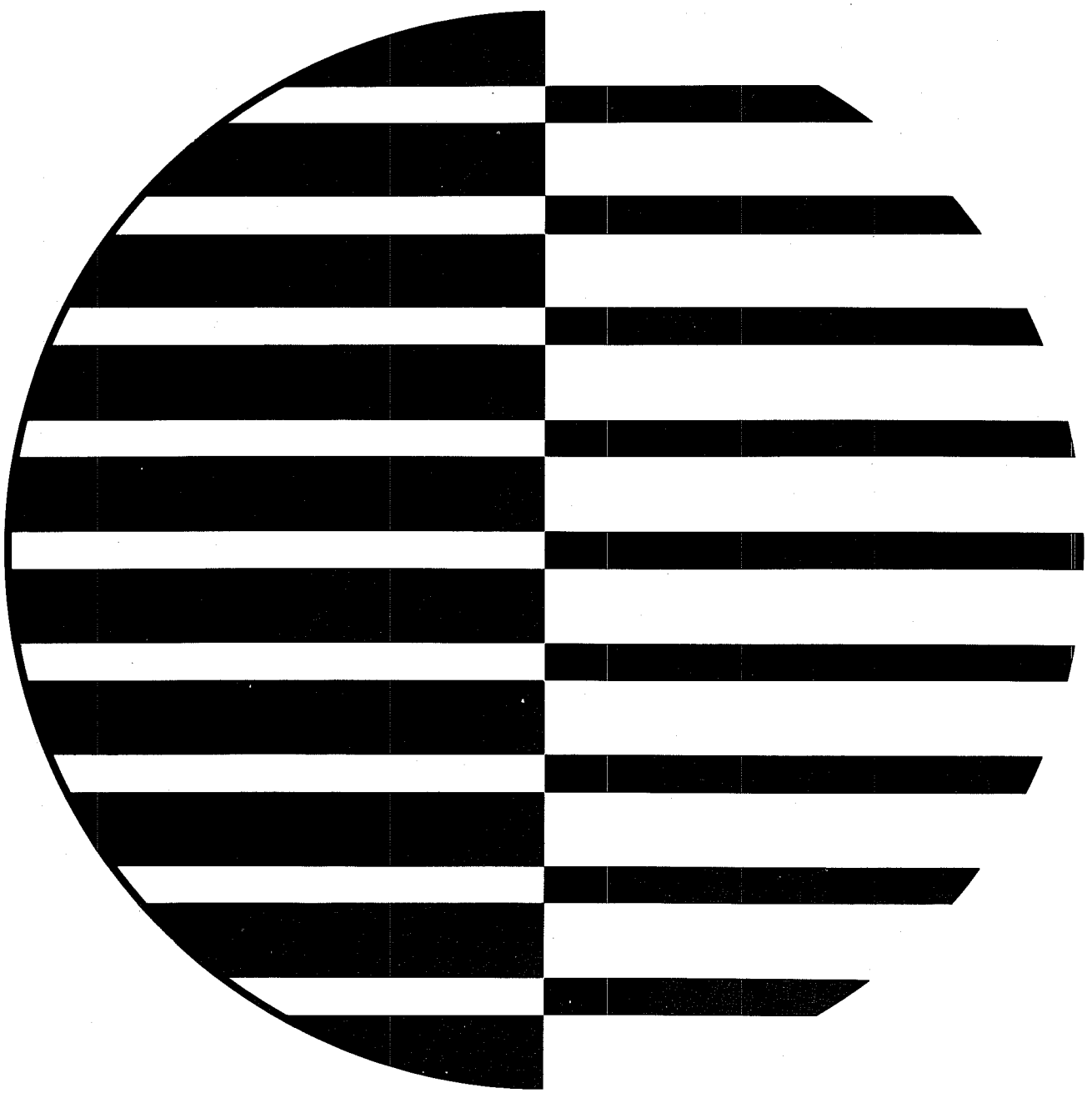
CONTROL DATA® 6000 SERIES COMPUTER SYSTEMS

Reference Manual



60100000	**Record of Revisions**
REVISION	NOTES
	This manual obsoletes the 6600 Computer System
	Reference Manual, Pub. No. 60045000.

Pub. No. 60100000
July, 1965

Address comments concerning this manual to:

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

or use Comment Sheet in back of this manual.

