together with instruction 21, may be used whenever a data word is to be shifted by a predetermined amount. If the amount of shift is derived in the execution of the program, instruction 22 or 23 should be used.

SHIFT COUNT IS ZERO

If the j and k designators are zero, this instruction reads the operand from the Xi register and returns the result unaltered to the same register. The timing is the same as for the normal case.

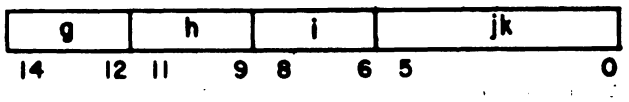
SHIFT COUNT IS GREATER THAN 60 (DECIMAL)

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 shift.

OPERAND ALL ONES OR ALL ZEROS

An all ones or all zeros word is treated in the same manner as any other bit pattern. The timing is the same as for the normal case.

21ijk RIGHT SHIFT (Xi) BY jk



This instruction causes the shift unit to read one operand from the Xi register, shift the 60-bit word right with sign extension by jk bit positions, and write the resulting 60-bit word back into the same Xi register. The designators j and k are treated as a single 6-bit positive integer operand in this instruction.

A right shift with sign extension implies that the bit pattern in the 60-bit word is displaced toward the lowest order bit positions. The bits which are shifted off the lower end of the word are discarded. The highest order bit positions are filled with copies of the original sign bit.

Two sample computations are listed in octal notation to illustrate the operation performed. The first example contains a positive operand and the second example contains a negative operand.

Initial (Xi) = 2004 7655 0002 3400 0004

jk = 30 (octal)

Final (Xi) = 0000 0000 2004 7655 0002

Initial (Xi) = 6000 4420 2222 0000 5643

jk = 10 (octal)

Final (Xi) = 7774 0011 0404 4440 0013

This instruction, together with instruction 20, may be used whenever a data word is to be shifted by a predetermined amount. If the amount of shift is derived in the execution of the program, instruction 22 or 23 should be used.

SHIFT COUNT IS ZERO

If the j and k designators are zero, this instruction reads the operand from the Xi register and returns the result unaltered to the same register. The timing is the same as for the normal case.

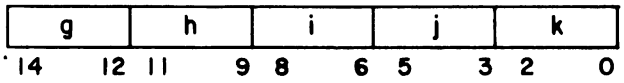
SHIFT COUNT IS GREATER THAN 60 (DECIMAL)

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

OPERAND ALL ONES OR ALL ZEROS

An all ones or all zeros word is treated in the same manner as any other bit pattern. The timing is the same as for the normal case.

22ijk LEFT SHIFT (Xk) NOMINALLY (Bj) PLACES TO Xi



This instruction causes the shift unit to read a 60-bit operand from the Xk register, shift the data either left or right as specified by (Bj), and write the resulting 60-bit word into the Xi register. If (Bj) is positive, the data is shifted to the left in a circular mode the number of bit positions designated by (Bj). If (Bj) is negative, the data is shifted to the right with sign extension the number of bit positions designated by (Bj).

A left circular shift implies that the bit pattern in the 60-bit word is displaced toward the highest order bit positions. The bits which are shifted off the upper end are inserted in the lowest order bit positions in the same sequence. The resulting 60-bit word has the same quantity of bits with values of one and zero as in the original operand.

A right shift with sign extension implies that the bit pattern in the 60-bit word is displaced toward the lowest order bit positions. The bits which are shifted off the lower end are discarded. The highest order bit positions are filled with copies of the original sign bit.

Two sample computations are listed in octal notation to illustrate the operation performed. The first example contains a positive shift count resulting in a left circular shift. The second example illustrates the right shift with sign extension.

(Xk) = 2323 6600 0000 0000 0111

(Bj) = 00 0012

(Xi) = 7540 0000 0000 0022 2464

(Xk) = 1327 6000 0000 3333 2422

(Bj) = 77 7771

(Xi) = 0013 2760 0000 0033 3324

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 correcting the coefficient of a floating point number when the exponent has been unpacked into a B register.

(Bj) IS ZERO

If (Bj) is zero (000000 or 777777), this instruction reads the operand from the Xk register and copies it unaltered into the Xi register. The timing is the same as for the normal case.

(Bj) POSITIVE WITH MAGNITUDE GREATER THAN 60 (DECIMAL)

If (Bj) is positive, only the lowest order six bits are used in determining the shift count. The highest order bits are ignored. The resulting 6-bit shift count is treated modulo 60 (decimal). For example, a shift count of 63 (decimal) results in a left circular shift of three bit positions.

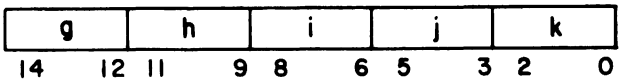
(Bj) NEGATIVE WITH MAGNITUDE GREATER THAN 60 (DECIMAL)

If (Bj) is negative, only the lowest order 12 bits are used in determining the shift count. The highest order bits are ignored. The lowest order 12 bits of (Bj) are complemented, and the resulting positive integer determines the shift count. If this shift count is greater than 60 (decimal), the resulting word stored in the Xi register consists of 60 copies of the original operand sign bit.

OPERAND ALL ONES OR ALL ZEROS

An all ones or all zeros word is treated in the same manner as any other bit pattern. The timing is the same as for the normal case.

23ijk RIGHT SHIFT (Xk) NOMINALLY (Bj) PLACES TO Xi



This instruction causes the shift unit to read a 60-bit operand from the Xk register, shift the data either left or right as specified by (Bj), and write the resulting 60-bit word into the Xi register. If (Bj) is positive, the data is shifted to the right with sign extension the number of bit positions designated by (Bj). If (Bj) is negative, the data is shifted to the left in a circular mode the number of bit positions designated by (Bj).

A left circular shift implies that the bit pattern in the 60-bit word is displaced toward the highest order bit positions. The bits which are shifted off the upper end are inserted in the lowest order bit positions in the same sequence. The resulting 60-bit word has the same quantity of bits with values of one and zero as in the original operand.

A right shift with sign extension implies that the bit pattern in the 60-bit word is displaced toward the lowest order bit positions. The bits which are shifted off the lower end of the word are discarded. The highest order bit positions are filled with copies of the original sign bit.

Two sample computations are listed in octal notation to illustrate the operation performed. The first example contains a positive shift count and results in a right shift with sign extension. The second example contains a negative shift count and results in a left circular shift.

(Xk) = 1327 6000 0000 3333 2422

(Bj) = 00 0006

(Xi) = 0013 2760 0000 0033 3324

(Xk) = 2323 6600 0000 0000 0111

(Bj) = 77 7765

(Xi) = 7540 0000 0000 0022 2464

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 correcting the coefficient of a floating point number when the exponent has been unpacked into a B register.

(Bj) IS ZERO

If (Bj) is zero (000000 or 777777), this instruction reads the operand from the Xk register and copies it unaltered into the Xi register. The timing is the same as for the normal case.

(Bj) POSITIVE WITH MAGNITUDE GREATER THAN 60 (DECIMAL)

If (Bj) is positive, only the lowest order 12 bits are used in determining the shift count. The highest order bits are ignored. If this resulting 12-bit shift count is greater than 60 (decimal), the resulting word stored in the Xi register consists of 60 copies of the original operand sign bit.

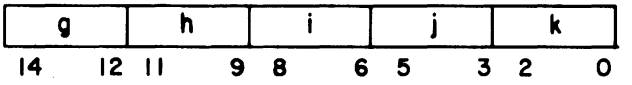
(Bj) NEGATIVE WITH MAGNITUDE GREATER THAN 60 (DECIMAL)

If (Bj) is negative, only the lowest order six bits are used in determining the shift count. The highest order bits are ignored. The lowest order six bits of (Bj) are complemented, and the resulting positive integer shift count is treated modulo 60 (decimal). For example, a shift count of 63 (decimal) results in a left circular shift of three bit positions.

OPERAND ALL ONES OR ALL ZEROS

An all ones or all zeros word is treated in the same manner as any other bit pattern. The timing is the same as for the normal case.

24ijk NORMALIZE (Xk) TO Xi AND Bj



This instruction causes the normalize unit to read one operand from the Xk register, perform a normalizing operation on this word in a floating point format, and deliver the normalized result to the Xi register. In addition, the normalize unit delivers a positive integer shift count to the Bj register. This shift count is the number of bit positions of shift required to normalize the original operand coefficient.

The normalizing operation consists of repositioning the coefficient portion of the operand and then adjusting the exponent portion of the operand to leave the value of the resulting word unaltered. The coefficient is displaced toward the higher order bit positions of the word. The coefficient is shifted the minimum number of bit positions required to make bit 47 different from the sign bit 59. This places the most significant bit of the coefficient in the highest order bit position. The exponent is then decreased by the number of bit positions shifted.

Two sample computations are listed in octal notation to illustrate the operation performed. The first example involves a positive floating point number and the second example involves a negative number.

(Xk) = 2034 0047 6500 0000 2262

(Xi) = 2026 4765 0000 0022 6200

(Bj) = 00 0006

(Xk) = 5743 7730 1277 7777 5515

(Xi) = 5751 3012 7777 7755 1577

(Bj) = 00 0006

This instruction is intended for use in normalized floating point computation in which rounding is not desired. If rounding is desired the 25 instruction should be used.

SPECIAL CASE OPERAND

The normalize unit makes a special case test on (Xk) at the beginning of execution for this instruction. If one of these special cases is present, the operand is copied unaltered to the Xi register. The shift count entered in the Bj register is zero for these cases. No condition flags are set in the PSD register by the normalize unit. The special case formats are listed and consist of partial overflow, complete overflow, and indefinite forms.

(Xk) = 3777 xxxx xxxx xxxx xxxx

(Xk) = 4000 xxxx xxxx xxxx xxxx

(Xk) = 1777 xxxx xxxx xxxx xxxx

(Xk = 6000 xxxx xxxx xxxx xxxx

COMPLETE UNDERFLOW

A complete underflow occurs whenever (Xk) is not a special case and the normalizing process results in an unpacked exponent more negative than -1777 (octal). The

normalize unit output to Xi is blocked, which results in the Xi register receiving all zero bits. The shift count delivered to the Bj register is a result of considering the coefficient of (Xk) without regard to the exponent. This quantity is the value which would be appropriate for normalizing the operand if the exponent were in range. There are no condition flags set in the PSD register.

PARTIAL UNDERFLOW

A partial underflow occurs whenever (Xk) is not a special case and the normalizing process results in an unpacked exponent equal to -1777 (octal). The result is delivered to the Xi and Bj registers as for a normal case even though subsequent computation may detect this operand as an underflow case.

COEFFICIENT IS ZERO

The normalize unit treats an operand with a zero coefficient as a special underflow situation. This exists for either positive or negative numbers whenever the sign bit is the same as each bit in the coefficient and the exponent does not qualify the operand as a special case format. There is no possibility of creating a normalized coefficient for this case. The sign of the result is positive and the Xi register receives all zero bits. The shift count delivered to the Bj register is 48 (decimal) for this case. There are no condition flags set in the PSD register.

UNDERFLOW OPERAND

A special situation exists if (Xk) is in one of the following formats.

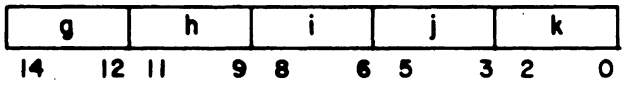
(Xk) = 0000 xxxx xxxx xxxx xxxx

(Xk) = 7777 xxxx xxxx xxxx xxxx

These formats include positive and negative numbers in either a partial or a complete underflow form. These cases are generally covered by one of the other special situations previously listed. If (Xk) is a partial underflow quantity and the coefficient is normalized, the execution proceeds as for a normal operand and the result is an unaltered copy of the original operand. The shift count delivered to the Bj register is zero. If (Xk) is a partial underflow quantity and the coefficient is not normalized, a complete underflow occurs.

If (Xk) is a complete underflow quantity, the execution is dominated by the fact that the coefficient is zero (described under that heading). The result is a complete underflow quantity delivered to the Xi register and a shift count of 48 (decimal) delivered to the Bj register.

25ijk ROUND NORMALIZE (Xk) TO Xi AND Bj



This instruction causes the normalize unit to read one operand from the Xk register, perform a rounding and then a normalizing operation in floating point format, and deliver the round normalized result to the Xi register. In addition, the normalize unit delivers a positive integer shift count to the Bj register. This shift count is the number of bit positions of shift required to normalize the original operand coefficient.

The rounding operation consists of adding a bit to the coefficient portion of the operand in a bit position immediately below the least significant bit position. This round bit has a value equal to the complement of the operand sign bit. The result is to increase the magnitude of the coefficient by one half the value of the least significant bit.

The normalizing operation consists of repositioning the coefficient and adjusting the exponent to leave the value of the resulting floating point quantity unaltered. The coefficient is displaced toward the higher order bit positions. The round bit is shifted along with the coefficient. The displacement is the minimum number of bit positions required to make bit 47 different from the sign bit 59. This places the most significant bit of the coefficient in the highest order bit position. The exponent is decreased by the number of bit positions shifted.

Two sample computations are listed in octal notation to illustrate the normalizing operation performed. The first example involves a positive floating point number and the second example involves a negative number.

(Xk) = 2034 0047 6500 0000 2262
(Xi) = 2026 4765 0000 0022 6240
(Bj) = 00 0006

(Xk) = 5734 7730 1277 7777 5515
(Xi) = 5751 3012 7777 7755 1537
(Bj) = 00 0006

SPECIAL CASE OPERAND

The normalize unit makes a special case test on (Xk) at the beginning of execution. If one of these special cases is present, the operand is copied unaltered to the Xi register. The shift count entered in the Bj register is zero for these cases. No condition flags are set in the PSD register. The special case formats are listed and consist of partial overflow, complete overflow, and indefinite forms.

(Xk) = 3777 xxxx xxxx xxxx xxxx
(Xk) = 4000 xxxx xxxx xxxx xxxx
(Xk) = 1777 xxxx xxxx xxxx xxxx
(Xk) = 6000 xxxx xxxx xxxx xxxx

COMPLETE UNDERFLOW

A complete underflow occurs whenever (Xk) is not a special case and the normalizing process results in an unpacked exponent more negative than -1777 (octal). The normalize unit output to Xi is blocked, which results in the Xi register receiving all zero bits. 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 the value which would be appropriate for normalizing the operand if the exponent were in range. No condition flags are set in the PSD register.

PARTIAL UNDERFLOW

A partial underflow occurs whenever (Xk) is not a special case and the normalizing process results in an unpacked exponent equal to -1777 (octal). The result is delivered to the Xi and Bj registers as for a normal case even though subsequent computation may detect this operand as an underflow case.

COEFFICIENT IS ZERO

A zero coefficient in the operand becomes nonzero with the addition of the round bit. The round bit is shifted to the left by 48 bit positions in the normalizing process to become the most significant bit of the result coefficient. The shift count delivered to Bj register is 48 (decimal) for this case. This case is superseded by one of the first two cases previously described if the operand is in a special case format, or if a complete underflow occurs.

UNDERFLOW OPERAND

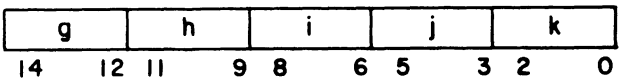
A special situation exists for executing this instruction if (Xk) is in one of the following underflow formats.

(Xk) = 0000 xxxx xxxx xxxx xxxx
(Xk) = 7777 xxxx xxxx xxxx xxxx

These formats include positive and negative numbers in either a partial or complete underflow form. These cases are generally covered by one of the other special situations previously listed. If (Xk) is a partial underflow quantity and the coefficient is normalized, the execution proceeds as for a normal operand and the result is an unaltered copy of the original operand. The shift count delivered to the Bj register is zero. If (Xk) is a partial underflow quantity and the coefficient is not normalized, a complete underflow occurs.

If (Xk) is a complete underflow quantity, the execution of this instruction is dominated by the fact that the coefficient is zero. The round bit is added and the shift count is 48 (decimal). This causes a complete underflow, and the result delivered to the Xi register is positive zero.

26ijk UNPACK (Xk) TO Xi AND Bj



This instruction causes the boolean unit to read one operand from the Xk register, unpack this word from floating point format, and deliver the coefficient to the Xi register and the exponent to the Bj register. The 60-bit word delivered to the Xi register consists of the lowest 48 bits unaltered from the original operand plus the upper 12 bits, each equal to the original sign bit. This is a signed integer equal to the value of the coefficient in the original operand. The 18-bit quantity delivered to the Bj register is a signed integer equal to the value of the exponent in the original operand. The 11-bit exponent field in the operand is altered to remove the bias and then sign extended to fill out the 18-bit quantity. The sign of the coefficient is removed in this process.

Four sample sets of operands and unpacked results are listed in octal notation to illustrate the operation performed. These examples contain the four combinations of coefficient sign and exponent sign.

(Xk) = 2034 4500 3333 2000 0077
(Xi) = 0000 4500 3333 2000 0077
(Bj) = 00 0034

(Xk) = 1743 4500 3333 2000 0077
(Xi) = 0000 4500 3333 2000 0077
(Bj) = 77 7743

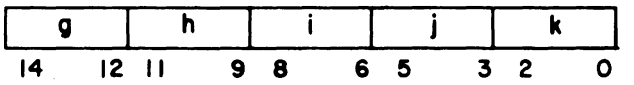
(Xk) = 5743 3277 4444 5777 7700
(Xi) = 7777 3277 4444 5777 7700
(Bj) = 00 0034

(Xk) = 6034 3277 4444 5777 7700
(Xi) = 7777 3277 4444 5777 7700
(Bj) = 77 7743

This instruction is intended for converting a number from floating point format to fixed point format as quickly as possible. This process is the reciprocal of the process used to implement the 27 instruction.

There are no special case tests made in the execution of this instruction. There are no condition flags set in the PSD register by the boolean unit. The special operand formats are treated in the same manner as a normal operand.

27ijk PACK (Xk) AND (Bj) TO Xi



This instruction causes the boolean unit to read (Xk) and (Bj), pack them into a single word in floating point format, and deliver this result to the Xi register. The coefficient for (Xi) is obtained from (Xk) treated as a signed integer. The exponent for (Xi) is obtained from (Bj) treated as a signed integer.

The lowest order 48 bits of (Xi) are copied directly from the lowest order 48 bits of (Xk). The sign bit in (Xi) is copied directly from the sign bit in (Xk). The exponent field in (Xi) is derived from (Bj) by extracting the lowest order 11 bits of (Bj) and modifying this quantity for exponent bias and coefficient sign.

Four sample sets of operands and packed results are listed in octal notation to illustrate the operation performed. These examples contain the four combinations of coefficient sign and exponent sign.

```
(Xk) = 0000 4500 3333 2000 0077
(Bj) = 00 0034
(Xi) = 2034 4500 3333 2000 0077

(Xk) = 0000 4500 3333 2000 0077
(Bj) = 77 7743
(Xi) = 1743 4500 3333 2000 0077

(Xk) = 7777 3277 4444 5777 7700
(Bj) = 00 0034
(Xi) = 5743 3277 4444 5777 7700

(Xk) = 7777 3277 4444 5777 7700
(Bj) = 77 7743
(Xi) = 6034 3277 4444 5777 7700
```

This instruction is intended for converting a number in fixed point format to floating point format as quickly as possible. This process is the reciprocal of the process used to implement the 26 instruction.

(Xk) HAS MAGNITUDE GREATER THAN 48 BITS

If (Xk) has more than 48 bits of significance, the higher order bits are ignored in packing this quantity into the floating point format. The lowest order 48 bits of (Xk) are masked out of the 60-bit word for the coefficient in (Xi). The sign bit in (Xk) is copied into (Xi) for the sign of the coefficient. The remaining bits of (Xk) are ignored.

