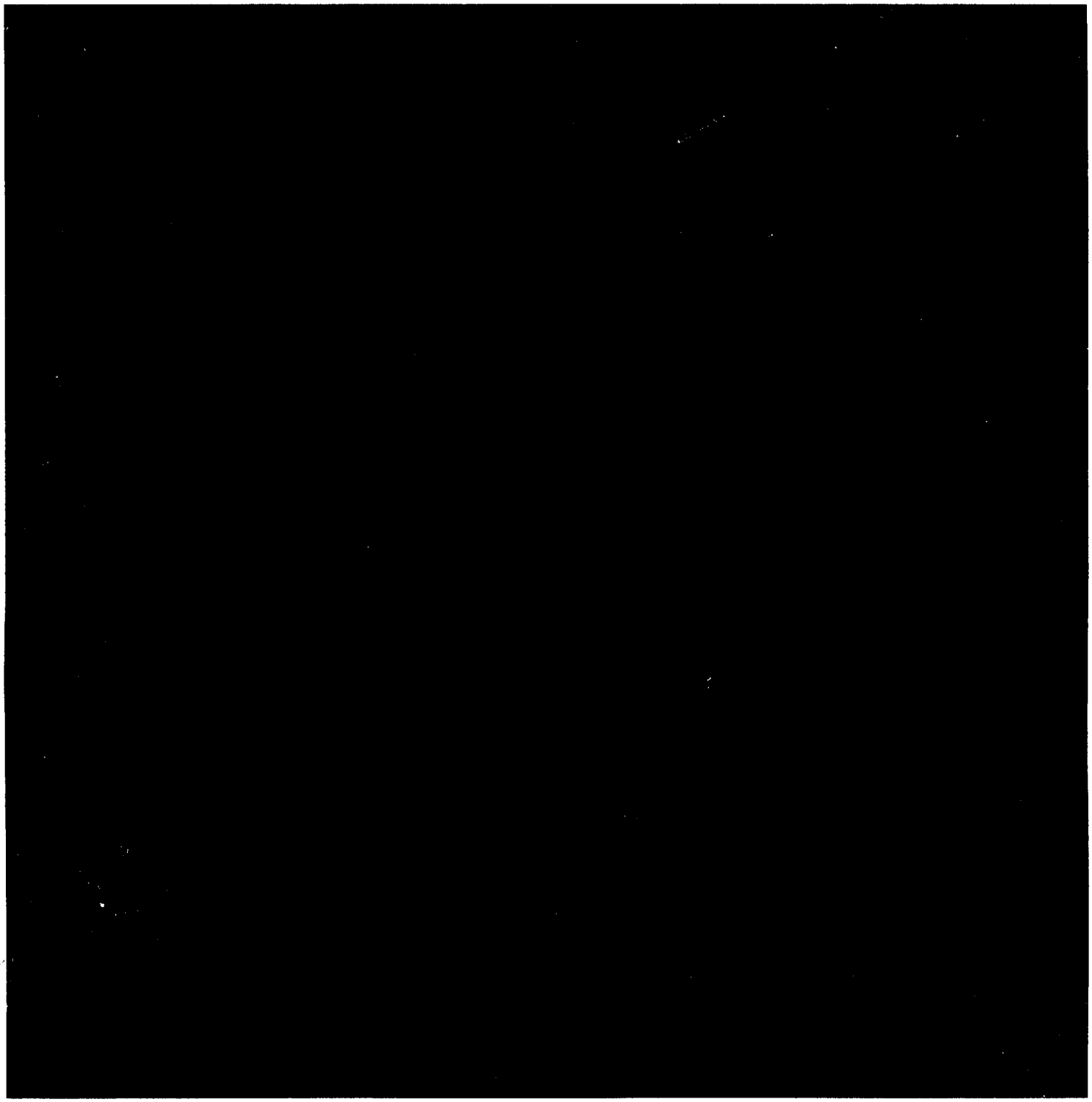
CONTROL DATA®
6400/6500/6600 COMPUTER SYSTEMS
Reference Manual



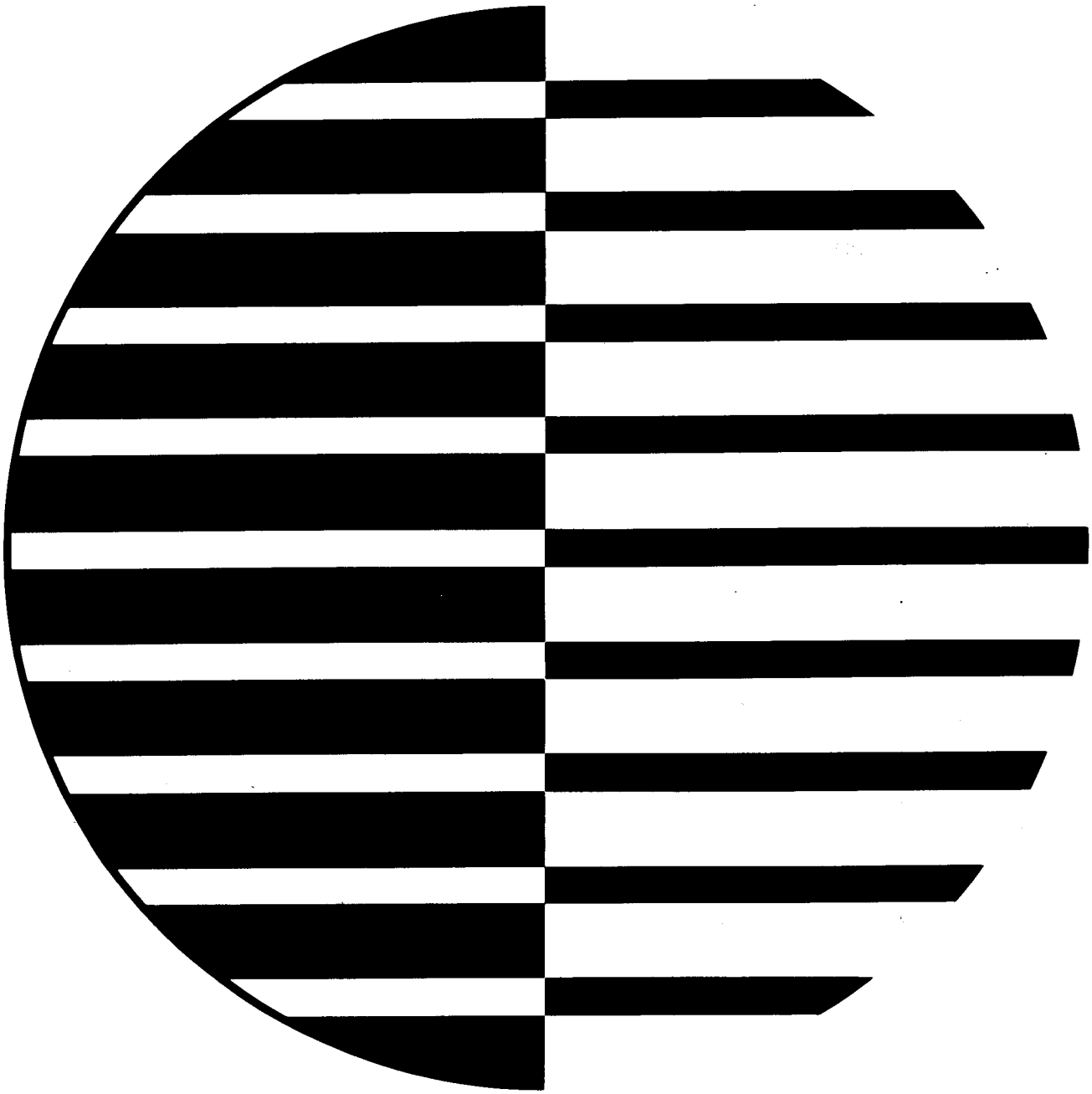
INDEX TO CENTRAL PROCESSOR INSTRUCTIONS

NUMERICAL

ALPHABETICAL

*Jump to K + Bi and Jump to K if Bi---tests made in Increment unit.
**Jump to K if Xj---tests made in Long Add unit.

CONTROL DATA®
6400/6500/6600 COMPUTER SYSTEMS
Reference Manual



RECORD of REVISIONS

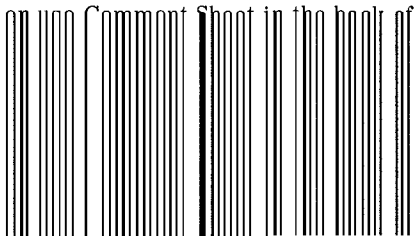
REVISION	NOTES
	This manual obsoletes the 6600 Computer System Reference Manual, Pub. No. 60045000.
A	Publication Change Order CA13186. Addition and deletion of information for technical accuracy.
(4-5-66)	Title changed to 6400/6600 Computer Systems Reference Manual. This edition obsoletes all
	previous editions.
B	Publication Change Order 14568, no Product Designation change. Pages 3-13, B-4, B-9, B-12,
(9-1-66)	B-13, B-14, B-15, B-16, D-5 and Index-2 revised.
C	Publication Change Order 15036, no Product Designation change. Page D-6 revised.
(10-27-66)	
D	Publication Change Order 15866; no Product Designation change. Addition of 6500 information;
(2-21-67)	title changed to 6400/6500/6600 Computer Systems Reference Manual. The following pages revised:
	cover and title page, iv, v, frontispiece, 1-1, 1-2, 1-3, 1-4, 1-5, 1-7, 1-8, 3-1, 3-2, 3-6, 3-7,
	3-12, 3-16, 3-20, 3-51, 4-1, 4-13, 4-24, 4-25, 4-29, 4-30, 4-36, 5-1, 6-1, 6-4, Appendix A title
	page, A-1, A-2, A-3, A-4, A-5, A-6, B-2, B-3, B-5, B-6, B-7, B-8, C-1, D-1, D-2, D-3, D-4,
	D-6, and Comment Sheet.

FORM CA230 REV. 1-67

Address comments concerning this manual to:

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

Pub No. 60100000

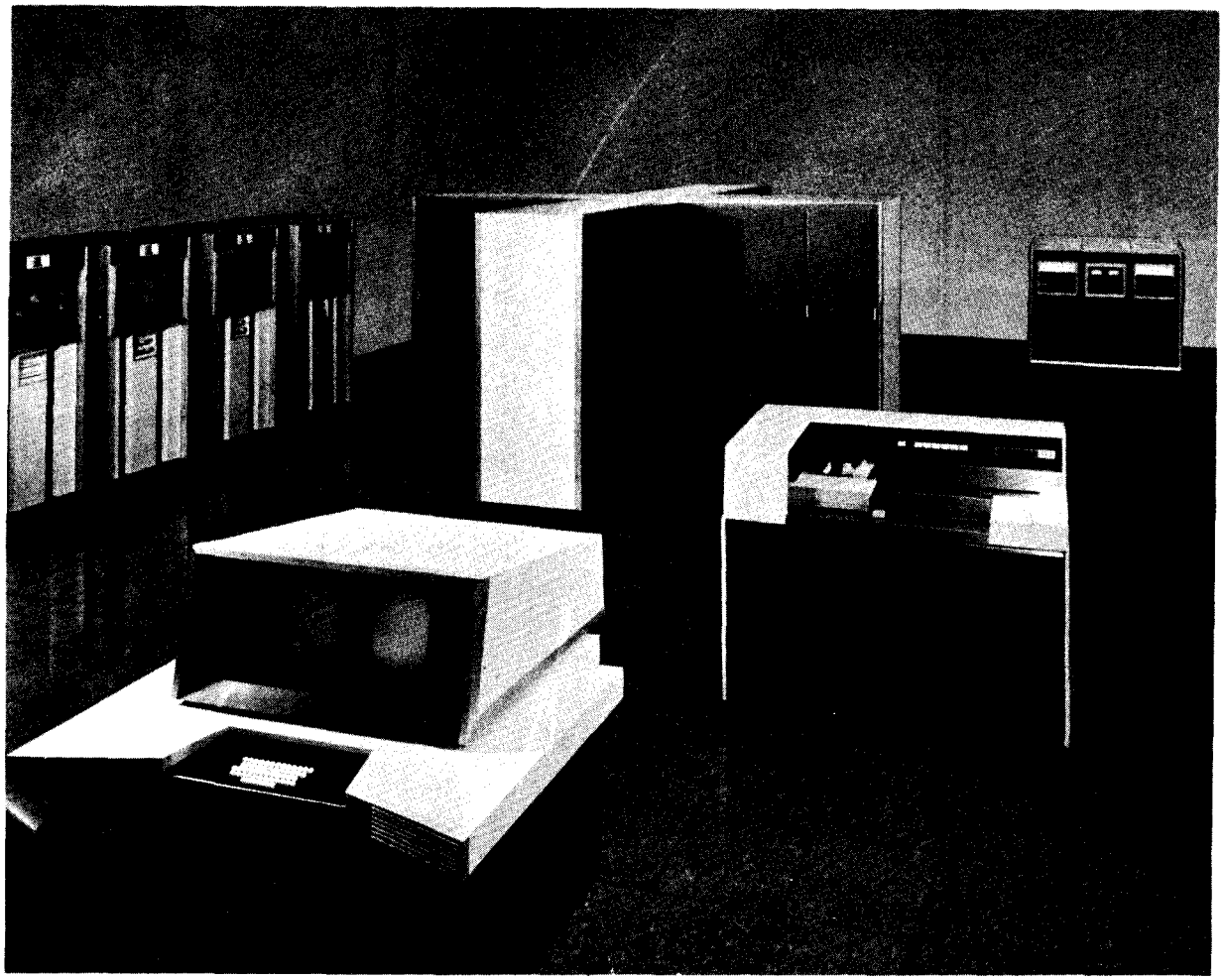


CONTENTS

CP MONITOR INFO

FIGURES

TABLES



A CONTROL DATA 6000 SERIES COMPUTER SYSTEM

Display console (foreground) - includes a keyboard for manual input and operator control and two 10-inch display tubes for display of problem status and operator directives.

Mainframe (center) - contains 10 Peripheral and Control Processors, Central Processor, Central Memory, some I/O synchronizers. The main frame in this photo is that of the 6600 Computer System; the mainframes for the 6400 and 6500 systems differ in physical appearance, depending on options included in the systems.

CONTROL DATA 607 Magnetic Tape Transport (left front) - 1/2-inch magnetic tape units for supplementary storage; binary or BCD data handled at 200, 556, or 800 bpi.

CONTROL DATA 626 Magnetic Tape Transport (left rear) - 1-inch magnetic tape units for supplementary storage; binary data handled at 800 bpi.

CONTROL DATA 405 Card Reader (right front) - reads binary or BCD cards at 1200 card per minute rate.

Disk file (right rear) - supplementary mass storage device; holds 500 million bits of information.

1. SYSTEM DESCRIPTION

INTRODUCTION

The CONTROL DATA* 6400, 6500, and 6600 Computer Systems are three large-scale, solid-state, general-purpose digital computing systems. The advanced design techniques incorporated in these systems provide for extremely fast solutions to data processing, scientific, and control center problems, as well as multiprocessing, time-sharing, and management information applications.

Each of the computing systems has at least eleven independent computers (Figure 1-1). Ten of these, constructed with the peripheral and operating system in mind, are Peripheral and Control Processors. Each of these ten has separate memory and can execute programs independently of each other or the Central Processor.

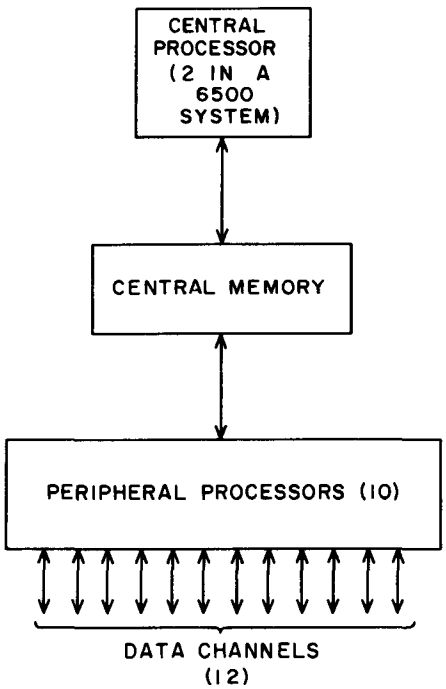


Figure 1-1. CONTROL DATA 6400/6500/6600 Computer Systems

*Registered trademark of Control Data Corporation

The eleventh computer, the Central Processor, is a very high speed arithmetic device. The common element of the Peripheral and Control Processors and the Central Processor is a large Central Memory.

In solving a problem, one or more Peripheral and Control Processors are used for high speed information transfer in and out of the system and to provide operator control. A number of problems may operate concurrently by time-sharing the Central Processor. (To facilitate this, the Central Processor may operate in Central Memory only within address bounds prescribed by a Peripheral and Control Processor.) Further concurrency is obtained within the Central Processor by parallel action of various functional segments. Similarly, Central Memory is organized in 32 logically independent banks of 4096 words (60-bit). Several banks may be in operation simultaneously, thereby minimizing execution time. The multiple operating modes of all segments of the computer, in combination with high-speed transistor circuits, produce a very high over-all computing speed.

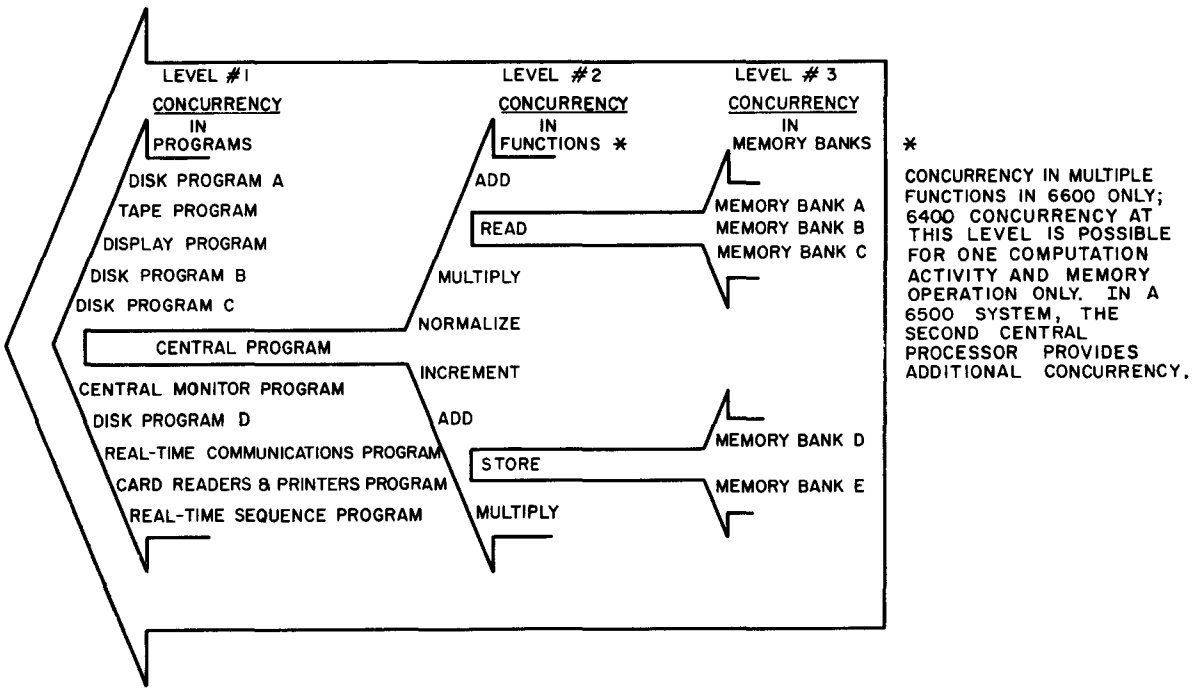


Figure 1-2. Concurrent Operations in the 6400/6500/6600

The Peripheral and Control Processor input/output facility provides a flexible arrangement for very high speed communication with a variety of I/O devices. Some of the I/O devices available with the 6400, 6500, and 6600 systems are listed below. (Refer to the 6000 Series Peripheral Equipment Reference Manual for additional external equipment information.)

- CONTROL DATA 6602/6612 Console Display: a display console with manual keyboard. This program-controlled unit displays problem status on two cathode ray tubes and handles operator directives from an alphanumeric keyboard similar to a standard typewriter keyboard.

- CONTROL DATA 6603 Disk System: a mass storage disk file providing nominal storage of 500 million bits.

- CONTROL DATA 626 Magnetic Tape Transports: one-inch magnetic tape units which handle binary data recording at 800 bpi on tapes up to 2400 feet long.

- CONTROL DATA 6682/6683 Satellite Coupler: a systems expansion device which permits direct connection between any two 6400 or 6600 systems via two standard 12-bit bi-directional data channels; two 6682/6683's are required for this.

- CONTROL DATA 6681 Data Channel Converter: a device which permits 6400, 6500, and 6600 systems to use CONTROL DATA 3000 Series peripheral equipment. Examples of available 3000 Series peripheral equipment are: card equipment (readers/punches), magnetic tape equipment, and line printers.

SYSTEMS CHARACTERISTICS SUMMARY

The following summary lists characteristics of the 6400, 6500, and 6600 Computer Systems. Where characteristics differ between the systems, differences are noted; otherwise, characteristics listed are common to all systems.

System Characteristics

- Large-scale, general-purpose computer system
- 11 independent computers; 12 in the Dual Processor 6500 system
  - 1 Central Processor (60-bit); 2 Central Processors in the 6500 system
  - 10 Peripheral and Control Processors (12-bit)

  Central Memory (60-bit)

  Display console and keyboard
- System communicates with a variety of external equipment
  - Disk files
  - Magnetic tapes
  - Card equipment
  - Printers
- Central Memory common to the system computers
- Maximum Central Memory storage capability 131,072 words (60-bit)

  Major Cycle = 1000 ns*

  Minor Cycle = 100 ns

  Memory organized in 32 banks of 4096 words

  Multiphase
- Central Processor instructions
  - Arithmetic, logical, indexing, branch
- Peripheral and Control Processor instructions
  - Add/Subtract, logical, input/output, access to Central Processor and Central Memory
- Each Peripheral and Control Processor has 12-bit 4096-word memory
- Solid-state system
  - Transistor logic

Central Processor Characteristics

6600

10 arithmetic and logical units

Add	Shift
Multiply	Branch
Multiply	Boolean
Divide	Increment
Long add	Increment

*ns = nanoseconds

- 24 operating registers for functional units
  - 8 operand (60-bit)
  - 8 address (18-bit)
  - 8 increment (18-bit)
- 8 transistor registers (60-bit) hold 32 instructions (15-bit) or 16 instructions (30-bit) or a combination of the two for servicing functional units.

6400 and 6500

- Unified arithmetic section, operating in sequential manner (one per processor in 6500)
- 24 operating registers (one set per processor in 6500)
  - 8 operand (60-bit)
  - 8 address (18-bit)
  - 8 increment (18-bit)
- Instruction Buffer register (60-bit)

Common Central Processor Characteristics

- Floating point arithmetic
  - Single and double precision
  - Optional rounding and normalizing
- Format
  - Integer coefficient - 48 bits
  - Biased exponent - 11 bits (2^{10})
  - Coefficient sign - 1 bit
- Fixed point arithmetic (subset of floating point arithmetic)
  - Full 60-bit add/subtract
- Controlled and started by Peripheral and Control Processors
- Addresses in Central Memory relative

Peripheral and Control Processor Characteristics

- 10 identical processors (characteristics as listed are per processor except as noted)
- 4096-word magnetic core memory (12-bit)
- Random access, coincident - current
  - Major Cycle = 1000 ns
  - Minor Cycle = 100 ns

- 12 input/output channels
  - All channels common to all processors
  - Maximum transfer rate per channel - one word/major cycle
  - All 12 channels may be active simultaneously
  - All channels 12-bit bi-directional
- Real-time clock (period = 4096 major cycles)
- Instructions
  - Add/Subtract
  - Logical
  - Branch
  - Input/Output
  - Central Processor access
  - Central Memory access
- Average instruction execution time = two major cycles
- Indirect addressing
- Indexed addressing

Central Memory Characteristics

- 131,072 words (maximum size)
- 60-bit words
- Memory organized in 32 logically independent banks of 4096 words with corresponding multiphasing of banks; (32 banks is maximum memory size)

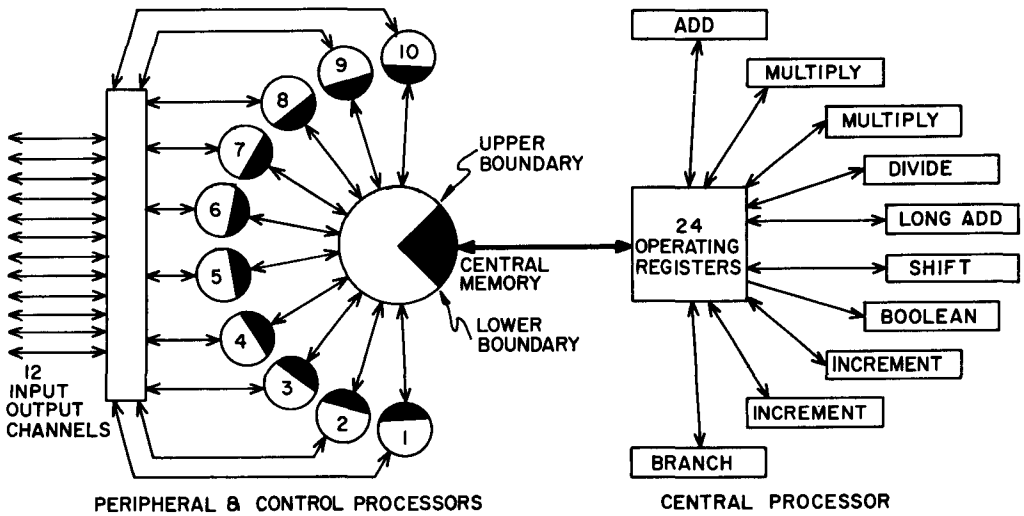


Figure 1-3. Block Diagram of 6600 System

- Random access, coincident-current, magnetic core
- One major cycle for read-write
- Maximum memory reference rate to all banks - one address/minor cycle
- Maximum rate of data flow to/from memory - one word/minor cycle

Display Console Characteristics

- Two display tubes
- Modes
  - Character
  - Dot
- Character size
  - Large - 16 characters/line
  - Medium - 32 characters/line
  - Small - 64 characters/line
- Characters
  - 26 alphabetic
  - 10 numeric
  - 11 special

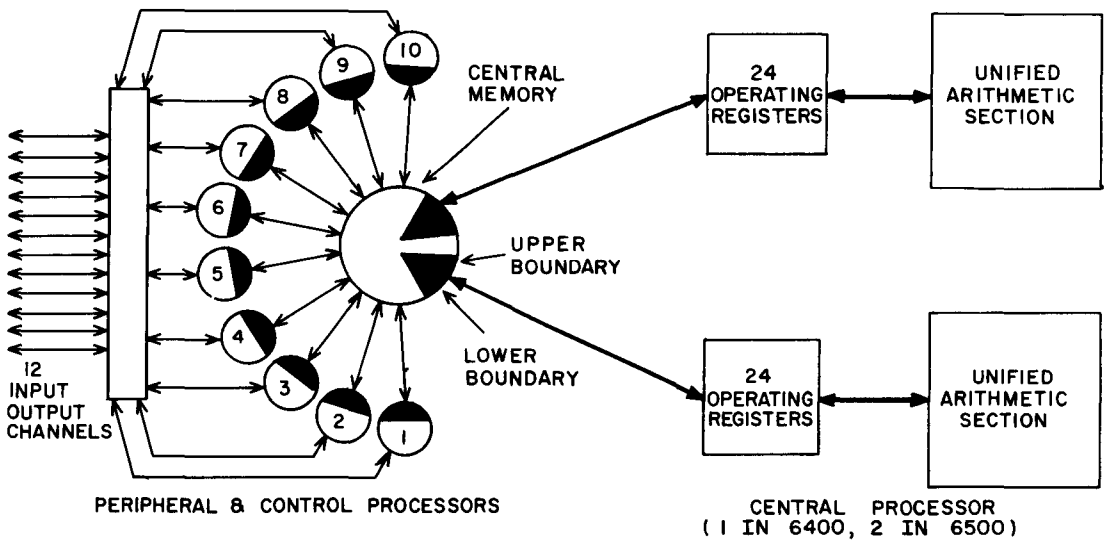


Figure 1-4. Block Diagram of 6400 and 6500 Systems

SYSTEMS OPTIONS

The foregoing summary of characteristics assumed a 6400, 6500, or 6600 system with 10 Peripheral and Control Processors, a Central Processor (except for the 6500 system with its two identical Central Processors), and Central Memory with 131,072 words (60-bit) of magnetic core storage.

Options listed below are available within each system unless otherwise noted.

- Central Memory with 131,072 words (60-bit) of magnetic core storage.
- Central Memory with 65,536 words (60-bit) of magnetic core storage. (This is the minimum Central Memory size available for the 6500 Computer System.)
- Central Memory with 32,768 words (60-bit) of magnetic core storage.
- Extended Core Storage: Magnetic core storage available in the following sizes:
  - 125,952 words (60-bit)
  - 251,904 words (60-bit)
  - 503,808 words (60-bit)
  - 1,007,616 words (60-bit)
  - 2,015,232 words (60-bit)
- Extended Core Storage Controller: couples up to 2,015,232 words of Extended Core Storage to from one to four 6400, 6500, or 6600 central computer(s) or Augmented I/O Buffer and Control unit(s) in any combination.
- Augmented I/O Buffer and Control: includes 16,384 words (60-bit) of magnetic core storage and 10 Peripheral and Control Processors with storage.
- Central Processor Monitor Facility (Central and Monitor Exchange Jump instructions). Refer to the publications for Standard Options 10103 and 10104.

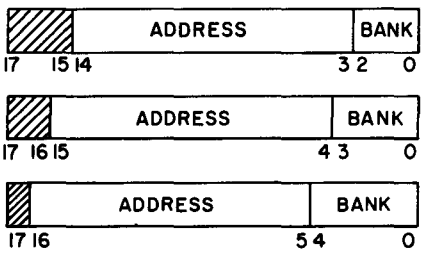
2. CENTRAL MEMORY

ORGANIZATION

Central Memory is organized into 32K, 65K, or 131K words (60-bit) in 8, 16, or 32 banks of 4096 words each. The banks are logically independent, and consecutive addresses go to different banks. Banks may be phased into operation at minor cycle* intervals, resulting in very high Central Memory operating speed. The Central Memory address and data control mechanisms permit a word to move to or from Central Memory every minor cycle.