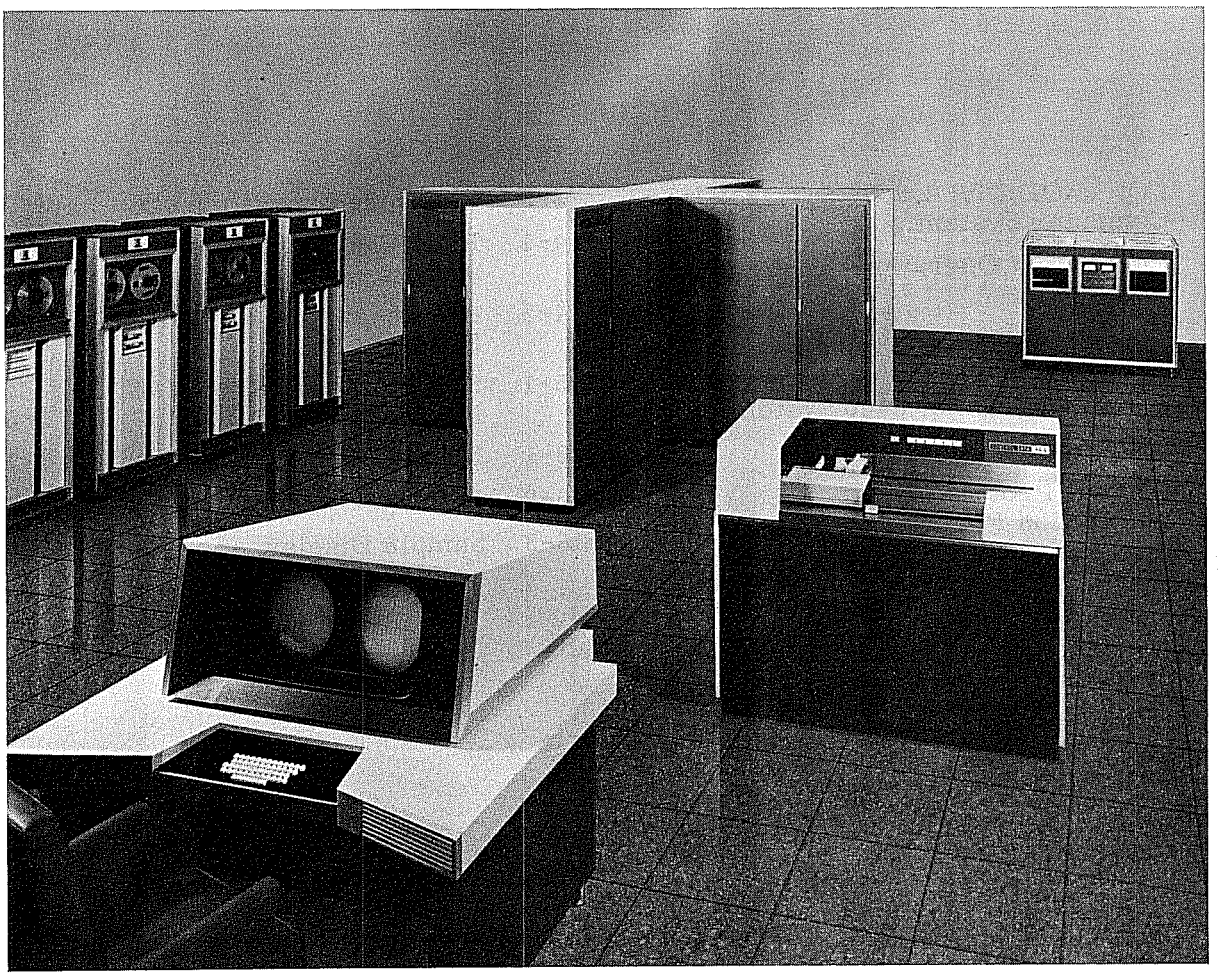
CONTENTS

5. System Interrupt

6. Manual Control

FIGURES

TABLES



1610

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.

Main frame (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; main frames for the 6400 or 6800 Systems vary in physical appearance, depending on options included in the system.

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* 6000 Series consists of three large-scale, solid-state, general-purpose digital computing systems: the 6400, 6600, and 6800. 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 in the 6000 Series 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.