(Bj) HAS MAGNITUDE GREATER THAN 10 BITS

If (Bj) has more than 10 bits of significance, an erroneous exponent is packed into floating point format for (Xi). In this case, the lowest order 11 bits of (Bj) are masked out of the 18-bit quantity. The highest order of these 11 bits is interpreted as the sign bit for the exponent. No error condition flags are set in the PSD register.

DESIGNATOR j IS ZERO

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

PACKING AN INDEFINITE QUANTITY

If the lowest order 11 bits of (Bj) all have a value of one, an indefinite quantity results in the floating point format. This is the case if (Bj) is a negative zero quantity. No error condition flags are set in the PSD register.

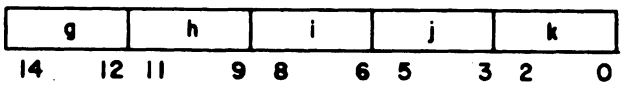
PACKING AN UNDERFLOW QUANTITY

An overflow quantity is generated in floating point format if (Bj) = 001777 (octal). No error condition flags are set in the PSD register.

PACKING AN OVERFLOW QUANTITY

An underflow quantity is generated in floating point format if (Bj) = 776000 (octal). No error condition flags are set in the PSD register.

30ijk FLOATING SUM OF (Xj) AND (Xk) TO Xi



This instruction causes the floating point add unit to read operands from two X registers, operate upon them to form a floating point sum, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are in floating point format and are not necessarily normalized. The result of (Xj) plus (Xk) is delivered to the Xi register in floating point format and is not necessarily normalized.

The operands are not rounded in this operation. The two operands are unpacked from floating point format and the exponents compared. The unpacked coefficients are positioned in a 99-bit ones complement adder to align bits of corresponding significance. A double precision ones complement sum is formed. A 48-bit result coefficient is read from the upper half of this sum. If an overflow of the highest order coefficient bit occurred, the result coefficient is read with a one-bit displacement to include this overflow bit. The result is corrected by one count in this case.

If the two operands have unlike signs, the result coefficient may have leading zeros. There is no normalize operation built into this instruction to correct this situation. A separate normalize instruction must be programmed if the result is to be kept in a normalized form.

This instruction is intended for use in floating point calculations where rounding of operands is not desired. This is the case in multiple precision arithmetic and in calculations involving error analysis.

CLEAN MISS

If the exponents of the two operands differ by more than 48 (decimal), the coefficient of the operand with the smaller exponent is shifted off the end of the double precision adder. If the exponent difference is exactly 48 (decimal), the two coefficients are aligned in a 96-bit field in the double precision adder with no bits matched. In either of these cases, the result is a copy of the operand with the larger exponent.

RESULT COEFFICIENT IS ZERO

If the two operands are of equal magnitude and opposite sign, the resulting sum has a zero coefficient. The exponent delivered to the Xi register is the same as the exponent for the operands even though the coefficient is zero. The sign of the result is positive. No error condition flags are set in the PSD register.

PARTIAL OVERFLOW

If the two operands are both in floating point range and one operand is at the upper limit, the resulting sum may overflow. The resulting exponent indicates the overflow condition. The coefficient is processed in a normal manner and the result is correct. No error indication is made and no condition flags are set in the PSD register. However, subsequent use of this number in a floating point unit results in overflow detection.

ONE OPERAND INDEFINITE

If either (or both) operand is indefinite, the result is indefinite. The resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

BOTH OPERANDS OVERFLOW WITH DIFFERENT SIGNS

If both operands have overflow exponents and the operand coefficients have different signs, the resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

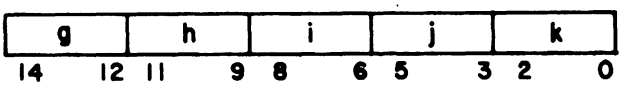
ONE OPERAND OVERFLOW

If either (or both) operand has an overflow exponent and the coefficient signs agree, the result is a complete overflow word. The resulting sign is the same as the sign of the operand with the overflow condition flag is set in the PSD register.

UNDERFLOW OPERAND

An operand with an underflow exponent is treated as a normal operand. No condition flags are set in the PSD register. If both operands are zero, the result is a positive zero word.

31ijk FLOATING DIFFERENCE OF (Xj) AND (Xk) TO Xi



This instruction causes the floating point add unit to read operands from two X registers, operate upon them to form a floating point difference, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are in floating point format and are not necessarily normalized. The result of (Xj) minus (Xk) is delivered to the Xi register in floating point format and is not necessarily normalized.

The operands are not rounded in this operation. The two operands are unpacked from floating point format and the exponents compared. The unpacked coefficients are positioned in a 99-bit ones complement adder to align bits of corresponding significance. A double precision ones complement difference is formed. A 48-bit result coefficient is read from the upper half of this difference. If an overflow of the highest order coefficient bit occurred, the result coefficient is read with one-bit displacement to include this overflow bit. The result exponent is corrected by one count in this case.

If the two operands have like signs, the result coefficient may have leading zeros. There is no normalize operation built into this instruction to correct this situation. A separate normalize instruction must be programmed if the result is to be kept in a normalized form.

This instruction is intended for use in floating point calculations where rounding of operands is not desired. This is the case in multiple precision arithmetic and in calculations involving error analysis.

CLEAN MISS

If the exponents of the two operands differ by more than 48 (decimal), the coefficient of the operand with the smaller exponent is shifted off the end of the double precision adder. If the exponent difference is exactly 48 (decimal), the two coefficients are aligned in the 96-bit field with no bits matched. In either of these cases, the result is a copy of the operand with the larger exponent.

RESULT COEFFICIENT IS ZERO

If the two operands are identical, the resulting difference has a zero coefficient. The resulting exponent is the same as the exponent for the operands even though the coefficient is zero. The sign of the result is positive. No error condition flags are set in the PSD register.

PARTIAL OVERFLOW

If the two operands are both in floating point range and one operand is at the upper limit, the resulting difference may overflow. The resulting exponent indicates the overflow condition. The coefficient is processed in a normal manner and the result is correct. No error indication is made and no condition flags are set in the PSD register. However, subsequent use of this number in a floating point unit results in overflow detection.

ONE OPERAND INDEFINITE

If either (or both) operand is indefinite, the result is indefinite. The resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

BOTH OPERANDS OVERFLOW WITH SAME SIGN

If both operands have overflow exponents and the operand coefficients have the same sign, the resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

BOTH OPERANDS OVERFLOW WITH DIFFERENT SIGNS

If both operands have overflow exponents and the coefficient signs differ, the result is a complete overflow word. The sign of the resulting word is the same as the sign of (Xj). The overflow condition flag is set in the PSD register.

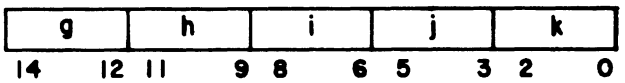
ONE OPERAND OVERFLOW

If either operand has an overflow exponent and the other operand is in floating point range or has an underflow exponent, the result is a complete overflow word. 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.

UNDERFLOW OPERAND

An operand with an underflow exponent is treated as a normal operand. No condition flags are set in the PSD register. If both operands are zero, the result is a positive zero word.

32ijk FLOATING DOUBLE PRECISION SUM OF (Xj) AND (Xk) TO Xi



This instruction causes the floating point add unit to read operands from two X registers, operate upon them to form a double precision floating point sum, and deliver the lower half of this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are in floating point format and are not necessarily normalized. The result of (Xj) plus (Xk) is delivered to the Xi register in floating point format and is not necessarily normalized.

The operands are not rounded in this operation. The two operands are unpacked from floating point format and the exponents compared. The unpacked coefficients are then positioned in a 99-bit ones complement adder to align bits of corresponding significance. A double precision ones complement sum is formed. A 48-bit result coefficient is read from the lower half of this sum. The result exponent is exactly 48 (decimal) less than the exponent which would be delivered with the upper half of the double precision sum.

If an overflow of the highest order coefficient bit occurred, the result coefficient is read with a one-bit displacement to take into account the overflow bit. The result exponent is corrected by one count in this case.

If the two operands have unlike signs, the double precision sum may have leading zeros. No normalize operation is built into this instruction to correct this situation. Whether this situation exists or not, there may be leading zeros in the lower half of the double precision sum. These zero bits are not detected. Therefore, the coefficient in the result may have leading zeros.

This instruction is intended for use in floating point calculations involving double precision or multiple precision. This instruction, together with the 30 instruction, forms a double precision sum in two X registers with no loss of significance.

CLEAN MISS

If the exponents of the two operands differ by more than 48 (decimal), the coefficient of the operand with the smaller exponent is shifted off the end of the double precision adder. If the exponent difference is exactly 48 (decimal), the two coefficients are aligned in the 96-bit field with no bits matched. In either of these cases, the result contains a coefficient from the 48-bit field corresponding to the lower half of a 96-bit sum. The resulting exponent is exactly 48 (decimal) less than the exponent which would result from the upper half of the 96-bit sum. If the difference of the exponents is greater than 48 (decimal), the lower half of this 96-bit sum has leading zeros. If the difference of the exponents is 96 (decimal) or greater, the lower half of the 96-bit sum is all zeros.

COEFFICIENT SUM IS ZERO

If the two operands are of equal magnitude and opposite sign, the resulting coefficient sum is zero. This condition is not sensed, and the exponent in the result has the same value as for a nonzero coefficient. The sign of the resulting zero coefficient is positive. No error condition flags are set in the PSD register.

PARTIAL OVERFLOW

If the two operands are in floating point range and one operand is at the upper limit, the result may overflow and cause the exponent for the upper half to go out of range. This condition is not sensed. The exponent for the lower half is 48 (decimal) less than this overflow value. The result is processed as a normal floating point result and no error condition flags are set in the PSD register.

PARTIAL UNDERFLOW

If the two operands are near the lower limit of the floating point range, the exponent for the lower half may be exactly -1777 (octal). This result is processed as a normal floating point number. No error condition flags are set in the PSD register. The resulting coefficient may be nonzero even though the exponent indicates an underflow condition. However, subsequent use of this number in a floating point unit may result in underflow detection.

COMPLETE UNDERFLOW

If the two operands are near the lower limit of the floating point range, the exponent for the lower half may be less than -1777 (octal). This results in a complete underflow word. The underflow condition flag is set in the PSD register.

UNDERFLOW OPERAND

An operand with an underflow exponent is treated as a normal operand. No condition flags are set in the PSD register if the other operand is sufficiently large so that the result does not underflow the floating point range. If the other operand is near the lower limit of the range, the result may be either a partial or a complete underflow.

ONE OPERAND INDEFINITE

If either (or both) operand is indefinite, the result is indefinite. The resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

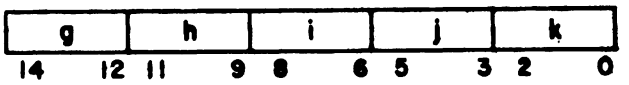
BOTH OPERANDS OVERFLOW WITH DIFFERENT SIGNS

If both operands have overflow exponents and the coefficients have different signs, the resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

ONE OPERAND OVERFLOW

If either (or both) operand has an overflow exponent and the coefficient signs agree, the result is a complete overflow word. 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.

33ijk FLOATING DOUBLE PRECISION DIFFERENCE OF (Xj) AND (Xk) TO Xi



This instruction causes the floating point add unit to read operands from two X registers, operate upon them to form a double precision floating point difference, and deliver the lower half of this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are in floating point format and are not necessarily normalized. The result of (Xj) minus (Xk) is delivered to the Xi register in floating point format and is not necessarily normalized.

The operands are not rounded in this operation. The two operands are unpacked from floating point format and the exponents are compared. The unpacked coefficients are positioned in a 99-bit ones complement adder to align bits of corresponding significance. A double precision ones complement difference is formed. A 48-bit result coefficient is read from the lower half of this difference. The result exponent is exactly 48 (decimal) less than the exponent which would result from the upper half of the double precision difference. If an overflow of the highest order coefficient bit occurred, the result coefficient is read with a one-bit displacement to take into account the overflow bit. The result exponent is then corrected by one count in this case.

If the two operands have like signs, the double precision difference may have leading zeros. No normalize operation is built into this instruction to correct this situation. Whether this situation exists or not, there may be leading zeros in the lower half of the double precision difference. These zero bits are not detected, and the coefficient in the result may have leading zeros.

This instruction is intended for use in floating point calculations involving double precision or multiple precision. This instruction, together with the 31 instruction, forms a double precision difference in two X registers with no loss of significance.

CLEAN MISS

If the exponents of the two operands differ by more than 48 (decimal), the coefficient of the operand with the smaller exponent is shifted off the end of the double precision adder. If the exponent difference is exactly 48 (decimal), the two coefficients are aligned in the 96-bit field with no bits matched. In either of these cases, the result

contains a coefficient from the 48-bit field corresponding to the lower half of a 96-bit difference. The resulting exponent is exactly 48 (decimal) less than the exponent which would result for the upper half of the 96-bit difference. If the difference of the operand exponents is greater than 48 (decimal), the lower half of this 96-bit difference has leading zeros. If the difference of the operand exponents is 96 (decimal) or greater, the lower half of the 96-bit difference is all zeros.

COEFFICIENT SUM IS ZERO

If the two operands are identical, the result is zero. This condition is not sensed. The resulting exponent is the same value as for a nonzero coefficient. The sign of the resulting zero coefficient is positive. No error condition flags are set in the PSD register.

PARTIAL OVERFLOW

If the two operands are in floating point range and one operand is at the upper limit, the resulting double precision difference may overflow and cause the exponent for the upper half to go out of range. This condition is not sensed. The exponent for the lower half is 48 (decimal) less than this overflow value. The result is processed as a normal floating point result, and no error condition flags are set in the PSD register.

PARTIAL UNDERFLOW

If the two operands are near the lower limit of the floating point range, the resulting exponent for the lower half may be exactly -1777 (octal). This result is processed as a normal floating point number. No error condition flags are set in the PSD register. The resulting coefficient may be nonzero even though the exponent in the resulting word indicates an underflow condition. However, subsequent use of this number in a floating point unit may result in underflow detection.

COMPLETE UNDERFLOW

If the two operands are near the lower limit of the floating point range, the resulting exponent for the lower half may be less than -1777 (octal). The result is a complete underflow word. The underflow condition flag is set in the PSD register.

UNDERFLOW OPERAND

An operand with an underflow exponent is treated as a normal operand. No condition flags are set in the PSD register if the other operand is large enough so that the result does not underflow the floating point range. If the other operand is near the lower limit, the result may be either a partial or complete underflow.

ONE OPERAND INDEFINITE

If either (or both) operand is indefinite, the result is indefinite. The resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

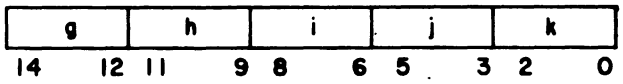
BOTH OPERANDS OVERFLOW WITH SAME SIGN

If both operands have overflow exponents and the coefficient signs agree, the resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

ONE OPERAND OVERFLOW

If either (or both) operand has an overflow exponent and the coefficient signs disagree, the result is a complete overflow word. The sign of the resulting word is the same as the sign of (Xj) if (Xj) has the overflow exponent. The sign of the resulting word is the complement of the sign of (Xk) if (Xk) has the overflow exponent. The overflow condition flag is set in the PSD register.

34ijk ROUND FLOATING SUM OF (Xj) AND (Xk) TO Xi



This instruction causes the floating point add unit to read operands from two X registers, operate upon them to form a rounded floating point sum, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are in floating point format and are not necessarily normalized. The result is delivered to the Xi register in floating point format and is not necessarily normalized.

The floating point add unit unpacks the two operands from floating point format and compares the exponents. The unpacked coefficients are then positioned in a 99-bit ones complement adder to align bits of corresponding significance. The two coefficients are rounded according to the following rules. A double precision ones complement sum is formed. A 48-bit result coefficient is read from the upper half of this sum. If an overflow of the highest order coefficient bit occurred, the result coefficient is read with a one-bit displacement to include this overflow bit. The result exponent is corrected by one count in this case.

If the two operands have unlike signs, the result coefficient may have leading zeros. No normalize operation is built into this instruction to correct this situation. A separate normalize instruction must be programmed if the result is to be kept in a normalized form.

This instruction is intended for use in floating point calculations involving single precision accuracy. For multiple precision calculations, the 30 and 32 instructions must be used.

ROUNDING

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 with 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 increases the magnitude of the coefficient by 1/2 of the least significant bit. A second round bit is added to the other coefficient if both operands were normalized or had unlike signs.

The amount of error introduced by the rounding operation is a function of the relative magnitudes of the operands. If the two exponents differ significantly, the rounding is relatively free of bias and the maximum error is bounded by +1/2 and -1/2 of the least significant bit of the larger coefficient. If the exponents differ only slightly, the rounding introduces some bias because of the discrete combinations involved. An additional complication is introduced by the possibility of overflow. If an overflow occurs, the result coefficient is displaced by one bit position to include the overflow bit. This introduces a negative bias on the rounding operation whenever it occurs.

CLEAN MISS

If the exponents differ by more than 48 (decimal), the coefficient with the smaller exponent is shifted off the end of the double precision adder. The result is a copy of the operand with the larger exponent. If the exponents differ by exactly 48 (decimal), the two coefficients are aligned in a 96-bit field with no bits matched. However, the round bit for the larger number is aligned with the highest order bit for the smaller number. The result is a rounded version of the operand with the larger exponent.

RESULT COEFFICIENT IS ZERO

If the two operands are of equal magnitude and opposite sign, the result has a zero coefficient. The resulting exponent is the same as the exponent for the operands even though the coefficient is zero. The sign of the result is positive. No error condition flags are set in the PSD register.

PARTIAL OVERFLOW

If the two operands are both in floating point range and one operand is at the upper limit, the result may overflow. The resulting exponent indicates the overflow condition. The coefficient is processed in a normal manner and the resulting floating point number is correct. No error indication is made and no condition flags are set in the PSD register. However, subsequent use of this number in a floating point unit results in overflow detection.

ONE OPERAND INDEFINITE

If either (or both) operand is indefinite, the result is indefinite. The resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

BOTH OPERANDS OVERFLOW WITH DIFFERENT SIGNS

If both operands have overflow exponents and the coefficients have different signs, the resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

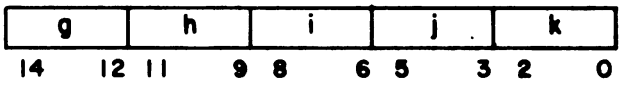
ONE OPERAND OVERFLOW

If either (or both) operand has an overflow exponent and the coefficient signs agree, the result is a complete overflow word. 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.

UNDERFLOW OPERAND

An operand with an underflow exponent is treated as a normal operand. No condition flags are set in the PSD register. If both operands are positive or negative zero in any combination, the result is a positive zero word.

35ijk ROUND FLOATING DIFFERENCE OF (Xj) AND (Xk) TO Xi



This instruction causes the floating point add unit to read operands from two X registers, operate upon them to form a rounded floating point difference, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are in floating point format and are not necessarily normalized. The result of (Xj) minus (Xk) is delivered to the Xi register in floating point format and is not necessarily normalized.

The floating point add unit unpacks the two operands from floating point format and compares the exponents. The unpacked coefficients are positioned in a 99-bit ones complement adder to align bits of corresponding significance. The two coefficients are rounded according to the following rules. A double precision ones complement difference is formed. A 48-bit result coefficient is read from the upper half of this difference. If an overflow of the highest order coefficient bit occurred, the result coefficient is read with a one-bit displacement to include this overflow bit. The result exponent is corrected by one count in this case.

If the two operands have like signs, the result coefficient may have leading zeros. No normalize operation is built into this instruction to correct this situation. A separate normalize instruction must be programmed if the result is to be kept in a normalized form.

This instruction is intended for use in floating point calculations involving single precision accuracy. For multiple precision calculations, the 31 and 33 instructions must be used.