ADDRESS FORMAT

The location of each word in Central Memory is identified by an assigned number (address), which consists of 18 bits. Address formats are shown below for 8-bank (32K), 16-bank (65K), and 32-bank (131K) systems. Within the address format, the bank portion specifies one of 8, 16, or 32 banks; the 12-bit address defines one of 4096 separate



locations within the specified bank. Addresses written or compiled in the conventional manner reference consecutive banks and hence make most efficient use of the bank phasing feature.

CENTRAL MEMORY ACCESS

References to Central Memory from all areas of the system (Central Processor and Peripheral and Control Processors) go to a common address clearing house called a stunt box and are sent from there to all banks in Central Memory. The stunt box accepts addresses from the various sources under a priority system and at a maximum rate of one address every minor cycle.

*Minor cycle=100 ns

An address is sent to all banks, and the correct bank, if free, accepts the address and indicates this to the stunt box. The associated data word is then sent to or stored from a central data distributor. The bank ignores the address if it is busy processing a previous address. The stunt box issues addresses at a maximum rate of one every minor cycle.

The stunt box saves, in a hopper mechanism, each address that it sends to Central Memory and then reissues it (and again saves it) under priority control in the event it is not accepted because of bank conflict. The address issue-save process repeats until the address is accepted, at which time the address is dropped from the hopper and the read or store data word is distributed. A fixed time lapse from address-issue to the memory-accept synchronizes the action taken.

The hopper (i.e., a previously unaccepted address) has highest priority in issuing addresses to Central Memory. The Central Processor and Peripheral and Control Processors (all 10 share a common path to the stunt box) follow in that order.

A data distributor which is common to all processors handles all data words to and from Central Memory (the Peripheral and Control Processors share one read path and one write path to the distributor). A series of buffer registers in the distributor provides temporary storage for words to be written into storage when the addresses are not immediately accepted because of bank conflict.

Each group of four banks communicates with the distributor on separate 60-bit read and write paths, but only one word moves on the data paths at one time. However, words can move at minor cycle intervals between the distributor and Central Memory or distributor and address-sender.

Data words and addresses are correlated by control information (tags) entered in the stunt box with the address. The tags define the address sender, origin/destination of data, and whether the address is a Read, Write, or Exchange Jump address.

MEMORY PROTECTION

All Central Processor references to Central Memory for new instructions, or to read and store data, are made relative to the Reference Address. The Reference Ad-

dress defines the lower limit of a Central Memory program. Changes to the Reference Address permit easy relocation of programs in Central Memory.

During an Exchange Jump, an 18-bit Reference Address and an 18-bit Field Length (parts of the Exchange Jump package) are loaded into their respective registers to define the Central Memory limits of the program initiated by the Exchange Jump.

The relationship between absolute memory address, relative memory address, Reference Address (RA), and Field Length (FL) is indicated in Figure 2-1.

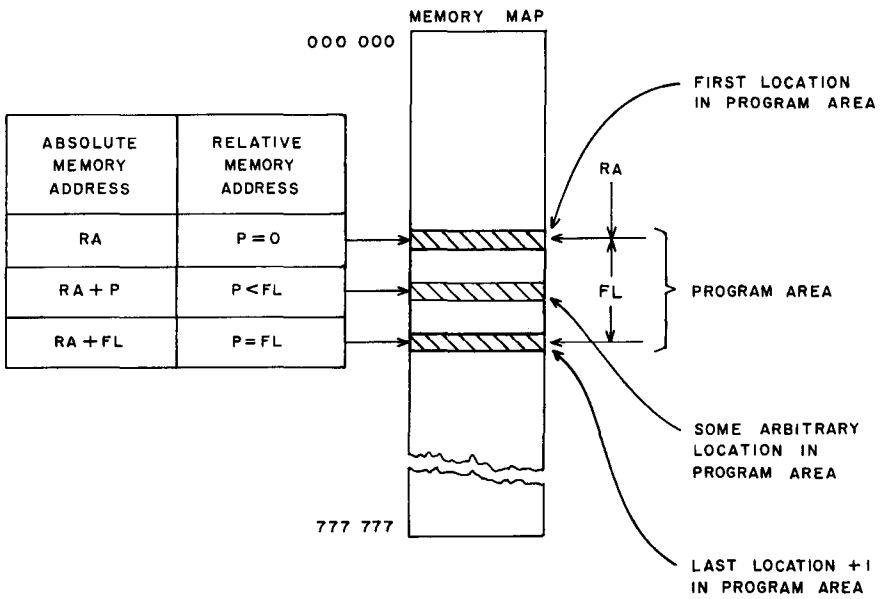


Figure 2-1. Memory Map

The following relationships must be true if the program is to operate within its bounds:

$$RA \leq (RA + P) < (RA + FL) \quad \text{(Absolute Memory Addresses), or}$$
$$0 \leq P < FL \quad \text{(Relative Memory Addresses)}$$

NOTE

1) FL is the number of 60-bit words comprising the program, not an address.

2) To avoid possible "artificial" range faults, instructions should not be stored near the upper limit address of the Field Length. For example, using absolute address $[(RA + FL) - 1]$ for an instruction produces a

range fault when the (look-ahead) Read Next Instruction occurs to (RA + FL). Data should always be stored in addresses near or approaching absolute location (RA + FL), rather than instructions.

An optional exit condition (EM in the Exchange Jump package) allows the Central Processor to stop on a memory reference outside the limits expressed above.

3. CENTRAL PROCESSOR

ORGANIZATION

The Central Processor is an extremely high-speed arithmetic processor which communicates only with Central Memory. It consists (functionally) of an arithmetic unit and a control unit. The arithmetic unit contains all logic necessary to execute the arithmetic, manipulative and logical operations. The control unit directs the arithmetic operations and provides the interface between the arithmetic unit and Central Memory. It also performs instruction fetching, address preparation, memory protection, and data fetching and storing.

The Central Processor is isolated from the Peripheral and Control Processors and is thus free to carry on high-speed computation unencumbered by input/output requirements.

The organization of the Central Processor in the 6400 system differs from the 6600 Central Processor in two important respects. The 6500 system has two Central Processors; each similar to the 6400 Central Processor. Central Processor differences are tabulated in Table 3-1.

TABLE 3-1. CENTRAL PROCESSOR DIFFERENCES

SYSTEM	INSTRUCTION REGISTERS	ARITHMETIC SECTION
6400 and 6500 Central Processors	Instruction Buffer Register; holds one 60-bit instruction word.	Unified Arithmetic Section; executes instructions in serial order. Requires no reservation control.
6600 Central Processor	Instruction Stack; holds eight 60-bit instruction words.	Ten functional (arithmetic & logical) units; operate concurrently on unrelated instructions. Require reservation control.

The following discussion details the operation of the Central Processor in the 6600 system. With the exception of differences noted in the above table (and the inherent effects on Central Processor operation), the 6400 system Central Processor operation is identical. Each of the two 6500 Central Processors operates identically with the 6400 Central Processor.

Programs for the Central Processor are held in Central Memory. A program is begun by an Exchange Jump instruction from a Peripheral and Control Processor. This instruction also specifies a segment of Central Memory for the central program, specifies the mode of exit (normal or error) of the program, and sets initial quantities in the X, B, and A registers.

High speed in the Central Processor depends first on minimizing memory references. Twenty-four registers are provided to lower the Central Memory requirements for arithmetic operands and results. These 24 are divided into:

- 8 address registers of 18 bits in length
- 8 increment registers of 18 bits in length
- 8 operand registers of 60 bits in length

Eight 60-bit registers are provided to hold instructions (6600), thereby limiting the number of memory reads for repetitive instructions, especially in inner loops. Multiple banks of Central Memory are also provided to minimize memory reference time. References to different banks of memory may be handled without wait.

Speed of operation in a conventional computer is also limited by the serial manner in which instructions are executed; instructions are executed sequentially in time with little or no concurrency.

In the 6600 Computer System, this delay is minimized by providing 10 arithmetic (functional) units and a reservation control. Unrelated instructions are executed simultaneously, provided no conflicts exist in the arithmetic units.

The 6400 or 6500, with its unified arithmetic section, executes instructions serially, with little concurrency.

Programs are written for the Central Processor in a conventional manner, specifying a sequence of arithmetic and control operations to be executed. Each instruction in a program is brought up in its turn from one of the instruction registers. These registers are filled from Central Memory in a manner sufficient to keep a reasonable flow of instructions available. A branch to another area of the program voids the old instructions in the registers and brings in new instructions. When a new instruction is brought up, a test is made on it to determine which of the 10 arithmetic units is needed, if it is busy, and if reservation conflict is possible. If the unit is free and no conflict is present, the entire instruction is given to the specified arithmetic unit for further action. Another instruction may then be brought up for issuance.

The original sequence of the program is established at the time each instruction is issued. Only those operations which depend on previous steps prevent the issuing of instructions, and then only if the steps are incomplete. The reservation control keeps a running account of the address, increment, and operand registers and of the arithmetic units in order to preserve the original sequence.

Nearly all Central Memory references for information or instructions are made on an implicit or secondary basis. Instructions are fetched from memory only if the instruction registers are nearly empty (or when ordered by a branch). Information is brought to or from the operand registers only when appropriate address registers are referenced during the course of a program. Such references are also accounted for in the reservation control.

All Central Processor references to Central Memory are made relative to the lower boundary address assigned by a Peripheral and Control Processor. A Central Processor program may therefore be relocated in Central Memory by modifying the boundaries only. Any attempt by the Central Processor to reference memory outside of its boundaries causes an immediate exit which can be readily examined by a Peripheral and Control Processor and displayed for the operator.

The Exchange Jump instruction described on page 3-9 starts a central program. This instruction starts a sequence of Central Memory references which exchanges 16 words in memory with the contents of the address, increment, and operand registers of the Central Processor. Also exchanged are the program address, the Central Memory and

Extended Core Storage boundaries, and choice of program exit. This instruction may be executed by any Peripheral and Control Processor and acts as an interrupt to an active central program as well as a start from an inactive state. The Exchange Jump is used by the operating system to switch between two central programs, leaving the first program in a usable state for later re-entry.

CENTRAL PROCESSOR PROGRAMMING

Central Processor program instructions are stored in Central Memory. A 60-bit memory location may hold 60 data bits, four 15-bit instructions, two 30-bit instructions or a combination of 15 or 30-bit instructions. Figure 3-1 shows all instruction combinations in a 60-bit word and the two instruction word formats.

The Central Processor reads 60-bit words from Central Memory and stores them in an instruction stack which is capable of holding up to eight 60-bit words.

Each instruction in turn is sent to a series of instruction registers for interpretation and testing and is then issued to one of 10 functional units for execution. The functional units obtain the instruction operands from and store results in the 24 operating registers. The reservation control records active operating registers and functional units to avoid conflicts and insure that the original instructions do not get out of order.

Functional Units

The 10 functional units in the 6600 system handles the requirements of the various instructions. The Multiply and Increment units are duplexed, and an instruction is sent to the second unit if the first is busy. The general function of each unit is listed in Table 3-2.

Instruction Formats

Groups of bits in an instruction are identified by the letters f, m, i, j, k, and K (Figure 3-1). All letters represent octal digits except K, which is an 18-bit constant.

TABLE 3-2. FUNCTIONAL UNITS

UNIT	GENERAL FUNCTION
Branch	Handles all jumps or branches from the program.
Boolean	Handles the basic logical operations of transfer, logical product, logical sum, and logical difference.
Shift	Handles operations basic to shifting. This includes left (circular) and right (end-off sign extension) shifting, and Normalize, Pack, and Unpack floating point operations. The unit also provides a mask generator.
Add	Performs floating point addition and subtraction on floating point numbers or their rounded representation.
Long add	Performs one's complement addition and subtraction of 60-bit fixed point numbers.
Multiply	Performs floating point multiplication on floating point numbers or their rounded representation.
Divide	Performs floating point division of floating point quantities or their rounded representation. Also sums the number of "1's" in a 60-bit word.
Increment	Performs one's complement addition and subtraction of 18-bit numbers.

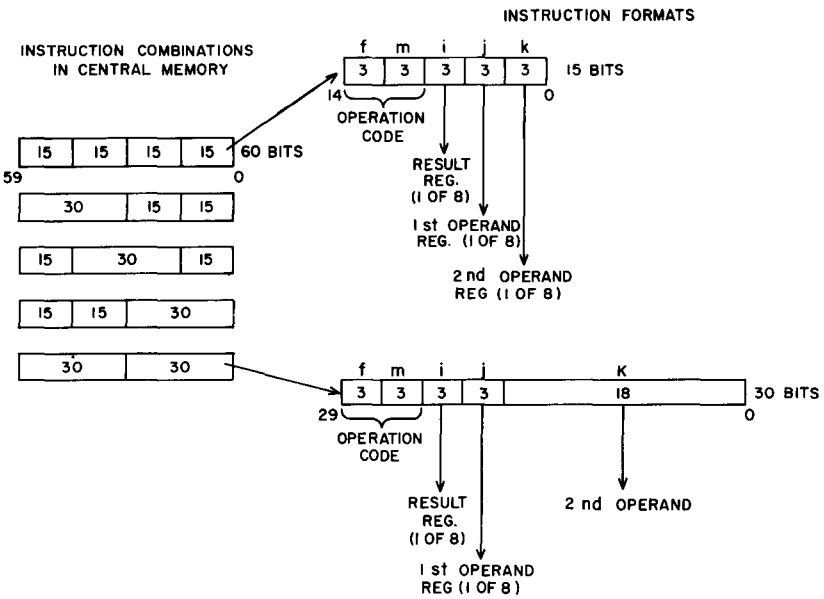


Figure 3-1. Central Processor Instruction Formats

The f and m digits are the operation code and identify the type of instruction. In a few instructions the i designator becomes a part of the operation code.

In most 15-bit instructions the i, j, and k digits each specify one of eight operating registers where operands are found and where the result of the operation is to be stored. In other 15-bit instructions, the j and k digits provide a 6-bit shift count.

In 30-bit instructions the i and j digits each specify one of eight operating registers where one operand is found and where the result is to be stored, and K is taken directly as an 18-bit second operand.

NOTE

In the 6600, it is permissible to pack the upper-order 15 bits (fmij portion) of a 30-bit instruction in the lower-order 15-bit portion of an instruction word. When this 30-bit instruction is executed, the lower-order 15-bits of K are taken from the upper-order 15 bits of the instruction word.

In the 6400 and 6500, any 30-bit instruction with its fmij portion packed in the lower-order 15 bits of an instruction word will be executed as a STOP instruction.

Operating Registers

In order to provide a compact symbolic language, the 24 operating registers are identified by letters and numbers:

A = address register (A0, A1 . . . A7)
B = increment register (B0, B1 . . . B7)
X = operand register (X0, X1 . . . X7)

The operand registers hold operands and results for servicing the functional units. Five registers (X1 - X5) hold read operands from Central Memory, and two registers (X6 - X7) hold results to be sent to Central Memory (Figure 3-2). Operands and results transfer between memory and these registers as a result of placing a quantity into a corresponding address register (A1 - A7).

Placing a quantity into an address register A1 - A5 produces an immediate memory reference to that address and reads the operand into the corresponding operand register X1 - X5. Similarly, placing a quantity into address register A6 or A7 stores the word in the corresponding X6 or X7 operand register in the new address.

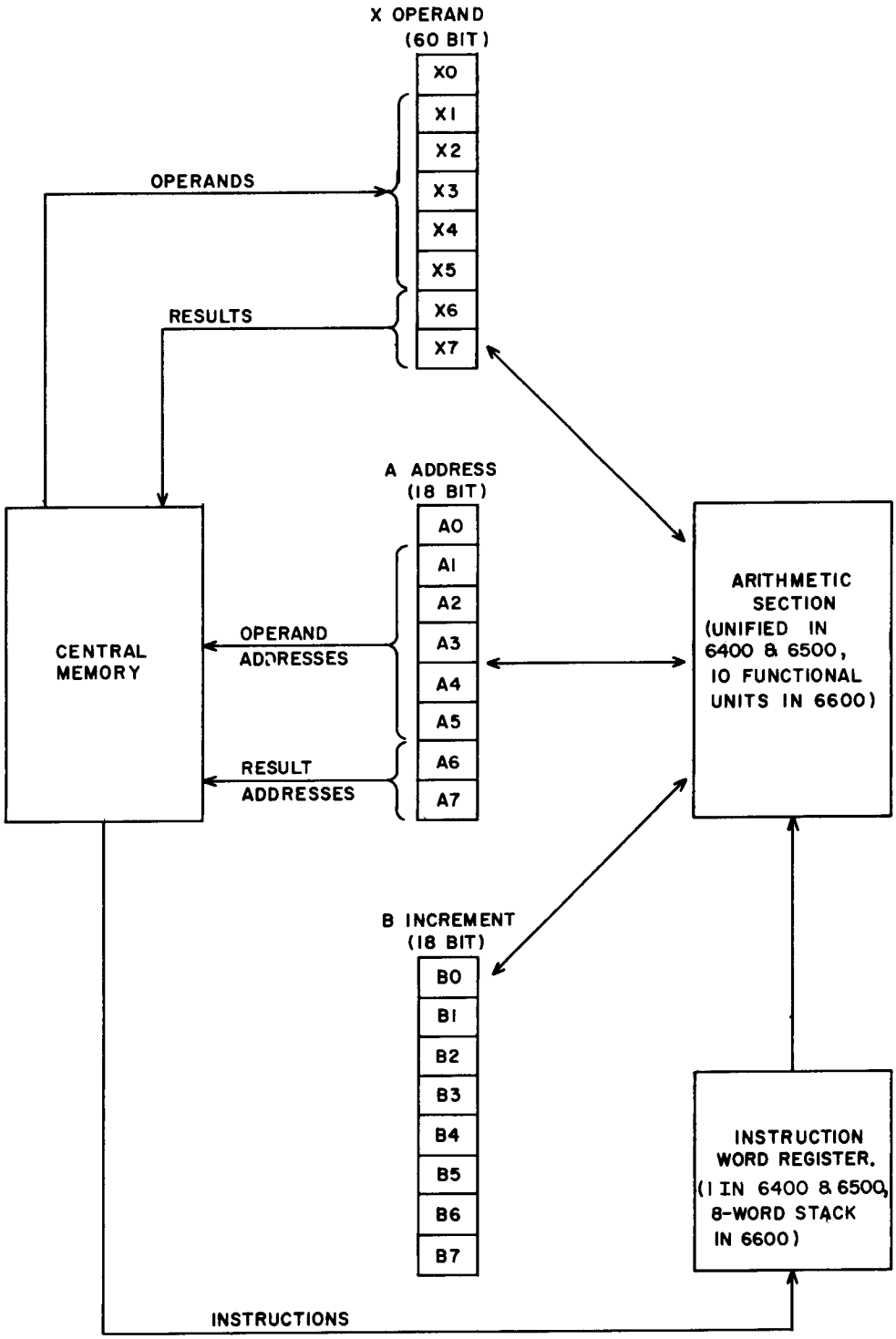


Figure 3-2. Central Processor Operating Registers

The increment instructions place a result in address register Ai (where "i" = 1-5) in three ways:

- By adding an 18-bit signed constant K to the contents of any A, B, or X register.
- By adding the content of any B register to any A, B, or X register.
- By subtracting the content of any B register from any A register or any other B register.

The A0 and X0 registers are independent and have no connection with Central Memory. They may be used for scratch pad or intermediate results. Note the special use of A0 and X0 when executing Extended Core Storage communication instructions.

The B registers have no connection with Central Memory. The B0 register is fixed to provide a constant zero (18-bit) which is useful for various tests against zero, providing an unconditional jump modifier, etc. In general, the B registers provide means for program indexing. For example, B4 may store the number of times a program loop has been traversed, thereby providing a terminal condition for a program exit.

An Exchange Jump instruction from a Peripheral and Control Processor enters initial values in the operating registers to start Central Processor operation. Subsequent address modification instructions executed in the increment functional units provide the addresses required to fetch and store data.

Program Address

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

1) P is advanced by one when all instructions in a 60-bit word have been extracted and sent to the instruction registers.

2) P is set to the address specified by a Go To ... (branch) instruction. If the instruction is a Return Jump, (P) + 1 is stored before the branch to allow a return to the sequence after the branch.

3) P is set to the address specified in the Exchange Jump package.

All branch instructions to a new program start the program with the instruction located in the highest order position of the 60-bit word.

Exchange Jump

A Peripheral and Control Processor Exchange Jump instruction starts or interrupts the Central Processor and provides Central Memory with the first address (which is the address in the Peripheral and Control Processor A register) of a 16-word package in Central Memory. The Exchange Jump package (Figure 3-3) provides the following information on a program to be executed:

1) Program address (P)
2) Reference Address for Central Memory (RA_{CM})
3) Field length of program for Central Memory (FL_{CM})
4) Reference Address for Extended Core Storage (RA_{ECS})
5) Field length of program for Extended Core Storage (FL_{ECS})
6) Program exit mode (EM)
7) Initial contents of the eight A registers
8) Initial contents of the eight X registers
9) Initial contents of B registers B1 - B7 (B0 is fixed at 0)

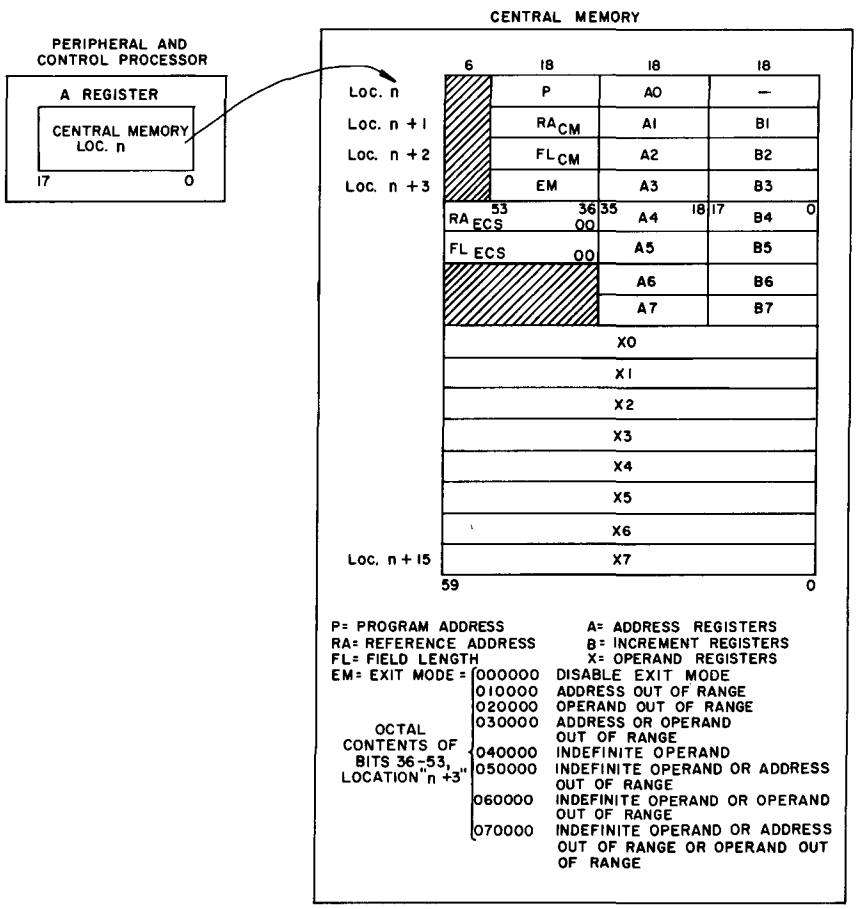


Figure 3-3. Exchange Jump Package

The Central Processor enters the information about a new program into the appropriate registers and stores the corresponding and current information from the interrupted program at the same 16 locations in Central Memory. Hence, the controlling information for two programs is exchanged. A later Exchange Jump may return an interrupted program to the Central Processor for completion. The normal relation of the A and X registers (described earlier) is not active during the Exchange Jump so that the new entries in A are not reflected into changes in X.

PROGRAMMING NOTE

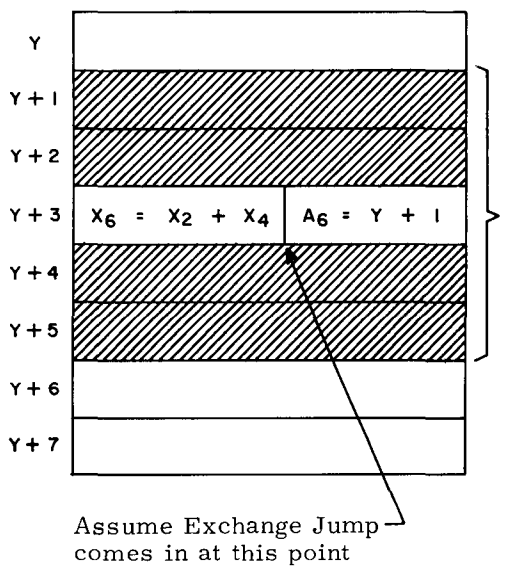
When an Exchange Jump interrupts the Central Processor, several steps occur to insure leaving the interrupted program in a usable state for re-entry:

1) Issue of instructions halts after issuing all instructions from the current instruction word in the instruction stack.
2) The Program Address register, P, is set to the address of the next instruction word to be executed.
3) The issued instructions are executed, and then
4) The parameters for the two programs are exchanged.

A subsequent Exchange Jump can then re-enter the interrupted program at the point it was interrupted, with no loss of program continuity.

To preserve the integrity of an "in-stack" loop (in the event of an Exchange Jump), it is illegal to modify the contents of any memory address which holds an executable instruction (or instruction word) contained within the loop.

EXAMPLE:



These instruction words in stack (from memory locations [Y + 1] through [Y + 5]) constitute a loop.

After executing the lower instruction at [Y + 3], the contents of memory location [Y + 1] differ from the contents of [Y + 1] in the stack. If the Exchange Jump comes in as indicated, subsequent reentry will call up the modified loop from memory, rather than the stack loop in its original un-modified form.

All Central Processor references to Central Memory for new instructions, or to fetch and store data, are made relative to the Reference Address. This allows easy relocation of a program in Central Memory. The Reference Address or beginning address and the Field Length define the Central Memory limits of the program. An Exit Selection allows the Central Processor to stop on a memory reference outside these limits.

The Program Address register P defines the location of a program step within the limits prescribed. Each reference to memory to fetch instructions is made to the address specified by P + RA. Hence program relocation is conveniently handled through a single change to RA.

A P = 0 condition specifies address zero and hence RA. This address is reserved for recording program exit (error) conditions and should not, therefore, be used to store data or instructions of a program.

Exit Mode

The Exit mode feature allows the programmer to select Exit or Stop conditions for the Central Processor. Exit selections are loaded into bits 36-53 of memory location "n+3" of the Exchange Jump package (Figure 3-3). When the Exchange Jump occurs to that package, the exit selections are stored in the Central Processor and the exit occurs as soon as the selected condition is sensed. The Exit conditions, as stored in bits 36-53 of address "n+3" in the Exchange Jump package, are shown below in octal format:

EM	=	000000	Disable Exit mode - no Exit selections made.
	=	010000	Address out of range - a) an attempt to reference either Central Memory or Extended Core Storage outside established limits, or b) the word count, [(Bj) + K], in an Extended Core Storage Communication instruction is negative, or c) an attempt to reference last 60-bit word (word 7) in relative address FL_{ECS}. (For details on action when an address is out of range, refer to the Increment and Extended Core Storage instruction descriptions.)
	=	020000	Operand out of range - floating point arithmetic unit received an infinite operand (see Range Definitions, page 3-17).

=	030000	Address or operand out of range
=	040000	Indefinite operand - floating point arithmetic unit (Add, Multiply, or Divide) attempted to use an indefinite operand (see Range Definitions, page 3-17).
=	050000	Indefinite operand or address out of range
=	060000	Indefinite operand or operand out of range
=	070000	Indefinite operand or operand or address out of range

Typically, the Reference Address (RA) for any program is left cleared to all zeros. When an error exit is taken, the Central Processor records at RA the exit condition (upper 2 octal digits only) and the Program Address at exit time (refer to the format below).

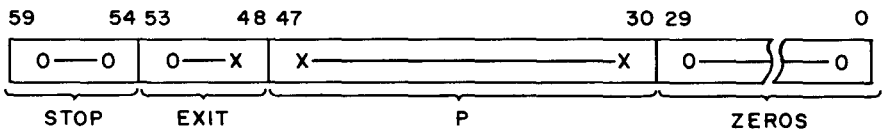
NOTE

The Exit condition(s) recorded at RA comprises all the Exit conditions detected since the last Exchange Jump, regardless of whether they were selected. Thus, combinations of error Exit conditions (03, 05, 06 or 07) can appear at RA:

a) When at least one Exit condition was selected and the selected condition plus another condition occurred since the last Exchange Jump, or

b) When more than one Exit condition was selected and each occurred in the same minor cycle.

The contents of RA are then read up, interpreted as a Stop instruction, and the Central Processor stops.



P = (P) + 1; AT TIME OF ERROR EXIT.

For error stops, (P) + 1 gives only an approximate location of the error since the Central Processor may have issued other instructions to the functional units (one of which may have been a branch) before the exit was sensed.

On an Address Out of Range, hardware action differs from that outlined above. In some cases, a stop occurs when an address is out of bounds even though an Exit mode stop is not selected for this condition. Table 3-3 summarizes hardware action for operations which may reference addresses that are out of bounds.

TABLE 3-3. EXIT MODE: ADDRESS OUT OF BOUNDS

	HARDWARE ACTION	
OPERATION	EXIT MODE SELECTED	EXIT MODE NOT SELECTED
RNI to an address that is out-of bounds (occurs when an instr. is located in absolute address (RA + FL) - 1).	1. Detect error condition 2. Clear P 3. Stop by reading (AAZ)* 4. Write EM and (P) + 1 into RA	1. Detect error condition 2. Stop by reading (AAZ) 3. Nothing stored in RA 4. (P) = out of range P or (P) + 1
Branch to an address that is out-of-bounds.	1. Detect error condition 2. Clear P 3. Stop by reading (AAZ) 4. Write EM and jump address + 1 in RA	1. Detect error condition 2. Stop by reading (AAZ) 3. Nothing stored in RA 4. (P) = out of range P or (P) + 1
Read Operand	1. Detect error condition 2. Clear P 3. Stop by reading (AAZ) 4. Write EM and (P) + 1 into RA 5. (X_i) = (AAZ)	1. Detect error condition 2. Read (AAZ) into X_i 3. Continue program
Write Operand	1. Detect error condition 2. Clear P 3. Stop by reading (AAZ) 4. Write EM and (P) + 1 into RA	1. Detect error condition 2. Read (AAZ), but (X_i) not stored; (X_i) and (A_i) unchanged. 3. Continue program

Action After Exit Mode or Normal Stop

Typically, a Peripheral and Control Processor periodically searches for an unchanging Central Processor Program Address register (any value) to determine if the Central Processor has stopped. Once it has been determined that the Central Processor has stopped, the examining Peripheral and Control Processor can transfer control to an error routine to determine the nature of the condition causing the Stop. Figure 3-4 illustrates sample steps for processing Central Processor stops (either Exit mode or normal).