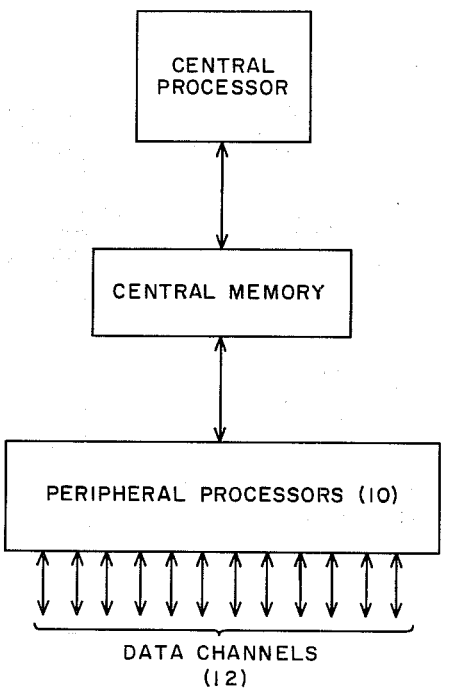


Figure 1-1. CONTROL DATA 6000 Series Computer System

*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 of more Peripheral and Control Processors are used for high speed information transfer in and out of the system and to provide operator control. If the problem requires significant arithmetic speed, the Peripheral and Control Processors may call on the Central Processor. A number of problems may operate concurrently by time-sharing with 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.

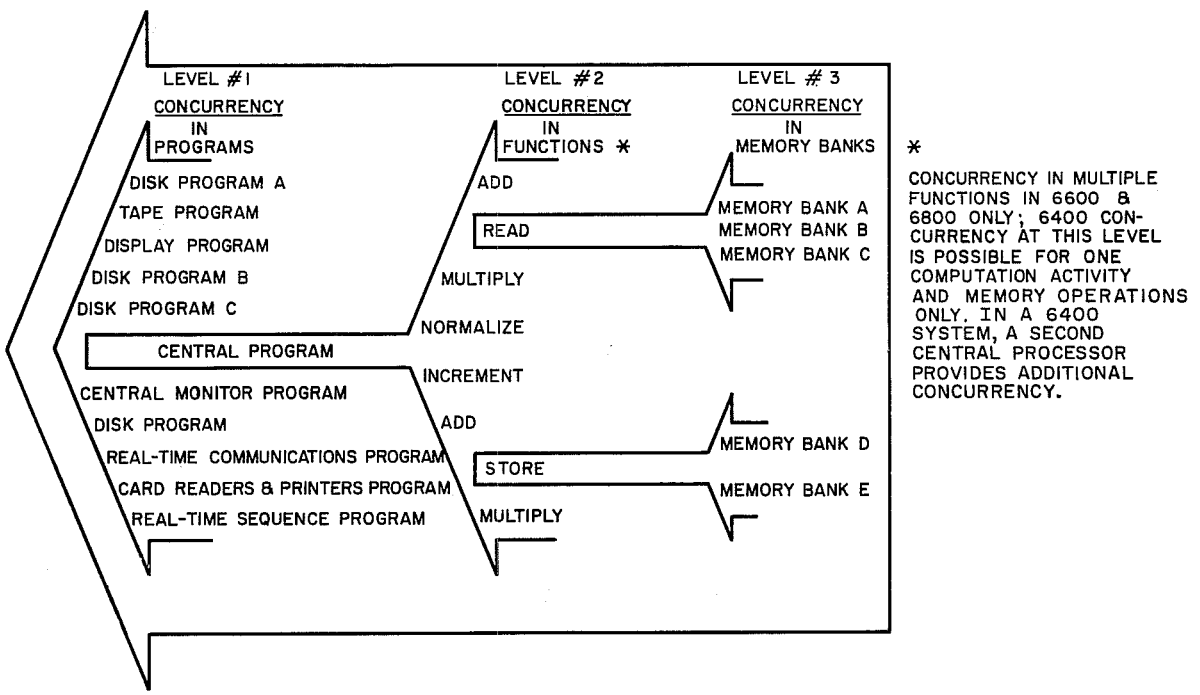


Figure 1-2. Concurrent Operations in the 6000 Series

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 6000 Series are listed below. (Refer to the 6000 Series Peripheral Equipment Reference Manual for additional external equipment information.)