ROUNDING

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 with 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 increases the magnitude of the coefficient by 1/2 the value of the least significant bit. A second round bit is added to the other coefficient if both operands were normalized or had like signs.

The amount of error introduced by the rounding operation is a function of the relative magnitudes of the operands. If the two exponents differ significantly, the rounding is relatively free of bias and the maximum error is bounded by +1/2 and -1/2 of the least significant bit of the larger coefficient. If the exponents differ only slightly, the rounding introduces some bias because of the discrete combinations involved. An additional complication is introduced by the possibility of overflow. If an overflow occurs, the result coefficient is displaced by one bit position to include the overflow bit. This introduces a negative bias on the rounding operation whenever it occurs.

CLEAN MISS

If the exponents differ by more than 48 (decimal), the coefficient with the smaller exponent is shifted off the end of the double precision adder. The result is a copy of the operand with the larger exponent. If the exponents differ by exactly 48 (decimal), the two coefficients are aligned in the 96-bit field with no bits matched. However, the round bit for the larger number is aligned with the highest order bit for the smaller number. The result is a rounded version of the operand with the larger exponent.

RESULT COEFFICIENT IS ZERO

If the two operands are identical, the result has a zero coefficient. The resulting exponent is the same as the exponent for the operands even though the coefficient is zero. The sign of the result is positive. No error condition flags are set in the PSD register.

PARTIAL OVERFLOW

If the two operands are both in floating point range and one operand is at the upper limit, the resulting difference may overflow. The resulting exponent indicates the overflow condition. The coefficient is processed in a normal manner and the result is correct. No error indication is made and no condition flags are set in the PSD register. However, subsequent use of this number in a floating point unit results in overflow detection.

One operand indefinite

If either (or both) operand is indefinite, the result is indefinite. The resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

BOTH OPERANDS OVERFLOW WITH SAME SIGN

If both operands have overflow exponents and the coefficients have the same sign, the resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

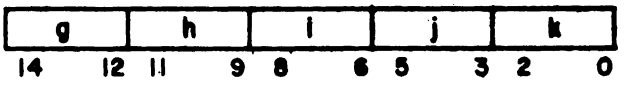
ONE OPERAND OVERFLOW

If either (or both) operand has an overflow exponent and the coefficient signs disagree, the result is a complete overflow word. The sign of the resulting word is the same as the sign of (Xj) if (Xj) has the overflow exponent. The sign of the result is the complement of the sign of (Xk) if (Xk) has the overflow exponent. The overflow condition flag is set in the PSD register.

UNDERFLOW OPERAND

An operand with an underflow exponent is treated as a normal operand. No condition flags are set in the PSD register. If both operands are positive or negative zero in any combination, the result is a positive zero word.

36ijk INTEGER SUM OF (Xj) AND (Xk) TO Xi



This instruction causes the long add unit to read operands from two X registers, operate upon them to form a 60-bit integer sum, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are signed integers. The resulting integer sum is delivered to the Xi register.

The long add unit executes this instruction in a 60-bit ones complement mode. The two operands are read directly to a 60-bit integer adder. The resulting sum is delivered directly to the Xi register. Overflow is not detected.

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

BOTH OPERANDS ZERO

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

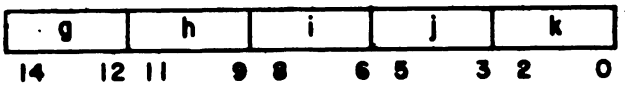
DESIGNATORS j AND k HAVE THE SAME VALUE

If the j and k designators have the same value, the designated 60-bit operand is added to itself, and the resulting sum is delivered to the Xi register.

DESIGNATORS i AND j OR k HAVE THE SAME VALUE

If the i designator has the same value as the j or k designator, this instruction becomes a replace add instruction. The initial (Xi) is added to the other operand. The result is stored back in the Xi register.

37ijk INTEGER DIFFERENCE OF (Xj) AND (Xk) TO Xi



This instruction causes the long add unit to read operands from two X registers, operate upon them to form a 60-bit integer difference, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are signed integers. The result of (Xj) minus (Xk) is delivered to the Xi register.

The long add unit executes this instruction in a 60-bit ones complement mode. The two operands are read directly to a 60-bit integer adder. (Xj) is transmitted unaltered from the register to the adder. (Xk) is complemented in the transmission from the register to the adder. The resulting sum of (Xj) and the complement of (Xk) is delivered directly to the Xi register. Overflow is not detected.

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.

BOTH OPERANDS ZERO

If (Xj) is negative zero and (Xk) is positive zero, the result is negative zero. For the other three combinations of positive and negative zero operands, the result is positive zero.

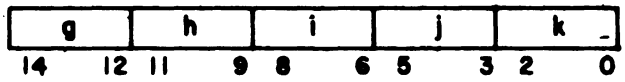
DESIGNATORS j AND k HAVE THE SAME VALUE

If the j and k designators have the same value, the designated 60-bit operand is subtracted from itself. The result is positive zero.

DESIGNATORS i AND j OR k HAVE THE SAME VALUE

If the i designator has the same value as the j or k designator, this instruction becomes a replace subtract instruction. The initial (Xi) is read as an operand, and the resulting difference is then stored in the same register.

40ijk FLOATING PRODUCT OF (Xj) AND (Xk) TO Xi



This instruction causes the multiply unit to read operands from two X registers, operate upon them to form a floating point product, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are in floating point format and are not necessarily normalized. The result is delivered to the Xi register in floating point format. If both operands are normalized, the result is also normalized. If both operands are not normalized, the result is not normalized.

The operands are not rounded in this operation. The two operands are unpacked from floating point format. The exponents are added with a correction factor to determine 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 extracted to form the coefficient for the result. An alternate output path provides a one-bit displacement to normalize the result coefficient if the original operands are normalized and the product has only 95 significant bits. The resulting exponent is corrected by one count in this case.

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

This instruction is intended for use in floating point calculations where rounding of operands is not desired. This is the case in multiple precision arithmetic and in calculations involving error analysis.

RESULT COEFFICIENT IS ZERO

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. The exponent for the result is processed without regard to the zero coefficient. This results in a zero coefficient and a nonzero exponent. No error flags are set in the PSD register.

PARTIAL OVERFLOW

A partial overflow occurs whenever the resulting exponent is +1777 (octal) and the double precision product has 96 bits of significance. No error condition flags are set in the PSD register. The result is delivered to the Xi register in a normal manner. However, subsequent use of this result in a floating point unit results in overflow

detection. If the resulting exponent is +1777 (octal) and the alternate output path is used, the exponent is reduced one count and the result is in floating point range.

COMPLETE OVERFLOW

A complete overflow occurs whenever the resulting exponent is greater than +1777 (octal). This results in a complete overflow word with the sign being calculated the same as for a result that is in floating point range. The overflow condition flag is set in the PSD register.

PARTIAL UNDERFLOW

A partial underflow occurs whenever the resulting exponent is -1776 (octal) and the alternate output path is used to normalize the coefficient. The exponent is reduced one count to create an underflow exponent with a valid coefficient. No condition flags are set in the PSD register. However, subsequent use of this result in a floating point unit may result in underflow detection.

COMPLETE UNDERFLOW

A complete underflow occurs whenever the resulting exponent is less than -1776 (octal). This results in a complete underflow word. The underflow condition flag is set in the PSD register.

ONE OPERAND INDEFINITE

If either (or both) operand is indefinite, the result is indefinite. The resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

ONE OPERAND OVERFLOW

If either (or both) operand has an overflow exponent, the result is a complete overflow word. The sign of the result is calculated the same as for operands in range. The overflow condition flag is set in the PSD register.

UNDERFLOW TIMES OVERFLOW

If one operand has an underflow exponent and the other operand has an overflow exponent, the result is indefinite. The resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

ONE OPERAND UNDERFLOW

If one operand has an underflow exponent ($\pm$ 0) while the other operand exponent is in range (an indefinite, overflow, or underflow does not exist) the result is a complete underflow word. This causes the underflow condition flag in the PSD register to set.

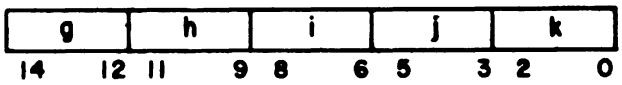
BOTH OPERANDS UNDERFLOW

If both operands have an underflow exponent ($\pm$ 0) and both coefficients are normalized, the result is the same as described for One Operand Underflow.

If both operands have underflow exponents and both coefficients are not normalized, the integer multiply hardware is enabled. In this case, the underflow condition flag in the PSD register is not set and the integer result delivered to Xi is 48 bits with sign extension to 60 bits.

Since the result would be taken from the upper half of the 96-bit coefficient product, the result delivered to Xi will be all zeros unless the integer product exceeds 42 bits. The 42 instruction, which returns the lower 48 bits of the 96-bit product, is the most applicable instruction for integer multiply. The 40 instruction may be used to check for coefficient overflow. This is an integer product exceeding 48 bits.

41ijk ROUND FLOATING PRODUCT OF (Xj) AND (Xk) TO Xi



This instruction causes the multiply unit to read operands from two X registers, operate upon them to form a rounded floating point product, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are in floating point format and are not necessarily normalized. The result is delivered to the Xi register in floating point format. If both operands are normalized, the result is also normalized. If both operands are not normalized, the result is not normalized.

The multiply unit unpacks the two operands from floating pointing format. The exponents are added with a correction factor to determine the exponent for the result. The coefficients are multiplied as signed integers to form a 96-bit integer product. A rounding bit is added in bit position 46 of this product. The upper half of this product is extracted to form the coefficient for the result. An alternate output path provides a one-bit displacement to normalize the result coefficient if the original operands are normalized and the product has only 95 significant bits. The resulting exponent is corrected by one count in this case.

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

This instruction is intended for use in single precision floating point calculations. For multiple precision calculations, the 40 and 42 instructions must be used.

Rounding of the result coefficient occurs in the final addition of the partial products to form a 96-bit double precision result. The rounding is accomplished by adding a bit in position 46 of the adder. This additional bit reduces the maximum amount of truncation error and also the average bias.

The special situations for this instruction are the same as the special situations for the 40 instructions.

42ijk FLOATING DOUBLE PRECISION PRODUCT OF (Xi) AND (Xk) TO Xi

This instruction causes the multiply unit to read operands from two X registers, operate upon them to form a floating point double precision product, and deliver the lower half of this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are in floating point format and are not necessarily normalized. The lower half of the double precision product is delivered to the Xi register in floating point format and is not necessarily normalized.

The operands are not rounded in this operation. The two operands are unpacked from floating point format. The exponents are added to determine the exponent for the result. The result exponent is exactly 48 less than the exponent for a 40 instruction. The coefficients are multiplied as signed integers to form a 96-bit integer product. The lower half of this product is extracted to form the coefficient for the result. An alternate output path provides a one-bit displacement when both operands are normalized and the double precision product has only 95 significant bits. The resulting exponent is corrected by one count if this path is used.

If the two operands are not both normalized, the resulting double precision product has less than 96 significant bits. The alternate output path is never used in this case. No test is made for the position of the most significant bit in the product. The lower 48 bits are always read from the 96-bit product register in this case.

This instruction is intended for use in multiple precision floating point calculations. This instruction is also intended for integer multiplication where both operands have an exponent value of ± zero and the coefficients are not both normalized. The integer result sent to the Xi register is 48 bits with sign extension to 60 bits. If the result exceeds 48 bits (coefficient overflow), the hardware does not detect this overflow. A coefficient overflow check can be made by executing a 40 or 41 instruction using the same two operands. If the result is nonzero, a coefficient overflow condition exists. An integer multiply operation is not intended to be used with normalized operands. If both operands are normalized, the integer multiply hardware is not enabled and underflow condition flag is set in the PSD register.

NOTE

To ensure that floating point operands are not mistaken for integers, normalize all floating point quantities that are used as operands.

PARTIAL OVERFLOW

A partial overflow occurs whenever the resulting exponent is +1777 (octal) and the double precision product has 96 bits of significance. No error condition flags are set in the PSD register. The result is delivered to the Xi register in a normal manner. However, subsequent use of this result in a floating point unit results in overflow detection. If the resulting exponent is +1777 (octal) and the alternate output path is used, the exponent is reduced one count and the result is in floating point range.

COMPLETE OVERFLOW

A complete overflow occurs whenever the resulting exponent is greater than +1777 (octal). This results in a complete overflow word with the sign being calculated the same as for a result that is in floating point range. The overflow condition flag is set in the PSD register.

PARTIAL UNDERFLOW

A partial underflow occurs whenever the resulting exponent is -1776 (octal) and the alternate output path is used. The exponent is reduced one count and creates an underflow exponent with a valid coefficient. No condition flags are set in the PSD register. However, subsequent use of this result in a floating point unit may result in underflow detection.

COMPLETE UNDERFLOW

A complete underflow occurs whenever the resulting exponent is less than -1776 (octal). This results in a complete underflow word. The underflow condition flag is set in the PSD register.

ONE OPERAND INDEFINITE

If either (or both) operand is indefinite, the result is indefinite. The resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

ONE OPERAND OVERFLOW

If either (or both) operand has an overflow exponent, the result is a complete overflow word. The sign of the result is calculated the same as for operands in range. The overflow condition flag is set in the PSD register.

ONE OPERAND UNDERFLOW

If one operand has an underflow exponent ($\pm$ 0) while the other operand exponent is in range (an indefinite, overflow, or underflow does not exist) the result is a complete underflow word. This causes the underflow condition flag in the PSD register to set.

BOTH OPERANDS UNDERFLOW

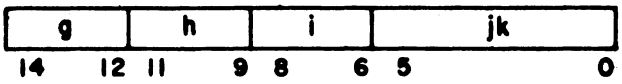
If both operands have an underflow exponent ($\pm$ 0) and both coefficients are normalized, the result is the same as described for One Operand Underflow.

If both operands have underflow exponents and both coefficients are not normalized, the integer multiply hardware is enabled. In this case, the underflow condition flag in the PSD register is not set and the integer result delivered to Xi is 48 bits with sign extension to 60 bits.

UNDERFLOW TIMES OVERFLOW

If one operand has an underflow exponent and the other operand has an overflow exponent, the result is indefinite. The resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

43ijk FORM MASK OF jk BITS TO Xi



This instruction causes the shift unit to generate a masking word using the j and k designators as parameters. No operands are read from operating registers. The j and k designators are treated as a single 6-bit quantity to designate the width of the masking field. A field of ones, beginning at the highest order end of the word, is extended downward on a background of zeros. The completed masking word consists of one bits in the highest order jk bit positions and zero bits in the remainder of the word. This masking word is then delivered to the Xi register. The following are sample parameters.

j = 2
k = 4
(Xi) = 7777 7760 0000 0000 0000

This instruction is intended for generating variable width masks for logical operations. This instruction, together with a shift instruction, generally creates an arbitrary field mask faster than reading a pregenerated mask from storage.

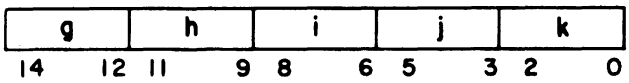
DESIGNATORS j AND k ARE ZERO

If the j and k designators are zero, a word containing all zero bits is written into the Xi register. The timing is the same as for the normal case.

QUANTITY jk GREATER THAN 60 (DECIMAL)

If the quantity jk is 60 (decimal) or greater, a word containing all one bits is written into the Xi register.

44ijk FLOATING DIVIDE (Xj) BY (Xk) TO Xi



This instruction causes the divide unit to read operands from two X registers, operate upon them to form a floating point quotient, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are in floating point format. The result of dividing (Xj) by (Xk) is delivered to the Xi register. If both operands are normalized, the quotient is also normalized. The remainder from the division process is discarded.

The operands are not rounded in this operation. The two operands are unpacked from floating point format. The exponents are subtracted with a correction factor to determine the exponent for the result. The coefficient from (Xj) is positioned in a dividend register. The coefficient from (Xk) is trial subtracted repeatedly from the dividend, and the dividend is shifted to form the quotient bits. The quotient bits are assembled in a quotient register. When 48 bits of the quotient have been assembled, they are packed with the result exponent into floating point format and delivered to the Xi register.

If the dividend is not normalized, the quotient may not be normalized. However, the quotient is correct even though there may be leading zeros in the coefficient. If the divisor is not normalized, the quotient may be incorrect. If the coefficient for (Xj) is larger than the coefficient for (Xk) by a factor of two or more, the quotient is incorrect.

This instruction is intended for use in floating point calculations where rounding of operands is not desired. In multiple precision division, this instruction must be followed by a multiplication of the quotient by the divisor and subtracted from the dividend in order to reconstruct the remainder.

QUOTIENT IS INCORRECT

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 is incorrect. The quotient is disregarded and the resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

PARTIAL OVERFLOW

A partial overflow occurs whenever the resulting exponent is +1777 (octal). No error condition flags are set in the PSD register, and the result is delivered to the Xi register in a normal manner. However, subsequent use of this result in a floating point unit results in overflow detection.

COMPLETE OVERFLOW

A complete overflow occurs whenever the resulting exponent is greater than +1777 (octal). This results in a complete overflow word with the sign being calculated the same as for a result that is in floating point range. The overflow condition flag is set in the PSD register.

PARTIAL UNDERFLOW

A partial underflow occurs whenever the resulting exponent is -1777 (octal). No error condition flags are set in the PSD register, and the result is delivered to the Xi register in a normal manner. However, subsequent use of this result in a floating point unit may result in underflow detection.

COMPLETE UNDERFLOW

A complete underflow occurs whenever the resulting exponent is less than -1777 (octal). This results in a complete underflow word. The underflow condition flag is set in the PSD register.

ONE OPERAND INDEFINITE

If either (or both) operand is indefinite, the result is indefinite. The resulting word is positive indefinite with a zero coefficient. The indefinite condition flag is set in the PSD register.

(Xj) IS OVERFLOW WORD

If (Xj) has an overflow exponent and (Xk) is in range or has an underflow exponent, the result is a complete overflow word. The sign of the result is calculated the same as for operands in range. The overflow condition flag is set in the PSD register.

(Xj) IS UNDERFLOW WORD

If (Xj) has an underflow exponent and (Xk) is in range or has an overflow exponent, the result is a complete underflow word. The underflow condition flag is set in the PSD register.

(Xk) IS OVERFLOW WORD

If (Xk) has an overflow exponent and (Xj) is in range, the result is a complete underflow word. The underflow condition flag is set in the PSD register.

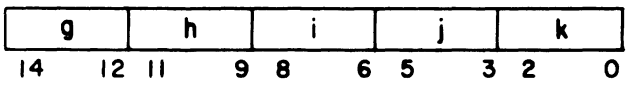
(Xk) IS UNDERFLOW WORD

If (Xk) has an underflow exponent and (Xj) is in range, the result is a complete overflow word. The sign of the result is calculated the same as for operands in range. The overflow condition flag is set in the PSD register.

UNDERFLOW DIVIDED BY UNDERFLOW AND OVERFLOW DIVIDED BY OVERFLOW

These combinations result in a positive indefinite word with a zero coefficient. The indefinite condition flag is set in the PSD register.

45ijk ROUND FLOATING DIVIDE (Xj) BY (Xk) TO Xi



This instruction causes the divide unit to read operands from two X registers, operate upon them to form a rounded floating point quotient, and deliver this result to a third X register. The operands for this instruction are (Xj) and (Xk). These operands are in floating point format. The result of dividing (Xj) by (Xk) is delivered to the Xi register. If both operands are normalized, the quotient is also normalized. The remainder from the division process is discarded.