* Absolute Address Zero

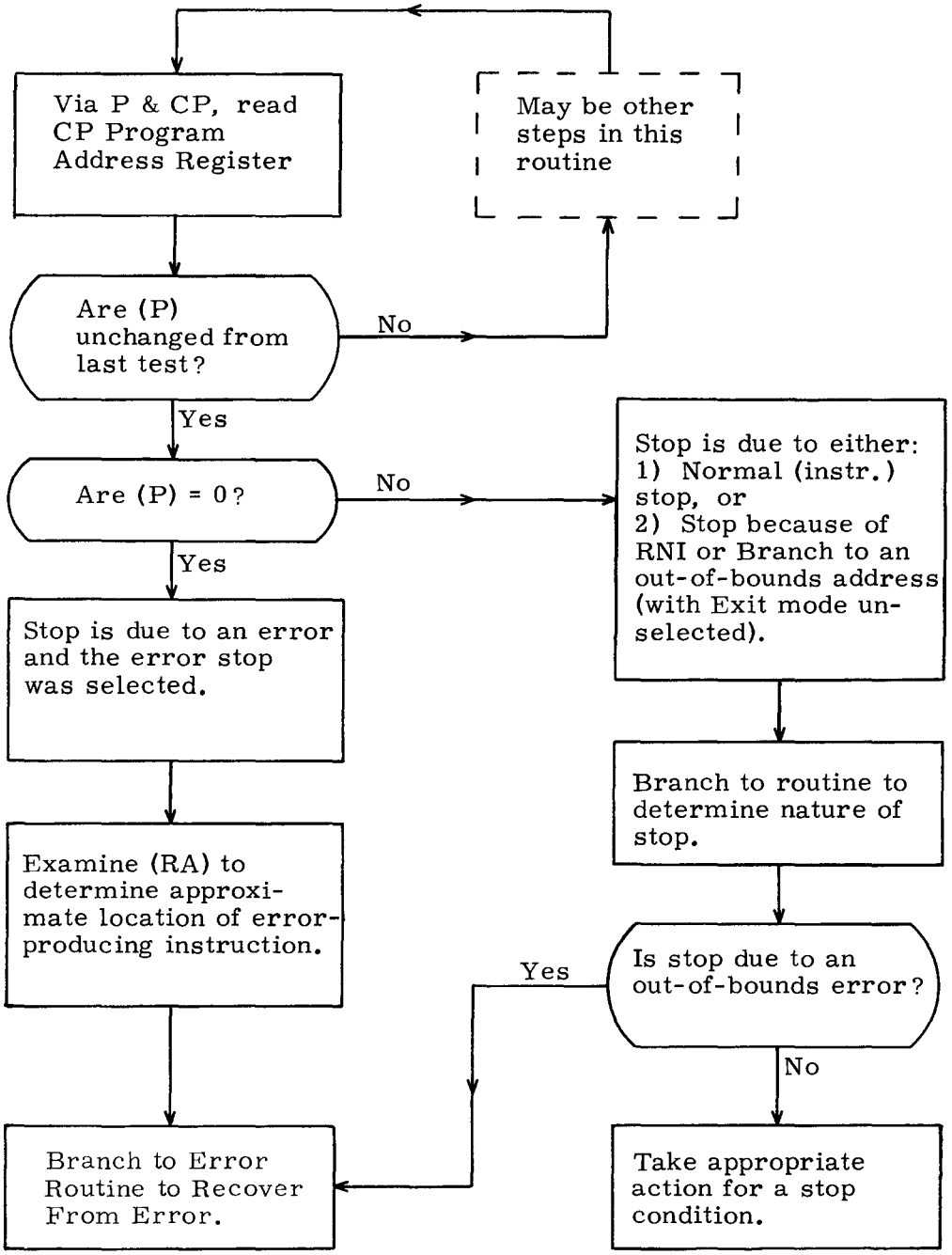


Figure 3-4. Detecting and Handling Central Processor Stops

Floating Point Arithmetic

Format

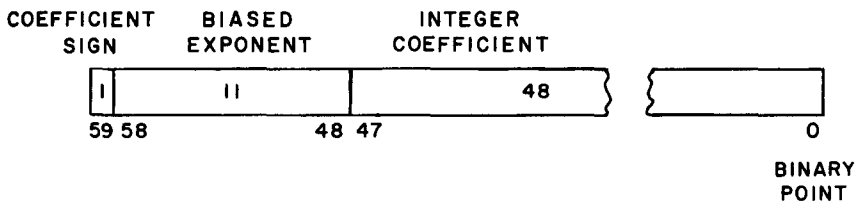
Floating point arithmetic takes advantage of the ability to express a number with the general expression kB^n, where:

k = coefficient

B = base number

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

The base number is constant (2) for binary-coded quantities and is not included in the general format. The 60-bit floating-point format is shown below. The binary point is considered to be to the right of the coefficient, thereby providing a 48-bit integer coefficient, the equivalent of about 14 decimal digits. The sign of the coefficient is carried in the highest order bit of the packed word. Negative numbers are represented in one's complement notation.



The 11-bit exponent carries a bias of 2^{10} (2000_8) when packed in the floating point word (biased exponent sometimes referred to as characteristic). The bias is removed when the word is unpacked for computation and restored when a word is packed into floating format. Table 3-4 lists (in decimal and octal notation) the complete range of permissible exponents and the octal form of the corresponding positive and negative floating point words.

Thus, a number with a true exponent of 342 would appear as 2342; a number with a true exponent of -160 would appear as 1617. Exponent arithmetic is done in one's complement notation. Floating point numbers can be compared for equality and threshold.

TABLE 3-4. RANGE OF PERMISSIBLE EXPONENTS

EXPONENT (n)			REPRESENTATION OF $k \times B^n$ (OCTAL)	
DECIMAL	OCTAL		POSITIVE COEFFICIENT	NEGATIVE COEFFICIENT
+1023	+1777	(infinite operand)	3777 X X	4000 X X
+1022	+1776		3776 X X	4001 X X
.	.		.	.
.	.		.	.
.	.		.	.
.	.		.	.
+1	+1		2001 X X	5776 X X
+0	+0		2000 X X	5777 X X
-0	-0	(indefinite operand)	1777 X X	6000 X X
-1	-1		1776 X X	6001 X X
.	.		.	.
.	.		.	.
.	.		.	.
.	.		.	.
-1023	-1777		0000 X X	7777 X X

Normalizing and Rounding

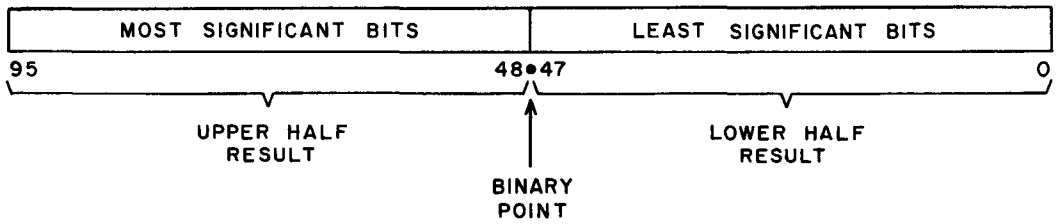
Normalizing a floating point quantity shifts the coefficient left until the most significant bit is in bit 47. Sign bits are entered in the low-order bits of the coefficient as it is normalized. Each shift decreases the exponent by one.

A round bit is added (optionally) to the coefficient during an arithmetic process and has the effect of increasing the absolute value of the operand or result by one-half the value of the least significant bit. Normalizing and rounding are not automatic during pack or unpack operations so that operands and results may not be normalized.

Single and Double Precision

The floating point arithmetic instructions generate double-precision results. Use of unrounded operations allows separate recovery of upper and lower half results with proper exponents; only upper half results can be obtained with rounded operations.

Double length registers appear as follows:



Range Definitions

A result with an exponent so large that it exceeds the upper limit of octal 3777 (overflow case) is treated as an infinite quantity. A coefficient of all zeros and an exponent of octal 3777 or 4000 is packed for this case. An optional exit is provided when an attempt is made to use an infinite operand in the floating arithmetic units since its use may propagate an indefinite result as shown in Table 3-5. No error exit occurs when an infinite or indefinite result is generated in a functional unit.

TABLE 3-5. INDEFINITE FORMS

$\infty - \infty$	= INDEFINITE	$\infty \div N = \infty$
$\infty + \infty$	= INDEFINITE	$\infty + N = \infty$
$\infty \bullet 0$	= INDEFINITE	$\infty - N = \infty$
$0 \div 0$	= INDEFINITE	$N \div 0 = \infty$
INDEFINITE $+, -, \div, \bullet$ (X)	= INDEFINITE	$0 \div \infty = 0$
$\infty + \infty$	$= \infty$	$0 \bullet 0 = 0$
$\infty \bullet \infty$	$= \infty$	$0 \div N = 0$
$\infty \div 0$	$= \infty$	$N \div \infty = 0$

WHERE: ∞ = INFINITY, N = INTEGER,
X = ∞, N OR 0.

A result the exponent of which is less than the lower limit of octal 0000 (underflow case) is treated as a zero quantity. This quantity is packed with a zero exponent and zero coefficient. No exit is provided for underflow. A result with an exponent of octal 0000 and a coefficient which is not zero is a non-zero quantity and is packed with a zero exponent and the non-zero coefficient.

Use of either infinity or zero as operands may produce an indefinite result. An exponent of octal 1777 and a zero coefficient are packed in this case, and an optional exit provided. Note that zero, infinite, and indefinite results are generated or regenerated in floating arithmetic operations only. The branch instructions test for infinite or indefinite quantities.

In all floating arithmetic operations, an attempt to normalize an indefinite quantity returns the original quantity, e. g., if the number 17770237... were to be normalized, the result would be the same as the original number. Note that Exit mode does not occur on detecting an indefinite quantity in the Shift Unit.

Exit mode tests for infinite and indefinite operands are made only in the Floating Add, Multiply, and Divide Units. The 12 most significant bits of each operand are tested for these special forms.

In the Multiply and Divide Units (but not in the Floating Add Unit) there is a special test for zero operands as determined by the 12 most significant bits.

Thus the special operand forms (in octal) are:

3777X...X	(+ ∞)	infinite operands
4000X...X	(- ∞)	
1777X...X	(+IND)	indefinite operands
6000X...X	(-IND)	
0000X...X	(+0)	zero operands for Multiply and Divide units only
7777X...X	(-0)	

Whenever infinite, indefinite, or zero results are generated in accordance with the rules given in Table 3-5 and Appendix C, only the following octal words can occur as results:

37770...0	= + ∞	(result)
40000...0	= - ∞	(result)
17770...0	= +IND	(result)
00000...0	= +0	(result)

Note that in these cases the 48 least significant bits of the result are zeros. Indefinite and zero results generated in accordance with Table 3-5 and Appendix C are always positive, but the sign of infinite results is determined by the usual algebraic sign convention. For example:

(+0)/(-0)	= +IND	= 17770...0
(+N)*(-0)	= +0	= 00000...0
(-∞)/(-0)	= +∞	= 37770...0
(+∞)/(-0)	= -∞	= 40000...0

There is no special treatment of zero operands in the Floating Add unit. Zero coefficients and the forms 0000X...X and 7777X...X are not specially detected, and unstandardized zero results can be produced. (See description of 30 instruction, page 3-37.)

Overflow and Underflow

Exponents lying outside the range -1777_8 to $+1777_8$ cannot be generated during execution of a floating point arithmetic instruction or during execution of a Normalize instruction. An attempt to generate an exponent greater than $+1777_8$ yields an infinite result (overflow case). An attempt to generate an exponent less than -1777_8 yields a zero result (underflow case). All cases of overflow and underflow are listed in Table 3-6.

Converting Integers to Floating Format

Conversion of integers to floating point format makes use of the Shift Unit and the zero constant in increment register B0. The B0 quantity provides for generation of exponent bias in this case. For example, the instructions:

- Sum of Bj and Bk to Xi (where i = 2, j = 3, k = 4)
- Pack Xi from Xk and Bj (where i = 2, j = 0, k = 2)

form an 18-bit signed integer in operand register X2 as a result of the addition of the contents of increment registers B3 and B4. The integer coefficient with its sign, plus the octal 2000 exponent is then packed into the floating format shown earlier. The coefficient is not normalized; normalizing may be accomplished with a Normalize instruction.

TABLE 3-6. OVERFLOW AND UNDERFLOW CONDITIONS

OVERFLOW		
INSTRUCTIONS	OVERFLOW CONDITION	RESULT
Normalize (24, 25)	None	---
Upper Sum (30, 31, 34, 35)	None (see Note 1)	---
Lower Sum (32, 33)	None	---
Upper Product (40, 41)	$*n_1 + n_2 + 60_8 \geq 2000_8$	X_i = 3777 0....0_8 or 4000 0....0_8
Lower Product (42)	$n_1 + n_2 \geq 2000_8$	(True Sign)
Quotient (44, 45)	$n_1 - n_2 - 57_8 \geq 2000_8$	
UNDERFLOW		
INSTRUCTIONS	UNDERFLOW CONDITION	RESULT
Normalize (24 only)	Initial coefficient = ±0	X_i = 0000 0....0_8, $(B_j) = 60_8$
Normalize (24, 25)	Final Exponent $\leq -2000_8$	X_i = 0000 0....0_8, (B_j) are correct. (See Note 2.)
Upper Sum (30, 31, 34, 35)	None	---
Lower Sum (32, 33)	Final Exponent $\leq -2000_8$	X_i = 0000 0....0_8
Upper Product (40, 41)	$n_1 + n_2 + 57_8 \leq -2000_8$	X_i = 0000 0....0_8
Lower Product (42)	$n_1 + n_2 - 1 \leq -2000_8$	
Quotient (44, 45)	$n_1 - n_2 - 60_8 \leq -2000_8$	

*n_1 and n_2 are the initial exponents.

Note 1. Overflow of Upper Sum: Overflow cannot occur unless one operand is infinite. In this case the result is as indicated. If a one-place Right Shift occurs when the larger operand exponent is equal to $+1776_8$, a correct result with exponent $+1777_8$ is generated.

Note 2. Underflow of Exponent During Normalization: The final (B_j) are the same as if underflow had not occurred. In particular, if the initial coefficient is zero, (B_j) are equal to 60_8.

Fixed Point Arithmetic

Fixed point addition and subtraction of 60-bit numbers are handled in the Long Add Unit (6600). Negative numbers are represented in one's complement notation, and overflows are ignored. The sign bit is in the high-order bit position (bit 59) and the binary point is at the right of the low-order bit position (bit 0).

The Increment Units provide an 18-bit fixed point add and subtract facility. Negative numbers are represented in one's complement notation and overflows are ignored. The sign bit is in the high-order bit position (bit 17), and the binary point is at the right of the low-order bit position (bit 0). The Increment Units allow program indexing through the full range of Central Memory addresses.

Fixed point integer addition and subtraction are possible in the Floating Add Unit providing the exponents of both operands are zero and no overflow occurs. The unit performs the one's complement addition (or subtraction) in the upper half of a 98-bit accumulator. If overflow occurs, the unit shifts the result one place right and adds one to the exponent, thereby producing a floating point quantity. Thus, care must be used in performing fixed point arithmetic in the Floating Add Unit.

Fixed point integer multiplication is handled in the multiply functional units as a subset operation of the unrounded Floating Multiply (40, 42) instructions. The multiply is double precision (96 bits) and allows separate recovery of upper and lower products. The multiply requires that both of the integer operands be converted (by program) to floating format to provide biased exponents. This insures that results are not sensed as underflow conditions. The bias is removed when the result is unpacked.

An integer divide takes several steps and makes use of the Divide and Shift Units. For example, an integer quotient X1 = X2/X3 is produced by the following steps:

	Instructions	Remarks
1)	Pack X2 from X2 and B0	Pack X2
2)	Pack X3 from X3 and B0	Pack X3
3)	Normalize X3 in X0 and B0	Normalize X3 (divisor)
4)	Floating quotient of X2 and X0 to X1	Divide
5)	Unpack X1 to X1 and B7	Unpack quotient
6)	Shift X1 nominally left B7 places	Shift to integer position

The divide requires that:

1) both integer (2^{47} maximum) operands be in floating format

and 2) the divisor be shifted 48 places left

or 3) the quotient be shifted 48 places right

or 4) any combination of n left-shifts of the divisor and 48-n right shifts of the quotient be accomplished.

The Normalize X3 instruction shifts the divisor n places left ($n \geq 0$), providing a divisor exponent of -n. The quotient exponent then is: $0 - (-n) - 48 = n - 48 \leq 0$.

After unpacking and shifting nominally left, the negative (or zero) value in B7 shifts the quotient 48 - n places right, producing an integer quotient in X1. A remainder may be obtained by an integer multiply of X1 and X3 and subtracting the result from X2.

Description of Central Processor Instructions

This section describes the Central Processor instructions. Instruction grouping follows a somewhat pedagogical approach (i.e., simple to complex) and does not necessarily relate instructions to the functional units (6600 system) which execute them. Central Processor instructions as related to functional units are tabulated in Appendix B, Instruction Execution Times.

TABLE 3-7. CENTRAL PROCESSOR INSTRUCTION DESIGNATORS

DESIGNATOR	USE
A	Specifies one of eight 18-bit address registers.
B	Specifies one of eight 18-bit index registers; B0 is fixed and equal to zero.
fm	A 6-bit instruction code.
i	A 3-bit code specifying one of eight designated registers (e.g., Ai).
j	A 3-bit code specifying one of eight designated registers (e.g., Bj).
jk	A 6-bit constant, indicating the number of shifts to be taken.
k	A 3-bit code specifying one of eight designated registers (e.g., Bk).
K	An 18-bit constant, used as an operand or as a branch destination (address).
X	Specifies one of eight 60-bit operand registers.

Preceding the description of each instruction is the octal code, mnemonic code and address field, the instruction name and length. Mnemonic codes and address field mnemonics are from ASCENT, the Central Processor Assembly language.

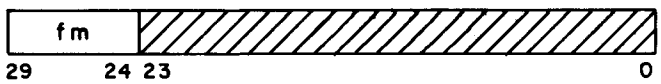
EXAMPLE:

12	BXi	Xj+Xk	Logical Sum of Xj and Xk to Xi	(15 Bits)
Octal Code	Mnemonic Code	Address Field	Instruction Name	Instruction Length

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

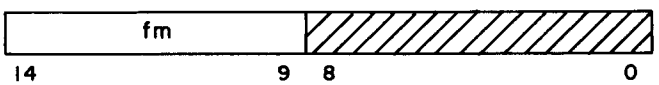
Program Stop and No Operation

00 PS Program Stop (30 Bits)



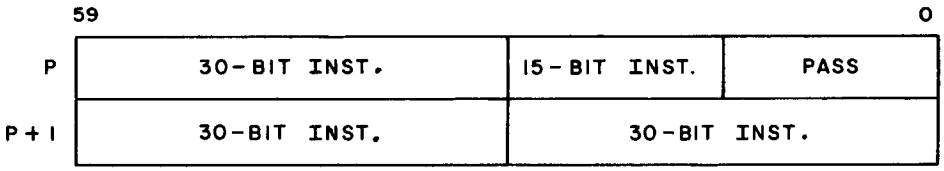
This instruction stops the Central Processor at the current step in the program. An exchange Jump is necessary to restart the Central Processor.

46 NO No operation (Pass) (15 Bits)



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

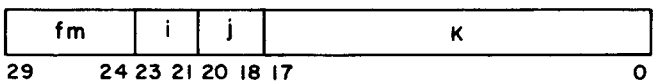
EXAMPLE:



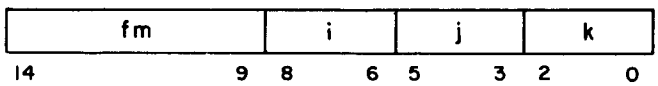
In this example, a Pass instruction is used to pad the remainder of the word at P. Since the next instruction is 30 bits, it cannot fit in P and must be placed in P + 1.

Increment

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



53	***SAi***	***Xj + Bk***	***Set Ai to Xj + Bk***	***(15 Bits)***
54	***SAi***	***Aj + Bk***	***Set Ai to Aj + Bk***	***(15 Bits)***
55	***SAi***	***Aj − Bk***	***Set Ai to Aj − Bk***	***(15 Bits)***
56	***SAi***	***Bj + Bk***	***Set Ai to Bj + Bk***	***(15 Bits)***
57	***SAi***	***Bj − Bk***	***Set Ai to Bj − Bk***	***(15 Bits)***



These instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in address register i. Overflow, in itself, is ignored, but an address range fault may result from overflow in this set of instructions.

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

Note that an immediate memory reference is performed to the address specified by the final content of address registers A1 - A7. The operand read from memory address specified by A1 - A5 is sent to the corresponding operand register X1 - X5. When A6 or A7 is referenced, the operand from the corresponding X6 or X7 operand register is stored at the address specified by A6 or A7.

NOTE

If, in this category of instructions, the result placed in address register Ai is an address out of range, the following occurs: (Note that this action is independent of an Exit selection on Address Out of Range.)

If i = 1-5: Operand register Xi is loaded with the contents of absolute address zero and the contents of memory location (Ai) are unchanged.

If i = 6 or 7: Operand register Xi retains its original contents and the contents of memory location (Ai) are unchanged.

EXAMPLE:

50 SAi Aj + K i = 4

SA_4 A_6 + K j = 6

$SA_4 = 432100_8 + 234567_8$

$SA_4 = 666667_8$

Initial Quantities:

$K = 234567_8$

$A_4 = 321110_8$

$A_6 = 432100_8$

$X_4 = 00.....00_8$

Storage location $666667 = 7...75342104600_8$

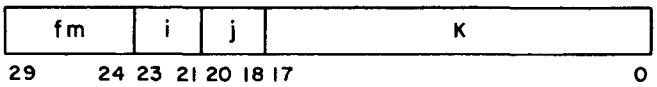
Final Quantities:

$A_4 = 666667_8$

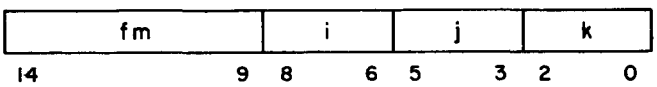
$A_6 = 432100_8$

$X_4 = 7...75342104600_8$

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



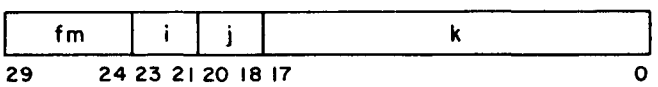
63	*SBi*	*Xj + Bk*	*Set Bi to Xj + Bk*	*(15 Bits)*
64	*SBi*	*Aj + Bk*	*Set Bi to Aj + Bk*	*(15 Bits)*
65	*SBi*	*Aj − Bk*	*Set Bi to Aj − Bk*	*(15 Bits)*
66	*SBi*	*Bj + Bk*	*Set Bi to Bj + Bk*	*(15 Bits)*
67	*SBi*	*Bj − Bk*	*Set Bi to Bj − Bk*	*(15 Bits)*



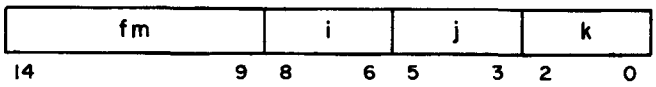
These instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in increment register Bi. An overflow condition is ignored.

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

70	*SXi*	*Aj + K*	*Set Xi to Aj + K*	*(30 Bits)*
71	*SXi*	*Bj + K*	*Set Xi to Bj + K*	*(30 Bits)*
72	*SXi*	*Xj + K*	*Set Xi to Xj + K*	*(30 Bits)*



73	*SXi*	*Xj + Bk*	*Set Xi to Xj + Bk*	*(15 Bits)*
74	*SXi*	*Aj + Bk*	*Set Xi to Aj + Bk*	*(15 Bits)*
75	*SXi*	*Aj − Bk*	*Set Xi to Aj − Bk*	*(15 Bits)*
76	*SXi*	*Bj + Bk*	*Set Xi to Bj + Bk*	*(15 Bits)*
77	*SXi*	*Bj − Bk*	*Set Xi to Bj − Bk*	*(15 Bits)*



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

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

EXAMPLE:

73 SXi Xj + Bk i = 2

SX_2 $X_3 + B_1$ j = 3, K = 1

$SX_2 = 0...0652224310_8 + 511245_8$

$SX_2 = 7...7777735555_8$

Initial Quantities:

$X_2 = 0...0745321402_8$

$X_3 = 0...0652224310_8$

$B_1 = 511245_8$

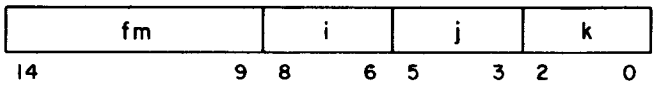
Final Quantities:

$X_2 = 7...7777735555_8$

$X_3 = 0...0652224310_8$

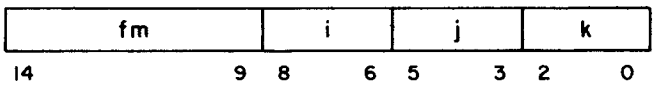
$B_1 = 511245_8$

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



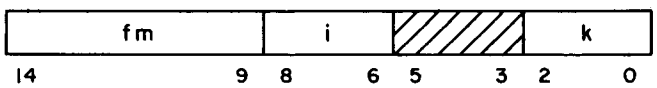
This instruction forms a 60-bit one's complement sum of the quantities from operand registers Xj and Xk and stores the result in operand register Xi. An overflow condition is ignored.

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



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

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



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

EXAMPLE:

47 CXi Xk i = 4

CX$_4$ X$_1$ k = 1

CX$_4$ = 11$_8$

Initial Quantities:

X$_1$ = 0 ... 0543321$_8$

X$_4$ = 23420... 0005547$_8$

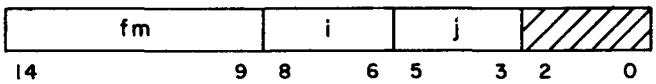
Final Quantities:

X$_1$ = 0 ... 0543321$_8$

X$_4$ = 0 ... 0000011$_8$

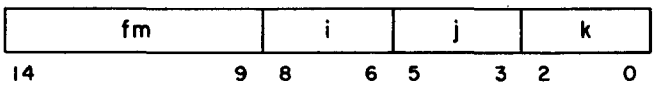
Logical

10 BXi Xj Transmit Xj to Xi (15 Bits)



This instruction transfers a 60-bit word from operand register Xj to operand register Xi.

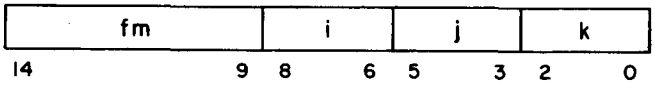
11 BXi Xj * Xk Logical Product of Xj and Xk to Xi (15 Bits)



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

Xj = 0101
Xk = 1100
Xi = 0100

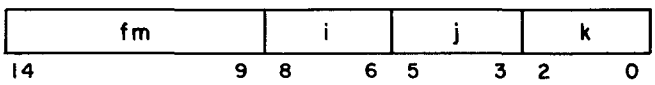
12 **BXi** ***Xj + Xk*** ***Logical sum of Xj and Xk to Xi*** ***(15 Bits)***



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

Xj = 0101
Xk = 1100
Xi = 1101

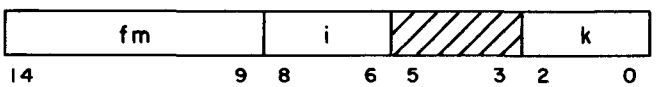
13 **BXi** ***Xj − Xk*** ***Logical difference of Xj and Xk to Xi*** ***(15 Bits)***



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

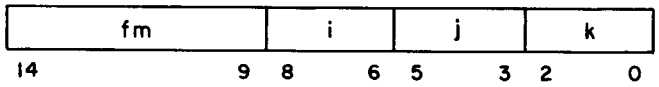
Xj = 0101
Xk = 1100
Xi = 1001

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



This instruction extracts the 60-bit word from operand register Xk, complements it, and transmits this complemented quantity to operand register Xi.

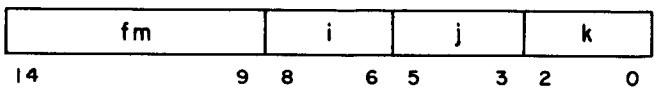
15 BXi −Xk * Xj Logical product of Xj and complement of Xk to Xi (15 Bits)



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

```
          Xj = 0101
Complemented Xk = 0011
          Xj = 0001
```

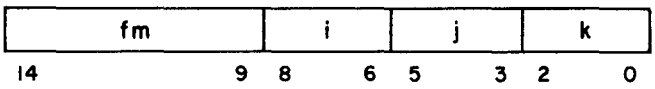
16 BXi −Xk + Xj Logical sum of Xj and complement of Xk to Xi (15 Bits)



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

```
          Xj = 0101
Complemented Xk = 0011
          Xi = 0111
```

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

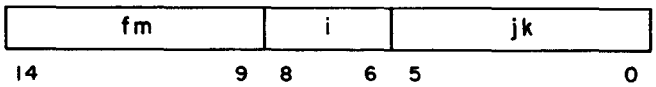


This instruction forms the logical difference (exclusive OR) of the quantity from operand register Xj and the complement of the 60-bit word from operand register Xk, and places the result in operand register Xi. Thus, bits of Xi are set to "1" if the corresponding bits of register Xj and the complement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

Xj = 0101
Complemented Xk = 0011
Xi = 0110

Shift

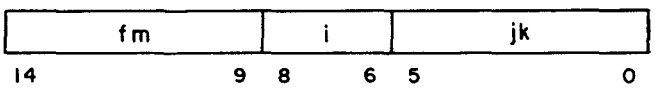
20 **LXi** ***jk*** ***Left shift Xi, jk places*** ***(15 Bits)***



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

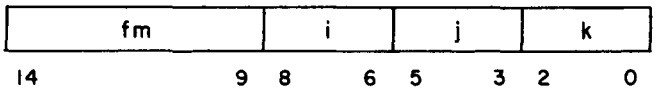
The 6-bit shift count jk allows a complete circular shift of register Xi.