- CONTROL DATA 6602 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 alpha-numeric 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 Satellite Coupler: a systems expansion device which permits direct connection between any two 6000 Series systems via two standard 12-bit bi-directional data channels; two 6682's are required for this.

- CONTROL DATA 6681 Data Channel Converter: a device which permits 6000 Series 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 6000 Series computer systems. Where characteristics differ within the systems comprising the 6000 Series, differences are noted. Otherwise, systems characteristics listed are common to the Series.

System Characteristics

- Large-scale, general-purpose computer system
- 11 independent computers; 12 in a dual Central Processor 6400 system

  1 Central Processor (60-bit); an optional second Central Processor is available in 6400 systems

  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)

		6400 & 6600	6800
Major Cycle	=	1000 ns*	250 ns
Minor Cycle	=	100 ns	25 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 & 6800

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 combination of two for servicing functional units.

6400

- Unified arithmetic section, operating in sequential manner
- 24 operating registers
  - 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

		6400 & 6600	6800
Major Cycle	=	1000 ns	250 ns
Minor Cycle	=	100 ns	25 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)

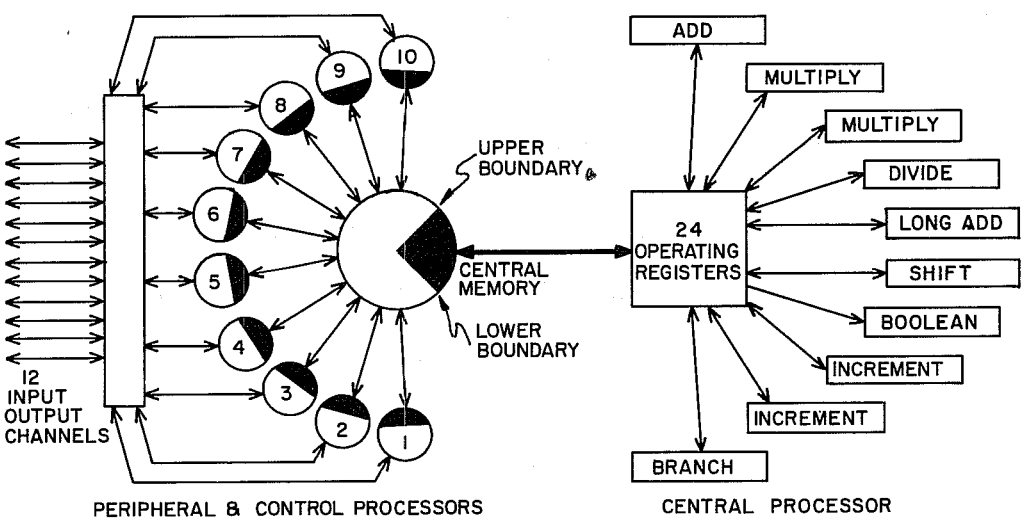


Figure 1-3. Block Diagram of 6600 and 6800 Systems

- 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

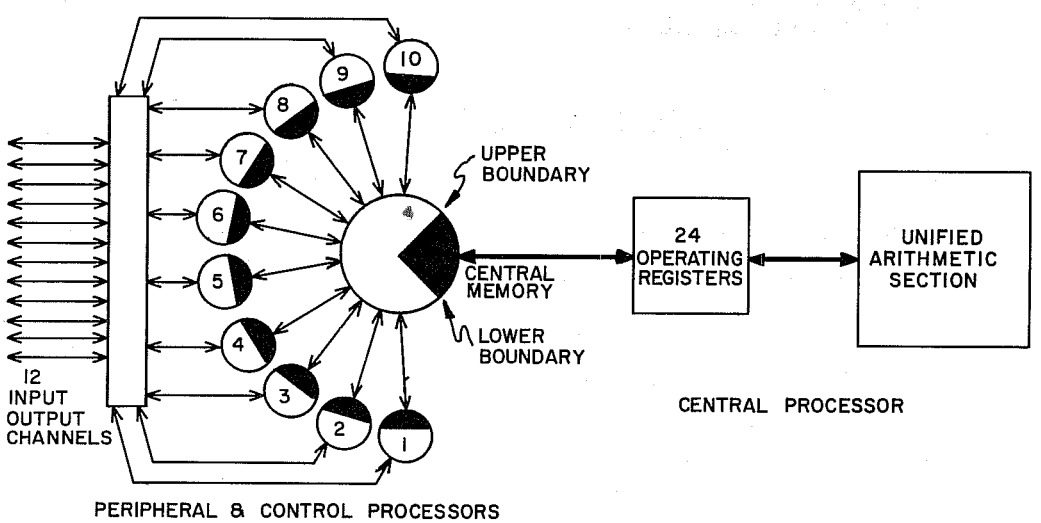


Figure 1-4. Block Diagram of 6400 System

SYSTEMS OPTIONS

The foregoing summary of characteristics assumed a standard 6000 Series system with 10 Peripheral and Control Processors, a Central Processor, and Central Memory with 131,072 words (60-bit) of magnetic core storage.

Options listed below are available within each system in the 6000 Series unless otherwise noted.

- Central Memory with 65,536 words (60-bit) of magnetic core storage.
- Central Memory with 32,768 words (60-bit) of magnetic core storage.
- Extended Core Storage: Magnetic core storage available in the following sizes:

  131,072 words (60-bit)

  262,144 words (60-bit)

  524,288 words (60-bit)*

  1,048,576 words (60-bit)*

  2,097,152 words (60-bit)
- Augmented I/O Buffer and Control (6400 and 6600 systems only): includes 16,384 words (60-bit) of magnetic core storage and 10 Peripheral and Control Processors with storage.
- Additional Central Processor (6400 system only): includes arithmetic and control functions of the 6400 system Central Processor. One such processor may be added to a 6400 system to provide increased processing capability.

SYSTEMS SOFTWARE

The basic programming package for the CONTROL DATA 6000 Series Computer Systems provides a comprehensive operational system which includes:

1) A universal operating system
2) A FORTRAN compiler, compatible with FORTRAN 63
3) An assembly system for the Central Processor
4) An assembly system for the Peripheral Processors