The two operands are unpacked from floating point format in this operation. The exponents are subtracted with a correction factor to determine the exponent for the result. The coefficient from (Xj) is positioned in a dividend register. This quantity is modified by adding a round bit just below the lowest order bit of the coefficient from (Xj). This round bit increases the magnitude of the dividend by 1/2 the value of the least significant bit. The coefficient from (Xk) is trial subtracted repeatedly from the dividend, and the dividend is shifted to form the quotient bits. The quotient bits are assembled in a quotient register. When 48 bits of the quotient have been assembled, they are packed with the result exponent into floating point format and delivered to the Xi register.

If the dividend is not normalized, the quotient may not be normalized. However, the quotient is correct even though there may be leading zeros in the coefficient. If the divisor is not normalized, the quotient may be incorrect. If the coefficient for (Xj) is larger than the coefficient for (Xk) by a factor of two or more, the quotient is incorrect.

This instruction is intended for use in single precision floating point calculations where rounding of operands is desired to reduce truncation errors.

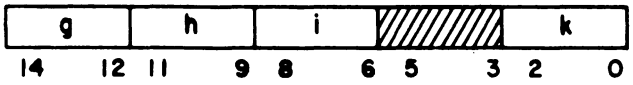
The rounding step occurs in the dividend register just prior to the first trial subtraction. A round bit is added to the dividend which has the effect of increasing the dividend by 1/2 the value of the least significant bit. The effect on the quotient varies depending upon the value of the divisor and upon the truncation point in the quotient. If the dividend is smaller than the divisor, the quotient is truncated one bit position lower than if the dividend is equal to or larger than the divisor. These effects cause the rounding to vary in the quotient from a value of 1/4 of the least significant bit in the result to almost one.

The special situations for this instruction are the same as the special situations listed for the 44 instruction.

46xxx PASS

This instruction causes no action in any functional unit. It is used to fill program instruction words where necessary to match jump destinations with word boundaries. The i, j, and k designators are normally zero in this instruction. However, these designators are ignored and a nonzero value has no effect.

47ixk POPULATION COUNT OF (Xk) TO Xi



This instruction causes the population count unit to read one operand from the Xk register, count the number of one bits in this word, and store this count in the Xi register. The word delivered to the Xi register is a positive integer. 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.

This instruction is intended for use in data processing where a degree of coincidence is desired.

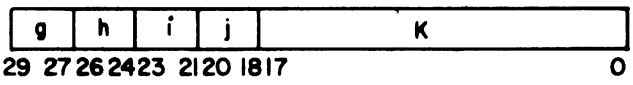
DESIGNATOR j NOT ZERO

The j designator is normally zero. However, a nonzero value has no effect on the results.

DESIGNATORS i AND k HAVE THE SAME VALUE

If the i and k designators have the same value, the operand is read from the designated X register and the count is stored back in the same X register. The timing is the same as for the normal case.

50ijK SET Ai TO (Aj) + K



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction causes the increment unit to read an operand from the Aj register, form the sum of (Aj) + K, and deliver this result to the Ai register. If the i designator is nonzero, a storage reference is made to SCM using this resulting sum as the relative storage address. The type of storage reference is a function of the i designator value.

0: No storage reference
1, 2, 3, 4, 5: Read from SCM to the Xi register
6, 7: Write into SCM from the Xi register

The increment unit forms the sum of (Aj) + K in an 18-bit ones complement mode. The resulting sum is simultaneously delivered to the Ai register and to the SAS, if required. The SAS treats this quantity as an 18-bit positive integer address. This address is relative to the beginning of the SCM field for the current program. If the i designator value causes a read from SCM, the result arrives at the Xi register a minimum of 6 clock periods later than the result delivered to the Ai register. This X register is reserved until the data arrives from storage. If the i designator value causes a write into SCM, the 60-bit word being stored is read from the Xi register into SCM in the same clock period in which this instruction issues. This X register may be used for unrelated computation in the next clock period.

This instruction is intended for fetching operands from storage for computation and for delivering results back into storage.

LAST PARCEL

This instruction normally requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If this instruction begins in the last parcel, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

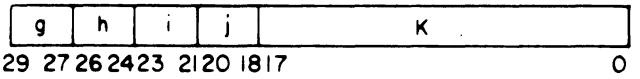
SCM ADDRESS OUT OF RANGE

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 causes the current program sequence to terminate with an exchange jump to (EEA). If the reference involved writing into an X register, a positive zero word is written into the X register before the interrupt occurs. If the reference involved writing into SCM, the memory sequence is aborted.

DESIGNATORS i AND j ARE THE SAME

If the i and j designators have the same value, the initial contents of the designated A register is sent to the increment unit and the result is stored back into the same A register.

51ijK SET Ai TO (Bj) + K



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction causes the increment unit to read an operand from the Bj register, form the sum of (Bj) + K, and deliver this result to the Ai register. If the i designator is nonzero, a storage reference is made to SCM using this resulting sum as the relative storage address. The type of storage reference is a function of the i designator value.

0: No storage reference
1, 2, 3, 4, 5: Read from SCM to the Xi register
6, 7: Write into SCM from the Xi register

The increment unit forms the sum of (Bj) + K in an 18-bit ones complement mode. The resulting sum is simultaneously delivered to the Ai register and to the SAS, if required. The SAS treats this quantity as an 18-bit positive integer address. This address is relative to the beginning of the SCM field for the current program. If the i designator value causes a read from SCM, the result arrives at the Xi register a minimum of 6 clock periods later than the result delivered to the Ai register. This X register is reserved until the data arrives from storage. If the i designator causes a write into SCM, the 60-bit word being stored is read from the Xi register into SCM

in the same clock period in which this instruction issues. This X register may be used for unrelated computation in the next clock period.

This instruction is intended for fetching operands from storage for computation and for delivering results back into storage.

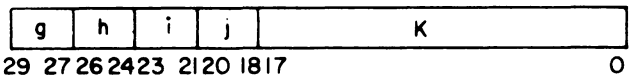
LAST PARCEL

This instruction normally requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If this instruction begins in the last parcel, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

SCM ADDRESS OUT OF RANGE

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 causes the current program sequence to terminate with an exchange jump to (EEA). If the reference involved writing into an X register, a positive zero word is written into the X register before the interrupt occurs. If the reference involved writing into SCM, the memory sequence is aborted.

52ijK SET Ai TO (Xj) + K



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction causes the increment unit to read an operand from the Xj register, form the sum of (Xj) + K, and deliver this result to the Ai register. If the i designator is nonzero, a storage reference is made to SCM using this resulting sum as the relative storage address. The type of storage reference is a function of the i designator value.

0: No storage reference

1, 2, 3, 4, 5: Read from SCM to the Xi register

6, 7: Write into SCM from the Xi register.

The increment unit forms the sum of (Xj) + K in an 18-bit ones complement mode. Only the lower order 18 bits of (Xj) are transmitted to the increment unit. The resulting sum is simultaneously delivered to the Ai register and to the SAS, if required. The SAS treats this quantity as an 18-bit positive integer address. This address is relative to the beginning of the SCM field for the current program. If the i designator value causes a read from SCM, the result arrives at the Xi register a minimum of 6 clock periods later than the result delivered to the Ai register. This X register is reserved until the data arrives from storage. If the i designator value causes a write into SCM, the 60-bit word being stored is read from the Xi register into SCM in the same clock period in which this instruction issues. This X register may be used for unrelated computation in the next clock period.

This instruction is intended for fetching operands from storage for computation and for delivering results back into storage.

LAST PARCEL

This instruction normally requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If this instruction begins in the last parcel, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

SCM ADDRESS OUT OF RANGE

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 causes the current program sequence to terminate with an exchange jump to (EEA). If the reference involved writing into an X register, a positive zero word is written into the X register before the interrupt occurs. If the reference involved writing into SCM, the memory sequence is aborted.

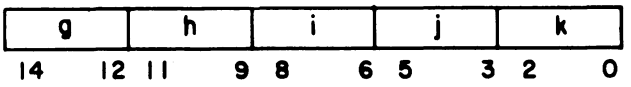
DESIGNATORS i AND j ARE THE SAME

If the i and j designators have the same value, the initial contents of the designated X register is sent to the increment unit. The resulting sum may then be used as an address for an SCM reference which reads a word from storage into this same X register.

(Xj) HAS MORE THAN 18 SIGNIFICANT BITS

If (Xj) is not an integer with less than 18 bits of significance, the lower order 18 bits are extracted and treated as an integer. The higher order bits in (Xj) are ignored.

53ijk SET Ai TO (Xj) + (Bk)



This instruction causes the increment unit to read operands from the Xj and Bk registers, form the sum of (Xj) + (Bk), and deliver this result to the Ai register. If the i designator is nonzero, a storage reference is made to SCM using this resulting sum as the relative storage address. The type of storage reference is a function of the i designator value.

0: No storage reference
1, 2, 3, 4, 5: Read from SCM to the Xi register
6, 7: Write into SCM from the Xi register.

The increment unit forms the sum of (Xj) + (Bk) in an 18-bit ones complement mode. Only the lowest order 18 bits of (Xj) are transmitted to the increment unit. The resulting sum is delivered simultaneously to the Ai register and to the SAS, if required. The SAS treats this quantity as an 18-bit positive integer address. This address is relative to the beginning of the SCM field for the current program. If the i designator value causes a read from SCM, the result arrives at the Xi register a minimum of 6 clock periods later than the result delivered to the Ai register. This X register is reserved until the data arrives from storage. If the i designator value causes a write into SCM, the 60-bit word being stored is read from the Xi register into SCM in the same clock period in which this instruction issues. This X register may be used for unrelated computation in the next clock period.

This instruction is intended for fetching operands from storage for computation and for delivering results back into storage.

SCM ADDRESS OUT OF RANGE

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 causes the current program sequence to terminate with an exchange jump to (EEA). If the reference involved writing into an X register, a positive zero word is written into the X register before the interrupt occurs. If the reference involved writing into SCM, the memory sequence is aborted.

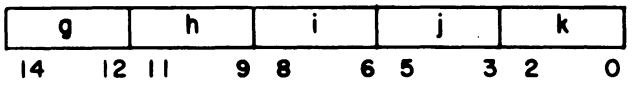
DESIGNATORS i AND j ARE THE SAME

If the i and j designators have the same value, the initial contents of the designated X register is sent to the increment unit. The resulting sum may then be used as an address for an SCM reference which reads a word from storage into this same X register.

(Xj) HAS MORE THAN 18 SIGNIFICANT BITS

If (Xj) is not an integer with less than 18 bits of significance, the lowest order 18 bits are extracted and treated as an integer. The highest order bits in (Xj) are ignored.

54ijk SET Ai TO (Aj) + (Bk)



This instruction causes the increment unit to read operands from the Aj and Bk registers, form the sum of (Aj) + (Bk), and deliver this result to the Ai register. If the i designator is nonzero, a storage reference is made to SCM using this resulting sum as the relative storage address. The type of storage reference is a function of the i designator value.

0: No storage reference
1, 2, 3, 4, 5: Read from SCM to the Xi register
6, 7: Write into SCM from the Xi register

The increment unit forms the sum of (Aj) + (Bk) in an 18-bit ones complement mode. The resulting sum is delivered simultaneously to the Ai register and to the SAS, if required. The SAS treats this quantity as an 18-bit positive integer address. This address is relative to the beginning of the SCM field for the current program. If the i designator value causes a read from SCM, the result arrives at the Xi register a minimum of 6 clock periods later than the result delivered to the Ai register. This X register is reserved until the data arrives from storage. If the i designator value causes a write into SCM, the 60-bit word being stored is read from the Xi register into SCM in the same clock period in which this instruction issues. This X register may be used for unrelated computation in the next clock period.

This instruction is intended for fetching operands from storage for computation and for delivering results back into storage.

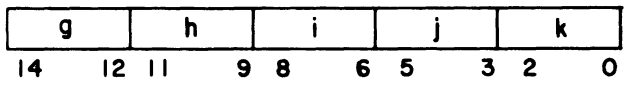
SCM ADDRESS OUT OF RANGE

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 causes the current program sequence to terminate with an exchange jump to (EEA). If the reference involved writing into an X register, a positive zero word is written into the X register before the interrupt occurs. If the reference involved writing into SCM, the memory sequence is aborted.

DESIGNATORS i AND j ARE THE SAME

If the i and j designators have the same value, the initial contents of the designated A register is sent to the increment unit and the result is stored back into the same A register.

55ijk SET Ai TO (Aj) - (Bk)



This instruction causes the increment unit to read operands from the Aj and Bk registers, form the difference of (Aj) - (Bk), and deliver this result to the Ai register. If the i designator is nonzero, a storage reference is made to SCM using this result

as the relative storage address. The type of storage reference is a function of the i designator value.

0: No storage reference
1, 2, 3, 4, 5: Read from SCM to the Xi register
6, 7: Write into SCM from the Xi register

The increment unit forms the difference of (Aj) - (Bk) in an 18-bit ones complement mode. The result is delivered simultaneously to the Ai register and to the SAS, if required. The SAS treats this quantity as an 18-bit positive integer address. This address is relative to the beginning of the SCM field for the current program. If the i designator value causes a read from SCM, the result arrives at the Xi register a minimum of 6 clock periods later than the result delivered to the Ai register. This X register is reserved until the data arrives from storage. If the i designator value causes a write into SCM, the 60-bit word being stored is read from the Xi register into SCM in the same clock period in which this instruction issues. This X register may be used for unrelated computation in the next clock period.

This instruction is intended for fetching operands from storage for computation and for delivering results back into storage.

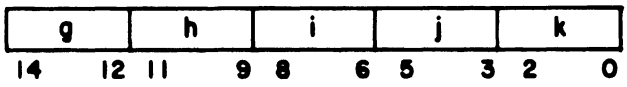
SCM ADDRESS OUT OF RANGE

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 causes the current program sequence to terminate with an exchange jump to (EEA). If the reference involved writing into an X register, a positive zero word is written into the X register before the interrupt occurs. If the reference involved writing into SCM, the memory sequence is aborted.

DESIGNATORS i AND j ARE THE SAME

If the i and j designators have the same value, the initial contents of the designated A register is sent to the increment unit and the result is stored back into the same A register.

56ijk SET Ai TO (Bj) + (Bk)



This instruction causes the increment unit to read operands from the Bj and Bk registers, form the sum of (Bj) + (Bk), and deliver this result to the Ai register. If the i designator is nonzero, a storage reference is made to SCM using this result as the relative storage address. The type of storage reference is a function of the i designator value.

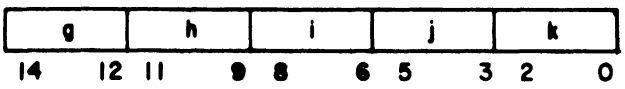
0: No storage reference
1, 2, 3, 4, 5: Read from SCM to the Xi register
6, 7: Write into SCM from the Xi register

The increment unit forms the sum of (Bj) + (Bk) in an 18-bit ones complement mode. The result is delivered simultaneously to the Ai register and to the SAS, if required. The SAS treats this quantity as an 18-bit positive integer address. This address is relative to the beginning of the SCM field for the current program. If the i designator value causes a read from SCM, the result arrives at the Xi register a minimum of 6 clock periods later than the result delivered to the Ai register. This X register is reserved until the data arrives from storage. If the i designator value causes a write into SCM, the 60-bit word being stored is read from the Xi register into SCM in the same clock period in which this instruction issues. This X register may be used for unrelated computation in the next clock period.

This instruction is intended for fetching operands from storage for computation and for delivering results back into storage.

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 causes the current program sequence to terminate with an exchange jump to (EEA). If the reference involved writing into an X register, a positive zero word is written into the X register before the interrupt occurs. If the reference involved writing into SCM, the memory sequence is aborted.

57ijk SET Ai TO (Bj) - (Bk)



This instruction causes the increment unit to read operands from the Bj and Bk registers, form the difference of (Bj) - (Bk), and deliver this result to the Ai register. If the i designator is nonzero, a storage reference is made to SCM using this result as the relative storage address. The type of storage reference is a function of the i designator value.

0: No storage reference

1, 2, 3, 4, 5: Read from SCM to the Xi register

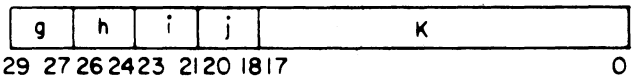
6, 7: Write into SCM from register

The increment unit forms the difference of (Bj) - (Bk) in an 18-bit ones complement mode. The result is delivered simultaneously to the Ai register and to the SAS, if required. The SAS treats this quantity as an 18-bit positive integer address. This address is relative to the beginning of the SCM field for the current program. If the i designator value causes a read from SCM, the result arrives at the Xi register a minimum of 6 clock periods later than the result delivered to the Ai register. This X register is reserved until the data arrives from storage. If the i designator value causes a write into SCM, the 60-bit word being stored is read from the Xi register into SCM in the same clock period in which this instruction issues. This X register may be used for unrelated computation in the next clock period.

This instruction is intended for fetching operands from storage for computation and for delivering results back into storage.

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 causes the current program sequence to terminate with an exchange jump to (EEA). If the reference involved writing into an X register, a positive zero word is written into the X register before the interrupt occurs. If the reference involved writing into SCM, the memory sequence is aborted.

60ijK SET Bi TO (Aj) + K



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction causes the increment unit to read an operand from the Aj register, form the sum of (Aj) + K, and deliver this result to the Bi register.

The increment unit forms the sum of (Aj) + K in an 18-bit ones complement mode. This instruction is intended for address modification in the increment registers.

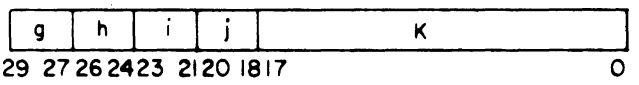
DESIGNATOR i IS ZERO

If the i designator is zero, this instruction becomes a pass instruction.

LAST PARCEL

This instruction normally requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If this instruction begins in the last parcel, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

61ijK SET Bi TO (Bj) + K



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction causes the increment unit to read an operand from the Bj register. The increment unit forms the sum of (Bj) + K in an 18-bit ones complement mode.

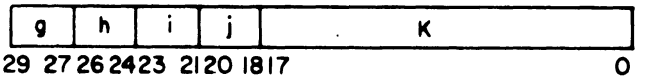
DESIGNATOR i IS ZERO

If the i designator is zero, this instruction becomes a pass instruction.

LAST PARCEL

This instruction normally requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If this instruction begins in the last parcel, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

62ijK SET Bi TO (Xj) + K



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction causes the increment unit to read an operand from the Xj register, form the sum of (Xj) + K, and deliver this result to the Bi register. The increment unit forms the sum of (Xj) + K in an 18-bit ones complement mode.

DESIGNATOR i IS ZERO

If the i designator is zero, this instruction becomes a pass instruction.

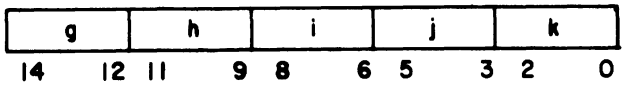
LAST PARCEL

This instruction normally requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If this instruction begins in the last parcel, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

(Xj) HAS MORE THAN 18 SIGNIFICANT BITS

If (Xj) is not an integer with less than 18 bits of significance, the lower order 18 bits are extracted and treated as an integer. The higher order bits in (Xj) are ignored.

63ijk SET Bi TO (Xj) + (Bk)



This instruction causes the increment unit to read operands from the Xj and Bk registers, form the sum of (Xj) + (Bk), and deliver this result to the Bi register. The increment unit forms the sum of (Xj) + (Bk) in an 18-bit ones complement mode. Only the lower order 18 bits of (Xj) are transmitted to the increment unit.

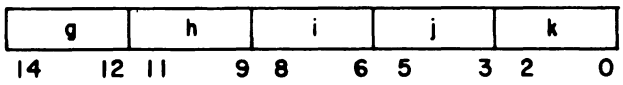
DESIGNATOR i IS ZERO

If the i designator is zero, this instruction becomes a pass instruction.