21 **AXi** ***jk*** ***Arithmetic right shift Xi, jk places*** ***(15 Bits)***



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

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

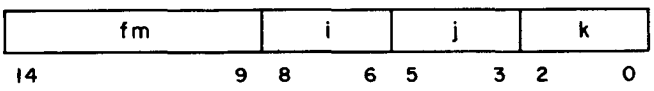


This instruction shifts the 60-bit quantity from operand register Xk the number of places specified by the quantity in increment register Bj and places the result in operand register Xi.

1) If Bj is positive (i.e., bit 17 of Bj = 0), the quantity from Xk is shifted left-circular. (The low order six bits of Bj specify the shift count.)

2) If Bj is negative (i.e., bit 17 of Bj = 1), the quantity from Xk is shifted right (end off with sign extention). (The one's complement of the low order eleven bits of Bj specify the shift count.) If any of bits 2^6-2^{10}, after complementing, are "1's", the shift is not performed and the result register Xi is cleared to all zeros.

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



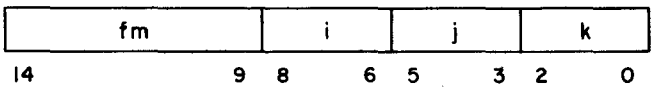
This instruction shifts the 60-bit quantity from operand register Xk the number of places specified by the quantity in increment register Bj and places the result in operand register Xi.

1) If Bj is positive (i.e., bit 17 of Bj = 0), the quantity from register Xk is

shifted right (end-off with sign extension). (The low order eleven bits of Bj specify the shift count.) If any of bits 2^6-2^{10} are "1's", the shift is not performed and the result register Xi is cleared to all zeros.

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

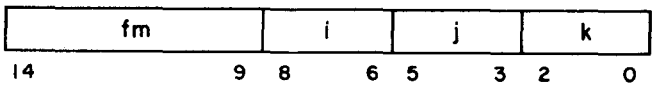
24 NXi Bj Xk Normalize Xk in Xi and Bj (15 Bits)



This instruction normalizes the floating point quantity from operand register Xk and places it in operand register Xi. The number of left shifts necessary to normalize the quantity is entered in increment register Bj. A Normalize operation may cause underflow which will clear Xi to all zeros regardless of the original sign of Xk. Normalizing either a plus or minus zero coefficient sets the shift count (Bj) to 48_{10} and clears Xi to all zeros.

If Xk contains an infinite quantity (3777X...X or 4000X...X) or an indefinite quantity (1777X...X or 6000X...X), no shift takes place. The contents of Xk are copied into Xi and Bj is set equal to zero. Optional error exits do not occur.

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

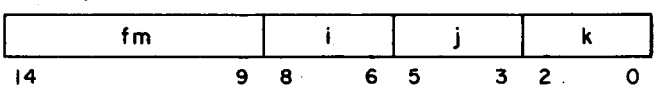


This instruction performs the same operation as instruction 24 except that the quantity

from operand register Xk is rounded before it is normalized. Rounding is accomplished by placing a "1" round bit immediately to the right of the least significant coefficient bit. Normalizing a zero coefficient places the round bit in bit 47 and reduces the exponent by 48. Note that the same rules apply for underflow.

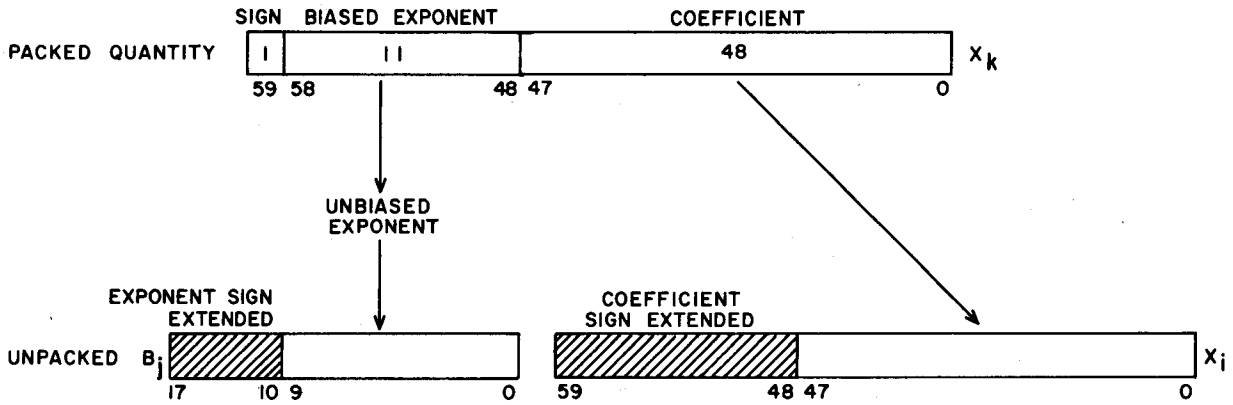
If Xk contains an infinite quantity (3777X...X or 4000X...X) or an indefinite quantity (1777X...X or 6000X...X), no shift takes place. The contents of Xk are copied into Xi and Bj is set equal to zero. Optional error exits do not occur.

26 UXi Bj Xk Unpack Xk to Xi and Bj (15 Bits)

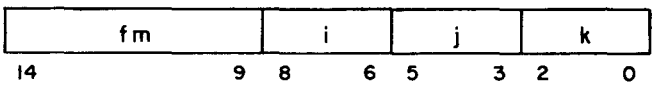


This instruction unpacks the floating point quantity from operand register Xk and sends the 48-bit coefficient to operand register Xi and the 11-bit exponent to increment register Bj. The exponent bias is removed during Unpack so that the quantity in Bj is the true one's complement representation of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

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



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

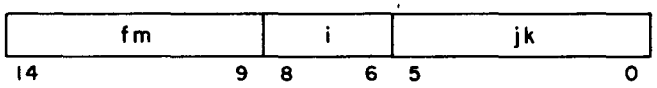


This instruction packs a floating point number in operand register Xi. The coefficient of the number is obtained from operand register Xk and the exponent from increment register Bj. Bias is added to the exponent during the Pack operation. The instruction does not normalize the coefficient.

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

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

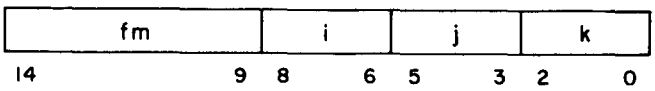
43 MXi jk Form mask in Xi, jk bits (15 Bits)



This instruction forms a mask in operand register Xi. The 6-bit quantity jk defines the number of "1's" in the mask as counted from the highest order bit in Xi.

The contents of operand register i = 0 when jk = 0.

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

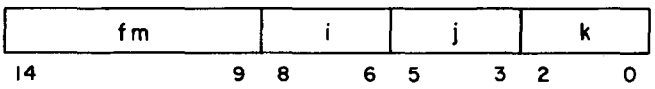


This instruction forms the sum of the floating point quantities from operand registers Xj and Xk and packs the result in operand register Xi. The packed result is the upper half of a double precision sum.

At the start both arguments are unpacked, and the coefficient of the argument with the smaller exponent is entered into the upper half of a 98-bit accumulator. The coefficient is shifted right by the difference of the exponents. The other coefficient is then added into the upper half of the accumulator. If overflow occurs, the sum is right-shifted one place and the exponent of the result increased by one. The upper half of the accumulator holds the coefficient of the sum, which is not necessarily in normalized form. The exponent and upper coefficient are then repacked in operand register Xi.

If both exponents are zero* and no overflow occurs, the instruction effects an ordinary integer addition. For treatment of special operands and/or indefinite forms, refer to Table 3-5 and Appendix C.

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

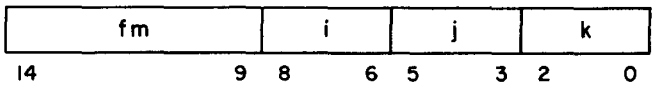


This instruction forms the difference of the floating point quantities from operand registers Xj and Xk and packs the result in operand register Xi. Alignment and overflow operations are similar to the Floating Sum (30) instruction, and the difference is not necessarily normalized. The packed result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

An ordinary integer subtraction is performed when the exponents are zero. For treatment of special operands and/or indefinite forms, refer to Table 3-5 and Appendix C.

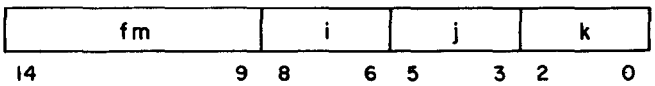
*A zero exponent is 2000_8.

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



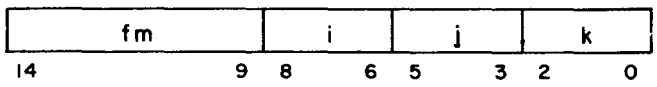
This instruction forms the sum of two floating point numbers as in the Floating Sum (30) instruction, but packs the lower half of the double precision sum with an exponent 48 less than the upper sum. For treatment of special operands and/or indefinite forms, refer to Table 3-5 and Appendix C.

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



This instruction forms the difference of two floating point numbers as in the Floating Difference (31) instruction, but packs the lower half of the double precision difference with an exponent of 48 less than the upper sum. For treatment of special operands and/or indefinite forms, refer to Table 3-5 and Appendix C.

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



This instruction forms the round sum of the floating point quantities from operand registers Xj and Xk and packs the upper sum of the double precision result in operand register Xi. The sum is formed in the same manner as the Floating Sum instruction but the

operands are rounded before the addition, as shown below, to produce a round sum.

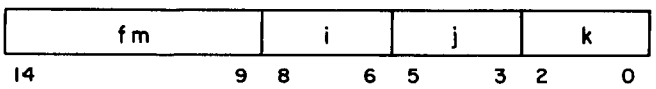
1) A round bit is attached at the right end of both operands if:

   a) both operands are normalized, or
   b) the operands have unlike signs.

2) A round bit is attached at the right end of the operand with the larger exponent for all other cases.

For treatment of special operands and/or indefinite forms, refer to Table 3-5 and Appendix C.

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



This instruction forms the round difference of the floating point quantities from operand registers Xj and Xk and packs the upper difference of the double precision result in operand register Xi. The difference is formed in the same manner as the Floating Difference (31) instruction but the operands are rounded before the subtraction, as shown below, to produce a round difference.

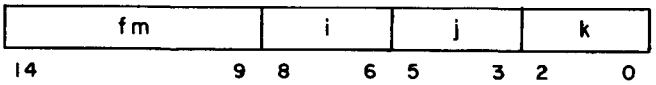
1) A round bit is attached at the right end of both operands if:

   a) both operands are normalized, or
   b) the operands have like signs.

2) A round bit is attached at the right end of the operand with the larger exponent for all other cases.

For treatment of special operands and/or indefinite forms, refer to Table 3-5 and Appendix C.

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



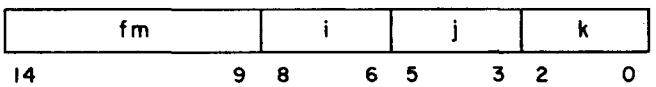
This instruction multiplies two floating point quantities obtained from operand registers Xj (multiplier) and Xk (multiplicand) and packs the upper product result in operand register Xi.

The two 48-bit coefficients are multiplied together to form a 96-bit product. The upper 48 bits of the product (bits 48-95) are then packed together with the resulting exponent. Note that when using unnormalized quantities, the entire result could lie in the lower-order 48 bits of the product; hence, this result would be lost when packing occurs.

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

The result is unnormalized when either or both operands are unnormalized; the exponent in this case is the sum of the exponents plus 48. For treatment of special operands and/or indefinite forms, refer to Table 3-5 and Appendix C.

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



This instruction multiplies the floating point number from operand register Xk (multiplicand), by the floating point number from operand register Xj. The upper product result is packed in operand register Xi. (No lower product available.) The multiply operation is identical to that of instruction 40 with the following exception:

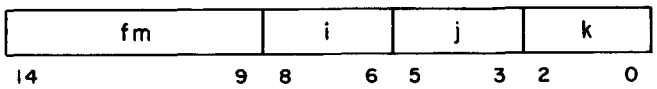
Before the left shift of the final product and during the merge operation to form the final product, a "1" bit is added to bit 2^{46}. The following rounded result is the net effect of this action:

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

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

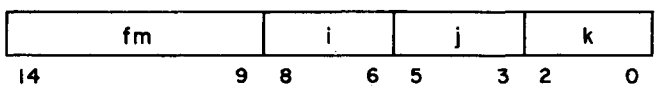
The result is unnormalized when either or both operands are unnormalized; the exponent in this case is the sum of the exponents plus 48. For treatment of special operands and/or indefinite forms, refer to Table 3-5 and Appendix C.

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



This instruction multiplies two floating point quantities obtained from operand registers Xj and Xk and packs the lower product in operand register Xi. The two 48-bit coefficients are multiplied together to form a 96-bit product. The lower-order 48 bits of this product (bits 47-00) are then packed together with the resulting exponent. The result is not necessarily a normalized quantity. The exponent of this result is 48 less than the exponent resulting from a 40 instruction using the same operands. For treatment of special operands and/or indefinite forms, refer to Table 3-5 and Appendix C.

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



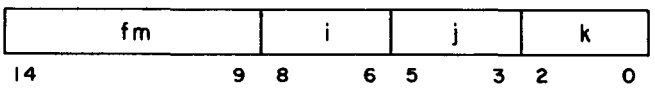
This instruction divides two normalized floating point quantities obtained from operand registers Xj (dividend) and Xk (divisor) and packs the quotient in operand register Xi.

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

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

The result is a normalized quantity when both the dividend and the divisor are normalized. Note that the machine makes no note of divide faults, i.e., when the absolute value of the coefficient of the dividend $\geq$ two times the absolute value of the coefficient of the divisor. To avoid possible incorrect results from using unnormalized operands, the operands in this instruction should be normalized. For treatment of special operands and/or indefinite forms, refer to Table 3-5 and Appendix C.

45 RXi Xj / Xk Round floating divide Xj by Xk to Xi (15 Bits)



This instruction divides the floating quantity from operand register j (dividend) by the floating point quantity from operand register Xk (divisor) and packs the round quotient in operand register Xi. Rounding is accomplished by adding one-third during the division process. In effect, the quantity "$2525....2525_8$" resides immediately to the right of the dividend binary point prior to starting the divide operation. On the first iteration, a "1" is added to the least significant bit of the dividend. After each iteration (subtraction of divisor from partial dividend) a two-place left shift occurs and a "1" is again added to the least significant bit of the partial dividend. Thus, successive iterations gradually bring in the one-third round "quantity" ($25....25_8$).

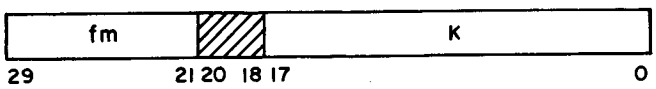
The result exponent in a no-overflow case is the difference of the dividend and divisor exponents minus 48.

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

The result is a normalized quantity when both the dividend and the divisor are normalized. Note that the machine makes no note of divide faults, i.e., when the coefficient of the dividend $\geq$ two times the coefficient of the divisor. To avoid possible incorrect results from using unnormalized operands, the operands in this instruction should be normalized. For treatment of special operands and/or indefinite forms, refer to Table 3-5 and Appendix C.

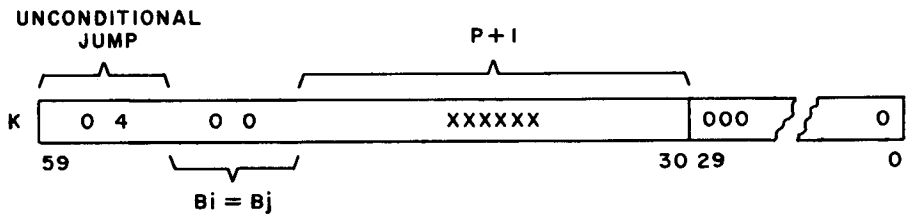
Branch

010 RJ K Return jump to K (30 Bits)



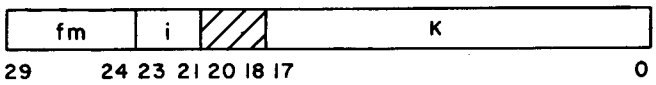
The instruction stores an 04 unconditional jump and the current address plus one [(P) + 1] in the upper half of address K, then branches to K + 1 for the next instruction. Note that this instruction is always out of the instruction stack, thus voiding the stack.

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



A jump to address K at the end of the branch routine returns the program to the original sequence.

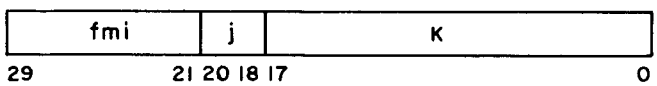
02 *JP* *Bi + K* *Jump to Bi +K* *(30 Bits)*



This instruction adds the contents of increment register Bi to K and branches to the address specified by the sum. The branch address is K when i = 0. Addition is performed modulo $2^{18}-1$.

Note that this instruction is always out of the instruction stack, thus voiding the stack. For an unindexed, unconditional jump, the 04 instruction with i = j = 0 is a better choice. Thus, if this instruction is contained in a tight loop, the instruction at K can be obtained from the stack, if possible.

030	*ZR*	*Xj*	*K*	*Jump to K if Xj = 0*	*(30 Bits)*
031	*NZ*	*Xj*	*K*	*Jump to K if Xj ≠ 0*	*(30 Bits)*
032	*PL*	*Xj*	*K*	*Jump to K if Xj = plus (positive)*	*(30 Bits)*
033	*NG*	*Xj*	*K*	*Jump to K if Xj = negative*	*(30 Bits)*
034	*IR*	*Xj*	*K*	*Jump to K if Xj is in range*	*(30 Bits)*
035	*OR*	*Xj*	*K*	*Jump to K if Xj is out of range*	*(30 Bits)*
036	*DF*	*Xj*	*K*	*Jump to K if Xj is definite*	*(30 Bits)*
037	*ID*	*Xj*	*K*	*Jump to K if Xj is indefinite*	*(30 Bits)*



These instructions branch to K when the 60-bit word in operand register X_j meets the condition specified by the i digit. The instruction allows zero, sign, and indefinite forms tests for fixed or floating point words.

The following applies to tests made in this instruction group:

a) The 030 (ZR) and 031 (NZ) operations test the full 60-bit word in Xj. The words 000... 000 and 777... 777 are treated as zero. All other words are non-zero.

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

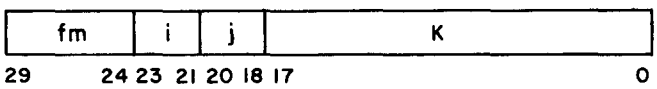
c) The 034 (IR) and 035 (OR) operations examine the upper-order 12 bits of Xj. Both plus and minus infinity are detected:

   3777XX... XX and 4000XX... XX are out of range; all other words are in range.

d) The 036 (DF) and 037 (ID) operations examine the upper-order 12 bits of Xj. Both plus and minus indefinite forms are detected:

   1777XX... XX and 6000XX... XX are indefinite; all other words are definite.

04	***EQ***	***Bi Bj K***	***Jump to K if Bi = Bj***	***(30 Bits)***
05	***NE***	***Bi Bj K***	***Jump to K if Bi ≠ Bj***	***(30 Bits)***
06	***GE***	***Bi Bj K***	***Jump to K if Bi ≥ Bj***	***(30 Bits)***
07	***LT***	***Bi Bj K***	***Jump to K if Bi < Bj***	***(30 Bits)***



These instructions test an 18-bit word from register Bi against an 18-bit word from register Bj (both words signed quantities) for the condition specified and branch to address K on a successful test. All tests against zero (all zeros) can be made by setting Bj = B0.

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

a) Positive zero is recognized as unequal to negative zero, and

b) Positive zero is recognized as greater than negative zero, and

c) A positive number is recognized as greater than a negative number.

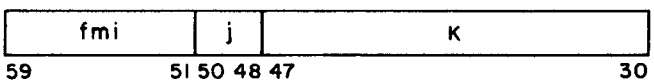
Note that the 06 and 07 instructions first perform a sign test on Bi and Bj and the Branch/No Branch determination is based on the above rules. If Bi and Bj are of the same sign, a subtract test is performed (in the Increment Unit) and the sign of the result (Bi-Bj) determines whether a Branch is made.

Extended Core Storage Communication

This category of instructions provides the ability to communicate with Extended Core Storage (ECS). Extended Core Storage communication instructions as related to the 6411/6416 are described in Appendix A.

This section describes Extended Core Storage communication instructions (and ramifications) only; information on Extended Core Storage itself is presented in Appendix D.

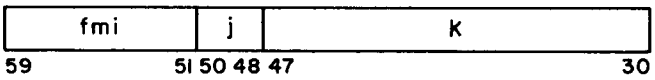
011* REC Bj + K Read Extended Core Storage (30 Bits)



This instruction initiates a Read operation to transfer $\left[(Bj) + K\right]$ 60-bit words from Extended Core Storage to Central Memory. The initial Extended Core Storage address is $\left[(X0) + RA_{ECS}\right]$; the initial Central Memory address is $\left[(A0) + RA_{CM}\right]$.

*This instruction must be located in the upper order position of the instruction word.

*012** *WEC* *Bj + K* *Write Extended Core Storage* *(30 Bits)*



This instruction initiates a Write operation to transfer $[(Bj) + K]$ 60-bit words from Central Memory to Extended Core Storage. The initial Central Memory address is $[(A0) + RA_{CM}]$; the initial Extended Core Storage address is $[(X0) + RA_{ECS}]$.

Address Formation: The starting address in Extended Core Storage is formed by taking the truncated lower-order 24 bits of operand register X0 and adding this quantity to RA_{ECS}. In the addition, both quantities are taken as positive with the upper-order 36 sign bits (zeros) extended.

RA_{ECS} is the Reference Address within Extended Core Storage, and FL_{ECS} is the allotted Field Length within Extended Core Storage. Both are 24-bit quantities contained in the Exchange Jump package; when the program specified by this package is being executed, these quantities are held in registers in the Central Processor. The lower-order six bits (2^0 - 2^5) of the RA_{ECS} and FL_{ECS} registers do not exist. The lower-order six bits in either of these 24-bit quantities always appear, therefore, as zeros.

The starting address in Central Memory is formed by a similar process; the contents of address register A0 are added to RA_{CM}. RA_{CM} is the Reference Address within Central Memory, and FL_{CM} is the allotted Field Length within Central Memory. Both are 18-bit quantities contained in the Exchange Jump package.

Note that adding the Reference Addresses to (A0) and (X0) is accomplished automatically when the Read or Write instructions are executed. The relative addresses in A0 and X0, however, must be placed there by the program prior to executing the Extended Core Storage Communication instructions.

*This instruction must be located in the upper order position of the instruction word.

An example of a typical Read Extended Core Storage operation follows:

EXAMPLE: Read Extended Core Storage

Assume a program with relative addresses in the range 0-400. The program, at relative address 200, contains a Read Extended Core Storage (011) instruction. The instruction specifies the number of words to be transferred as (Bj) + K. Prior to execution of this instruction, it is assumed that the program loaded registers Bj, A0 and X0 with block control parameters. Because the program was initiated by executing an Exchange Jump, the Central Processor holds the Reference Addresses RA_{CM} and RA_{ECS} and the Field Lengths FL_{CM} and FL_{ECS} as part of the Exchange Jump package.

It is desired, in this example, to block-transfer 300 sixty-bit words from Extended Core Storage to Central Memory. The various control parameters are assumed to be as follows:

(Bj)	= 100	RA_{ECS}	= 26500
K	= 200	FL_{ECS}	= 1600
RA_{CM}	= 1400	(A0)	= 4600
FL_{CM}	= 5300	(X0)	= 603

A map of Central Memory and Extended Core Storage would then appear as indicated in Figure 3-5.

A similar operation occurs for the Write Extended Core Storage (012) instruction.

For both Read and Write operations, the parameters held with the Central Processor which control the block transfer (namely Bj, X0, A0, RA_{CM}, RA_{ECS}, FL_{CM}, and FL_{ECS}), do not vary during the transfer. Therefore, an Exchange Jump occurring during a transfer may be effected. When the transfer program is again resumed however, the transfer is reinitiated from the initial (original) parameters, and not from the addresses used just before the interruption in the program.

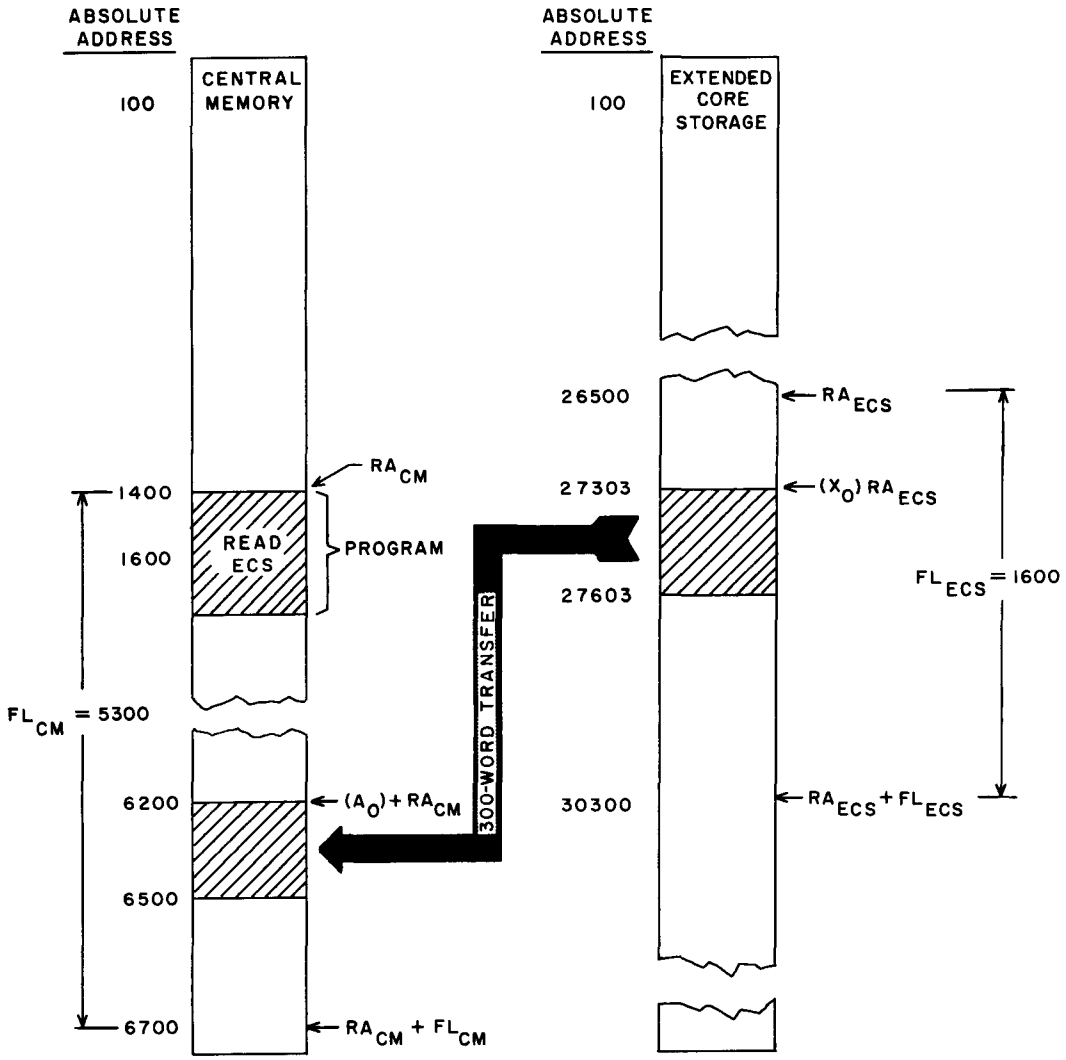


Figure 3-5. Memory Map (Read ECS Example)

Address Range Faults: Four address range fault conditions can arise when executing the Extended Core Storage Communication instructions:

- Word count fault
- Central Memory address out of range
- Extended Core Storage address out of range
- Last 60-bit word (word 7) in FL_{ECS} is referenced

a) Word Count

If, in forming the word count $\left[(Bj) + K\right]$, the result is negative, an address range fault occurs. If the Address Out of Range bit is set in the Exit Mode register, an error stop occurs; if this bit is clear, the Central Processor passes to the next instruction word at (P)+1 with no data transfer.

b) Central Memory Address

Central Memory address out of range is checked by comparing FL_{CM} with the sum $\left[(A0) + (Bj) + K\right]$. FL_{CM} must be greater than this sum or an address range fault occurs. If the Address Out of Range bit is set in the Exit Mode register, an error stop occurs; if this bit is clear, the Central Processor passes to the next instruction word at (P)+1 with no data transfer.

c) Extended Core Storage Address

Extended Core Storage address out of range is checked by comparing FL_{ECS} with the sum $\left[(X0) + (Bj) + K\right]$. In the comparison, FL_{ECS} is a 24-bit quantity with 36 upper-order bits of sign extended; X0 holds the 24-bit address quantity with 36 zeros occupying the upper-order bit positions. The result of this subtraction should always be negative; if positive, an address range fault occurs. If the Address Out of Range bit is set in the Exit Mode register, an error stop occurs; if this bit is clear, the Central Processor passes to the next instruction word at (P)+1 with no data transfer.

d) Word 7 reference in FL_{ECS}

If, after formation of the ECS address, the address format specifies a reference to word 7 in relative address FL_{ECS}, an address range fault occurs. If the Address Out of Range bit is set in the Exit Mode register, an error stop occurs; if this bit is clear, the Central Processor passes to the next instruction word at (P) + 1 with no data transfer.

Note that address range checks are made on the entire block of both Extended Core Storage and Central Memory addresses before the transfer (Read or Write) is begun. If any address in the block to be transferred is out of range, either in Central Memory or Extended Core Storage, no data is transferred, regardless of whether or not the Address Out of Range bit is set in the Exit Mode register.

Error Action: An error exit is an exit to the lower-order 30 bits of the instruction word containing the ECS Read or Write instruction. These 30 bits should always hold a jump to an error routine.

Three error conditions cause an error exit:

1) Parity error(s) when reading ECS. If a parity error is detected, the entire block of data is transferred before the exit is taken.

2) The ECS bank from/to which data is to be transferred is not available because the bank is in Maintenance mode, or the bank has lost power. If either of these conditions exists on an attempted Read or Write, an immediate error exit is taken.