5) A library system including mathematical function subroutines, input/output routines, and utility programs.

* Only sizes available for 6800 systems; all sizes are available for 6400 and 6600 systems.

These programming systems have been given abbreviated names derived from the initials of the descriptions and will be referred to hereafter by the following titles:

SIPROS	Simultaneous Processing Operating System
FORTRAN 66	FORTRAN for the 6000 Series
ASCENT	Assembly System, Central Processor
ASPER	Assembly System, Peripheral Processors
LIBRIOUS	Library System of I/O and Utility Systems

Following are brief descriptions of the basic 6000 Series software systems. For more definitive information of these systems, consult the individual software systems manuals.

SIPROS

The Simultaneous Processing Operating System (SIPROS) is the most vital part of the 6000 Series software. Its design has been carefully chosen to provide a universal system to fit the wide range of applications for the 6000 Series. The system fully utilizes the concurrencies possible in the 6000 Series and assumes that the using installation has, at any one time, a multiplicity of jobs to run.

The major objective of the system is the handling, with a minimum of operator intervention, of a dynamic situation in which many jobs such as I/O operations, computations, compilations, and debugging operations are proceeding simultaneously. At the same time, a maximum of choice is left to the using installation on the selection of certain features and options. Operator manual intervention and override capabilities are provided which allow simple insertion of new jobs, change in priorities and equipment assignments, etc., without disrupting operations. Full use of the operating console display and keyboard is made for status display and two-way communication with the operator.

Some of the more important features of SIPROS are:

For the users:

- Ease of use for the programmer and operator
- Flexible provisions for operator intervention and override
- Maximum use of console display and keyboard for two-way communications
- Provision of a comprehensive accounting system for event and time-use recording for Central and Peripheral Processors and I/O equipment.

As a dynamic multi-processing system:

- Provision for automatic batch processing
- Multi-processing capabilities using a flexible priority scheme
- Dynamic assignment and release of I/O equipment
- Ability to handle I/O requests through the system
- Provision of a queuing system to handle data for relatively slow I/O devices
- Dynamic memory re-assignment during multiple operations

From a systems-programming point of view:

- Maximum exploitation of 6000 Series systems memory protection features
- Integration of a diagnostic system under operating system control
- Universal adaptability to special executives for real-time or hybrid applications
- Orientation around use of a disk file, but not restricted or limited by the actual configuration

FORTRAN 66

FORTRAN 66 is a compiler system adapting current techniques to the particular capabilities of the 6000 Series computer systems hardware. It is written to operate independently on the 6000 Series systems, or under control of SIPROS.