(Xj) HAS MORE THAN 18 SIGNIFICANT BITS

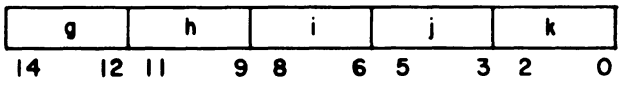
If (Xj) is not an integer with less than 18 bits of significance, the lower order 18 bits of (Xj) are extracted and treated as an integer. The higher order bits in (Xj) are ignored.

64ijk SET Bi TO (Aj) + (Bk)



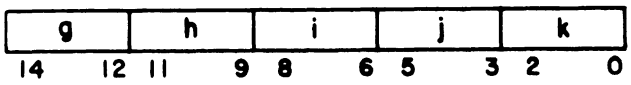
This instruction causes the increment unit to read operands from the Aj and Bk registers, form the sum of (Aj) + (Bk), and deliver this result to the Bi register. The increment unit forms the sum of (Aj) + (Bk) in an 18-bit ones complement mode. If the i designator is zero, this instruction becomes a pass instruction.

65ijk SET Bi TO (Aj) - (Bk)



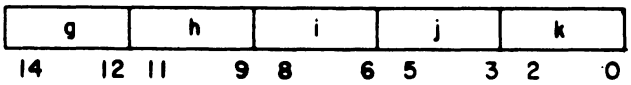
This instruction causes the increment unit to read operands from the Aj and Bk registers, form the difference of (Aj) - (Bk), and deliver this result to the Bi register. The increment unit forms the difference of (Aj) - (Bk) in an 18-bit ones complement mode. If the i designator is zero, this instruction becomes a pass instruction.

66ijk SET Bi TO (Bj) + (Bk)



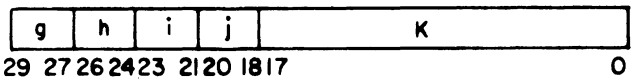
This instruction causes the increment unit to read operands from the Bj and Bk registers, form the sum of (Bj) + (Bk), and deliver this result to the Bi register. The increment unit forms the sum of (Bj) + (Bk) in an 18-bit ones complement mode. If the i designator is zero, this instruction becomes a pass instruction.

67ijk SET Bi TO (Bj) - (Bk)



This instruction causes the increment unit to read operands from the Bj and Bk registers, form the difference of (Bj) - (Bk), and deliver this result to the Bi register. The increment unit forms the difference of (Bj) - (Bk) in an 18-bit ones complement mode. If the i designator is zero, this instruction becomes a pass instruction.

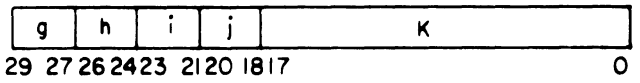
70ijK SET Xi TO (Aj) + K



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction causes the increment unit to read an operand from the Aj register, form the sum of (Aj) + K, and deliver this result to the Xi register. The increment unit forms the sum of (Aj) + K in an 18-bit ones complement mode. The resulting 18-bit quantity is sign extended by copying the highest order bit of the result into the upper 42 bit positions in the Xi register.

This instruction normally requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If this instruction begins in the last parcel, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

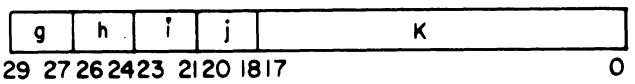
71ijK SET Xi TO (Bj) + K



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction causes the increment unit to read an operand from the Bj register, form the sum of (Bj) + K, and deliver this result to the Xi register. The increment unit forms the sum of (Bj) + K in an 18-bit ones complement mode. The resulting 18-bit quantity is sign extended by copying the highest order bit of the result into the upper 42 bit positions in the Xi register.

This instruction normally requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If this instruction begins in the last parcel, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

72ijK SET Xi TO (Xj) + K



This is a two-parcel instruction in which the lower order 18 bits are used as an operand K. This instruction causes the increment unit to read an operand from the Xj register, form the sum of (Xj) + K, and deliver this result to the Xi register. The increment unit forms the sum of (Xj) + K in an 18-bit ones complement mode. Only the lower order 18 bits of (Xj) are transmitted to the increment unit. The 18-bit result is sign extended by copying the highest order bit of the result into the upper 42 bit positions in the Xi register.

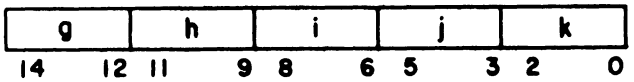
LAST PARCEL

This instruction normally requires two parcels of an instruction word. If this instruction begins in the first, second, or third parcel of an instruction word, the following parcel completes the instruction. If this instruction begins in the last parcel, it is not continued in the following word. The instruction is executed as if there were a fifth parcel in the instruction word and this parcel contained all zeros.

(Xj) HAS MORE THAN 18 SIGNIFICANT BITS

If (Xj) is not an integer with less than 18 bits of significance, the lowest order 18 bits are extracted and treated as an integer. The higher order bits in (Xj) are ignored.

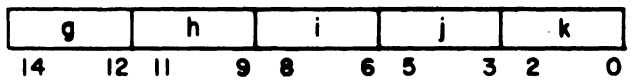
73ijk SET Xi TO (Xj) + (Bk)



This instruction causes the increment unit to read operands from the Xj and Bk registers, form the sum of (Xj) + (Bk), and deliver this result to the Xi register. The increment unit forms the sum of (Xj) + (Bk) in an 18-bit ones complement mode. Only the lower order 18 bits of (Xj) are transmitted to the increment unit. The 18-bit result is sign extended by copying the highest order bit of the result into the upper 42 bit positions in the Xi register.

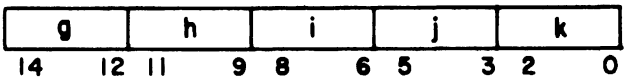
If (Xj) is not an integer with less than 18 significant bits, the lower order 18 bits are extracted and treated as an integer. The higher order bits in (Xj) are ignored.

74ijk SET Xi TO (Aj) + (Bk)



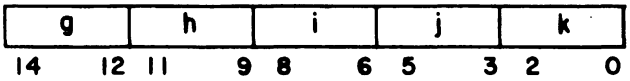
This instruction causes the increment unit to read operands from the Aj and Bk registers, form the sum of (Aj) + (Bk), and deliver this result to the Xi register. The increment unit forms the sum of (Aj) + (Bk) in an 18-bit ones complement mode. The 18-bit result is sign extended by copying the highest order bit of the result into the upper 42 bit positions in the Xi register.

75ijk SET Xi TO (Aj) - (Bk)



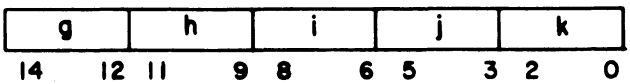
This instruction causes the increment unit to read operands from the Aj and Bk registers, form the difference of (Aj) - (Bk), and deliver this result to the Xi register. The increment unit forms the difference of (Aj) - (Bk) in an 18-bit ones complement mode. The 18-bit result is sign extended by copying the highest order bit of the result into the upper 42 bit positions in the Xi register.

76ijk SET Xi TO (Bj) + (Bk)



This instruction causes the increment unit to read operands from the Bj and Bk registers, form the sum of (Bj) + (Bk), and deliver this result to the Xi register. The increment unit forms the sum of (Bj) + (Bk) in an 18-bit ones complement mode. The 18-bit result is sign extended by copying the highest order bit of the result into the upper 42 bit positions in the Xi register.

77ijk SET Xi TO (Bj) - (Bk)



This instruction causes the increment unit to read operands from the Bj and Bk registers, form the difference of (Bj) - (Bk), and deliver this result to the Xi register. The increment unit forms the difference of (Bj) - (Bk) in an 18-bit ones complement mode. The 18-bit result is sign extended by copying the highest order bit of the result into the upper 42 bit positions in the Xi register.

PERIPHERAL PROCESSOR UNIT INSTRUCTIONS 6

This section describes the PPU instructions. Most of these instructions involve manipulation of internal registers in the PPU. The timing of execution for these instructions is dominated by the access time of the core storage banks. There are two independent banks of storage. One bank contains all of the even storage addresses and the other bank contains all of the odd storage addresses. If references to storage alternate between even and odd addresses, each reference requires 5 clock periods. If two even references (or two odd references) occur consecutively, the storage read/write cycle for the first reference must be completed before the second reference can begin. In this case, a storage reference requires 10 clock periods. As a result, the execution time for most of the PPU instructions is a multiple of 5 clock periods with variation in increments of 5 clock periods depending upon the storage addresses involved.

INSTRUCTION FORMATS

An instruction may have a 12-bit (Figure 6-1) or a 24-bit (Figure 6-2) format. The 12-bit format has a six-bit operation code f and a six-bit operand or operand address code d. 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 location d to form an 18-bit operand or a 12-bit operand address.

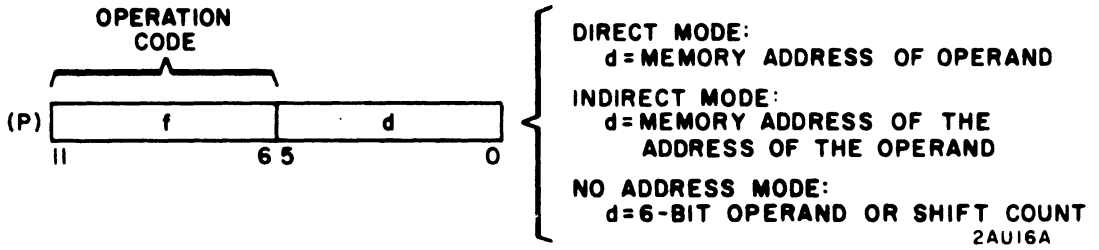


Figure 6-1. PPU 12-Bit Instruction Format

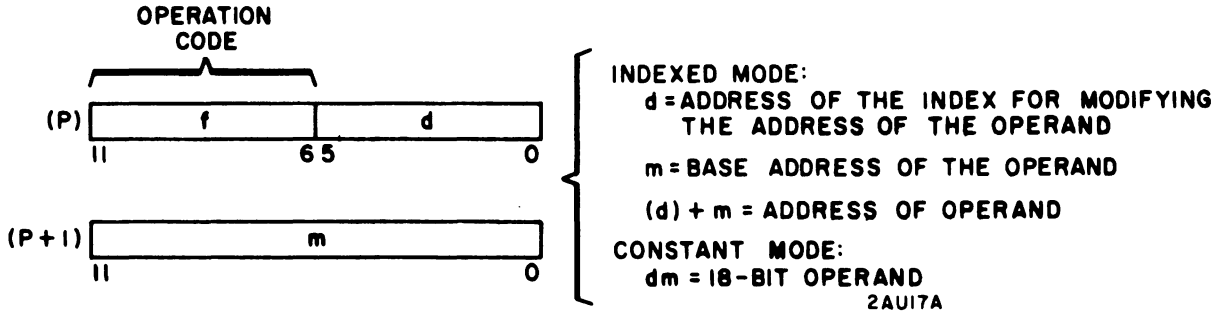


Figure 6-2. PPU 24-Bit Instruction Format

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 PPU instructions.

TABLE 6-1. ADDRESSING MODES FOR PPU 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	
Nonzero Jump				05	
Positive Jump				06	
Negative 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 six-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 through 0077).

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 through 0077). 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 through 7776). It is not possible to reference address 7777. If d is nonzero, the contents 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 contents of location d 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:

Mode	Instruction	A Register
No address (6-bit operand)	1425	000025
Constant (18-bit operand)	2025	250100
	0100	
Direct address	3025	000150
Indexed direct address	5025	001234
	0100	
Indirect address	4025	007776

RESTRICTIONS ON INSTRUCTION LOOPS

Program loops in the PPU must be four instruction words or longer. Also, there is a restriction on the number of memory references allowed within the instruction loop. The following criteria may be used for any size instruction loop.

The number of clock periods required to execute each pass through the instruction loop must be equal to or greater than 15N.

N = the number of memory references in the instruction loop to any one stack, including the references to obtain instructions in that stack.

In the following example, there are seven references to bank 1, stack 1. The total execution time for the pass is 80 clock periods. Since 15N equals 105 clock periods, this instruction loop violates the restriction.

Address	Instruction	Time (CP)	Location
1000	4023	25	0023=1023
1001	4133	25	0033=1033
1002	4243	25	0043=1043
1003	0474	5	

The restriction may be ignored if the number of passes through the loop does not exceed 2000/N. In the previous example, the loop may be used if the number of passes does not exceed 2000/7 or 285.

The restrictions ensure that the PPU memory stacks do not exceed a duty cycle of 66-2/3 percent, or if this limit is exceeded, the stack is not referenced more than 2000 times at this duty cycle.

INSTRUCTION TIMING

Execution times for the PPU instructions are listed in Table 6-2. The Timing Notes column indicates the assumptions made for the time listed in the Execution Time column for that instruction. The Timing Notes are at the end of the table.

TABLE 6-2. PERIPHERAL PROCESSOR UNIT INSTRUCTION TIMING

Mnemonic Code	Instruction Code	Description	Execution Time (CP)	Timing Notes
ESN	00xx	Error Stop	-	-
LJM	0100m	Long Jump to m	10 or 15	1, 2
LJM	01dm	Long Jump to m + (d)	15, 20, or 25	1, 2
RJM	0200m	Return Jump to m	15 or 20	1, 2
RJM	02dm	Return Jump to m + (d)	20, 25 or 30	1, 2
UJN	03d	Unconditional Jump d	7 or 10	2
ZJN	04d	Zero Jump d	5	3, 4
NJN	05d	Nonzero Jump d	5	3
PJN	06d	Positive Jump d	5	3
MJN	07d	Negative Jump d	5	3
SHN	10d	Shift (A) by d	6	5
LMN	11d	Logical Difference (A) and d	5	6
LPN	12d	Logical Product (A) and d	5	6
SCN	13d	Selective Clear (A) by d	5	6
LDN	14d	Load d	5	6
LCN	15d	Load Complement d	5	6
ADN	16d	Add (A) + d	5	6
SBN	17d	Subtract (A) - d	5	6
LDC	20dm	Load dm	10	1, 6
ADC	21dm	Add (A) + dm	10	1, 6
LPC	22dm	Logical Product (A) and dm	10	1, 6
LMC	23dm	Logical Difference (A) and dm	10	1, 6

TABLE 6-2. PERIPHERAL PROCESSOR UNIT INSTRUCTION TIMING (Cont'd)

Mnemonic Code	Instruction Code	Description	Execution Time (CP)	Timing Notes
PSN	24xx	Pass	5	6
	25xx	Pass	5	6
	26xx	Pass	5	6
	27xx	Pass	5	6
LDD	30d	Load (d)	15	7
ADD	31d	Add (A) + (d)	15	7
SBD	32d	Subtract (A) - (d)	15	7
LMD	33d	Logical Difference (A) and (d)	15	7
STD	34d	Store (A) at (d)	15	7
RAD	35d	Replace Add (A) + (d)	25	7
AOD	36d	Replace Add One (d)	25	7
SOD	37d	Replace Subtract One (d)	25	7
LDI	40d	Load ((d))	15 or 25	2
ADI	41d	Add (A) + ((d))	15 or 25	2
SBI	42d	Subtract (A) - ((d))	15 or 25	2
LMI	43d	Logical Difference (A) and ((d))	15 or 25	2
STI	44d	Store (A) at ((d))	15 or 25	2
RAI	45d	Replace Add (A) + ((d))	25 or 35	2
AOI	46d	Replace Add One ((d))	25 or 35	2
SOI	47d	Replace Subtract One ((d))	25 or 35	2
LDM	5000m	Load (m)	20	1, 7
LDM	50dm	Load (m + (d))	20 or 30	1, 2
ADM	5100m	Add (A) + (m)	20	1, 7
ADM	51dm	Add (A) + (m+(d))	20 or 30	1, 2
SBM	5200m	Subtract (A) - (m)	20	1, 7
SBM	52dm	Subtract (A) - (m + (d))	20 or 30	1, 2
LMM	5300m	Logical Difference (A) and (m)	20	1, 7
LMM	53dm	Logical Difference (A) and (m+(d))	20 or 30	1, 2
STM	5400m	Store (A) at (m)	20	1, 7
STM	54dm	Store (A) at (m + (d))	20 or 30	1, 2
RAM	5500m	Replace Add (A) + (m)	30	1, 7
RAM	55dm	Replace Add (A) + (m + (d))	30 or 40	1, 2
AOM	5600m	Replace Add One (m)	30	1, 7
AOM	56dm	Replace Add One (m + (d))	30 or 40	1, 2
SOM	5700m	Replace Subtract One (m)	30	1, 7
SOM	57dm	Replace Subtract One (m + (d))	30 or 40	1, 2
FIM	60dm	Jump to m if Channel d Input Word Flag Set	10	1, 8

TABLE 6-2. PERIPHERAL PROCESSOR UNIT INSTRUCTION TIMING (Cont'd)

Mnemonic Code	Instruction Code	Description	Execution Time (CP)	Timing Notes
EIM	61dm	Jump to m if Channel d Input Word Flag Not Set	10	1, 8
IRM	62dm	Jump to m if Channel d Input Record Flag Set	10	1, 8
NIM	63dm	Jump to m if Channel d Input Record Flag Not Set	10	1, 8
FOM	64dm	Jump to m if Channel d Output Word Flag Set	10	1, 8
EOM	65dm	Jump to m if Channel d Output Word Flag Not Set	10	1, 8
ORM	66dm	Jump to m if Channel d Output Record Flag Set	10	1, 8
NOM	67dm	Jump to m if Channel d Output Record Flag Not Set	10	1, 8
IAN	70d	Input to A on Channel d	9	9
IAM	71dm	Input (A) Words to m on Channel d	24 or 42	1, 10
OAN	72d	Output from A on Channel d	9	11
OAM	73dm	Output (A) Words from m on Channel d	34	1, 12
RFN	74d	Set Output Record Flag on Channel d	5	6
	75xx	Pass	5	6
	76xx	Pass	5	6
ESN	77xx	Error Stop	-	-

TIMING NOTES

1. The storage reference for the second word of the current instruction word must be to the alternate bank.
2. The shorter time is obtained when full use is made of bank phasing (back-to-back storage references to alternate banks).
3. The time listed assumes that the jump conditions are not met. If the jump is taken, the time is the same as for the 03d instruction.
4. The d designator cannot be 00 or 77.
5. The time listed assumes that d equals three or less. The time increases by 1 clock period for each shift beyond three. The maximum time is 34 clock periods.

6. The storage reference(s) following the one for the current instruction word must be to alternate bank(s).

7. The storage reference(s) following the one for the current instruction word may be to either bank.

8. The time listed assumes that either the jump conditions are not met or the jump is taken to the alternate bank. If the jump is taken to the same bank, the time is 15 clock periods.

9. The time listed assumes that channel d input word flag is set. If not set, add the time waiting for the flag to set.

10. The first time listed is for a two-word block input terminated by reducing (A) to zero. The following assumptions are made.

    a. A count of 2 is in the A register.

    b. The channel d input word flag is initially set.

    c. The first data storage reference is to the alternate bank.

    d. There is a 2-clock period response time between the resume pulse and the setting of the input word flag.

    The second time listed is for a three-word block input terminated by setting the channel d input record flag. The following assumptions are made.

    a. The channel d input word flag is initially set.

    b. The first data storage reference is to the alternate bank.

    c. There is a 2-clock period response time between the resume pulse and the setting of the input word flag.

11. The time listed assumes that the channel d output word flag is clear. If not clear, add the time waiting for the flag to clear.

12. The time listed is for a three-word block output. The following assumptions are made.

    a. A count of 3 is in the A register

    b. The channel d output word flag is initially clear.

    c. The first data storage reference is to the alternate bank.

    d. The device has a 2-clock period response time from receipt of word pulse to transmission of resume pulse.