3) An attempt to reference a nonexistent address. On an attempted Write operation, no data transfer occurs and an immediate error exit is taken. If the attempted operation is a Read, and addresses are in range, zeros are transferred to Central Memory. This is a convenient high-speed method of clearing blocks of Central Memory.

Exchange Jump During ECS Communication: If an Exchange Jump occurs while an Extended Core Storage transfer is in progress, the exchange waits until completion of a record. Action is then as follows:

a) If the record just completed is the last record of the block transfer, and the transfer was error-free, the Central Processor exits to (P)+1. The Exchange Jump then takes place.

b) If the record just completed is the last record of the block transfer, and an error condition exists, the Central Processor exits to the lower instruction, executes it, and the Exchange Jump is performed.

c) If the record just completed does not complete the block transfer, the Exchange Jump occurs, and (P) are stored in the Exchange Jump package. A return Exchange Jump to this program begins execution with the ECS Read or Write instruction and restarts the transfer. Note the transfer does not resume at the point it was truncated; rather, the entire transfer must be repeated.

4. PERIPHERAL AND CONTROL PROCESSORS

ORGANIZATION

The ten Peripheral and Control Processors are identical and operate independently and simultaneously as stored-program computers. Thus ten programs may be running at one time. A combination of processors can be involved in one problem, the solution of which may require a variety of I/O tasks plus use of Central Memory and Central Processor(s). Figure 4-1 shows data flow between I/O devices, the processors, and Central Memory.

The Peripheral and Control Processors act as system control computers and I/O processors. This permits the Central Processor to continue high-speed computations while the Peripheral and Control Processors do the slower I/O and supervisory operations.

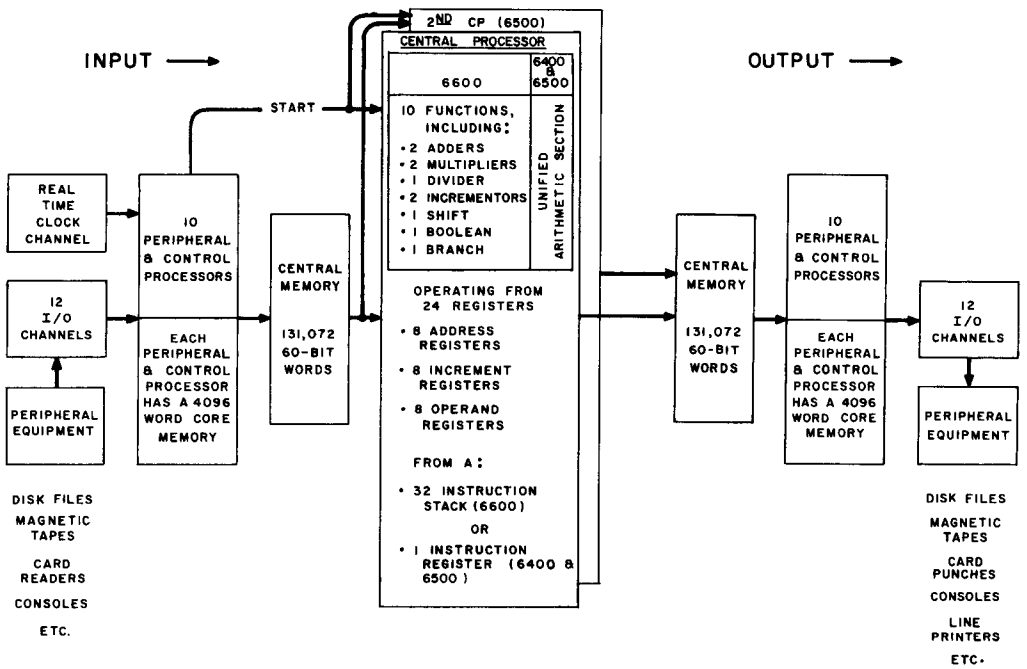


Figure 4-1. Flow Chart: 6400/6500/6600 Systems

Each processor has a 12-bit, 4096 word random-access memory (not a part of Central Memory) with a cycle time of 1000 ns (major cycle). Execution time of processor instructions is based on memory cycle time. A minor cycle is 1/10 of a major cycle and is another basic time interval.

All processors communicate with external equipment and each other on 12 independent, bi-directional I/O channels. All channels are 12-bit (plus control) and each may be connected to one or more external devices. Only one external equipment can communicate on one channel at one time, but all 12 channels can be active at one time. Data is transferred into or out of the system in 12-bit words; each channel has a single register which holds the data word being transferred in or out. Each channel operates at a maximum rate of one word per major cycle.

Data flows between a processor memory and the external device in blocks of words (a block may be as small as one word). A single word may be transferred between an external device and the A register of a processor.

The I/O instructions direct all activity with external equipment. These instructions determine the status of and select an equipment on any channel and transfer data to or from the selected device. Two channel conditions are made available to all processors as an aid to orderly use of channels.

- Each channel has an active/inactive flag to signal that it has been selected for use and is busy with an external device.
- Each channel has a full/empty flag to signal that a word (function or data) is available in the register associated with the channel.

Either state of both flags can be sensed. In general, I/O operation involves the following steps:

1) Determine channel inactive
2) Determine equipment ready
3) Select equipment
4) Activate channel
5) Input/Output data
6) Disconnect channel

One processor may communicate with another over a channel which is selected as output by one and input by the other. A common channel can be reserved for interprocessor communication and order preserved by determining equipment and channel status.

A real-time clock reading is available on a channel which is separate from the twelve I/O channels. The clock period is 4096 major cycles. The clock starts with power on and runs continuously and cannot be preset or altered. The clock may be used to determine program running time or other functions such as time-of-day, as required.

Each processor exchanges data with Central Memory in blocks of n words. Five successive 12-bit processor words are assembled into a 60-bit word and sent to Central Memory. Conversely, a 60-bit Central Memory word is disassembled into five 12-bit words and sent to successive locations in a processor memory. Separate assembly (write) and disassembly (read) paths to Central Memory are shared by all ten processors. Up to four processors may be writing in Central Memory while another four are simultaneously reading from Central Memory.

The processors generally do not solve complex arithmetic and logical problems;usually they perform I/O operations for running Central Processor programs and organize problem data (operands, addresses, constants, length of program, relative starting address, exit mode), and store it in Central Memory. Then, an Exchange Jump instruction starts (or interrupts) the Central Processor and provides it with the starting address of a problem on file in Central Memory. At the next convenient breakpoint, the Central Processor exchanges the contents of its A, B, and X registers, program address, relative starting address, length of program, Exit mode and Extended Core Storage parameters with the same information for the new program. A later Exchange Jump may return to complete the interrupted program.

Programs for the ten processors are written in the conventional manner and are executed in a multiplexing arrangement which uses the principle of time-sharing. Thus, the ten programs operate from separate memories, but all share a common facility for add/subtract, I/O, data transfer to/from Central Memory, and other necessary instruction control facilities. The multiplex consists of a 10-position barrel, which stores information (in parallel) about the current instruction in each of 10 programs, and a common instruction control device, or slot (Figure 4-2). The 10 program steps move

around the barrel in series, and each step is presented in turn to the slot. A portion of or all of the instruction steps are performed in one pass through the slot, and the altered instruction (or next instruction in a program) is reentered in the barrel for the next excursion. One or more trips around the barrel complete execution of an instruction. Thus, up to 10 programs are in operation at one time, and each program is acted upon once every 1000 ns.

One cycle of the multiplex is 1000 ns, with 900 ns consumed in the barrel and 100 ns (minor cycle) in the slot. Instructions in the barrel are interpreted at critical time intervals so that information is available in the slot at the time the instruction is ready to enter the slot. Hence, a reference to memory for data is determined ahead of time so that the data word is available in the slot when the instruction arrives. Similarly, instructions are interpreted before they reach the slot so that control paths in the slot are established when the instruction arrives.

The slot contains two adders as part of the instruction control. One adder is 12 bits, and the other is 18 bits. Both adders treat all quantities as one's complement.

For I/O instructions or communication with Central Memory, one pass through the slot transfers one 12-bit word to or from a peripheral memory. Thus, block transfer of data requires a number of trips around the barrel.

The barrel network holds four quantities which pertain to the current instruction in each of the programs. The quantities are held in registers which require a total of 51 bits. (The barrel can be considered as a 51 x 10 shifting matrix which is closed by the slot.) The barrel registers are referred to implicitly in the instruction steps and are discussed under Registers, page 4-8.

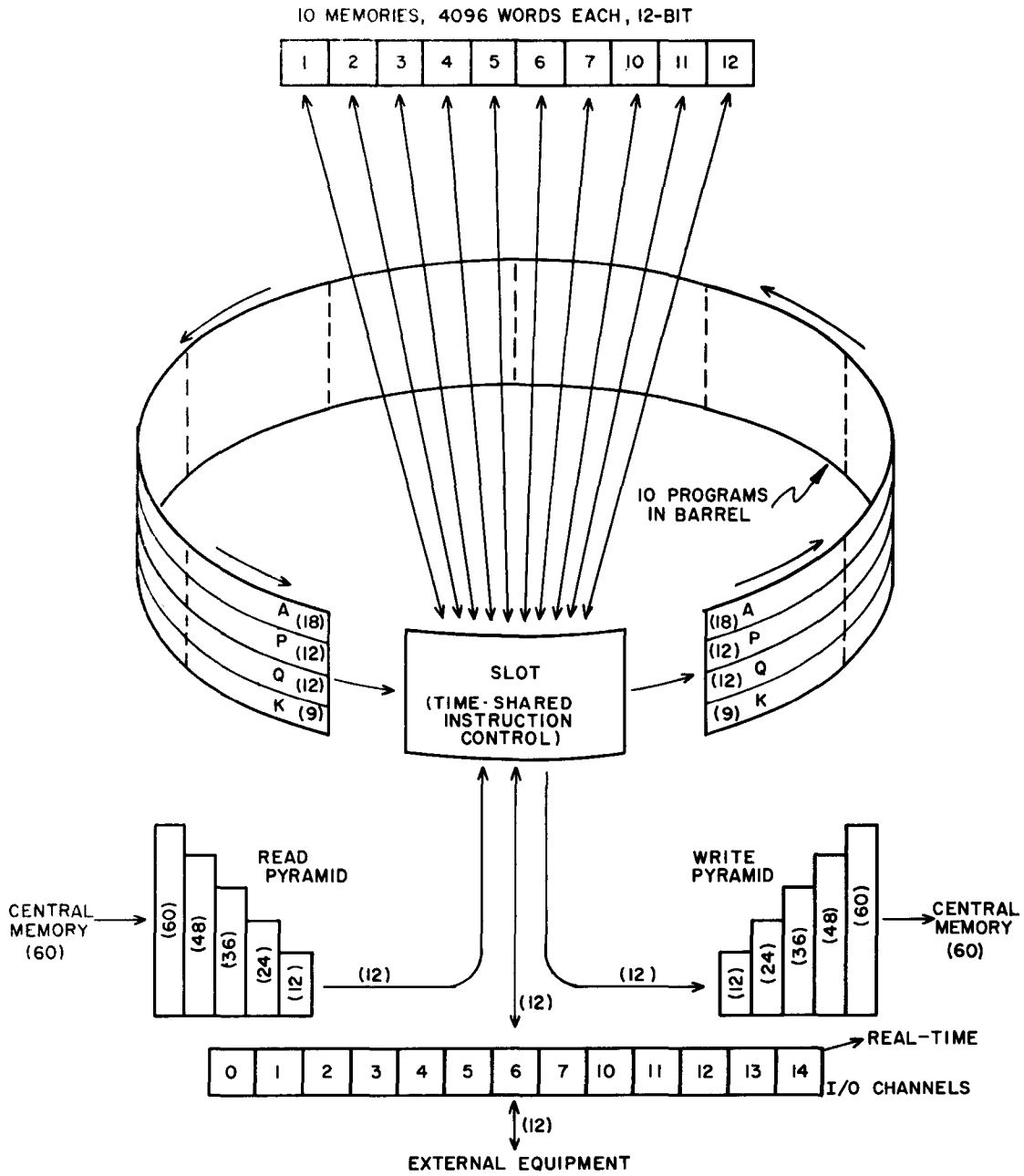
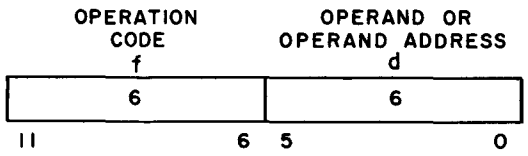


Figure 4-2. Peripheral and Control Processors

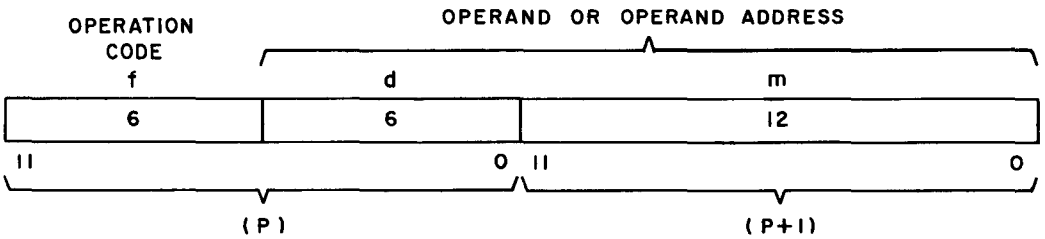
PERIPHERAL PROCESSOR PROGRAMMING

Instruction Formats

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



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



Address Modes

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

No Address

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

Direct Address

In this mode d or (m + (d)) is used as the address of the operand. The d quantity specifies one of the first 64 addresses in memory ($0000-0077_8$). The (m+(d)) quantity generates a 12-bit address for referencing all possible peripheral memory locations ($0000-7777_8$). If d ≠ 0, the content of address d is added to m to produce an operand address (indexed addressing). If d = 0, m is taken as the operand address.

EXAMPLE: Address Modes

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

Then:

MODE	INSTRUCTION	A REGISTER
No Address	LDN d	000025
	LDC dm	250100
Direct Address	LDD (d)	000150
	LDM (m + (d))	001234
Indirect Address	LDI ((d))	007776

Indirect Address

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

The Description of Instructions section, page 4-9, uses the expression (d) to define the contents of memory location d. An expression with double parentheses ((d)) refers to indirect addressing. The expression (m + (d)) refers to direct addressing when d = 0 and to indexed direct addressing when d ≠ 0. Table 4-1 summarizes the addressing modes used for the various Peripheral and Control Processor instructions.

TABLE 4-1. ADDRESSING MODES FOR PERIPHERAL AND CONTROL PROCESSOR INSTRUCTIONS

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

Registers

The four registers in the barrel are A, P, Q, and K. Each plays an important part in the execution of processor instructions.

A Register (18 bits)

The Arithmetic or A register is an adder. Quantities are treated as positive and overflows are ignored. No sign extension is provided for 6-bit or 12-bit quantities which are entered in the low order bits. However, the unused high-order bits are cleared to

zero. Zero is represented by all zeros. The A register holds an 18-bit Central Memory address during several instructions. A also participates in shift, logical, and some I/O instructions.

P Register (12 bits)

The Program Address register or P register holds the address of the current instruction. At the beginning of each instruction, the contents of P are advanced by one to provide the address of the next instruction in the program. If a jump is called for, the jump address is entered in P.

Q Register (12 bits)

The Q register holds the lower six bits of a 12-bit instruction word, or, when the six bits specify an address, Q holds the 12-bit word which is read from that address. Q is an adder which may add +1 or -1 to its content.

K Register (9 bits)

The K register holds the upper six bits (operation code) of an instruction and a 3-bit trip count designator. The trip count is a sequencing scheme to lend control to the sequential execution of an instruction.

There are other registers which provide indirect or transient control during execution of instructions. These include registers associated with the I/O channels, the registers in the read and write pyramids which assemble successive 12-bit words into 60-bit words or vice versa, and registers which hold the storage address and the word at that address for each peripheral memory.

Description of Peripheral Processor Instructions

This section describes the Peripheral and Control Processor instructions. Table 4-2 lists designators used throughout the section.

TABLE 4-2. PERIPHERAL AND CONTROL PROCESSOR INSTRUCTION DESIGNATORS

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

Preceding the description of each instruction is the octal code, mnemonic code and address field, the instruction name and instruction length. Mnemonic codes and address field mnemonics are from ASPER, the Peripheral and Control Processor Assembly language.

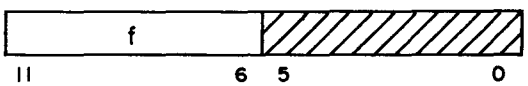
EXAMPLE:

52	SBM	md	Subtract (m + (d))	(24 Bits)
Octal Code	Mnemonic Code	Address Field	Instruction Name	Instruction Length

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

No Operation

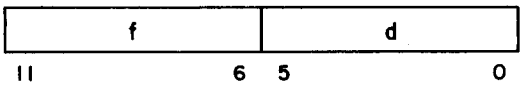
00	***PSN***	***Pass***	***(12 Bits)***
24	***PSN***	***Pass***	***(12 Bits)***
25	***PSN***	***Pass***	***(12 Bits)***



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

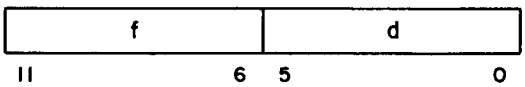
Data Transmission

14 LDN d Load d (12 Bits)



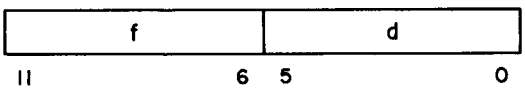
This instruction clears the A register and loads d. The upper 12 bits of A are zero.

15 LCN d Load Complement d (12 Bits)



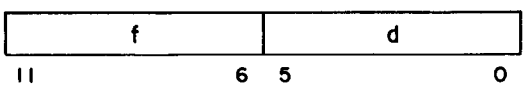
This instruction clears the A register and loads the complement of d. The upper 12 bits of A are set to one.

30 LDD d Load (d) (12 Bits)



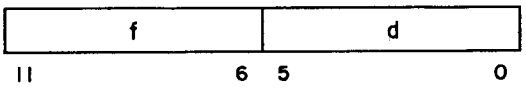
This instruction clears the A register and loads the contents of location d. The upper six bits of A are zero.

34 STD d Store (d) (12 Bits)



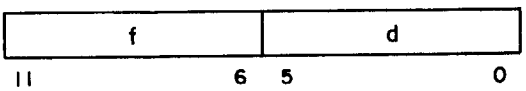
This instruction stores the lower 12 bits of A in location d.

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



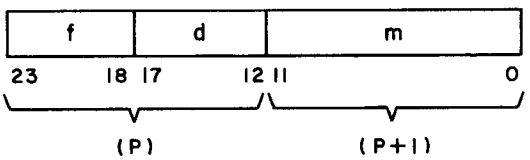
This instruction clears the A register and loads a 12-bit quantity that is obtained by indirect addressing. The upper six bits of A are zero. Location d is read out of memory, and the word obtained is used as the operand address.

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



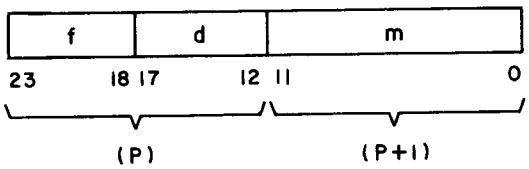
This instruction stores the lower 12 bits of A in the location specified by the contents of location d.

20 **LDC** *dm* ***Load dm*** ***(24 Bits)***



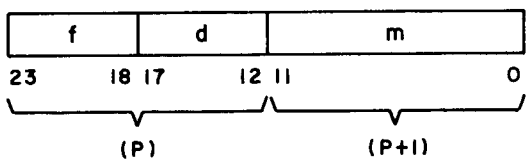
This instruction clears the A register and loads an 18-bit quantity consisting of d as the higher six bits and m as the lower 12 bits. The contents of the location following the present program address are read out to provide m.

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



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

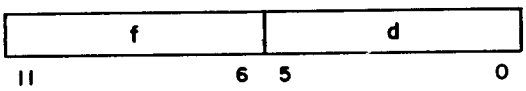
54 **STM** *m d* ***Store (m + (d))*** ***(24 Bits)***



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

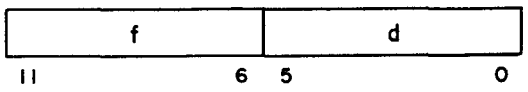
Arithmetic

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



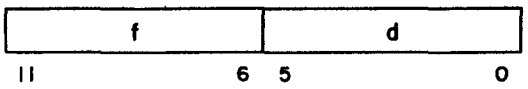
This instruction adds d (treated as a 6-bit positive quantity) to the content of the A register.

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



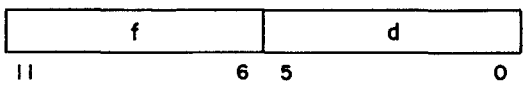
This instruction subtracts d (treated as a 6-bit positive quantity) from the content of the A register.

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



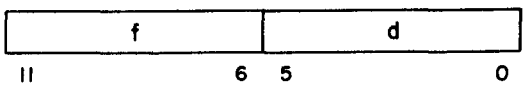
This instruction adds to the A register the contents of location d (treated as a 12-bit positive quantity).

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



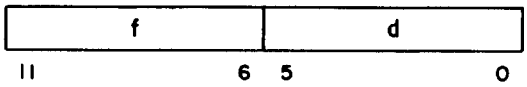
This instruction subtracts from the A register the contents of location d (treated as a 12-bit positive quantity).

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



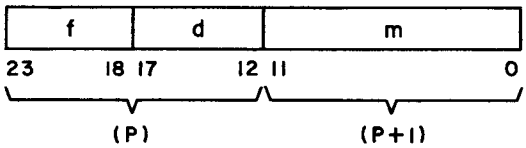
This instruction adds to the content of A a 12-bit operand (treated as a positive quantity) obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address.

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



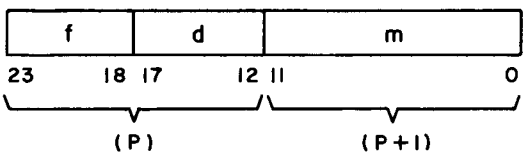
This instruction subtracts from the A register a 12-bit operand (treated as a positive quantity) obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address.

21 **ADC** *dm* **Add dm** *(24 Bits)*



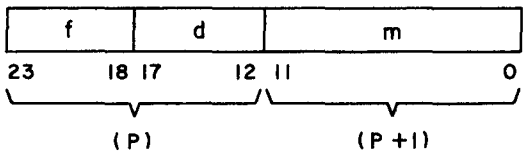
This instruction adds to the A register the 18-bit quantity consisting of d as the higher six bits and m as the lower 12 bits. The contents of the location following the present program address are read out to provide m.

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



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

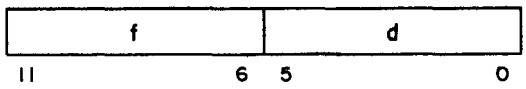
52 **SBM** *m d* ***Subtract (m + (d))*** ***(24 Bits)***



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

Shift

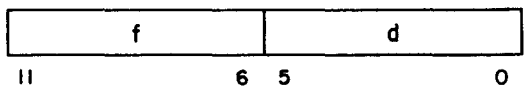
10 **SHN** *d* ***Shift d*** ***(12 Bits)***



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

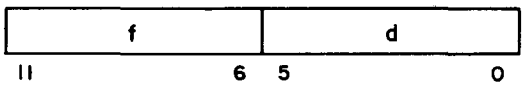
Logical

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



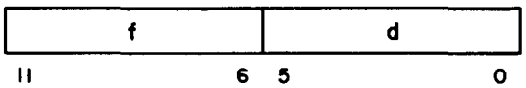
This instruction forms in A the bit-by-bit logical difference of d and the lower six bits of A. This is equivalent to complementing individual bits of A that correspond to bits of d that are one. The upper 12 bits of A are not altered.

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



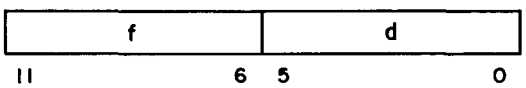
This instruction forms the bit-by-bit logical product of d and the lower six bits of the A register, and leaves this quantity in the lower 6 bits of A. The upper 12 bits of A are zero.

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



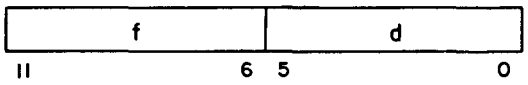
This instruction clears any of the lower six bits of the A register where there are corresponding bits of d that are one. The upper 12 bits of A are not altered.

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



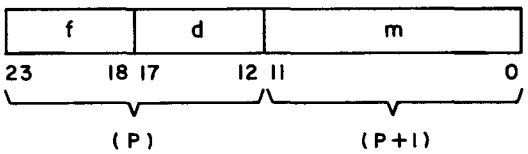
This instruction forms in A the bit-by-bit logical difference of the lower 12 bits of A and the contents of location d. This is equivalent to complementing individual bits of A which correspond to bits of (d) that are one. The upper six bits of A are not altered.

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



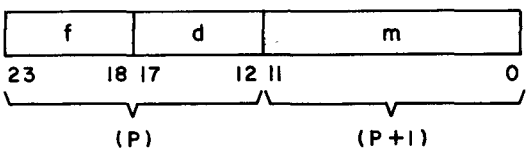
This instruction forms in A the bit-by-bit logical difference of the lower 12 bits of A and the 12-bit operand obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address. The upper six bits of A are not altered.

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



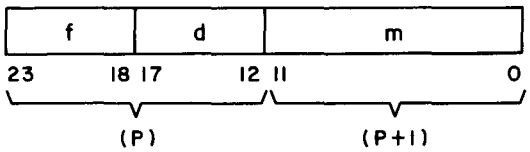
This instruction forms in the A register the bit-by-bit logical product of the contents of A and the 18-bit quantity dm. The upper six bits of this quantity consist of d and the lower 12 bits are the content of the location following the present program address.

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



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

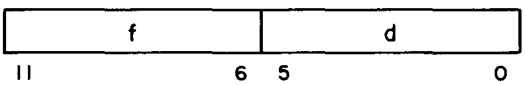
53 **LMM** ***m d*** ***Logical difference (m + (d))*** ***(24 Bits)***



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

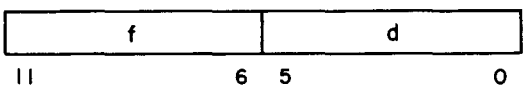
Replace

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



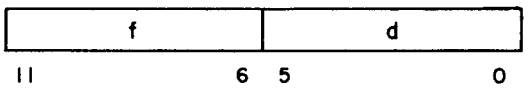
This instruction adds the quantity in location d to the contents of A and stores the lower 12 bits of the result at location d. The resultant sum is left in A at the end of the operation and the original contents of A are destroyed.

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



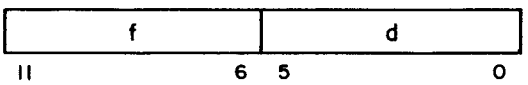
The quantity in location d is replaced by its original value plus one. The resultant sum is left in A at the end of the operation, and the original contents of A are destroyed.

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



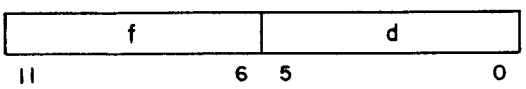
The quantity in location d is replaced by its original value minus one. The resultant difference is left in A at the end of the operation, and the original contents of A are destroyed.

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



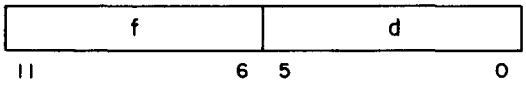
The operand which is obtained from the location specified by the contents of location d, is added to the contents of A, and the lower 12 bits of the sum replace the original operand. The resultant sum is also left in A at the end of the operation.

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



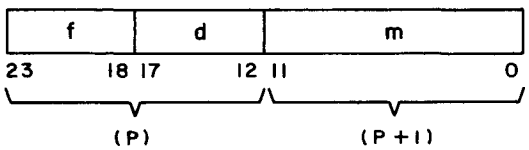
The operand, which is obtained from the location specified by the contents of location d, is replaced by its original value plus one. The resultant sum is also left in A at the end of the operation, and the original contents of A are destroyed.

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



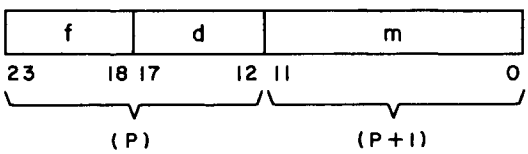
The operand, which is obtained from the location specified by the contents of location d, is replaced by its original value minus one. The resultant difference is also left in A at the end of the operation, and the original contents of A are destroyed.

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



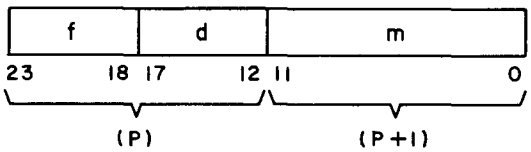
The operand, which is obtained from the location determined by indexed direct addressing, is added to the contents of A, and the lower 12 bits of the sum replace the original operand in memory. The resultant sum is also left in A at the end of the operation, and the original contents of A are destroyed.

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



The operand, which is obtained from the location determined by indexed direct addressing, is replaced by its original value plus one (see instruction 50, page 4-13 for explanation of addressing). The resultant sum is also left in A at the end of the operation, and the original contents of A are destroyed.

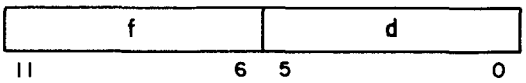
57 *SOM* *m d* ***Replace subtract one (m + (d))*** ***(24 Bits)***



The operand, which is obtained from the location determined by indexed direct addressing, is replaced by its original value minus one (see instruction 50, page 4-13 for explanation of addressing). The resultant difference is also left in A at the end of the operation, and the original contents of A are destroyed.

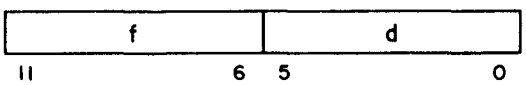
Branch

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



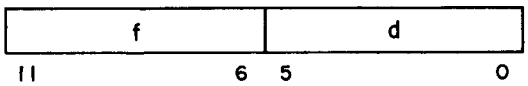
This instruction provides an unconditional jump to any instruction up to 31 steps forward or backward from the current program address. The value of d is added to the current program address. If d is positive (01 - 37), then 0001 (+1) - 0037 (+31) is added and the jump is forward. If d is negative (40 - 76) then 7740 (-31) - 7776 (-1) is added and the jump is backward. The program stops (a Dead Start is necessary to restart the machine) when d = 00 or 77.