DESCRIPTION OF INSTRUCTIONS

This part of the manual describes the PPU instruction in detail. Each instruction is described separately. Instruction designators are listed and defined in Table 6-3.

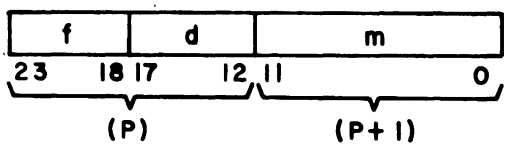
TABLE 6-3. PPU INSTRUCTION DESIGNATORS

Designator	Use
f	6-bit instruction code
fd	12-bit instruction code
d	6-bit operand or address
m	12-bit operand or address
dm	18-bit operand
(d)	Contents of location specified by d
((d))	Contents of location whose address is contained in location specified by d
(m)	Contents of location specified by m
x	Unused designator
A	Arithmetic register
(A)	Contents of A register

00xx ERROR STOP

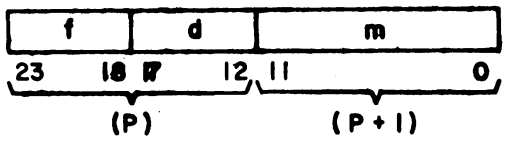
This instruction causes the PPU program execution to stop and to indicate a program error condition to the MCU. The PPU can be restarted only by a dead start condition.

0100m LONG JUMP TO m



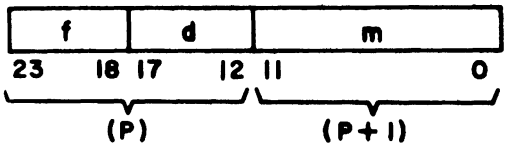
This instruction terminates the current program sequence with a jump to a new sequence beginning at address m. The value of d must be zero for this instruction. The instruction begins by reading the quantity m from storage location (P) + 1 to the X register. The address for the new program sequence is formed by adding (X) to a zero value in the Q register. This address is then used to fetch the first word of the new program sequence.

01dm LONG JUMP TO m + (d)



This instruction terminates the current program sequence with a jump to a new sequence beginning at address m + (d). The value of d must be nonzero for this instruction. The instruction begins by reading the quantity m from storage location (P) + 1 and holding this quantity in the Q register. The contents of location d is then read into the X register. The address for the new program sequence is formed by adding (Q) to (X) in a 12-bit ones complement mode. The resulting address is used to fetch the first word of the new program sequence.

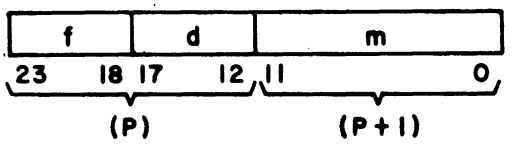
0200m RETURN JUMP TO m



This instruction interrupts the current program sequence and inserts the execution of a subroutine between the current instruction in the present sequence and the following instruction. The called subroutine must have a common exit point in the form of a long jump to m instruction preceding the entry point. The return jump instruction inserts the exit address in the m location of the subroutine exit and then jumps to the entry point in the following word.

The value of d in this instruction must be zero. The instruction begins by reading the quantity m from storage location (P) + 1 to the Q register. This quantity is then used as a storage address to store (P) + 2 at storage location m. The first word of the new program sequence is then read from storage location m + 1.

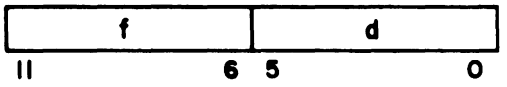
02dm RETURN JUMP TO m + (d)



This instruction interrupts the current program sequence and inserts the execution of a subroutine between the current instruction in the present sequence and the following instruction. The called subroutine must have a common exit point in the form of a long jump to m instruction preceding the entry point. The return jump instruction inserts the exit address in the m location of the subroutine exit and then jumps to the entry point in the following word.

The value of d in this instruction must be nonzero. The instruction begins by reading the quantity m from storage location (P) + 1 to the Q register. The contents of location d is then read into the X register. The address for the new program sequence is formed by adding (Q) to (X) in a 12-bit ones complement mode. The resulting address is used to store (P) + 2 in the m field of the called subroutine exit instruction. The first word of the new program is then read from the following storage location.

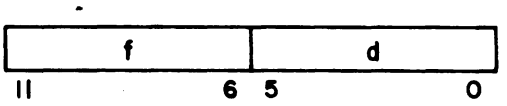
03d UNCONDITIONAL JUMP d



This instruction interrupts the current program sequence with a jump to a new sequence beginning at an address incrementally related to the current program address. The d designator may specify a new sequence which begins at an address either forward or backward from the current address by an amount no greater than 31 (decimal) locations. The d designator is considered as a six-bit ones complement number in determining the increment for the jump.

As an example, consider a d value of 16 (octal). The new program sequence in this case begins with an instruction word located 16 (octal) locations beyond the location of the 03d instruction. Now consider a d value of 55 (octal). The new program sequence in this case begins with an instruction word located 22 (octal) locations before the location of the 03d instruction. Values of 00 and 77 for the d designator must not be used with this instruction. These two values cause the PPU program to lock up and require dead starting the system with a new program.

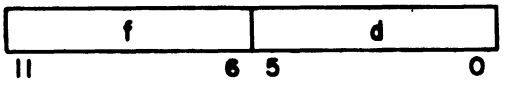
04d ZERO JUMP d



This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the contents of the A register. If (A) = 000000, the current program sequence is terminated with a jump to an address specified by the d designator. If (A) $\neq$ 000000, the current program sequence continues with the execution of the next instruction. A value of (A) = 777777 is not considered as zero for this instruction.

If the jump is taken, the new program sequence begins at an address either forward or backward from the current address by an amount not greater than 31 (decimal) locations. The d designator is considered as a six-bit ones complement number in determining the increment for the jump. (Refer to instruction 03d for examples.)

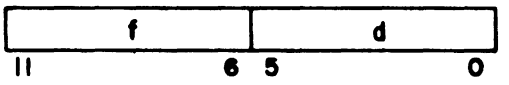
05d NONZERO JUMP d



This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the contents of the A register. If (A) $\neq$ 000000, the current program sequence is terminated with a jump to an address specified by the d designator. If (A) = 000000, the current program sequence continues with the execution of the next instruction. A value of (A) = 777777 is not considered as zero for this instruction.

If the jump is taken, the new program sequence begins at an address either forward or backward from the current address by an amount not greater than 31 (decimal) locations. The d designator is considered as a six-bit ones complement number in determining the increment for the jump. (Refer to instruction 03d for examples.)

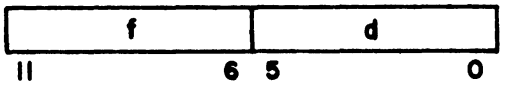
06d POSITIVE JUMP d



This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the contents of the A register. If the highest order bit in the A register has a zero value, the current program sequence is terminated with a jump to an address specified by the d designator. If the highest order bit in the A register has a one value, the current program sequence continues with the execution of the next instruction.

If the jump is taken, the new program sequence begins at an address either forward or backward from the current address by an amount not greater than 31 (decimal) locations. The d designator is considered as a six-bit ones complement number in determining the increment for the jump. (Refer to instruction 03d for examples.)

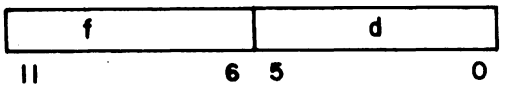
07d NEGATIVE JUMP d



This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the contents of the A register. If the highest order bit in the A register has a one value, the current program sequence is terminated with a jump to an address specified by the d designator. If the highest order bit in the A register has a zero value, the current program sequence continues with the execution of the next instruction.

If the jump is taken, the new program sequence begins at an address either forward or backward from the current address by an amount no greater than 31 (decimal) locations. The d designator is considered as a six-bit ones complement number in determining the increment for the jump. (Refer to instruction 03d for examples.)

10d SHIFT (A) BY d

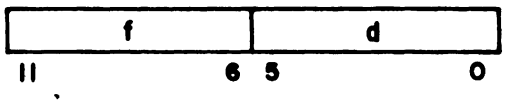


This instruction shifts the contents of the A register either to the right open ended or to the left circularly as specified by the d designator. The d designator is treated as a six-bit ones complement number in this instruction. If the highest order bit in the d designator is zero, the contents of the A register is shifted circularly to the left by the number of bit positions indicated in the value of the d designator. If the highest order bit in the d designator is one, the contents of the A register is shifted open ended to the right by the complement of the value of the d designator.

In a left circular shift, the contents of the A register is shifted one bit position at a time. In each shift, the lowest order bit position in the register is filled by the bit previously held in the highest order bit position. No bits are lost in this process but are repositioned toward the higher order bit positions. A d designator value of 00 causes no shift to take place. A d designator value greater than 18 (decimal) causes the contents of the A register to shift completely around the register. A maximum of 31 (decimal) shift count may be used.

In a right open ended shift, the contents of the A register is shifted one bit position at a time toward the lower order bit positions in the register. The highest order bit position in the A register is filled with a zero value as each shift occurs. The lowest order bit in the A register is discarded as each shift occurs. A maximum of 31 (decimal) shift count may be used. For all shift counts larger than 17 (decimal), the final A register value is 000000. A designator value of 77 causes no shift to take place.

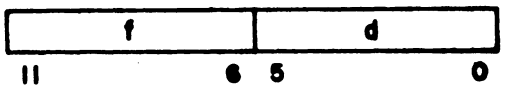
11d LOGICAL DIFFERENCE (A) AND d



This instruction forms in the A register the logical difference of the original (A) and the d designator considered as a six-bit positive integer. The highest order 12 bits in the A register are not affected by this operation.

The logical difference is the result of a bit-by-bit comparison of the two binary quantities. If two corresponding bits are equal, the resulting bit is zero. If unequal, the result is one.

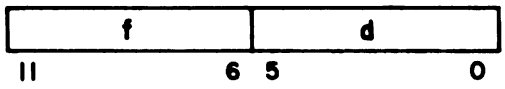
12d LOGICAL PRODUCT (A) AND d



This instruction forms in the A register the logical product of the original (A) and the d designator considered as a six-bit positive integer. The highest order 12 bits in the A register are always cleared to zero by this instruction.

The logical product is the result of a bit-by-bit comparison of the two binary quantities. If two corresponding bits are ones, the resulting bit is one. If not, the result is zero.

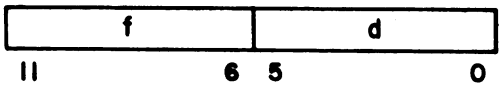
13d SELECTIVE CLEAR (A) BY d



This instruction forms in the A register the logical product of the original (A) and the complement of the d designator considered as a six-bit positive integer. The highest order 12 bits in the A register are not affected by this instruction.

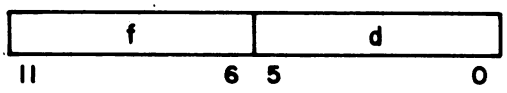
The selective clear is a bit-by-bit comparison of the two binary quantities. Any of the lower six bits in the A register are cleared if the corresponding bits of d are set.

14d LOAD d



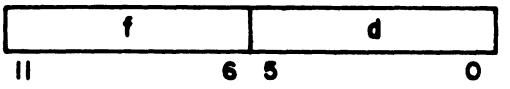
This instruction enters in the A register a copy of the d designator considered as a six-bit positive integer. The highest order 12 bits in the A register are always cleared to zero by this instruction.

15d LOAD COMPLEMENT d



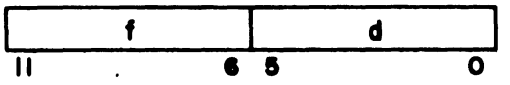
This instruction enters in the A register a complemented copy of the d designator. The highest order 12 bits in the A register are always set to one by this instruction. The lowest order six bits are bit-by-bit complements of the corresponding bits in the d designator.

16d ADD (A) + d



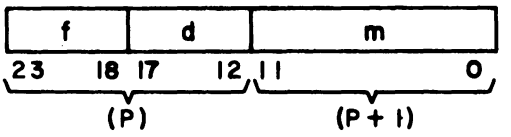
This instruction adds the d designator, considered as a six-bit positive quantity, to the current contents of the A register. The result is left in the A register. The addition is performed in an 18-bit ones complement mode. An 18-bit operand is formed from the d designator by adding 12 higher order zero bits.

17d SUBTRACT (A) - d



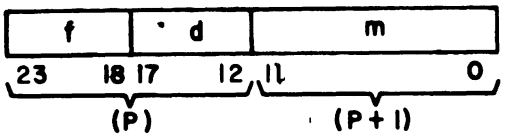
This instruction subtracts the d designator, considered as a six-bit positive quantity, from the current contents of the A register. The result is left in the A register. An 18-bit operand is formed from the d designator. This operand consists of 12 one bits in the highest order bit positions and six lowest order bits which are bit-by-bit complements of the corresponding bits in the d designator. This 18-bit operand is added to the original contents of the A register in an 18-bit ones complement mode.

20dm LOAD dm



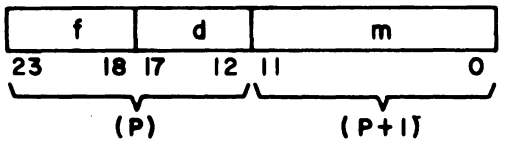
This instruction clears the A register and enters an 18-bit operand consisting of the d and m designators. The d designator is inserted in the highest order six-bit positions and the m designator is inserted in the lowest order 12-bit positions.

21dm ADD (A) + dm



This instruction adds an 18-bit operand consisting of the d and m designators to the current contents of the A register. The result is left in the A register. The addition is performed in an 18-bit ones complement mode. The d designator forms the highest order six bits and the m designator completes the lowest order 12 bits.

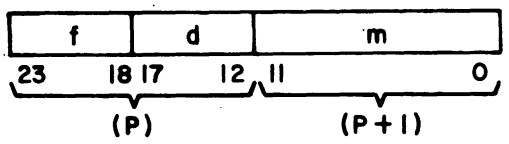
22dm LOGICAL PRODUCT (A) AND dm



This instruction forms the logical product of (A) and an 18-bit operand consisting of the d and m designators. The result is left in the A register. The d designator forms the highest order six bits and the m designator completes the lowest order 12 bits.

The logical product is the result of a bit-by-bit comparison of the two binary quantities. If two corresponding bits are ones, the resulting bit is one. If not, the result is zero.

23dm LOGICAL DIFFERENCE (A) AND dm



This instruction forms the logical difference of (A) and an 18-bit operand consisting of the d and m designators. The result is left in the A register. The d designator forms the highest order six bits and the m designator completes the lowest order 12 bits.

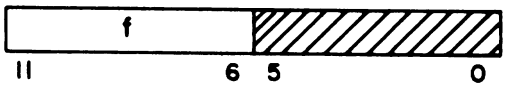
The logical difference is the result of a bit-by-bit comparison of the two binary quantities. If two corresponding bits are equal, the resulting bit is one. If unequal, the result is zero.

24xx PASS

25xx PASS

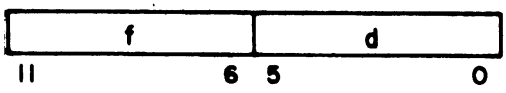
26xx PASS

27xx PASS



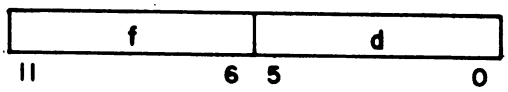
These four instructions are identical and perform no logical function. Each instruction results in a 5-clock-period delay.

30d LOAD (d)



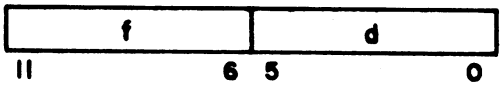
This instruction clears the A register and enters a 12-bit operand from location d. The operand is entered in the A register as a 12-bit positive integer. The highest order six bits in the A register are always cleared by this instruction.

31d ADD (A) + (d)



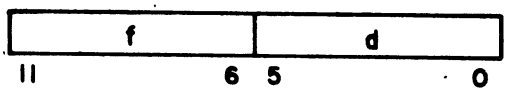
This instruction adds the contents of location d, considered as a 12-bit positive quantity, to the current contents of the A register. The result is left in the A register. The addition is performed in an 18-bit ones complement mode. An 18-bit operand is formed from location d contents by adding six higher order zero bits.

32d SUBTRACT (A) - (d)



This instruction subtracts the contents of location d, considered as a 12-bit positive quantity, from the current contents of the A register. The result is left in the A register. The operation is performed by adding the complement of location d contents to (A) in an 18-bit ones complement mode. An 18-bit operand is formed for the addition by forcing the highest order six bits to a one value. The lowest order 12 bits are the bit-by-bit complement of location d contents.

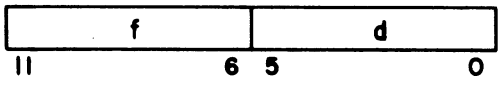
33d LOGICAL DIFFERENCE (A) AND (d)



This instruction forms in the A register the logical difference of the contents of location d, considered as a 12-bit positive quantity, and the original (A). The highest order six bits in the A register are not affected by this operation.

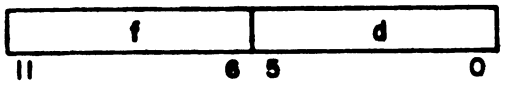
The logical difference is the result of a bit-by-bit comparison of the two binary quantities. If any corresponding bits are equal, the resulting bit is zero. If unequal, the result is one.

34d STORE (A) AT (d)



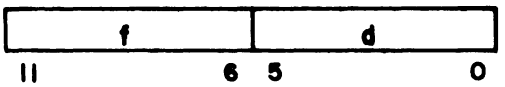
This instruction stores the lowest order 12 bits of (A) in location d. The contents of the A register is not altered in this process.

35d REPLACE ADD (A) + (d)



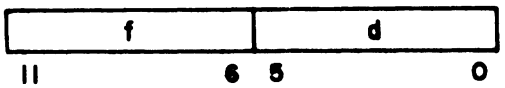
This instruction adds the contents of location d, considered as a 12-bit positive quantity, to the current contents of the A register. The result is left in the A register and is also stored in location d. The addition is performed in an 18-bit ones complement mode. An 18-bit operand is formed from location d contents by adding six higher order zero bits. The result stored in location d is the lowest order 12 bits of the resulting 18-bit sum.

36d REPLACE ADD ONE (d)



This instruction increases the contents of location d by one count. Execution begins by clearing the A register and entering a value of plus one. The contents of location d is read from storage to the X register and then added to (A) in an 18-bit ones complement mode. The location d value is treated as a 12-bit positive quantity in this process. An 18-bit operand is formed from (X) by adding six higher order zero bits. The result is left in the A register, and the lowest order 12 bits are stored in location d. Note that the arithmetic is essentially twos complement as viewed by location d, and the quantity in the A register is not necessarily equal to the result in location d.

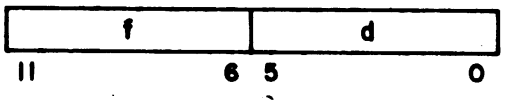
37d REPLACE SUBTRACT ONE (d)



This instruction decreases the contents of location d by one count. Execution begins by clearing the A register and entering a value of minus one. The contents of location d is read from storage to the X register and then added to (A) in an 18-bit ones complement mode. The location d value is treated as a 12-bit positive quantity in this process. An 18-bit operand is formed from (X) by adding six higher order zero bits. The result is left in the A register, and the lowest order 12 bits are stored in location d. Note

that the arithmetic is essentially twos complement as viewed by location d, and the quantity in the A register is not necessarily equal to the result in location d.

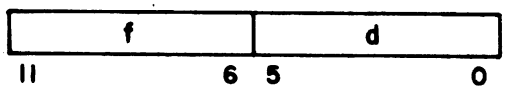
40d LOAD ((d))



This instruction clears the A register and enters a 12-bit operand from storage. The highest order six bits in the A register are always cleared by this instruction.

Instruction execution begins with a storage reference to location d. The contents of this location is read into the X register. A second storage reference is then made using (X) as the storage address. This operand is read into the A register, and the highest order six bits in the A register are cleared. A third storage reference is then initiated to read the next instruction word.

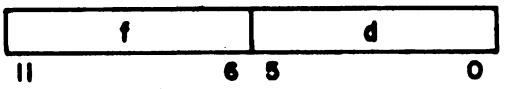
41d ADD (A) + ((d))



This instruction reads an operand from storage and adds it to the current contents of the A register. The addition is performed in an 18-bit ones complement mode. An 18-bit operand is formed from the 12-bit storage operand by adding six higher order zero bits. The address for the operand is contained in location d.

Instruction execution begins with a storage reference to location d. The contents of this location is read into the X register. A second storage reference is then made using (X) as the storage address. This operand is read into the X register and then added to the contents of the A register. A third storage reference is then initiated to read the next instruction word.

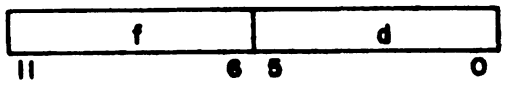
42d SUBTRACT (A) - ((d))



This instruction reads an operand from storage and subtracts it from the current contents of the A register. The result is left in the A register. The address for the operand is contained in location d. The operation is performed by adding the complement of the operand to (A) in an 18-bit ones complement mode. An 18-bit operand for the addition is formed from the 12-bit storage operand by forcing the highest order six bits to a one value. The lowest order 12 bits are the bit-by-bit complement of the storage operand values.

Instruction execution begins with a storage reference to location d. The contents of this location is read into the X register. A second storage reference is then made using (X) as the storage address. This operand is read into the X register and then subtracted from the contents of the A register. A third storage reference is then initiated to read the next instruction word.

43d LOGICAL DIFFERENCE (A) AND ((d))

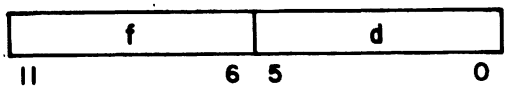


This instruction forms in the A register the logical difference of an operand read from storage and the original (A). The highest order six bits in the A register are not affected by this operation. The storage address for the operand is contained in location d.

The logical difference is the result of a bit-by-bit comparison of the two binary quantities. If the corresponding bits are equal, the resulting bit is zero. If unequal, the result is one.

Instruction execution begins with a storage reference to location d. The contents of this location is read into the X register. A second reference to storage is made using (X) as the storage address. This operand is read into the X register, and the logical difference is then formed and entered into the A register. A third storage reference then reads up the next instruction word.

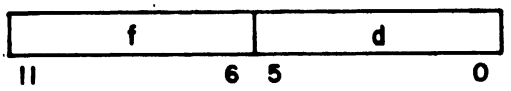
44d STORE (A) AT ((d))



This instruction stores the lowest order 12 bits of (A) in a storage location specified by the contents of location d. The contents of the A register is not altered in this process.

Execution begins with a storage reference to location d. The contents of this location is read into the X register. A second reference to storage is made using (X) as the storage address. The data read from storage is discarded in this reference, and the lowest order 12 bits of (A) are stored. A third storage reference then reads up the next instruction word.

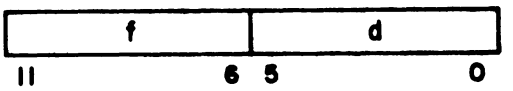
45d REPLACE ADD (A) + ((d))



This instruction reads an operand from storage and adds it to the current contents of the A register. The result is then left in the A register and is also stored in the same memory location from which the operand was read. The addition is performed in an 18-bit ones complement mode. An 18-bit operand is formed from the 12-bit storage operand by adding six higher order zero bits. The result returned to storage is the lowest order 12 bits of the final (A). The storage address for reading the operand and storing the result is contained in location d. Note that the result stored is not necessarily equal to the result left in the A register.

There are four storage references required in the execution of this instruction. The first reference reads the contents of location d into the X register and then into the Q register. A second storage reference is made using (X) as the storage address. This operand is read into the X register and then added to (A). A third storage reference stores the lowest order 12 bits of the resulting sum using (Q) as the storage address. The fourth storage reference reads up the next instruction word.

46d REPLACE ADD ONE ((d))

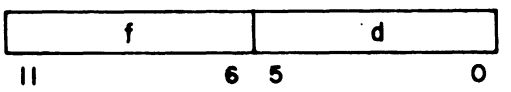


This instruction reads an operand from storage, increases its value by one count, and returns the result to the same storage location. The storage address for reading the operand and storing the result is contained in location d. The result is left in the A register as well as in storage.

Execution begins by clearing the A register and entering a value of plus one. The operand is then read from storage and added to (A) in an 18-bit ones complement mode. The operand is treated as a 12-bit positive quantity in this process. An 18-bit operand is formed from the 12-bit storage operand by adding six higher order zero bits. The result is left in the A register, and the lowest order 12 bits are returned to storage. Note that the arithmetic is essentially twos complement as viewed from storage, and the quantity in the A register is not necessarily equal to the result in storage.

There are four storage references required in the execution of this instruction. The first reference reads the contents of location d into the X register and then into the Q register. A second storage reference is made using (X) as the storage address. This operand is read into the X register and then added to (A). A third storage reference stores the lowest order 12 bits of the resulting sum using (Q) as the storage address. The fourth storage reference reads up the next instruction word.

47d REPLACE SUBTRACT ONE ((d))



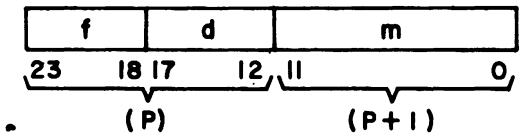
This instruction reads an operand from storage, decreases its value by one count, and returns the result to the same storage location. The storage address for reading the operand and storing the result is contained in location d. The result is left in the A register as well as in storage.

Execution begins by clearing the A register and entering a value of minus one. The operand is then read from storage and added to (A) in an 18-bit ones complement mode.

The operand is treated as a 12-bit positive quantity in this process. An 18-bit operand is formed from the 12-bit storage operand by adding six higher order zero bits. The result is left in the A register, and the lowest order 12 bits are returned to storage. Note that the arithmetic is essentially twos complement as viewed from storage, and the quantity in the A register is not necessarily equal to the result in storage.

There are four storage references required in the execution of this instruction. The first reference reads the contents of location d into the Q register. A second storage reference is made using (X) as the storage address. This operand is read into the X register and then added to (A). A third storage reference stores the lowest order 12 bits of the resulting sum using (Q) as the storage address. The fourth storage reference reads up the next instruction word.

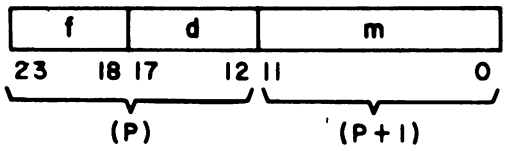
5000m LOAD (m)



This instruction clears the A register and enters a 12-bit operand from storage. The address for the operand is contained in the m designator for this instruction. The operand is entered in the A register as a 12-bit positive integer. The highest order six bits in the A register are always cleared.

Instruction execution begins with a storage reference for the m designator. This quantity is read into the X register. A second storage reference is then made using (X) as the storage address. This operand is read into the X register and then entered in the A register. A third storage reference is made to read the next instruction word.

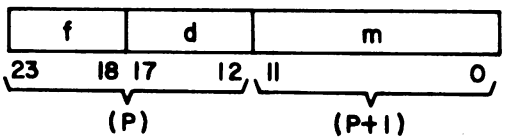
50dm LOAD (m + (d))



This instruction clears the A register and enters a 12-bit operand from storage. The address for the operand is formed by adding the m designator and the contents of location d in a 12-bit ones complement mode. The operand is entered in the A register as a 12-bit positive integer. The highest order six bits in the A register are always cleared. The d designator must have a nonzero value for this instruction.

There are four storage references required in the execution of this instruction. The first reference reads the m designator into the X register and then into the Q register. The second reference reads the contents of location d into the X register. The third reference uses (Q) + (X) as a storage address for the operand. This quantity is read into the X register and then entered in the A register. The fourth storage reference reads the next instruction word.

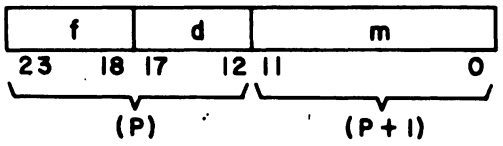
5100m ADD (A) + (m)



This instruction reads an operand from storage and adds it to the current contents of the A register. The addition is performed in an 18-bit ones complement mode. An 18-bit operand is formed from the 12-bit storage operand by adding six higher order zero bits. The storage address for the operand is contained in the m designator for this instruction.

Instruction execution begins with a storage reference for the m designator. This quantity is read into the X register. A second storage reference is then made using (X) as the storage address. This operand is read into the X register and then entered in the A register. A third storage reference is made to read the next instruction word.

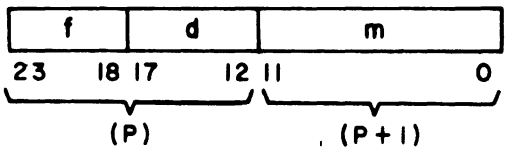
51dm ADD (A) + (m + (d))



This instruction reads an operand from storage and adds it to the current contents of the A register. The addition is performed in an 18-bit ones complement mode. An 18-bit operand is formed from the 12-bit storage operand by adding 6 higher order zero bits. The storage address for the operand is formed by adding the m designator and the contents of location d in a 12-bit ones complement mode. The d designator must have a nonzero value for this instruction.

There are four storage references required in the execution of this instruction. The first reference reads the m designator into the X register and then into the Q register. The second reference reads the contents of location d into the X register. The third reference uses (Q) + (X) as a storage address for the operand. This quantity is read into the X register and then entered in the A register. The fourth storage reference reads the next instruction word.

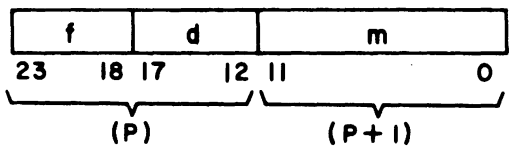
5200m SUBTRACT (A) - (m)



This instruction reads an operand from storage and subtracts it from the current contents of the A register. The result is left in the A register. The storage address for the operand is contained in the m designator for this instruction. The operation is performed by adding the complement of the operand to (A) in an 18-bit ones complement mode. An 18-bit operand for the addition is formed from the 12-bit storage operand by forcing the highest order six bits to a one value. The lowest order 12 bits are the bit-by-bit complement of the storage operand values.

Instruction execution begins with a storage reference for the m designator. This quantity is read into the X register. A second storage reference is then made using (X) as the storage address. This operand is read into the X register and then subtracted in the A register. A third storage reference is made to read the next instruction word.

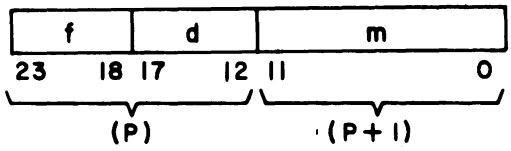
52dm SUBTRACT (A) - (m + (d))



This instruction reads an operand from storage and subtracts it from the current contents of the A register. The result is left in the A register. The address for the operand is formed by adding the m designator and the contents of location d in a 12-bit ones complement mode. The arithmetic operation is performed by adding the complement of the operand to (A) in an 18-bit ones complement mode. An 18-bit operand for the addition is formed from the 12-bit storage operand by forcing the highest order six bits to a value of one. The lowest order 12 bits are the bit-by-bit complement of the storage operand values. The d designator must have a nonzero value for this instruction.

There are four storage references required in the execution of this instruction. The first reference reads the m designator into the X register and then into the Q register. The second reference reads the contents of location d into the X register. The third reference uses (Q) + (X) as a storage address for the operand. This quantity is read into the X register and then subtracted in the A register. The fourth storage reference reads the next instruction word.

5300m LOGICAL DIFFERENCE (A) AND (m)

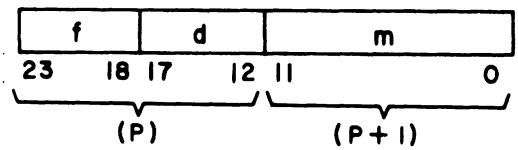


This instruction forms in the A register the logical difference of an operand read from storage and the original (A). The highest order six bits in the A register are not affected by this operation. The storage address for the operand is contained in the m designator for this instruction.

The logical difference is the result of a bit-by-bit comparison of the two binary quantities. If two corresponding bits are equal, the resulting bit is zero. If unequal, the result is one.

Instruction execution begins with a storage reference for the m designator. This quantity is read into the X register. A second storage reference is then made using (X) as the storage address. This operand is read into the X register and the logical difference entered in the A register. A third storage reference is made to read the next instruction word.

53dm LOGICAL DIFFERENCE (A) AND (m + (d))

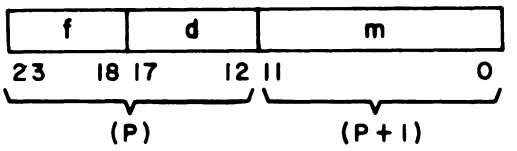


This instruction forms in the A register the logical difference of an operand read from storage and the original (A). The highest order six bits in the A register are not affected by this operation. The address for the operand is formed by adding the m designator and the contents of location d in a 12-bit ones complement mode. The d designator must have a nonzero value for this instruction.

The logical difference is the result of a bit-by-bit comparison of the two binary quantities. If two corresponding bits are equal, the resulting bit is zero. If unequal, the result is one.

There are four storage references required in the execution of this instruction. The first reference reads the m designator into the X register and then into the Q register. The second reference reads the contents of location d into the X register. The third reference uses (Q) + (X) as a storage address for the operand. This quantity is read into the X register and the logical difference entered in the A register. The fourth storage reference reads the next instruction word.

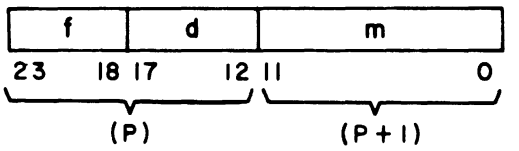
5400m STORE (A) AT (m)



This instruction stores the lowest order 12 bits of (A) in a storage location specified by the m designator. The contents of the A register is not altered in this process.

Execution begins with a storage reference for the m designator. This quantity is read into the X register. A second storage reference is made using (X) as the storage address. The lowest order 12 bits of (A) are stored during this storage cycle. A third storage reference is then made to read the next instruction word.

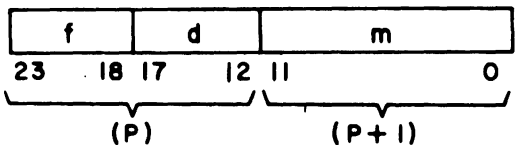
54dm STORE (A) AT (m + (d))



This instruction stores the lowest order 12 bits of (A). The storage address is formed by adding the m designator and the contents of location d in a 12-bit ones complement mode. The d designator must have a nonzero value for this instruction.

There are four storage references required in the execution of this instruction. The first reference reads the m designator into the X register and then into the Q register. The second reference reads the contents of location d into the X register. The third reference uses (Q) + (X) as a storage address for storing the lowest order 12 bits of the A register. The fourth storage reference reads the next instruction word.

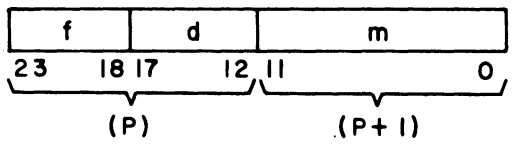
5500m REPLACE ADD (A) + (m)



This instruction reads an operand from storage and adds it to the current contents of the A register. The result is left in the A register and is also stored in the same memory location from which the operand was read. The addition is performed in an 18-bit ones complement mode. An 18-bit operand is formed from the 12-bit storage operand by adding six higher order zero bits. The result returned to storage is the lowest order 12 bits of the final (A). The storage address for reading the operand and storing the result is contained in the m designator for this instruction. Note that the result stored is not necessarily equal to the result left in the A register.

There are four storage references required in the execution of this instruction. The first reference reads the m designator into the X register and the Q register. A second reference is made using (X) as the storage address. This operand is read into the X register and is added into the A register. A third reference stores the lowest order 12 bits of (A) using (Q) as the storage address. The fourth reference reads up the next instruction word.

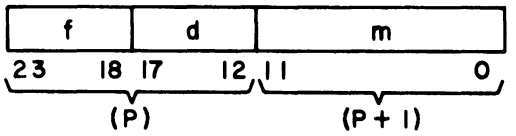
55dm REPLACE ADD (A) + (m + (d))



This instruction reads an operand from storage and adds it to the current contents of the A register. The result is left in the A register and is also stored in the same memory location from which the operand was read. The addition is performed in an 18-bit ones complement mode. An 18-bit operand is formed from the 12-bit storage operand by adding six higher zero bits. The result returned to storage is the lowest order 12 bits of the final (A). The storage address for reading the operand and storing the result is formed by adding the m designator to the contents of location d in a 12-bit ones complement mode. Note that the result stored is not necessarily equal to the result left in the A register.

There are five storage references required in the execution of this instruction. The first reference reads the m designator into the X register and then into the Q register. The second reference reads the contents of location d into the X register. The third reference uses (Q) + (X) to read the operand into the X register. The addition is performed in the A register. The quantity (Q) + (X) is entered in the Q register at this same time. The fourth storage reference stores the lowest order 12 bits of (A) using the new (Q) as a storage address. The last storage reference reads the next program instruction word.

5600m REPLACE ADD ONE (m)

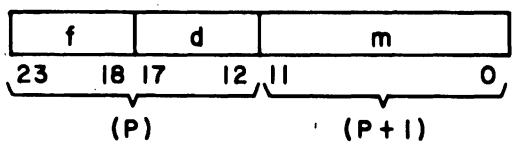


This instruction reads an operand from storage, increases its value by one count, and returns the result to the same storage location. The storage address for reading the operand and storing the result is contained in the m designator for this instruction. The result is left in the A register as well as in storage.

Execution begins by clearing the A register and entering a value of plus one. The operand is then read from storage and added to (A) in an 18-bit ones complement mode. The operand is treated as a 12-bit positive quantity in this process. An 18-bit operand is formed from the 12-bit storage operand by adding six higher order zero bits. The result is left in the A register, and the lowest order 12 bits are returned to storage. Note that the arithmetic is essentially twos complement as viewed from storage, and the quantity in the A register is not necessarily equal to the result in storage.

There are four storage references required in the execution of this instruction. The first reference reads the m designator from storage into the X register and then into the Q register. A second storage reference is made using (X) as the storage address. This operand is read into the X register and is added into the A register. A third reference stores the lowest order 12 bits of (A) using (Q) as the storage address. The fourth reference reads up the next instruction word.

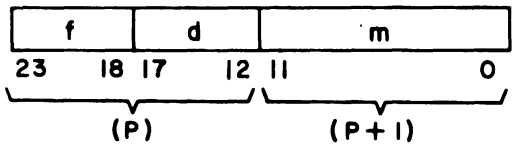
56dm REPLACE ADD ONE (m + (d))



This instruction reads an operand from storage, increases its value by one count, and returns the result to the same storage location. The storage address for reading the operand and storing the result is formed by adding the m designator to the contents of location d in a 12-bit ones complement mode. The result is left in the A register as well as in storage.

Execution begins by clearing the A register and entering a value of plus one. The m designator is read from storage and entered in the X register and then into the Q register. A second storage reference reads the contents of location d into the X register. A third reference reads the operand into the X register using (Q) + (X) as the storage address. The quantity (Q) + (X) is entered in the Q register at this same time. The operand is then added into the A register in an 18-bit ones complement mode. An 18-bit operand is formed from the 12-bit storage operand by adding six higher order zero bits. The result is left in the A register, and the lowest order 12 bits are returned to storage using (Q) as the storage address. Note that the arithmetic is essentially twos complement as viewed from storage, and the quantity in the A register is not necessarily equal to the result in storage. A fifth storage reference reads up the next instruction word.