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



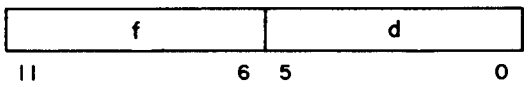
This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is zero, the jump is taken. If the content of A is non-zero, the next instruction is executed. Negative zero (777777) is treated as non-zero.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

05 NJN d Nonzero jump d (12 Bits)



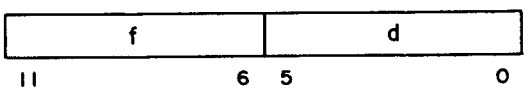
This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is nonzero, the jump is taken. If A is zero, the next instruction is executed. Negative zero (777777) is treated as nonzero. For interpretation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

06 PJN d Plus jump d (12 Bits)



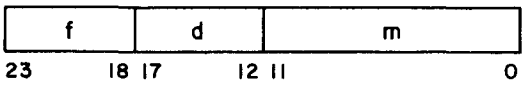
This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is positive, the jump is taken. If A is negative, the next instruction is executed. Positive zero is treated as a positive quantity; negative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

07 MJN d Minus jump d (12 Bits)



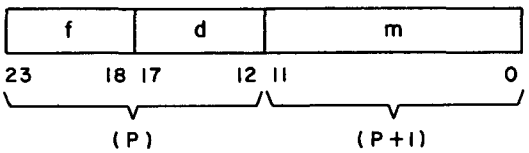
This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is negative, the jump is taken. If A is positive, the next instruction is executed. Positive zero is treated as a positive quantity; negative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

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



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

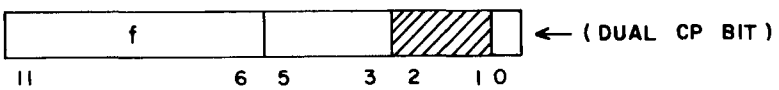
02 **RJM** *m d* ***Return jump to m + (d)*** ***(24 Bits)***



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

Central Processor and Central Memory

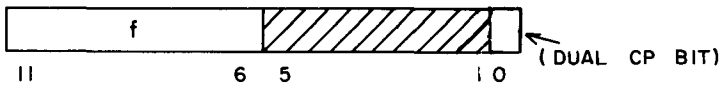
260 **EXN** ***Exchange jump*** ***(12 Bits)***



This instruction transmits an 18-bit (absolute) address (only 17 bits are used) from the A register to the Central Processor with a signal which tells the Central Processor to perform an Exchange Jump, with the address in A as the starting location of a file of 16 words containing information about the Central Processor program to be executed. The 18-bit initial address must be entered in A before this instruction is executed. The Central Processor replaces the file with similar information from the interrupted Central Processor program. The Peripheral Processor is not interrupted.

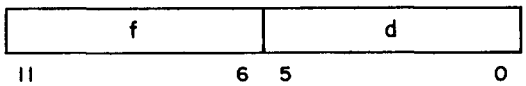
In 6500 systems with dual Central Processors, the lowest order bit of the instruction format specifies which Central Processor the Exchange Jump will interrupt. In 6400 and 6600 systems, this bit is not interpreted.

27 **RPN** ***Read program address*** ***(12 Bits)***



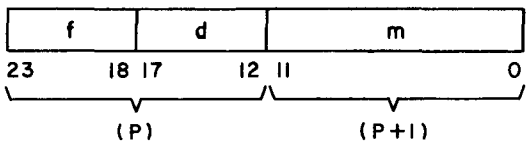
This instruction transfers the content of the Central Processor Program Address register, P, to the Peripheral Processor A register; this allows the Peripheral Processor to determine whether the Central Processor is running. In a 6500 system with dual Central Processors, the lowest order bit of the instruction format specifies which Central Processor P register is to be examined. In 6400 and 6600 systems, this bit is not interpreted.

60 **CRD** *d* ***Central read from (A) to d*** ***(12 Bits)***



This instruction transfers a 60-bit word from Central Memory to five consecutive locations in the processor memory. The 18-bit address of the Central Memory location must be loaded into A prior to executing this instruction. (Note that this is an absolute address.) The 60-bit word is disassembled into five 12-bit words beginning at the left. Location d receives the first 12-bit word. The remaining 12-bit words go to succeeding locations.

61 **CRM** *m d* ***Central read (d) words from (A) to m*** ***(24 Bits)***



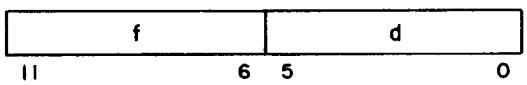
This instruction reads a block of 60-bit words from Central Memory. The content of location d gives the block length. The 18-bit address of the first central word must be loaded into A prior to executing this instruction. (Note that this is an absolute address.) During the execution of the instruction, (P) goes to processor address 0 and P holds m. Also, (d) goes to the Q register where it is reduced by one as each central word is processed. The original content of P is restored at the end of the instruction.

Each central word is disassembled into five 12-bit words beginning with the high-order 12 bits. The first word is stored at processor memory location m. The content of P

(which is holding m) is advanced by one to provide the next address in the processor memory as each 12-bit word is stored. If P overflows, operation continues as P is advanced from 7777_8 to 0000_8. These locations will be written into as if they were consecutive.

The content of A is advanced by one to provide the next Central Memory address after each 60-bit word is disassembled and stored. Also, the contents of the Q register are reduced by one. The block transfer is complete when Q = 0. The block of Central Memory locations goes from address (A) to address (A) + (d) -1. The block of processor memory locations goes from address m to m + 5(d) -1.

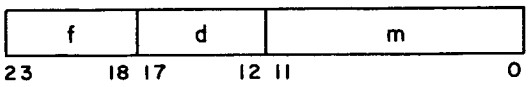
62 CWD d Central write to (A) from d (12 Bits)



This instruction assembles five successive 12-bit words into a 60-bit word and stores the word in Central Memory. The 18-bit address word designating the Central Memory location must be in A prior to execution of the instruction. (Note that this is an absolute address.)

Location d holds the first word to be read out of the processor memory. This word appears as the higher order 12 bits of the 60-bit word to be stored in Central Memory. The remaining words are taken from successive addresses.

63 **CWM** *m d* ***Central write (d) words to (A) from m*** ***(24 Bits)***



This instruction assembles a block of 60-bit words and writes them in Central Memory. The content of location d gives the number of 60-bit words. The content of the A register gives the beginning Central Memory address. (Note that this is an absolute address.) During the execution of this instruction (P) goes to processor address 0 and P holds m. Also, (d) goes to the Q register, where it is reduced by one as each central word is assembled. The original content of P is restored at the end of the instruction.

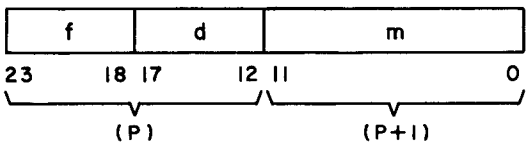
The content of P (the m portion of the instruction) gives the address of the first word to be read out of the processor memory. This word appears as the higher order 12 bits of the first 60-bit word to be stored in Central Memory.

The content of P is advanced by one to provide the next address in the processor memory as each 12-bit word is read. If P overflows, operation continues as P is advanced from 7777_8 to 0000_8. These locations will be read from as if they were consecutive.

The content of A is advanced by one to provide the next Central Memory address after each 60-bit word is assembled. Also, Q is reduced by one. The block transfer is complete when Q = 0.

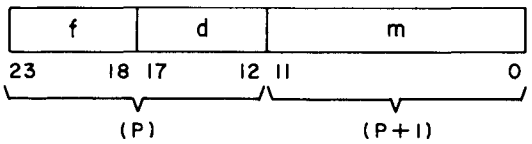
Input/Output

64 **AJM** *m d* ***Jump to m if channel d active*** ***(24 Bits)***



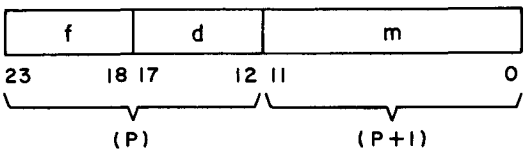
This instruction provides a conditional jump to a new program sequence beginning at an address given by the contents of m. The jump is taken if the channel specified by d is active. The current program sequence continues if the channel is inactive.

65 ***IJM*** *m d* ***Jump to m if channel d inactive*** *(24 Bits)*



This instruction provides a conditional jump to a new program sequence beginning at an address given by m. The jump is taken if the channel specified by d is inactive. The current program sequence continues if the channel is active.

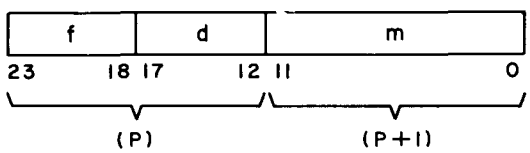
66 ***FJM*** *m d* ***Jump to m if channel d full*** *(24 Bits)*



This instruction provides a conditional jump to a new program sequence beginning at an address given by m. The jump is taken if the channel designated by d is full. The present program sequence continues if the channel is empty.

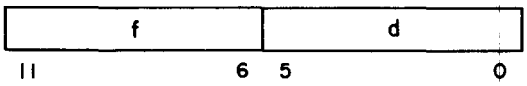
An input channel is full when the input equipment has placed a word on the channel and that word has not yet been sampled by a processor. The channel is empty when a word has been accepted. An output channel is full when a processor places a word on the channel. The channel is empty when the output equipment has sampled the word.

67 ***EJM*** *m d* ***Jump to m if channel d empty*** *(24 Bits)*



This instruction provides a conditional jump to a new program sequence beginning at an address specified by m. The jump is taken if the channel specified by d is empty. The current program sequence continues if the channel is full. (See instruction 66 for explanation of full and empty.)

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

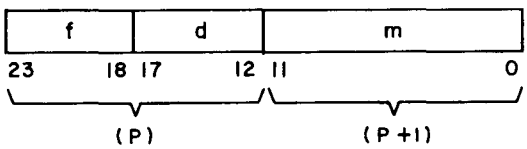


This instruction transfers a word from input channel d to the lower 12 bits of the A register. The upper 6 bits of the A register are cleared to zeros.

NOTE

This instruction will hang up the Peripheral Processor if executed when the channel is inactive.

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



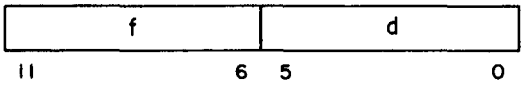
This instruction transfers a block of 12-bit words from input channel d to the processor memory. The content of A gives the block length. The contents of location m specifies the processor address which is to receive the first word. The content of A is reduced by one as each word is read. The input operation is complete when A = 0.

During this instruction address 0000 temporarily holds P, while m is held in the P register. The content of P advances by one to give the address for the next word as each word is stored.

NOTE

If this instruction is executed when the data channel is inactive, no input operation is accomplished and the program continues at P + 2.

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

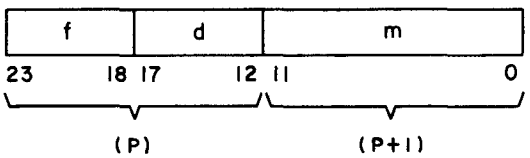


This instruction transfers a word from A (lower 12 bits) to output channel d.

NOTE

This instruction will hang up the Peripheral Processor if executed when the channel is inactive.

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



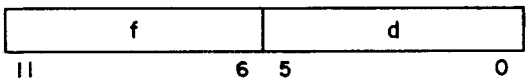
This instruction transfers a block of words from the processor memory to channel d. The first word comes from the address specified by m. The content of A specifies the number of words to be sent out. The content of A is reduced by one as each word is read out. The output operation is complete when A = 0.

During this instruction address 0000 temporarily holds P, while m is held in the P register. The content of P advances by one to give the address of the next word as each word is taken from memory.

NOTE

If this instruction is executed when the data channel is inactive, no output operation is accomplished and the program continues at P + 2.

74 *ACN* *d* ***Activate channel d*** ***(12 Bits)***

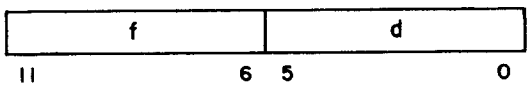


This instruction activates the channel specified by d. Activating a channel (must precede a 70 - 73 instruction) alerts and prepares the I/O equipment for the exchange of data.

NOTE

Activating an already active channel causes the Peripheral Processor to hang up.

75 *DCN* *d* ***Disconnect channel d*** ***(12 Bits)***

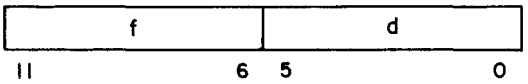


This instruction deactivates the channel specified by d. As a result, the I/O equipment stops and the buffer terminates.

NOTE

1) Do not deactivate an already inactive (i.e. disconnected) channel or the Peripheral Processor will hang up.
2) Do not disconnect the channel before first sensing for Channel Empty.
3) Do not deactivate a channel before stopping the associated processor.
4) Do not deactivate a channel before putting a useful program in the associated processor. Processors after Dead Start are hung up on an Input. Deactivating a channel after Dead Start causes an exit to address 0001 and execution of program.

76 **FAN** *d* ***Function (A) on channel d*** ***(12 Bits)***

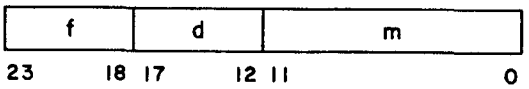


The external function code in the lower 12 bits of A is sent out on channel d.

NOTE

Do not execute this instruction when the channel is Active or the Peripheral Processor will hang up.

77 **FNC** *m d* ***Function m on channel d*** ***(24 Bits)***



The external function code specified by m is sent out on channel d.

Access to Central Memory

The Peripheral and Control Processors have access to all Central Memory storage locations. Four of the instructions (60, 61, 62, 63 - described previously) transfer one word or a block of words from a peripheral memory to Central Memory or vice versa. Data from an external equipment is read into a peripheral memory and, with separate instructions, transferred from there to Central Memory where it may be used by the Central Processor. Conversely, data is transferred from Central Memory to a peripheral memory and then transferred by separate instructions to external equipment. Note that all addresses sent to Central Memory from Peripheral and Control Processors are absolute addresses, rather than relative addresses.

Read Central Memory

The 60 and 61 instructions read one word or a block of 60-bit Central Memory words. The Central Memory words are delivered to a five stage read pyramid where they are disassembled into five 12-bit words, beginning with the high-order word. Successive

stages of the pyramid contain 60, 48, 36, 24 and 12 bits. The upper 12 bits of the word are removed and sent to a peripheral memory as the word is transferred through each stage. Thus, a 60-bit word is disassembled into five 12-bit words.

Words move through the pyramid when the stage ahead is clear. One pass through the slot determines that the next stage is clear, sends 12 bits of the word to a peripheral memory, and moves the word ahead to the cleared stage. The pyramid is a part of the slot and may be time shared by up to four processors. Thus four Central Memory words may be in the pyramid at one time in varying stages of disassembly. With a full pyramid, Read instructions from other processors are partially executed (housekeeping) and circulated unchanged in the barrel until the number of pyramid users drop below four. Waiting processors are serviced in the order in which they appear at the slot. Other instruction control provides address incrementing and keeps the word count.

The Central Memory starting address must be entered in A before a Read instruction is executed. A Load dm (20) instruction may be used for this. For a one word transfer, the d portion of the Read (60) instruction specifies the following:

d = peripheral address ($0000-0077_8$) of first 12-bit word; remaining words go to d + 1, d + 2, etc.

For a block transfer, d and m of the read (61) instruction specify the following:

(d) = number of Central Memory words to be transferred; reduced by one for each word transferred.

m = peripheral starting address; increased by one to provide locations for successive words. (A) is increased by one to locate consecutive Central Memory words.

Write Central Memory

The 62 and 63 instructions assemble 12-bit peripheral words into 60-bit words and write them in Central Memory. Peripheral words are assembled in a write pyramid and delivered from there to Central Memory. As in Read Central Memory, the pyramid is a part of the slot and is time-shared by up to four processors. Write pyramid action is similar to Read pyramid action except for the assembly.

The starting address in Central Memory is entered in A before the Write instruction is executed. For a one word transfer, the d portion of the Write (62) instruction specifies the following:

d = peripheral address ($0000-0077_8$) of first 12-bit word; remaining words are taken from d + 1, d + 2, etc.

For block transfer, d and m of the Write (63) instruction specify the following:

(d) = number of Central Memory words to be transferred; reduced by one for each word transferred.

m = peripheral starting address; increased by one to locate each successive peripheral word. (A) is increased by one to provide consecutive Central Memory locations.

Access to the Central Processor

The Peripheral and Control Processors use two instructions to communicate with the Central Processor. One instruction starts a program running in the Central Processor and the other instruction monitors the progress of the program.

Exchange Jump

The 260 instruction (described previously) starts a program running in the Central Processor or interrupts a current program and starts a new program running. In either case, the Central Processor is directed to a Central Memory file of 16 words which stores information about the new program to be executed (see Exchange Jump section, page 3-9). The 18-bit starting address of this file must be entered in A before the Exchange Jump instruction is executed. The Central Processor replaces the file with similar but current information from the interrupted program. A later Exchange Jump instruction referencing this file returns the interrupted program to the Central Processor for completion. This exchange feature permits the Peripheral Processor to time-share the Central Processor.

Read Program Address

The 27 instruction (described previously) transfers the content of the Central Processor P register into a peripheral A register. The peripheral program tests the A register content to determine the condition of the Central Processor. If A ≠ 0, the Central Processor is running a program, may have come to a normal (instruction) stop, or may have stopped due to an out-of-bounds error (unselected). (Refer to Exit Mode section, page 3-11.) If A = 0, the Central Processor has stopped due to a selected Exit mode error; the reference address for the Central Processor program is then examined to determine which error condition exists. A Stop instruction (00_8) in the upper six bits of the reference address signals a stop; the next lower six bits define the nature of the exit (see Exchange Jump section, page 3-9).

Input and Output

There are 12 instructions to direct activity on the I/O channels. These instructions select a unit of external equipment and transfer data to or from the equipment. The instructions also determine whether a channel or external equipment is available and ready to transfer data. The preparatory steps insure that the data transfer is carried out in an orderly fashion.

Each external equipment has a set of external function codes which are used by the processors to establish modes of operation and to start or stop data transfer. Also, the devices are capable of detecting certain errors (e.g., parity error) and provide an indication of these errors to the controlling processor. The external error conditions can be read into a processor for interpretation and further action. Details of mode selection and error flags in external devices such as card readers and magnetic tape systems are presented in the 6000 Series Peripheral Equipment Reference manual.

Data Channels

Each channel has a 12-bit bi-directional data register and two control flags which indicate:

- The channel is active or inactive
- The channel register is full or empty

The 64 and 65 instructions determine the state of the channel, and the 66 and 67 instructions determine the state of the register. The flags provide housekeeping information for the processors so that channels can be monitored and processed in an orderly way. The flags also provide control for the I/O operation.

Word Rate: Each processor is serviced by the slot once every major cycle. This sets the maximum word rate on a channel at one word each 1000 ns, a 1 megacycle word rate. Up to 10 processors can be communicating with I/O equipment over separate channels at this rate since each processor is regularly serviced at major cycle intervals.

Channel Active/Inactive Flag: A channel is made active by a Function (76, 77) instruction or an Activate Channel (74) instruction.

The Function instruction selects a mode of operation in the external equipment. The instruction places a 12-bit function word in the channel register and activates the channel. The external equipment accepts the function word, and its response to the processor clears the register and drops the channel active flag. The latter action produces the channel inactive flag.

The activate channel instruction prepares a channel for data transfer. Subsequent input or output instructions transfer the data. A disconnect channel instruction after data transfer is complete returns the channel to the inactive state.

Register Full/Empty Flag: A register is full when it contains a function or data word for an external equipment or contains a word received from an external equipment. The register is empty when it is cleared. The flags are turned on or off as the register changes state.

On data output, the processor places a word in the Channel register and sets the full flag. The external device accepts the word, clears the register, and sets the empty flag. The empty flag and channel active flag signal the processor to send another word to the register to repeat the sequence.

On input, the external device places a word in the register and sets the full flag. The processor stores the word, clears the register, and sets the empty flag. The empty flag and channel active flag signal the external device to deliver another word.

Data Input

Several instructions are necessary to transfer data from external equipment into a processor. The instructions prepare the channel and equipment for the transfer and then start the transfer. Some external equipment, when once started, send a series of words (record) spaced at equal time intervals and then stops automatically between records. Magnetic tape equipment is an example of this type of transfer. The processor can read all or a part of the record and then disconnect the channel to end the operation. The latter step makes the channel inactive. Other equipment, such as the display console, can send one word (or character) and then stop. The input instructions allow the input transfer to vary from one word to the capacity of the processor.

An input transfer may be accomplished in the following way:

1) Determine if the channel is inactive. A Jump to m on channel d Inactive (65) instruction does this. Here, m can be a function instruction to select Read mode or determine the status of the equipment.

2) Determine if the equipment is ready. A Function m on Channel d (77) instruction followed by an Activate channel d (74) followed by an Input to A from Channel d (70) instruction loads A with the status response of the desired equipment. Here, m is a status request code, and the status response in A can be tested to determine the course of action.

3) Select Read mode in the equipment. A Function m on Channel d (77) instruction or Function (A) on Channel d (76) instruction will send a code word to the desired device to prepare it for data transfer.

4) Enter the number of words to be transferred in A. A Load d (14) or Load (d) (30) instruction will accomplish this.

5) Activate the channel. An Activate Channel d (74) instruction sets the channel active flag and prepares for the impending data transfer.

6) Start input data transfer. An Input (A) Words to m on Channel d (71) instruction or an Input to A from Channel d (70) instruction starts data transfer. The 71 instruction transfers one word or up to the capacity of the processor memory. The 70 instruction transfers one word only.

7) Disconnect the channel. A Disconnect Channel d (75) instruction makes the channel inactive and stops the flow of input information.

The design of some external equipment requires timing considerations in issuing function, activate, and input instructions. The timing consideration may be based on motion in the equipment, i.e., the equipment must attain a given speed before sending data (e.g., magnetic tape). In general, timing considerations can be resolved by issuing the necessary instructions without an intervening time gap. The external equipment literature lists timing considerations to be taken into account.

Data Output

The data output operation is similar to data input in that the channel and equipment must be ready before the data transfer is started by an output instruction.

An output transfer may be accomplished in the following way:

1) Determine if the channel is inactive. A Jump to m on Channel d Inactive (65) instruction does this. Here, m can be a function instruction to select Write mode or determine the status of the equipment.

2) Determine if the equipment is ready. A Function m on Channel d (77) followed by an Activate channel d (74) followed by an Input to A from Channel d (70) instruction loads A with the status response of the desired equipment. Here, m is a status request code, and the status response in A can be tested to determine the course of action.

3) Select Write mode in the equipment. A Function m on Channel d (77) instruction or Function (A) on Channel d (76) instruction will send a code word to the desired device to prepare it for data transfer.

4) Enter the number of words to be transferred in A. A Load d (14) or Load d (30) instruction will accomplish this.

5) Activate the channel. An Activate Channel d (74) instruction signals an active channel and prepares for the impending data transfer.

6) Start data transfer. An Output (A) Words from m on Channel d (73) instruction or an Output from A on Channel d (72) instruction starts data transfer. The 73 instruction can transfer one or more words while the 72 instruction transfers only one word.

7) Test for channel empty. A Jump to m if Channel d Full (66) instruction where m = current address, provides this test. The instruction exits to

itself until the channel is empty. When the channel is empty, the processor goes on to the next instruction which generally disconnects the channel. The instruction acts to idle the program briefly to insure successful transfer of the last output word to the recording device.

8) Disconnect the channel. A Disconnect Channel d (75) instruction makes the channel inactive. Data flow in this case terminates automatically when the correct number of words is sent out.

Instruction timing considerations, as in a data input operation, are a function of the external device.

Real-Time Clock

The real-time clock runs continuously; its period is 4096 cycles (4.096 ms). The clock may be sampled by any Peripheral and Control Processor with an Input to A (70) instruction from channel 14_8. The clock is advanced by the storage sequence control and cannot be cleared or preset.

5. SYSTEM INTERRUPT

INTRODUCTION

Essentially, detecting and handling interruptible conditions in the 6400, 6500, and 6600 Computer Systems involves both hardware and software. This section describes hardware provisions for detecting and handling interrupt. The salient features of an operating system for implementing interrupt handling are described in the operating system reference manual.

HARDWARE PROVISIONS FOR INTERRUPT

Exchange Jump

Within a Peripheral Processor, execution of an Exchange Jump instruction initiates hardware action in the Central Processor to interrupt the current Central Processor program and substitute a program, the parameters of which are defined in the Exchange Jump package. Note that the Exchange Jump is also used to start the Central Processor from a Stop condition. (Refer to the Exchange Jump section, page 3-9.)

Channel and Equipment Status

Within the Peripheral Processors, hardware flags indicate the state of various conditions in the data channels, e.g., Full/Empty, and Active/Inactive. External equipments are capable of detecting certain errors (e.g., parity error) and hold status information reflecting their operating conditions (e.g., Ready, End of File, etc.). Channel and equipment status information may be examined by instructions in the Peripheral Processors. The Input/Output section describes these instructions. For detailed status information on external devices such as magnetic tape units and card readers, refer to literature associated with these devices.

Exit Mode

Central Processor hardware provides for three types of error halt conditions (Exit mode):

- Address out of range (i.e., out of bounds)
- Operand out of range (i.e., exponent overflow)
- Indefinite result

Detecting the occurrence of one or more of these conditions is accomplished by the hardware and causes an error halt. Note that halting on any of these conditions is selectable; selection is performed by setting appropriate flags in the Exit mode portion of the Exchange Jump package. (Refer to Exit Mode, page 3-11.)

6. MANUAL CONTROL

INTRODUCTION

Manual control of 6400/6500/6600 Computer Systems operation is provided through 1) the dead start panel and 2) the console keyboard. The Dead Start circuit is a means of manually entering a 12-word program (normally a load routine) to start operation. The console keyboard provides for the manual entry of data or instructions under program control.

DEAD START

The dead start panel (Figure 6-1) contains a 12 x 12 matrix of toggle switches, a MODE switch to select SWEEP, LOAD, or DUMP, and a DEAD START switch. The panel also contains memory margin switches which are used for maintenance checks. The three modes of operation (Load, Sweep, Dump) selectable via the dead start panel are described below.

Load Mode

To initially load programs and data into the computer system, the MODE switch is placed in the LOAD position. The matrix of toggle switches is set to a 12-word (or less) program (switch up = "1", switch down = "0"). The program set in the switch matrix is normally a load routine used to load a larger program from an input device such as a disk file or magnetic tape unit.

The DEAD START switch is turned on momentarily, then off. Turning on the DEAD START switch initiates the following operations:

1) Assigns processors $0\text{-}11_8$ to corresponding data channels.
2) Sends a Master Clear to all I/O channels. A Master Clear removes all equipment selections except the dead start panel, and sets all channels to the Active and Empty condition (ready for input).

3) Sets all processors to the Input (71) instruction.

4) Clears the A and P registers in all processors to zero.

5) Loads the 12 words from the toggle switches into memory locations $0001\text{-}0014_8$ of processor 0.

After the switch matrix program is read from the dead start panel, the panel is automatically disconnected. Processor 0 reads location 0000, adds one to its content, and begins executing the program at address 0001. The other processors are still set to the Input (71) instruction and may receive data from processor 0 via their assigned channels.

Sweep Mode

Placing the MODE switch in the SWEEP position and momentarily turning on the DEAD START switch results in the following:

1) Sets all processors to instruction 50X.
2) Clears all processor P registers to zero.

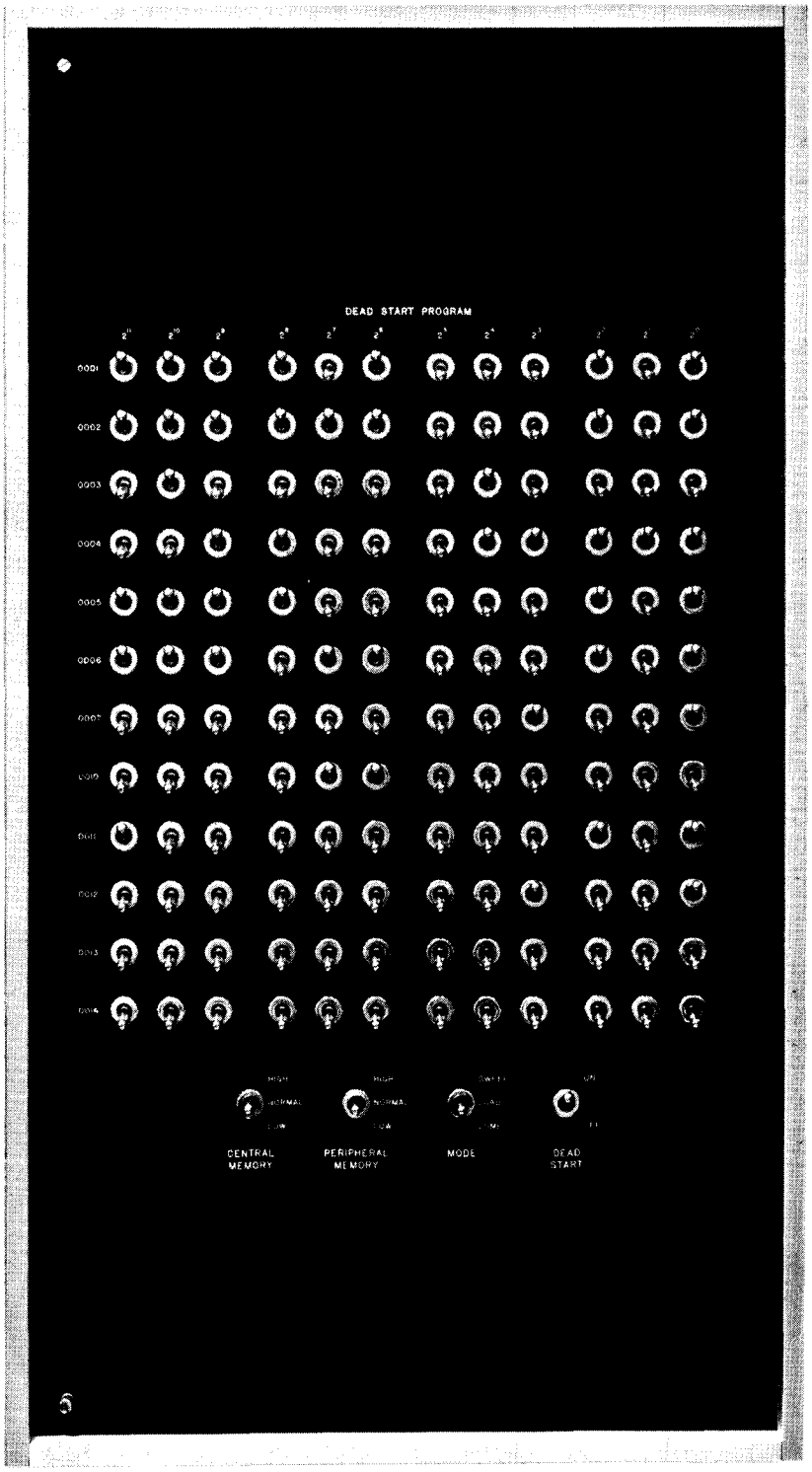
The translation of the 50X instruction in each processor causes each processor to sweep through its memory, reading and restoring the contents of each location, without executing instructions. Sweep mode is a maintenance tool useful in checking the operation of memory logic.

Dump Mode

Placing the MODE switch in the DUMP position and momentarily turning on the DEAD START switch initiates the following operations:

1) Assigns processors $0\text{-}11_8$ to corresponding data channels.
2) Sends a Master Clear to all I/O channels except channel 0.
3) Holds channel 0 to Active and Empty.
4) Sets all processors to the Output (73) instruction.
5) Clears the A and P registers in all processors to zero.

Each of the processors senses the Active and Empty condition of its assigned channel and outputs the content of its memory address zero. Each of the I/O channels is then set to Full (except channel 0), and the processors wait for an Empty signal. Each processor advances its P register by one and reduces the content of its A register by one



1593

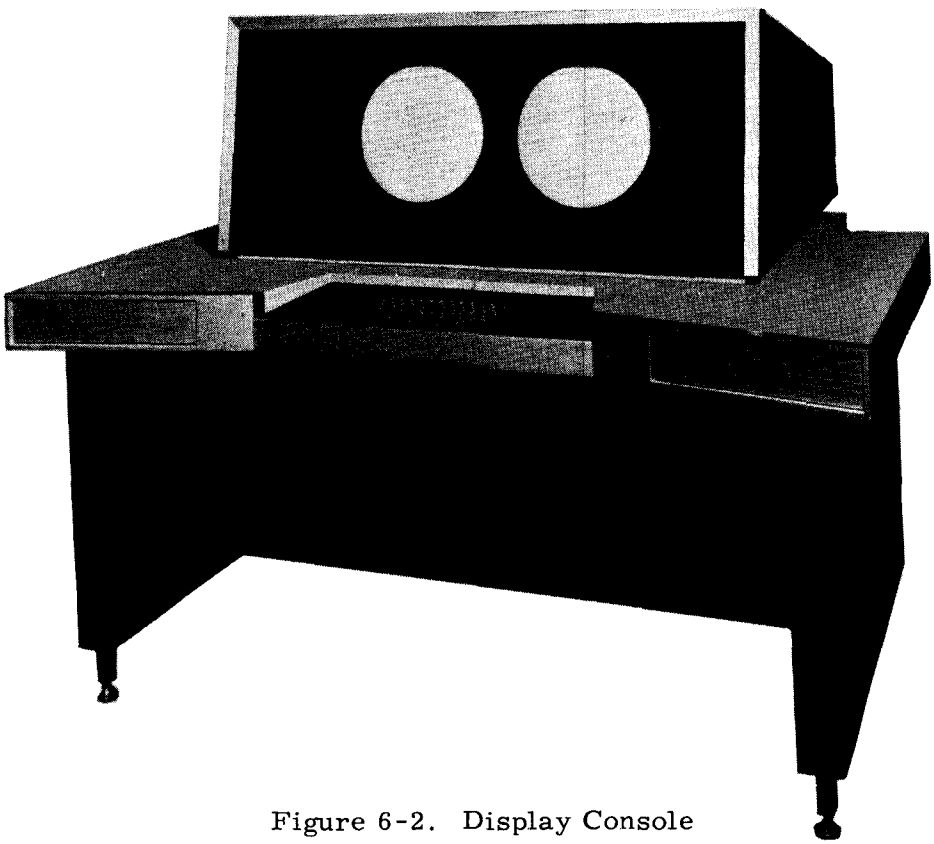
Figure 6-1. Dead Start Panel

(to 7776_8). At this point, the processors waiting for an Empty signal are hung up and cannot proceed.

Channel 0 (assigned to processor 0) is held to Empty by the DUMP position. Processor 0, therefore, proceeds through the 73 instruction until the contents of A are reduced to one. Processor 0 has now dumped its entire memory content on channel 0 (although no I/O device was selected to receive it). Processor 0 then exits to memory location 0001 for its next instruction; it is now free to execute a dump program which must have been previously stored in its memory (beginning at location 0001).

CONSOLE

The display console (Figure 6-2) consists of two cathode ray tube displays and a keyboard for manual entry of data. A typical 6400/6500/6600 Computer System may have several display consoles for controlling independent programs simultaneously.



1371

Figure 6-2. Display Console

Keyboard Input

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

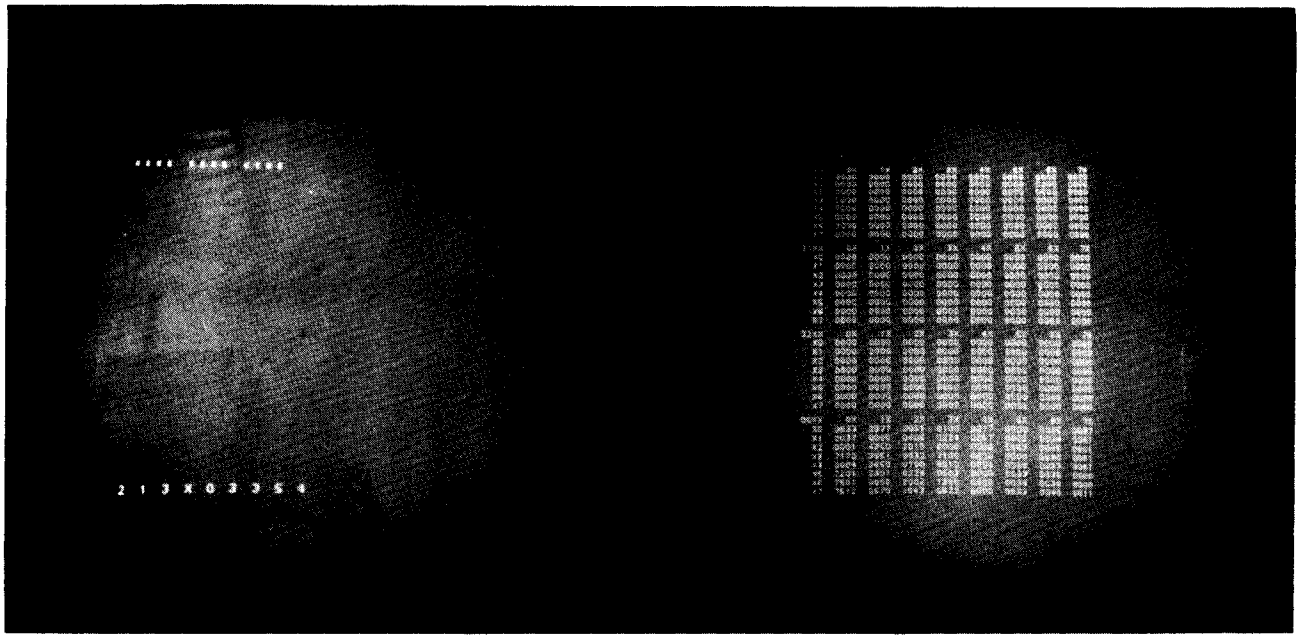
Display

The console may be selected to display (Figure 6-3) in either the Character or Dot mode. In the Character mode, two alphanumeric characters may be displayed for each 12-bit word sent from a processor. Character sizes are:

Small	-	64 characters/line
Medium	-	32 characters/line
Large	-	16 characters/line

In Dot mode, a pattern of dots (graph, figures, etc.) may be displayed. Each dot is located by two 12-bit words: a vertical coordinate and a horizontal coordinate.

A display program must repeat a display periodically in order to maintain persistence on the display tube.



1586

Figure 6-3. Sample Display

Appendix A

AUGMENTED I/O BUFFER AND CONTROL (6416)

CONTROL DATA 6416

AUGMENTED I/O BUFFER AND CONTROL

The CONTROL DATA 6416 Augmented I/O Buffer and Control unit is a large-scale, solid state device for communication with the Central Processor of 6400, 6500, and 6600 Computer Systems.

DESCRIPTION

The 6416 is comprised of ten Peripheral and Control Processors and a Central Memory. A summary of characteristics for the 6416 is tabulated below.

PERIPHERAL AND CONTROL PROCESSORS

- 10 identical processors
  Each processor has a 4096 word magnetic core memory (12-bit)
  Random access, coincident current
  Major cycle = 1000 ns; Minor cycle = 100 ns

- 12 input/output channels
  All channels common to all processors
  Maximum transfer rate per channel - one word/major cycle
  All channels may be active simultaneously
  All channels 12-bit bi-directional

- Real-time clock (period = 4096 major cycles)
- Instructions
  - Logical
  - Branch
  - Add/Subtract
  - Input/Output
  - Central Memory Access
  - Extended Core Storage Access

- Average instruction execution time = two major cycles
- Indirect addressing
- Indexed addressing

CENTRAL MEMORY

- 16,384 words (60-bit)
- Memory organized into four logically independent banks of 4096 words with corresponding multiphasing of banks
- Random-access, coincident-current, magnetic core
- One major cycle for read-write
- Maximum memory reference rate to all banks; four addresses/major cycle
- Maximum rate of data flow to/from memory; four words/major cycle

The 6416 has no Central Processor; otherwise, it is identical to the 6400, 6500, and 6600 Computer Systems. The following discussion assumes use of the 6416 in a typical 6400 or 6600 system; the 6416 can also be used in a 6500 system. Furthermore, it is a computer capable of operating alone.

SYSTEMS CONFIGURATIONS

The 6416, in typical systems configurations, provides an extremely useful and powerful system expansion. For installations with multiple on-line users, the 6416 provides additional data channels facilitating additional external equipments. The ten Peripheral and Control Processors, each capable of independently executing programs, and the 16,384 word 60-bit Central Memory significantly increase the multiprogramming and batch job processing capabilities of the 6400, 6500, and 6600 Computer Systems.

A typical configuration diagrammed in Figure A-1 illustrates the orientation of a 6416 with a 6400 or 6600 Computer System. The 6416 is attached to the 6400 or 6600 system via one of the Peripheral Processor Data Channels.

The 6682/6683 Satellite Coupler accepts and relays control signals and data to provide smooth information flow throughout the system.

In this configuration, the 6416 may be thought of as a batching terminal, where batch jobs may enter the system, be assembled, and placed in the 16K distributive memory. Access to the 6400 or 6600 Central Processor for job execution is then under operating system control.

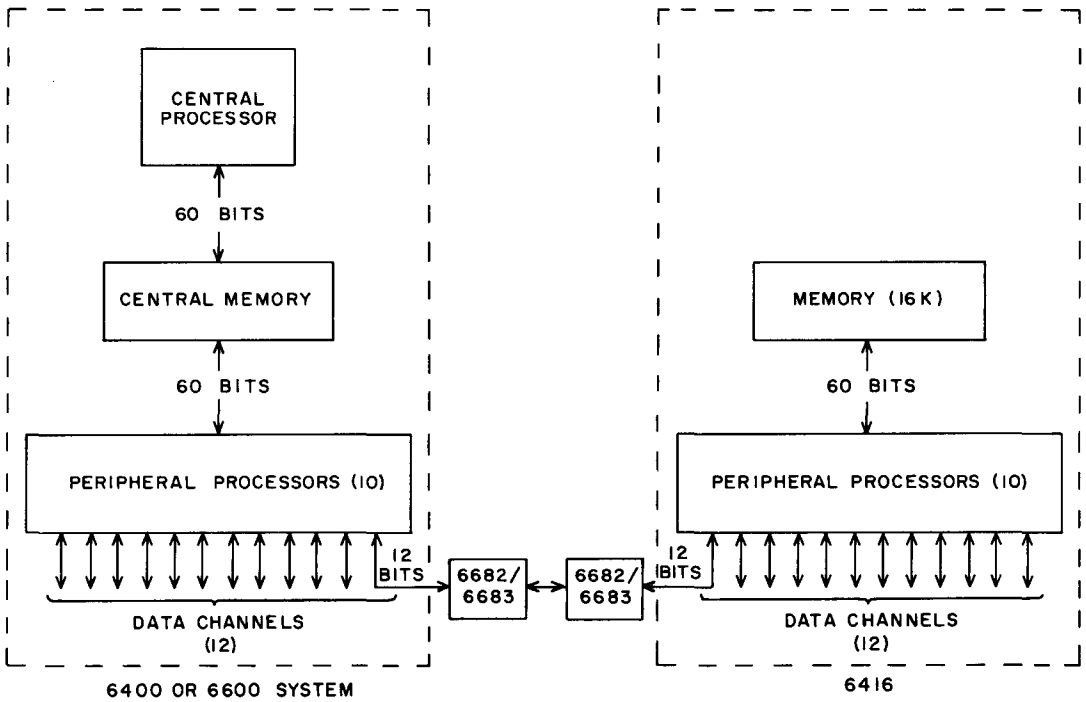


Figure A-1. Typical Configuration: 6416 with 6400 or 6600 System

Another possible systems configuration (Figure A-2) incorporates Extended Core Storage between the 6400 or 6600 Central Memory and the 6416 16K memory. This configuration implies a hierarchy of memories as follows:

1) Extended Core Storage as a system Central Memory
2) 6400 or 6600 Central Memory as a system Central Processor memory
3) 6416 16K memory as a distributive memory

Communication with Extended Core Storage (Figure A-2) is accomplished as follows:

1) Read and Write instructions in the 6400, 6500, and 6600 Central Processors initiate transfers between Extended Core Storage and Central Memory.

2) An Exchange Jump instruction in the 6416 Peripheral Processor initiates Read and Write operations between Extended Core Storage and the 6416 16K memory. (Refer to the instruction descriptions which follow.)

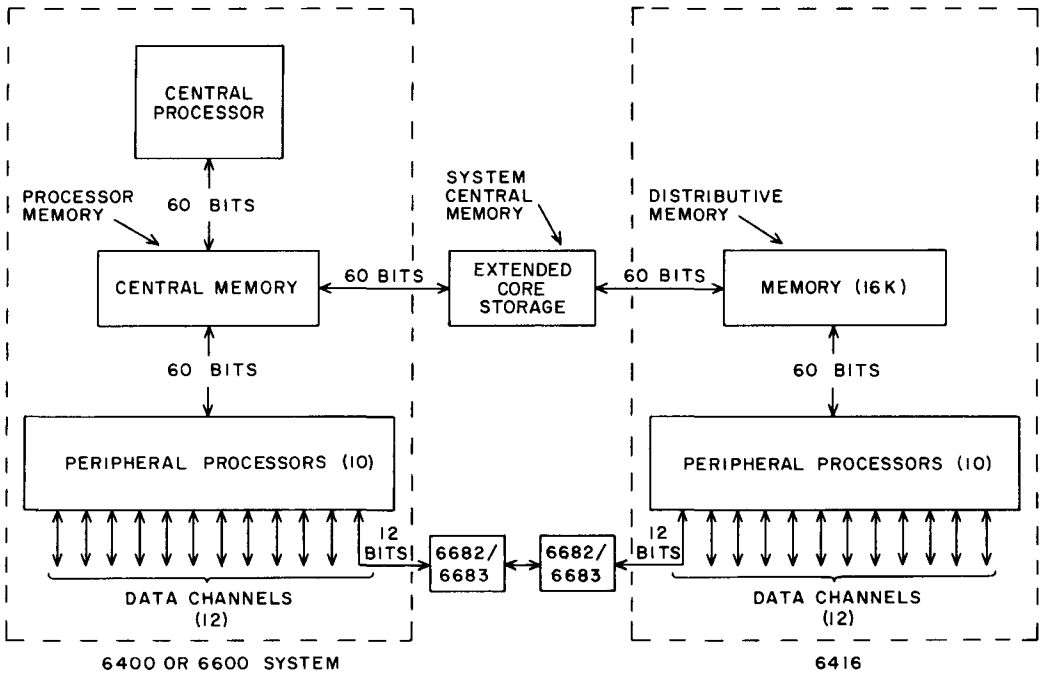
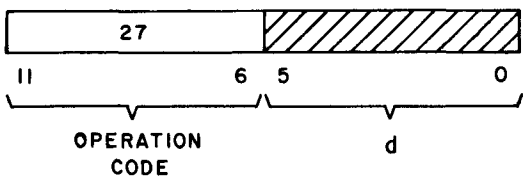


Figure A-2. Typical Configuration with Extended Core Storage

6416 INSTRUCTIONS

Within the 6416, Peripheral Processor instructions are identical to those of the 6400, 6500, and 6600 systems with two exceptions. Note that these two instructions (the exceptions) are meaningful only when Extended Core Storage is attached to the system.

27 RCS d Read Extended Core Coupler Status (12 bits)



This instruction reads the 6416 Extended Core Coupler status and places these status bits in the upper-order three bits of the Peripheral and Control Processor A register. The significance of these status bits (when set to "1") is as follows:

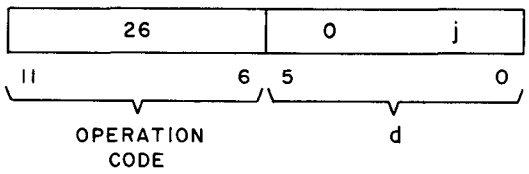
Bit 17 Extended Core Storage transfer is in progress.

Bit 16 Parity error(s) occurred during the last Read Extended Core Storage operation.

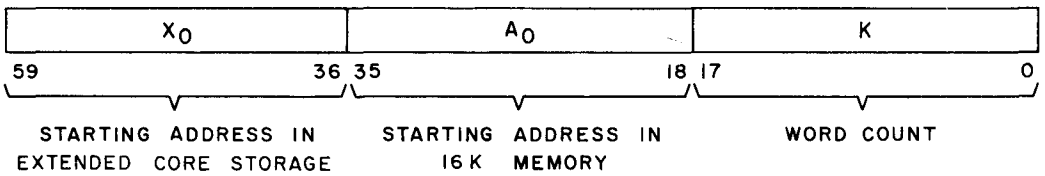
Bit 15 At least one address of the last Extended Core Storage transfer was not available (power off, in maintenance mode, address not in system).

Within the Extended Core Coupler, status bit 17 is dynamic; bits 16 and 15 are cleared each time an Extended Core Storage transfer is initiated.

26 ECT d Extended Core Transfer (12 bits)



Execution of the Extended Core Transfer instruction initiates memory operations by transmitting an 18-bit address, "n", from the Peripheral Processor A register to the 6416 16K memory. Address "n" holds a word, the format of which is as follows:



The "d" portion of this instruction specifies the storage operation to be performed:

If "j" = 0, Read "K" words from Extended Core Storage into 16K memory.
If "j" = 1, Write "K" words from 16K memory into Extended Core Storage.

NOTE

If this instruction is executed without Extended Core Storage in the system configuration, it acts as a Pass (Do-Nothing) instruction.

Note that addresses contained in the word at address "n" are absolute addresses. Operating systems may require relocation (adding RA to an address) and Field Length testing, e.g., is "address + RA" $\geq$ FL? (The Exchange Jump package contains RA and FL values for Central Memory and for Extended Core Storage.) The 6416 has no hardware for automatic relocation and Field Length testing; it is therefore incumbent upon the program to perform these functions whenever required by an operating system.

Appendix B

INSTRUCTION EXECUTION TIMES

INSTRUCTION EXECUTION TIMES

The execution times for Central and Peripheral and Control Processor instructions are given in the following paragraphs. Factors which influence instruction execution time and hence program running time are also given.

CENTRAL PROCESSOR (6600 SYSTEM)

The execution time of Central Processor instructions is given in minor cycles, and instructions are grouped under the functional unit (6600) which executes the instruction. Time is counted from the time the unit has both input operands to when the instruction result is available in the specified result register. Central Memory access time is not considered in those increment instructions which result in memory references to read operands or store results.

The following paragraphs give some general statements about Central Processor instruction execution and summarize the statements into a list which may be used as a guide to efficient use of the Central Processor functional units.

Central Processor programs are written in the conventional manner and are stored in Central Memory under direction of a Peripheral and Control Processor. After an Exchange Jump start by a Peripheral and Control Processor program, Central Processor instructions are sent automatically, and in the original sequence, to the instruction stack, which holds up to 32 instructions.

Instructions are read from the stack one at a time and issued to the functional units for execution. A scoreboard reservation system in Central Processor control keeps a current log of which units are busy (reserved) and which operating registers are reserved for results of computation in functional units.

Each unit executes several instructions, but only one at a time. Some branch instructions require two units, but the second unit receives its direction from the branch unit.

The instruction issue rate may vary from a theoretical maximum rate of one instruction every minor cycle (sustained issuing at this rate may not be possible because of unit and Central Memory conflict) and resulting parallel operation of many units to a slow issue rate and serial operation of units. The latter results when successive operations depend on results of previous steps. Thus, program running time can be decreased by efficient use of the many units. Instructions which are not dependent on previous steps may be arranged or nested in areas of the program where they may be executed during operation time of other units. Effectively, this eliminates dead spots in the program and steps up the instruction issue rate.

The following steps summarize instruction issuing and execution:

1) An instruction is issued to a functional unit when
   - the specified functional unit is not reserved
   - the specified result register is not reserved for a previous result.

2) Instructions are issued to functional units at minor cycle intervals when no reservation conflicts (see above) are present.

3) Instruction execution starts in a functional unit when both operands are available (execution is delayed when an operand(s) is a result of a previous step which is not complete.

4) No delay occurs between the end of a first unit and the start of a second unit which is waiting for the results of the first.

5) No instructions are issued after a Branch instruction until the Branch instruction has been executed. The Branch Unit uses
   - an Increment Unit to form the go to k + Bi and go to k if Bi . . . instructions, or
   - the Long Add unit to perform the go to k if Xj . . . instructions

   in the execution of a Branch instruction. The time spent in the Long Add or Increment Units is part of the total branch time.

6) Read Central Memory access time is computed from the end of Increment Unit time to the time operand is available in X operand register. Minimum time is 500 ns, assuming no Central Memory bank conflict.

CENTRAL PROCESSOR (6400 AND 6500 SYSTEMS)

Central Processors in the 6400 and 6500 systems have unified Arithmetic units, rather

than separate functional units as in the 6600 system. Instructions in these Central Processors, therefore, are executed in sequential fashion with little concurrency.

All execution times for instructions listed in Table B-1 include readying the next instruction for execution. For the Return Jump instruction and the Jump instructions (in which the jump condition is met), Table B-1 lists times which include obtaining the new instruction word from storage and readying it for execution. Times listed, then, are complete times except for possible additional time due to hardware limitations or memory bank conflicts. Factors which may add to the stated times in Table B-1 are summarized below:

1) Reading the next instruction word of a program from Central Memory (termed an RNI - Read Next Instruction) is in part concurrent with instruction execution. The RNI is initiated between execution of the first and second instructions of the instruction word being processed. Initiating the RNI operation requires 2 minor cycles; the remainder of the RNI time is in time parallel with the execution of the remaining instructions in the instruction word. (Refer to Figure B-1.)

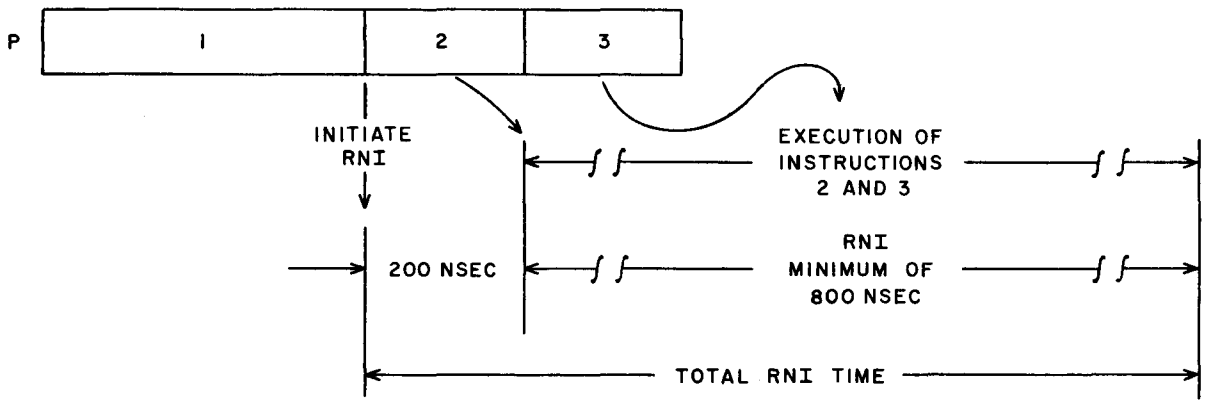


Figure B-1. RNI Timing Example

In the example diagrammed in Figure B-1, execution of instruction 2 is delayed 2 minor cycles until RNI initiation is complete.

In calculating execution times for a program, add 2 minor cycles to each instruction word in a program to cover the RNI initiation time. Exceptions to

this rule are the Return Jump and the Jump instructions (in which the jump condition is met) when these occupy the upper position of the instruction word. Since the stated times for these instructions in Table B-1 include the time required to read up the new instruction word at the jump address, no additional time is required.

Example:

P	JUMP TO K (MET)		PASS	PASS
K	ADD 1	ADD 2	LOAD	STORE

Instruction	Time Required
Jump	13 Minor Cycles
Add 1	5 Minor Cycles
RNI Initiation	2 Minor Cycles
Add 2	5 Minor Cycles
Load	12 Minor Cycles
Store	10 Minor Cycles
Total Time Required =	47 Minor Cycles

2) After RNI has been initiated (between the first and second instructions of the instruction word), a minimum of 8 minor cycles elapse before the next instruction word is available for execution. If the total time required by instructions in the lower order positions of the word is less than 8 minor cycles, allow a minimum of 8 minor cycles, regardless of the execution times stated in Table B-1.

Example:

P	JUMP TO K (NOT MET)	PASS	PASS
(P)+ 1			

Instruction		Time Required
Jump (not met)		5 Minor Cycles
RNI Initiation		2 Minor Cycles
Pass = 3 Pass = 3 } 6, but RNI Minimum	=	8 Minor Cycles
Minimum time before instruction word at P + 1 is available for execution	=	15 Minor Cycles

3) The Return Jump instruction, all Jump instructions in which the jump condition is met, and Load/Store Memory instructions always require additional time when located in the second instruction position of an instruction word. This additional time is caused by hardware limitations and is not due to memory bank conflicts.

Instruction	Additional Time Required If Used As Second Instruction in Word
a) Jumps (02 - 07) in which the jump condition is met	1 Minor Cycle
b) Return Jump (010)	2 Minor Cycles
c) Load/Store (5X instructions with i $\neq$ 0)	2 Minor Cycles

4) An additional 3 minor cycles due to bank conflict are required if the second instruction of a word references the memory bank in which (P)+1 is located.

5) A Store (not Load) as the first instruction of a word can cause a bank conflict with (P)+1. If this occurs, 3 minor cycles are added to the execution time.

Summary of guidelines for efficient coding in the 6400 and 6500 Central Processors:

- Always attempt to place Jump instructions in the upper parcel of the instruction word. In most cases, this avoids both the additional time for RNI (2 minor cycles) and the possibility of a memory bank conflict with (P) + 1.
- Where possible, place Load/Store instructions in the lower order two parcels to avoid lengthening execution times as outlined above.

CENTRAL PROCESSOR INSTRUCTION EXECUTION TIMES

Central Processor instruction execution times for the 6400, 6500, and 6600 systems are tabulated in Table B-1 (6500 times are for each Central Processor). Instructions are tabulated according to the functional units in which they are executed; this functional unit designation, of course, does not apply to the 6400 and 6500 systems. Their Central Processors have unified arithmetic sections. Instruction execution times are listed in minor cycles.

TABLE B-1. INSTRUCTION EXECUTION TIMES: CENTRAL PROCESSOR

Octal Code	BRANCH UNIT	6400 6500	6600
00	STOP	-	-
010	RETURN JUMP to K	21	13
011	READ EXTENDED CORE STORAGE	**	**
012	WRITE EXTENDED CORE STORAGE	**	**
02	GO TO K + Bi †	13	14
030	GO TO K if Xj = zero	13	9*
031	GO TO K if Xj ≠ zero	13	9*
032	GO TO K if Xj = positive	13	9*
033	GO TO K if Xj = negative	13	9*
034	GO TO K if Xj is in range	13	9*
035	GO TO K if Xj is out of range	*** 13	9*
036	GO TO K if Xj is definite	13	9*
037	GO TO K if Xj is indefinite	13	9*
04	GO TO K if Bi = Bj †	13	8*
05	GO TO K if Bi = Bj †	13	8*
06	GO TO K if Bi ≥ Bj †	13	8*
07	GO TO K if Bi < Bj †	13	8*

† GO TO K + Bi and GO TO K if Bi - - - tests made in Increment Unit

†† GO TO K if Xj - - - tests made in Long Add Unit

*Add 6 minor cycles to branch time for a branch to an instruction which is out of the stack (no memory conflict considered); add 2 minor cycles to branch time for a no branch condition in the stack. Add 5 minor cycles to branch time for a no branch condition out of the stack.

**Execution times for Extended Core Storage operations are dependent upon several factors; refer to Extended Core Storage literature for timing information.