5700m REPLACE SUBTRACT ONE (m)



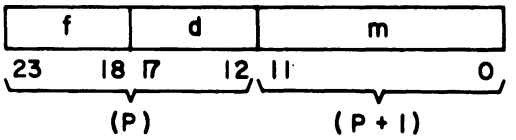
This instruction reads an operand from storage, decreases its value by one count, and returns the result to the same storage location. The storage address for reading the operand and storing the result is contained in the m designator for this instruction. The result is left in the A register as well as in storage.

Execution begins by clearing the A register and entering a value of minus one. The operand is then read from storage and added to (A) in an 18-bit ones complement mode.

The operand is treated as a 12-bit positive quantity in this process. An 18-bit operand is formed from the 12-bit storage operand by adding six higher order zero bits. The result is left in the A register, and the lowest order 12 bits are returned to storage. Note that the arithmetic is essentially twos complement as viewed from storage, and the quantity in the A register is not necessarily equal to the result in storage.

There are four storage references required in the execution of this instruction. The first reference reads the m designator from storage into the X register and then into the Q register. A second storage reference is made using (X) as the storage address. This operand is read into the X register and is added into the A register. A third reference stores the lowest order 12 bits of (A) using (Q) as the storage address. The fourth reference reads up the next instruction word.

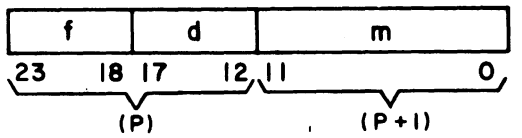
57dm REPLACE SUBTRACT ONE (m + (d))



This instruction reads an operand from storage, decreases its value by one count, and returns the result to the same storage location. The storage address for reading the operand and storing the result is formed by adding the m designator to the contents of location d in a 12-bit ones complement mode. The result is left in the A register as well as in storage.

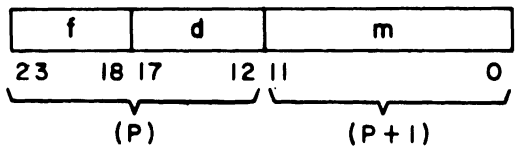
Execution begins by clearing the A register and entering a value of minus one. The m designator is read from storage and entered in the X register and then into the Q register. A second storage reference reads the contents of location d into the X register. A third reference reads the operand into the X register using (Q) + (X) as the storage address. The quantity (Q) + (X) is entered in the Q register at this same time. The operand is then added into the A register in an 18-bit ones complement mode. An 18-bit operand is formed from the 12-bit storage operand by adding six higher order zero bits. The result is left in the A register, and the lowest order 12 bits are returned to storage using (Q) as the storage address. Note that the arithmetic is essentially twos complement as viewed from storage, and the quantity in the A register is not necessarily equal to the result in storage. A fifth storage reference reads up the next instruction word.

60dm JUMP TO m IF CHANNEL d INPUT WORD FLAG SET



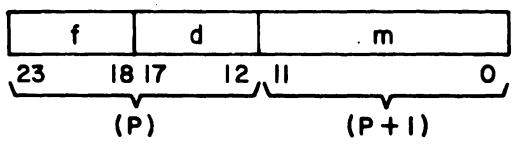
This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the condition of the channel d input word flag. A new program sequence is initiated beginning at address m if the channel d input word flag is set. The current program sequence is continued if the flag is not set.

61dm JUMP TO m IF CHANNEL d INPUT WORD FLAG NOT SET



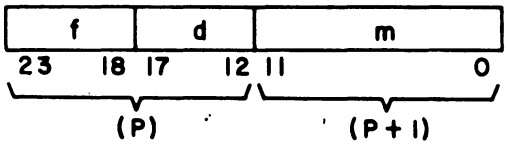
This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the condition of the channel d input word flag. A new program sequence is initiated beginning at address m if the channel d input word flag is not set. The current program sequence is continued if the flag is set.

62dm JUMP TO m IF CHANNEL d INPUT RECORD FLAG SET



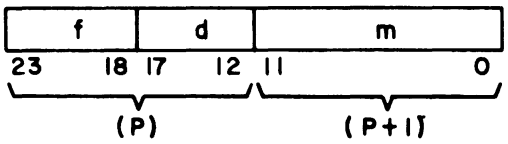
This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the condition of the channel d input record flag. A new program sequence is initiated beginning at address m if the channel d input record flag is set. The current program sequence is continued if the flag is not set.

63dm JUMP TO m IF CHANNEL d INPUT RECORD FLAG NOT SET



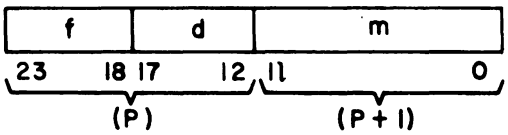
This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the condition of the channel d input record flag. A new program sequence is initiated beginning at address m if the channel d input record flag is not set. The current program sequence is continued if the flag is set.

64dm JUMP TO m IF CHANNEL d OUTPUT WORD FLAG SET



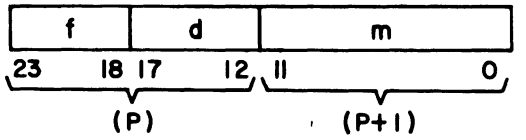
This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the condition of the channel d output word flag. A new program sequence is initiated beginning at address m if the channel d output word flag is set. The current program sequence is continued if the flag is not set.

65dm JUMP TO m IF CHANNEL d OUTPUT WORD FLAG NOT SET



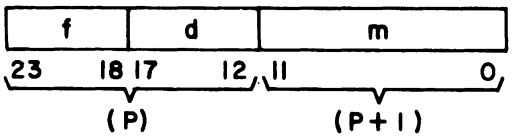
This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the condition of the channel d output word flag. A new program sequence is initiated beginning at address m if the channel d output word flag is not set. The current program sequence is continued if the flag is set.

66dm JUMP TO m IF CHANNEL d OUTPUT RECORD FLAG SET



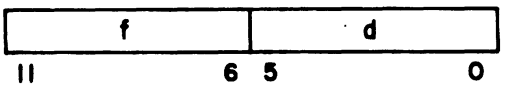
This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the condition of the channel d output record flag. A new program sequence is initiated beginning at address m if the channel d output record flag is set. The current program sequence is continued if the flag is not set.

67dm JUMP TO m IF CHANNEL d OUTPUT RECORD FLAG NOT SET



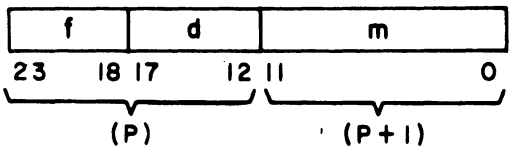
This is a conditional branch instruction which continues the current program sequence or jumps to a new program sequence, depending upon the condition of the channel d output record flag. A new program sequence is initiated beginning at address m if the channel d output record flag is not set. The current program sequence is continued if the flag is set.

70d INPUT TO A ON CHANNEL d



This instruction reads one word from input channel d and enters the word in the A register. This instruction is not executed until the channel d input word flag is set. If the flag is not set at the time the instruction is read from storage, the PPU program stops with the instruction in the fd register and waits until the flag is set by an external signal. The channel d input record flag does not affect execution of this instruction. This instruction clears the channel d input word flag and transmits a resume signal over the input cable after the word has been read into the A register. (Refer to appendix B for related information.)

71dm INPUT (A) WORDS TO m ON CHANNEL d



This instruction reads a block of data arriving on input channel d and stores the data in consecutive address locations in storage. 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 a record flag on the input channel.

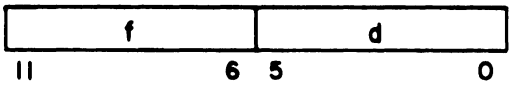
Instruction execution begins with a storage reference for the m designator. This quantity is read into the X register and then entered into the Q register. The Q register now contains the address for the first word of the data block. The d designator specifies the channel number, and the A register contains 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 channel d input 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 stops with the instruction in the fd register and waits until the flag is set by an external signal. The presence of a channel d input record flag is ignored for the first word of the block.

When the channel d input word flag is set, the word on the input channel data lines is read into PPU storage at location (Q). The contents of the A register is reduced by one count. The contents of the Q register is increased by one count in a 12-bit ones complement mode. The channel d input word and record flags are cleared, and a resume pulse is transmitted over the input cable. If the contents 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 channel d input word flag for the next word of the block.

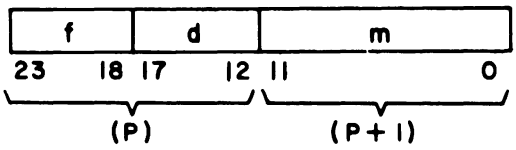
The setting of the channel d input record flag terminates the block input at any word after the first word. The sequence is terminated with (A) decremented by the number of words actually transmitted over the input channel. A noise word is entered in the next sequential storage location in the PPU block input storage area. The remaining locations in the PPU storage area are unaltered.

72d OUTPUT FROM A ON CHANNEL d



This instruction transmits one word over output channel d from the lowest order 12 bits of (A). The A register contents is not altered in the process. This instruction is not executed while the channel d output word flag is set. If the flag is set from a previous output instruction, the PPU program stops with this instruction in the fd register and waits for an external resume signal to clear the channel d output word flag. When this instruction is executed, the output word flag is set and a word pulse is transmitted over the output channel d cable.

73dm OUTPUT (A) WORDS FROM m ON CHANNEL d



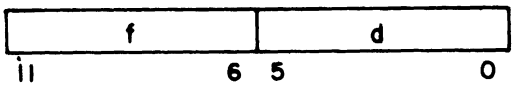
This instruction transmits a block of data over output channel d from consecutive storage locations beginning at address m. The length of the block is specified by the initial contents of the A register. A zero length causes the instruction to be executed as a pass instruction.

Instruction execution begins with a storage reference for the m designator. This quantity is read into the X register and then entered into the Q register. The Q register now contains the address for the first word of the data block. The d designator specifies 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 channel d output 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 stops with the instruction in the fd register and waits until the flag is cleared by a resume pulse over the output channel d cable. The presence of the channel d output record flag has no effect on the execution of this instruction.

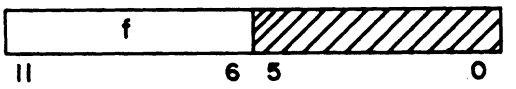
When the channel d output word flag is cleared, a word is read from storage location (Q) and is entered into the channel d output register. The channel d output word flag is set, and a word pulse is transmitted over the output cable. The contents of the A register is reduced by one count. The contents of the Q register is increased by one count in a 12-bit ones complement mode. If the contents 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 channel d output word flag to clear and repeats the sequence for the next word of the block.

74d SET OUTPUT RECORD FLAG ON CHANNEL d



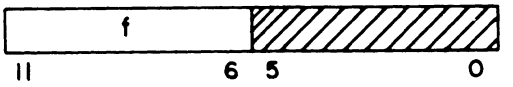
This instruction sets the channel d output record flag and transmits a record pulse over the output channel d cable. The previous status of the flag is ignored in this process. The instruction is executed and a record pulse transmitted even though the channel d output record flag was already set.

75xx PASS
76xx PASS



These two instructions are identical and perform no logical function. Each instruction results in a 5-clock-period delay.

77xx ERROR STOP



This instruction causes the PPU program to stop and indicate a program error condition to the MCU. The PPU can be restarted only by a dead start condition.

6000/7000 RESULT DIFFERENCES A

1. 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 = 2057 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

2. 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.

3. 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

4. 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

5. When instruction 22 or 23 is used for a right shift, the 7000 checks bits 6 through 11 for a shift greater than or equal to 64 (decimal) and ignores bits 12 through 16. For these instructions, the 6000 checks bits 6 through 10 and ignores bits 11 through 16.

6. 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)

7. 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 (decimal).

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 indicated previously.

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.

PROGRAMMING CONSIDERATIONS B

CPU PROGRAMMING CONSIDERATIONS

1. When the monitor mode flag is set in the PSD register, no interrupt requests will be honored (no I/O channel interrupt requests nor real time clock interrupt requests). When the monitor mode flag is clear, all interrupt requests will be honored (in priority order); I/O channel interrupt requests and real time clock interrupt requests.

2. I/O channel interrupt exchange packages must have the monitor mode flag bit set in PSD. If this bit is not set, the I/O channel interrupt request will cause repeated interrupts of the interrupt program.

3. The CPU dead start exchange jump is the result of the CPU dead start (master clear) signal clearing the entire I/O channel interrupt request register. This results in a channel 0 interrupt request which causes an exchange jump when the CPU dead start signal drops. This exchange jump uses the package at SCM absolute address 0. Because the CPU dead start exchange jump is the result of an I/O interrupt request, the dead start exchange package must have the monitor mode flag bit set in PSD. If this bit is not set, the CPU dead start program will be re-interrupted by the channel 0 interrupt request.

4. Like other exchange jump sequences, the CPU dead start exchange jump swaps register data with SCM exchange package data (locations 0 through 17). All exchange data swapped into SCM will be as it was in the CPU registers except for the PSD register data. The PSD bits will be correct except for the unconditional clearing of the monitor mode flag and the unconditional setting of the program range error flag. The program range error flag is set because of the time delay between the dropping of the CPU dead start signal and the setting of the request interrupt flag (RIF).

5. In the CPU hardware there is more than one P register. There are six P registers, each containing the same value at all times. The different P registers feed different circuits. Care should be taken to ensure that all P registers contain the same value when working on P related problems.

6. The P register is 18 bits in length and can contain an 18-bit program (P) address. The IAS registers are 18 bits in length, but bits 16 and 17 are always forced to zero. Therefore, the IAS registers contain only 16 usable bits. The NSA and IFA registers are only 16 bits in length.

   As long as proper programming is used and the FLS is never set to a value larger than 177777 (octal), no problems will occur if a P address larger than 177777 (Octal) is generated. A P address larger than 177777 (octal) would result in program termination because of an SCM direct range error (P GTE FLS). However, if FLS is set to a value larger than 177777 (octal), such as 400000 (octal), it is possible to hang up the CPU by generating a P address larger than 177777 (octal) but smaller than FLS (ie: P = 200000). This hang-up will occur because there can never be coincidence between IAS and P. This results in no instruction word being sent to CIW and instruction fetch control continually referencing SCM for the instruction word at P = 200000. Since NSA is set to 000000 (16 bits) and P is set to 200000, there will never be instruction address coincidence. The address sent to SCM will be actually 000000, since IFA is only 16 bits.

7. The 00 instruction can be blocked from setting the program range error flag (PSD) under the following condition.

   If an I/O interrupt request sets the RIF at the same time as the 00 instruction is entered into the translation bits of the CIW (top bits), the setting of the program range error flag is blocked by RIF. The P register will have been advanced to the next location. If the next location contains légal instruction code, the I/O interrupt program will return control to this instruction word and the 00 instruction will have been missed since the program range error flag was never set.

8. A master clear of the CPU can cause an SCM parity error. To prevent a parity error caused by a master clear from being confused with a parity error caused by a system failure, check SCM after each master clear to verify that it is free of parity errors. Verify the existence of any parity errors by reading all addresses. Eliminate any parity errors by writing into the affected addresses.

PPU PROGRAMMING CONSIDERATIONS

1. PPU memory parity errors can be caused by dead starting a PPU while it is executing a program. To eliminate the sensing of such parity errors (false parity errors), the MCU PPU dead start and loading program should do one of two things. Either the PPU's memory can be completely loaded, PPU resident padded out with zero words, and then the PPU clear parity sent, or the PPU resident can sweep the remainder of the PPU memory, and then allow the PPU clear parity signal to be sent by the MCU program.

2. If a data sample occurs at the same time that data is changing on the line during a block input, a parity error is likely to take place in the PPU memory.

3. If a MUX output channel is reset at the same time that the PPU is doing a read from the MUX, a parity error is likely to occur in the PPU memory.

4. Dead starting a PPU while it is executing a program will normally wipe out one or two words of the program. This is due to the dead start signal clearing the X register when the PPU is in the process of a memory read/write cycle. A zero word will replace the word just read from memory.

5. The dead start signal should be applied for a minimum of 32_{10} clock periods when performing a dead start or dead dump operation on a running PPU. Since the dead start signal does not clear the shift count register, application of the signal for at least 32 clock periods (allows for maximum number of shifts) ensures that shift operations are completed prior to the end of the dead start or dead dump operation. If the dead start signal were to drop before the shift count register has cleared itself, the contents of the A register would shift and become incorrect data.

6. The dead start signal from the MCU to the PPU (through the scanner) clears the PPU's input and output channels word and record flags. While the dead start signal is up, all of the PPU's input channel resume flags are forced to ones.

7. A PPU is set up for being dead dumped by sending the dead start signal, dropping the dead start signal, sending the dead dump signal and dropping the dead dump signal. This order of events is required to ensure that all PPU memory locations are dumped (excluding location 7777 octal).

8. If an input record flag is forced to a 1 during a block input, the block input instruction exits. The PPU processes only the data which was on the input channel at the time of the forced record flag.

9. Input channels with forced input word flags can be responsible for PPU parity errors. This problem occurs when reading data from such a channel directly into memory (71 instruction). This problem is alleviated if all such channel data is input to the A register (70 instruction), and then written into memory from the A register.

10. If a status word is entered into the A register while the channel input word flag is in a forced 1 condition, there is always the possibility that the word was in a transitional state at the time of entry. To allow for this possibility, consecutively input the status word twice and then compare the two inputs to make sure that they are the same.

11. When a block input instruction is terminated by a record pulse, the input record flag will remain set until the next input instruction has input at least one word. The last word received during the block input will be duplicated in the last block location + 1.

12. When terminating a block output by sending an output record pulse, the output record pulse should not be sent until the resume for the last word sent has been received. If this is not done (the output record pulse is sent before the resume is received), the receiving device (PPU or equipment) may lose the output record pulse and hang up waiting for another output record pulse.

13. When the PPU is not selected through the scanner (scanner channel selected for a different PPU), the PPU's output channel 0 will receive a constant output resume.

14. It is impossible for a PPU program to reference memory location 7777 (octal).

15. PPU program loops should not be smaller than four words. Program loops smaller than four words may cause marginal memory operation (bit dropping) due to core overheating.

GLOSSARY

Central Processor

A0-A7	Address Registers
B0-B7	Index Registers
BPA	Breakpoint Address Register
CIW	Current Instruction Word Register
CPU	Central Processing Unit
EEA	Error Exit Address Register
FLL	Field Length-LCM Register
FLS	Field Length-SCM Register
IAS	Instruction Address Stack
IWS	Instruction Word Stack
LCM	Large Core Memory
MUX	I/O Multiplexer
NEA	Normal Exit Address
P	Program Address Register
PSD	Program Status Designator Register
RAL	Reference Address-LCM Register
RAS	Reference Address-SCM Register
SAS	Storage Address Stack
SCM	Small Core Memory
X0-X7	Operand Registers

Peripheral Processor Unit (PPU), Maintenance Control Unit (MCU)

A	Arithmetic Register
fd	Instruction Register
P	Program Address Register
Q	Working Register
X	Memory Read Register

COMMENT SHEET

MANUAL TITLE CONTROL DATA® 7700 Dual-Processor Computer System Hardware Reference Manual

PUBLICATION NO. 60396300 REVISION A

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

AA3419 REV. 11/69 PRINTED IN USA

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

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

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

POSTAGE WILL BE PAID BY

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

MD 220

FOLD

FOLD

CUT ALONG LINE

CONTROL DATA
CORPORATION