***Jumps in which the jump condition is not met require 5 minor cycles.

TABLE B-1. (Cont'd)

Octal Code	BOOLEAN UNIT	6400 6500	6600
10	TRANSMIT Xj to Xi	5	3
11	LOGICAL PRODUCT of Xj and Xk to Xi	5	3
12	LOGICAL SUM of Xj and Xk to Xi	5	3
13	LOGICAL DIFFERENCE of Xj and Xk to Xi	5	3
14	TRANSMIT Xk COMP. to Xi*	5	3
15	LOGICAL PRODUCT of Xj and Xk COMP. to Xi	5	3
16	LOGICAL SUM of Xj and Xk COMP. to Xi	5	3
17	LOGICAL DIFFERENCE of Xj and Xk COMP. to Xi	5	3
Octal Code	SHIFT UNIT	6400 6500	6600
20	SHIFT Xi LEFT jk places	6	3
21	SHIFT Xi RIGHT jk places	6	3
22	SHIFT Xk NOMINALLY LEFT Bj places to Xi	6	3
23	SHIFT Xk NOMINALLY RIGHT Bj places to Xi	6	3
24	NORMALIZE Xk in Xi and Bj	7	4
25	ROUND AND NORMALIZE Xk in Xi and Bj	7	4
26	UNPACK Xk to Xi and Bj	7	3
27	PACK Xi from Xk and Bj	7	3
43	FORM jk MASK in Xi	6	3
Octal Code	ADD UNIT	6400 6500	6600
30	FLOATING SUM of Xj and Xk to Xi	11	4
31	FLOATING DIFFERENCE of Xj and Xk to Xi	11	4
32	FLOATING DP SUM of Xj and Xk to Xi*	11	4
33	FLOATING DP DIFFERENCE of Xj and Xk to Xi	11	4
34	ROUND FLOATING SUM of Xj and Xk to Xi	11	4
35	ROUND FLOATING DIFFERENCE of Xj and Xk to Xi	11	4
Octal Code	LONG ADD UNIT	6400 6500	6600
36	INTEGER SUM of Xj and Xk to Xi	6	3
37	INTEGER DIFFERENCE of Xj and Xk to Xi	6	3
Octal Code	MULTIPLY UNIT**	6400 6500	6600
40	FLOATING PRODUCT of Xj and Xk to Xi	57	10
41	ROUND FLOATING PRODUCT of Xj and Xk to Xi	57	10
42	FLOATING DP PRODUCT of Xj and Xk to Xi	57	10

*Comp. = Complement; DP = Double Precision
**Duplexed units - instruction goes to free unit

TABLE B-1. (Cont'd)

Octal Code	DIVIDE UNIT	6400 6500	6600
44	FLOATING DIVIDE Xj by Xk to Xi	57	29
45	ROUND FLOATING DIVIDE Xj by Xk to Xi	57	29
47	SUM of 1's in Xk to Xi	68	8
46	PASS	3	1

Octal Code	INCREMENT UNIT*	6400 6500	6600
50	SUM of Aj and K to Ai	**	3
51	SUM of Bj and K to Ai	**	3
52	SUM of Xj and K to Ai	**	3
53	SUM of Xj and Bk to Ai	**	3
54	SUM of Aj and Bk to Ai	**	3
55	DIFFERENCE of Aj and Bk to Ai	**	3
56	SUM of Bj and Bk to Ai	**	3
57	DIFFERENCE of Bj and Bk to Ai	**	3
60	SUM of Aj and K to Bi	5	3
61	SUM of Bj and K to Bi	5	3
62	SUM of Xj and K to Bi	5	3
63	SUM of Xj and Bk to Bi	5	3
64	SUM of Aj and Bk to Bi	5	3
65	DIFFERENCE of Aj and Bk to Bi	5	3
66	SUM of Bj and Bk to Bi	5	3
67	DIFFERENCE of Bj and Bk to Bi	5	3
70	SUM of Aj and K to Xi	6	3
71	SUM of Bj and K to Xi	6	3
72	SUM of Xj and K to Xi	6	3
73	SUM of Xj and Bk to Xi	6	3
74	SUM of Aj and Bk to Xi	6	3
75	DIFFERENCE of Aj and Bk to Xi	6	3
76	SUM of Bj and Bk to Xi	6	3
77	DIFFERENCE of Bj and Bk to Xi	6	3

* Duplexed units - instruction goes to free unit

** When: i = 0 the execution time is 6 minor cycles
i = 1-5 the execution time is 12 minor cycles
i = 6 or 7 the execution time is 10 minor cycles

6600 CENTRAL PROCESSOR TIMING NOTES

1. The times given in Table B-1 are computational times - the time needed after the execution start until the result is computed and ready to be stored into the result register.

2. The functional units are not freed until one minor cycle after the result has been stored into the result register.

3. A result register value may be used as an operand to another instruction as soon as the result has been stored into the register (same minor cycle). This result register will not be freed to be used as a result register of another instruction until one cycle after the result has been stored into that register (no trunk priority considered).

4. An instruction is issued to a functional unit if:

   a) The word containing the instruction is in the stack and the U registers,
   b) The functional unit(s) needed are free, and
   c) The result register(s) needed are free (note Table B-2 and B-3).

   If these three conditions are not met, all further instruction issues are held until they are satisfied. Each issued 15-bit instruction requires one minor cycle before the next instruction is available for issue. Each issued 30-bit instruction requires two minor cycles before the next instruction is available for issue.

5. Execution within a functional unit does not start until the operands are available (note Table B-3). The two operands required are fetched from the registers at the same time (one operand is not loaded while the unit waits for a second operand).

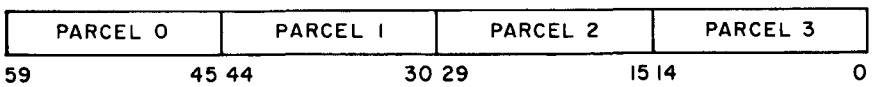
6. In instructions 02-07, where more than one functional unit is used, the instruction is not issued until both functional units involved are free.

7. Times given for instructions 01-07 and 50-57 do not consider any memory conflict conditions.

8. In instructions 50-57, if i = 1, 2...5 (load from memory instructions), the Xi register value is not available until 8 minor cycles after the start of the instruction execution (assuming no memory conflicts). When two load instructions begin execution one minor cycle apart, one extra minor cycle is required for execution of the later instruction. Therefore, the second executed instruction would require 9 cycles for the load, 5 cycles for the Increment Unit, and 4 cycles for the A register.

9. In instructions 50-57, if i = 6 or 7 (store to memory instructions), the Xi register is not available for a result register until 10 minor cycles after the instruction begins execution (assuming no memory conflicts). When two store instructions begin execution one minor cycle apart, one extra minor cycle is required for execution of the later instruction. Therefore, the second executed instruction would require 11 cycles for the store, 5 cycles for the Increment Unit, and 4 cycles for the A register.

10. When executing sequential instructions that are not in the stack, the minimum time is one word of instructions every 8 cycles. The time of issue of the last parcel of an instruction word to the time of issue of the first parcel of the next instruction word (while executing sequential instructions that are not in the stack) requires a minimum of 4 cycles. If the last instruction in an instruction word is a 30-bit instruction, a minimum of 5 cycles are required from the time of issue to a functional unit of this instruction to the time of issue of the first instruction in the next word. An instruction word is parcelled as diagrammed below:



11. When a branch out of the stack is taken, 15 minor cycles are normally required for a 03ijk instruction and 14 minor cycles are normally required for other branch instructions (considering no memory conflict). The latter timing is from the start of branch instruction execution to the point when the instruction at the branch address is ready for issue to a functional unit.

12. Nine cycles are required for 03ijk instructions when the branch is taken within the stack. The next sequential word is recognized as within the stack.

13. Eight cycles are required for 04ijk to 07ijk instructions when the branch is taken within the stack. The next sequential word is recognized as within the stack.

14. Eleven cycles are required for 03ijk instructions when the branch is not taken (time from branch execution to issue of next instruction) if in the stack or if falling through to the same word. Out of the stack fall-through to the next word takes 14 cycles.

15. Ten cycles are required for 04ijk to 07ijk instructions when the branch is not taken (time from branch execution to issue of next instruction) if in the stack or if falling through to the same word. Out of the stack fall-through to the next word takes 13 cycles.

16. The B0 register is handled as any other Bi register for timing purposes (i.e., B0 will hold up execution of an instruction if it is a result register of a previous non-completed instruction, etc.).

17. Neither Increment Unit may be involved in a load operation if a store operation is to be issued, and neither Increment Unit may be involved in a store operation if a load operation is to be issued. The sequential loading of instruction words does not affect the load/store conditions of the Increment Units. Increments of A0 are considered neither loads nor stores.

18. The operand registers are available to more than one functional unit in the same minor cycles if the units are in different groups.

Group 1	Group 2	Group 3
Boolean	Shift	Increment 1
Divide	Floating Add	Increment 2
Multiply 1	Long Add	
Multiply 2		

19. The time needed for a functional unit to operate on indefinite, out-of-range, or zero values is the same as for normal, in-range values (i.e., no gain or loss in execution time due to a unit recognizing an indefinite operand and setting an indefinite result).

20. An Index Jump instruction (02) will always destroy the stack. If an unconditional jump back in the stack is desired, a 0400K instruction may be used (to save memory access time for instructions).

21. A Return Jump instruction (01) will always destroy the stack.

22. After a result has been computed by a functional unit, the result register is checked to see if it is still needed as an operand register for a previously issued instruction. This is done so that a result will not overlay an operand to a previously issued instruction. If a unit (#1) is waiting for an operand to be fetched by another unit (#2) before storing its result, for timing considerations,

    a) The result register is available to a third unit (#3) as an operand, the cycle following the fetch, and
    b) The unit (#1) is freed two cycles following the fetch.

23. In cases of bank conflict, unaccepted addresses get a chance at access every three minor cycles. If the address can then be accessed, the memory operation proceeds. If the bank is still busy, the address circulates in the hopper, while access is permitted for any other source requesting access.

TABLE B-2. FUNCTIONAL UNIT DATA TRUNK ASSIGNMENTS AND PRIORITY

FUNCTIONAL UNIT		RESULT (i)		OPERAND (j)		OPERAND (k)	
		Trunk	Priority	Trunk	Priority	Trunk	Priority
Group 1:	Shift	3 (X), 4 (B) *	1	1	2	2	2
	Add	3	2	1	1	2	1
	Long Add	3	3	1	3	2	3
Group 2:	Boolean	7	1	5	4	6	4
	Divide	7	2	5	1	6	1
	Multiply 1	7	3	5	2	6	2
	Multiply 2	7	4	5	3	6	3
Group 3:	Increment 1	10	1	8	1	9	1
	Increment 2	10	2	8	2	9	2

*The Shift Unit is sometimes required to store two results at one time: one into an X register and one into a B register.

TABLE B-3. 6600 REGISTER RESERVATION CONTROL

INSTRUCTION	XBA RESULT REGISTER (ISSUE)	Q OPERAND REGISTER (EXECUTION)
Branch Unit		
02ijK	-	Bi & Bj
03ijK	-	Xi & Xj
04ijK	-	Bi & Bj
Boolean Unit		
10ijk - 17ijk	Xi	Xj & Xk
Shift Unit		
20ijk - 23ijk	Xi	Bj & Xk
24ijk - 26ijk	Xi & Bj	Bj & Xk
27ijk & 43ijk	Xi	Bj & Xk
Add Unit (Floating)		
30ijk - 35ijk	Xi	Xj & Xk
Long Add (Integer)		
36ijk - 37ijk	Xi	Xj & Xk
Multiply (2 Units)		
40ijk - 42ijk	Xi	Xj & Xk
Divide Unit		
44ijk - 47ijk	Xi	Xj & Xk
Increment (2 Units)		
50ijK	Ai & Xi *	Aj & Bk **
51ijK	Ai & Xi *	Bj & Bk **
52ijK	Ai & Xi *	Xj & Bk **
53ijk	Ai & Xi *	Xj & Bk
54ijk & 55ijk	Ai & Xi *	Aj & Bk
56ijk & 57ijk	Ai & Xi *	Bj & Bk
60ijK	Bi	Aj & Bk **
61ijK	Bi	Bj & Bk **
62ijK	Bi	Xj & Bk **
63ijk	Bi	Xj & Bk
64ijk & 65ijk	Bi	Aj & Bk
66ijk & 67ijk	Bi	Bj & Bk
70ijK	Xi	Aj & Bk **
71ijK	Xi	Bj & Bk **
72ijK	Xi	Xj & Bk **
73ijk	Xi	Xj & Bk
74ijk & 75ijk	Xi	Aj & Bk
76ijk & 77ijk	Xi	Bj & Bk

* The Xi register is considered only when i = 1, 2...7.
** k here refers to the high order 3 bits of 18-bit address field.

PERIPHERAL AND CONTROL PROCESSOR

The execution time of Peripheral and Control Processor instructions is influenced by the following factors:

- Number of memory references - indirect addressing and indexed addressing require an extra memory reference. Instructions in 24-bit format require an extra reference to read m.
- Number of words to be transferred - in I/O instructions and in references to Central Memory the execution times vary with the number of words to be transferred. The maximum theoretical rate of flow is one word/major cycle. I/O word rates depend upon the speed of external equipments which are normally much slower than the computer.
- References to Central Memory may be delayed if there is conflict with Central Processor memory requests.
- Following an Exchange Jump instruction, no memory references (nor other Exchange Jump instructions) may be made until the Central Processor has completed the Exchange Jump.

TABLE B-4.. PERIPHERAL AND CONTROL PROCESSOR INSTRUCTION EXECUTION TIMES

OCTAL CODE	NAME	TIME* (MAJOR CYCLES)
00	Pass	1
01	Long jump to m + (d)	2-3
02	Return jump to m + (d)	3-4
03	Unconditional jump d	1
04	Zero jump d	1
05	Nonzero jump d	1
06	Plus jump d	1
07	Minus jump d	1
10	Shift d	1
11	Logical difference d	1
12	Logical product d	1
13	Selective clear d	1
14	Load d	1

*Note that the shorter time is taken in certain instructions when d = 0.

TABLE B-4. (Cont'd)

OCTAL CODE	NAME	TIME* (MAJOR CYCLES)
15	Load complement d	1
16	Add d	1
17	Subtract d	1
20	Load dm	2
21	Add dm	2
22	Logical product dm	2
23	Logical difference dm	2
24	Pass	1
25	Pass	1
260	Exchange jump	1**
27	Read program address	1
30	Load (d)	2
31	Add (d)	2
32	Subtract (d)	2
33	Logical difference (d)	2
34	Store (d)	2
35	Replace add (d)	3
36	Replace add one (d)	3
37	Replace subtract one (d)	3
40	Load ((d))	3
41	Add ((d))	3
42	Subtract ((d))	3
43	Logical difference ((d))	3
44	Store ((d))	3
45	Replace add ((d))	4
46	Replace add one ((d))	4
47	Replace subtract one ((d))	4
50	Load (m + ((d))	3-4
51	Add (m + (d))	3-4
52	Subtract (m + (d))	3-4
53	Logical difference (m + (d))	3-4
54	Store (m + (d))	3-4

*Note that the shorter time is taken in certain instructions when d = 0.

**Though the execution time for this instruction in the Peripheral and Control Processor is only 1 major cycle, a minimum of 2 major cycles is required to complete the Exchange operation in Central Memory. Thus, Central Memory honors no requests for access for a minimum of 2 major cycles during an Exchange Jump.

TABLE B-4. (Cont'd)

OCTAL CODE	NAME	TIME* (MAJOR CYCLES)
55	Replace add (m + (d))	4-5
56	Replace add one (m + (d))	4-5
57	Replace subtract one (m + (d))	4-5
60	Central read from (A) to d	min. 6
61	Central read (d) words from (A) to m	5 plus 5/word
62	Central write to (A) from d	min. 6
63	Central write (d) words to (A) from m	5 plus 5/word
64	Jump to m if channel d active	2
65	Jump to m if channel d inactive	2
66	Jump to m if channel d full	2
67	Jump to m if channel d empty	2
70	Input to A from channel d	2
71	Input (A) words to m from channel d	4 plus 1/word
72	Output from A on channel d	2
73	Output (A) words from m on channel d	4 plus 1/word
74	Activate channel d	2
75	Disconnect channel d	2
76	Function (A) on channel d	2
77	Function m on channel d	2

*Note that the shorter time is taken in certain instructions when d = 0.

Appendix C

NON-STANDARD FLOATING POINT ARITHMETIC

NON-STANDARD FLOATING POINT ARITHMETIC

The following is a tabulation of operations (Add, Subtract, Multiply, Divide) using various combinations of operands to supplement Table 3-3 (page 3-13). The key to operands and results used in the table is as follows:

KEY:

OPERANDS			RESULTS		
+0	=	0000 X...X	0	=	0000 0...0
-0	=	7777 X...X	IND	=	1777 0...0
+ ∞	=	3777 X...X	+ ∞	=	3777 0...0
- ∞	=	4000 X...X	- ∞	=	4000 0...0
+IND	=	1777 X...X			
-IND	=	6000 X...X			
W	=	Any word except ± ∞ , ±IND			
N	=	Any word except ± ∞ , ±IND, or ±0			

ADD

Xi=Xj+Xk

(Instructions 30, 32, 34)

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

SUBTRACT

Xi=Xj-Xk

(Instructions 31, 33, 35)

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

MULTIPLY

Xi=Xj*Xk

(Instructions 40, 41, 42)

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

DIVIDE

Xi=Xj/Xk

(Instructions 44, 45)

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

Appendix D

EXTENDED CORE STORAGE

EXTENDED CORE STORAGE

This appendix describes characteristics of Extended Core Storage for the 6400, 6500, and 6600 systems and the 6416.

SUMMARY OF CHARACTERISTICS

The following summary lists characteristics of an Extended Core Storage (ECS) configuration.

- Bounds protection and relocation capabilities for ECS
- 125,952 60-bit words per bank (minimum available size)
- Optional sizes available: 1, 2, 4, 8, and 16-bank configurations (maximum available size is 2,015,232 60-bit words)
- Memory organized in logically independent banks of 488-bit words (eight 60-bit words plus parity bit for each) with corresponding multiphasing of banks
- Random access, word-oriented, magnetic core
- Approximately 3.2 microsecond cycle time (read-write time for 488-bit word)
- Approximately 1.86 microseconds for access to first 60-bit word
- Four access channels (60-bit) for communication with up to four 6400, 6500, 6600, or 6416 systems.
- Scanning mechanism services all channels equally; scan occurs after each record
- Assembly/Disassembly (60-bit words into 480-bit word plus 8-bit parity and vice versa)
- Parity bit generated for each 60-bit word; parity check on Read operations

DESCRIPTION

An Extended Core Storage configuration for a 6400, 6500, 6600, or 6416 basically involves three logical elements: Extended Core Storage, Extended Core Controller, and Extended Core Coupler. These logical elements are shown in Figure D-1.

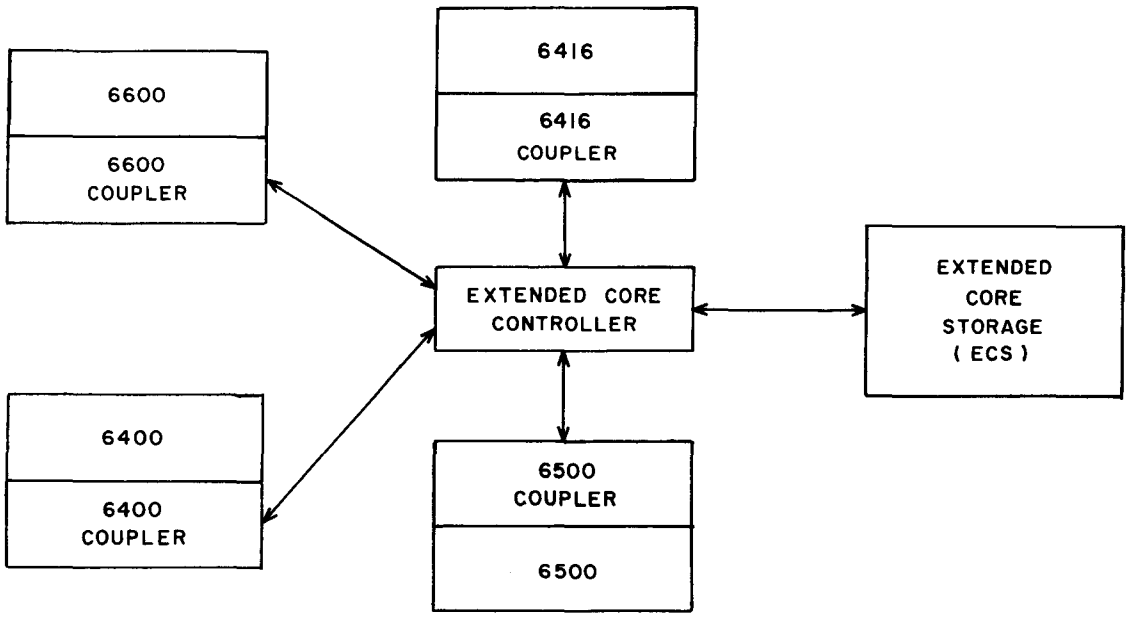


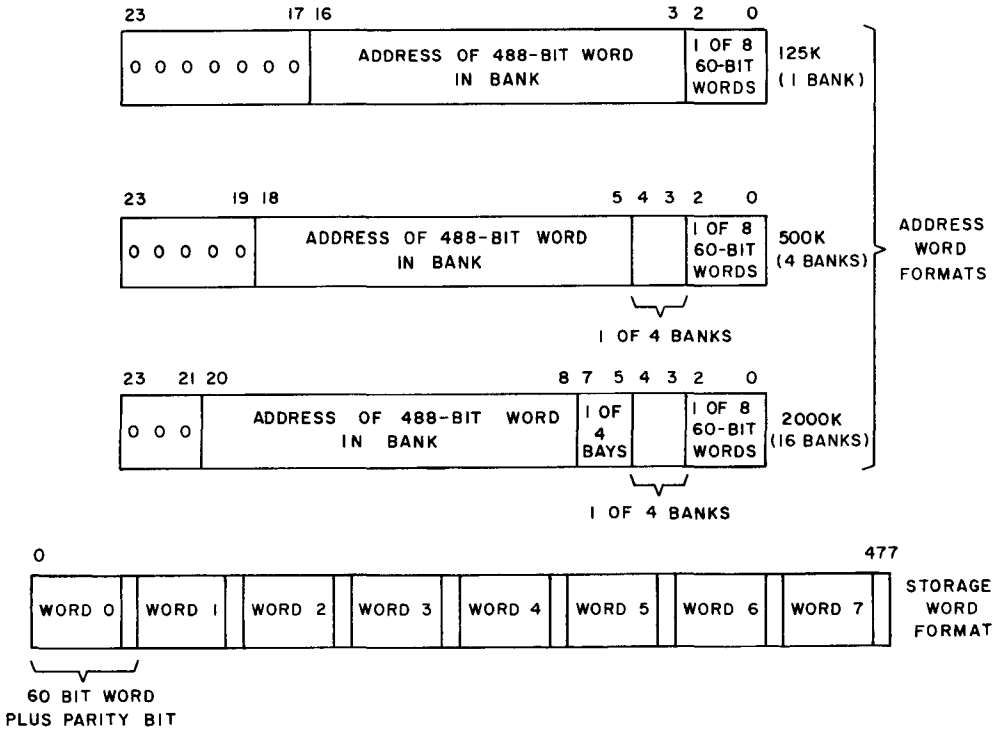
Figure D-1. Typical Extended Core Storage Configuration

EXTENDED CORE STORAGE

The Extended Core Storage unit provides up to two million directly addressable 60-bit words. Eight 60-bit words are organized into a 488-bit data word in ECS. A parity bit is attached to each 60-bit word in the controller on a Write ECS. Extended Core Storage (ECS) is organized into banks of 125,952 60-bit words per bank.

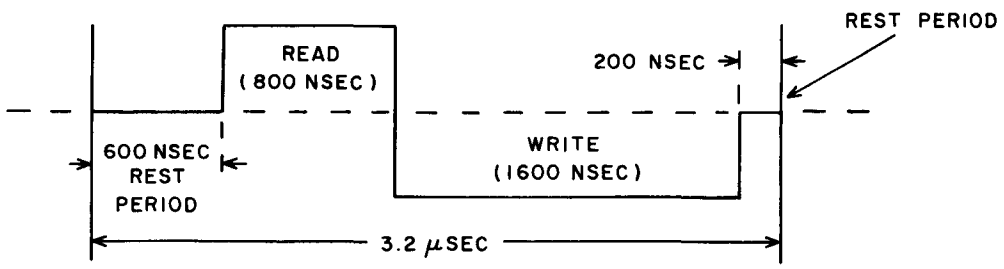
The minimum available ECS is a bank of 125,952 60-bit words. Expanding ECS to four banks provides a bay containing 503,808 60-bit words. Four bays provide the maximum available ECS capability - 2,015,232 60-bit words. Within this range of minimum to maximum (125K to 2000K), ECS is available in 1, 2, 4, 8, and 16-bank configurations.

Addressing a particular word in ECS is accomplished by transmitting a 24-bit address word to ECS. Read and Write instructions which initiate ECS communication are described in the Order of Instructions section for the Central Processor and in Appendix A. Successive 488-bit words are in different banks to permit bank phasing. Typical address word formats and an ECS data word format are diagrammed below:



An assembly/disassembly network in ECS assembles eight 60-bit words (plus eight parity bits) into a 488-bit word for Write operations. On Read operations, this network disassembles eight 60-bit words and their associated parity bits from the 488-bit word read from ECS. Each bank has an assembly/disassembly network.

Each storage bank has a Read/Write cycle time of 3.2 microseconds per 488-bit word selected. This storage cycle time is as diagrammed below:



EXTENDED CORE CONTROLLER

The Extended Core Controller provides four bidirectional access channels to read or write 60-bit data words, a scanning mechanism to service the requests of these channels, a parity generator and checker, and the associated control necessary to regulate these operations.

Access Channels

Bidirectional access channels on the controller provide the paths for data and control signals between ECS and the coupler. To permit access to ECS by other systems, a total of four access channels are provided. Data trunks in the access channels are 60-bits in length.

Data transfer (for block transfers) is accomplished in groups of eight words or less, called records. Single 60-bit word transfers can also be effected. Near the end of a record, the controller scans the other access channels for memory requests. If another channel is requesting access to ECS, that channel is serviced. If other channels are transferring data, each channel is serviced on a record basis. Thus, there may be time gaps between records on a given access channel.

Since ECS can handle 60-bit words at 100-nanosecond intervals for a complete block transfer, some restriction is placed on possible transfer rates with system elements having either 16K (4 bank) or 32K (8 bank) Central Memories. Since data can neither be sent nor received by the coupler or ECS at rates greater than the 16K or 32K Central Memories can handle, the couplers provide the control for proper transfer rates in these cases.

For a 6416 with a 16K Central Memory (4 banks), the maximum rate of data flow occurs in a 4-reinitiate ECS-4.....pattern (send 4 words, one to each of the 4 banks, reinitiate ECS to obtain the second half of the 488-bit word, send those 4 data words, etc.). This pattern is referred to as a type A transfer.

A similar operation occurs (a type B transfer) for a computer system with a 32K (8-bank) Central Memory. The maximum rate of data flow occurs in an 8-2-8-2.... pattern (send 8 data words, wait 200 nanoseconds, send 8 data words, etc.).

A type C transfer requires a Central Memory size of 65K (16 banks) or 131K (32 banks). This transfer type has a maximum data rate capability of one 60-bit word per 100 nanoseconds until the block transfer is completed.

The ECS Controller performs much the same function for ECS that Central Memory control does for Central Memory. The controller holds in its service registers the Requests, the ECS address, and the Store bit (Read or Write operation) from each of the four access channels.

If more than one request arrives at one time, the requests are scanned to prevent a request on a given channel from being locked out by other requests.

Parity Generator/Checker

For each 60-bit word to be stored in ECS, a parity bit is generated and stored along with that word (odd parity). Parity is checked on each 60-bit word as the storage word is disassembled after a Read operation. If a parity error occurs, a signal is sent to the coupler.

EXTENDED CORE COUPLER

In response to a Read or Write ECS instruction, the Extended Core Coupler performs the following operations:

- Receives the initial ECS address and relays this address and a request signal to the controller.
- Receives the word count, [(Bj) + K]. The coupler compares the number of words read (or written) with the word count to insure transferring the proper number of words.
- Keeps count of the words transferred. Increments ECS address once each eight-word record.
- Generates an End of Transfer signal when the transfer is completed.
- Sends a Go signal to Central Memory for every word to be read from Central Memory. (Central Memory control increments the Central Memory address during the transfer.)
- Regulates data transfer rate for a 16K or a 32K Central Memory which cannot give a sustained transfer of one word every 100 nanoseconds.
- Sets a "1" in bit 2^{17} of the Peripheral Processor A register to indicate an Extended Core Storage transfer is in progress.

DATA TRANSFER TIMING

Although the block length of Central Memory to ECS (and vice versa) transfers is limited only by the respective field lengths (FL_{CM} and FL_{ECS}), the actual transfer is accomplished in records. Near the end of a record, Central Memory control and the controller examine their inputs for memory requests. If any memory requests are present in either Central Memory or on some other controller access channel, they are honored before the next record is transferred. If no other requests are present, the transfer continues on that channel at the maximum rate.

Several variables exist in a typical ECS configuration which makes an attempt to state transfer times difficult. Several factors which influence transfer times are:

- The number of banks in Central Memory
- The number of banks in ECS
- Use of the bank phasing feature in addressing
- Conflicts in Central Memory and ECS
- First-word access time

From the foregoing, it is evident that any presentation of timing information is best accomplished with a specific configuration in mind as well as some knowledge of the use of the configuration, i.e., degree of overlapping operations. The times listed in Table D-1 are based on the following assumptions:

- Times are listed for continuous streaming of data after first-word access. (Continuous means uninterrupted except where waits are introduced to permit a bank to complete its storage cycle.)
- ECS is comprised of at least four banks (503K).
- Bank phasing is used in addressing.
- No other requests occur for Central Memory access.
- No conflicts are presented at the ECS access.

TABLE D-1. TYPICAL TRANSFER TIMES

NUMBER OF CENTRAL MEMORY BANKS	TRANSFER TIMES
Four	8 Words/2.0 usec
Eight	8 Words/1.0 usec
Sixteen or Thirty-Two	10 Words/1.0 usec

INDEX

COMMENT SHEET

CONTROL DATA 6400/6500/6600 COMPUTER SYSTEMS
Reference Manual
Pub. No. 60100000

FROM: NAME: _____

BUSINESS
ADDRESS: _____

THESE COMMENTS REFER TO REV. _____ OF THIS MANUAL.

COMMENTS: (DESCRIBE ERRORS, SUGGESTED ADDITIONS OR DELETIONS, ETC. INCLUDE PAGE NUMBER.)

CUT ALONG LINE

FORM CA 231 REV. 1-66

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

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
8100 34TH AVENUE SOUTH
MINNEAPOLIS 20, MINNESOTA

ATTN: TECHNICAL PUBLICATIONS DEPT.
COMPUTER DIVISION
PLANT TWO

FOLD

FOLD

CUT ALONG LINE

STAPLE

STAPLE

INDEX TO PERIPHERAL AND CONTROL PROCESSOR INSTRUCTIONS

NUMERICAL

ALPHABETICAL

CONTROL DATA
CORPORATION

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

Pub. No. 60100000

Litho in U.S.A.