The FORTRAN language used is completely compatible with FORTRAN 63 but provides additional features to fully exploit 6000 Series characteristics.

Implementation of FORTRAN 66 places heavy emphasis on efficiency, both in compilation and in object code produced. Special emphasis is placed on algorithms and numerical techniques which best exploit the word size and instruction repertoire of the Central Processor.

Some of the features of FORTRAN 66 are:

- Language includes FORTRAN 63 and hence is compatible with both FORTRAN 63 and, essentially, with FORTRAN IV.
- Mixed assembly language and FORTRAN statements are allowed.
- Provision is made for automatic I/O buffering.

- Register names can be used as operands.
- Object code is optimized by a variety of 6000 Series hardware-oriented techniques.
- Special numerical methods are used for certain algorithms.

ASCENT

The Central Processor Assembler (ASCENT) is a symbolic machine-oriented language which permits direct access to all features of the hardware, but which allows the use of various functions provided by the operating system. Machine language coding can be generated very rapidly by using the system features for control of I/O, subroutine call, and macros; yet the programmer is not restricted from using any feature of the Central Processor. Principal features of ASCENT are:

- Use of all I/O functions in the operating system.
- Macro system: use of system macros and programmer-defined macros.
- Ability to mix with FORTRAN language.
- Subroutine call
- Pseudo commands
- Peripheral Processor program call
- Access to all features of the Central Processor

ASPER

Normally, the Peripheral Processors are not programmed in the course of accomplishing day-to-day work. They are generally assigned the functions of the operating system and all input/output work. However, situations may arise which require very precise machine-language programming of the Peripheral Processors to obtain the desired results. In this event, a symbolic assembly system is provided for the Peripheral Processors. ASPER includes standard features such as subroutine call, pseudo commands, access to all features of the Peripheral Processors, a macro system, and access to all I/O functions provided in the operating system.

LIBRIOUS

The Library System of I/O and Utility Systems (LIBRIOUS) contains a set of programs available to the user via the programming languages. The library system has the capabilities necessary for updating the library store of routines. This store is composed of

routines used frequently enough to be kept on-line to the operating system, yet not frequently enough to be contained in the resident operating system. Initially, the library routines are of the following types:

1) I/O Routines

   System I/O Routines are provided to handle the requirements for communications between the Central and Peripheral Processors and all input/output devices. They include routines such as:

   - Disk or tape to Central Memory
   - Disk or tape to a particular Peripheral Processor memory
   - Central or Peripheral Processor memory to tape or disk

2) Utility Routines

   Utility routines for general operations are provided in the library. These routines are arranged so that they may be externally called and used by an operator through the operating system without interrupting other processing. The routines may also be called and executed by other programs as part of a total job. These routines provide such operations as:

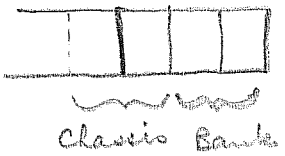
   - Card-to-tape, tape-to-cards in various modes
   - Card-to-disk and disk-to-cards with or without conversion
   - Disk-to-printer, cards-to-printer, tape-to-printer

3) Mathematical Subroutines

   The library contains a complete set of standard FORTRAN mathematical subroutines which have been specially developed to take advantage of the 6000 Series capabilities.

Additional Software Packages

To complement the basic programming package described in the foregoing, Control Data provides many comprehensive programming packages upon request, including Algol, Cobol-61 Extended, and Sort/Merge. Information on available software systems may be obtained from any Control Data sales representative.



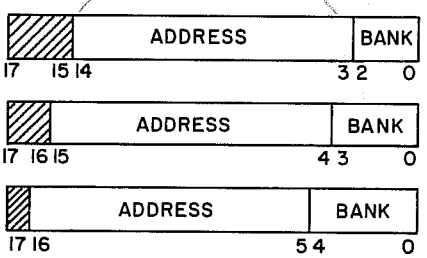
2. CENTRAL MEMORY

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 in 6400 and 6600 systems; 25 ns in 6800 system.

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-

Reference address = Relative add = Relocation Add.

dress allows easy relocation of a program in Central Memory and also defines the lower limit of a Central Memory program.

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.

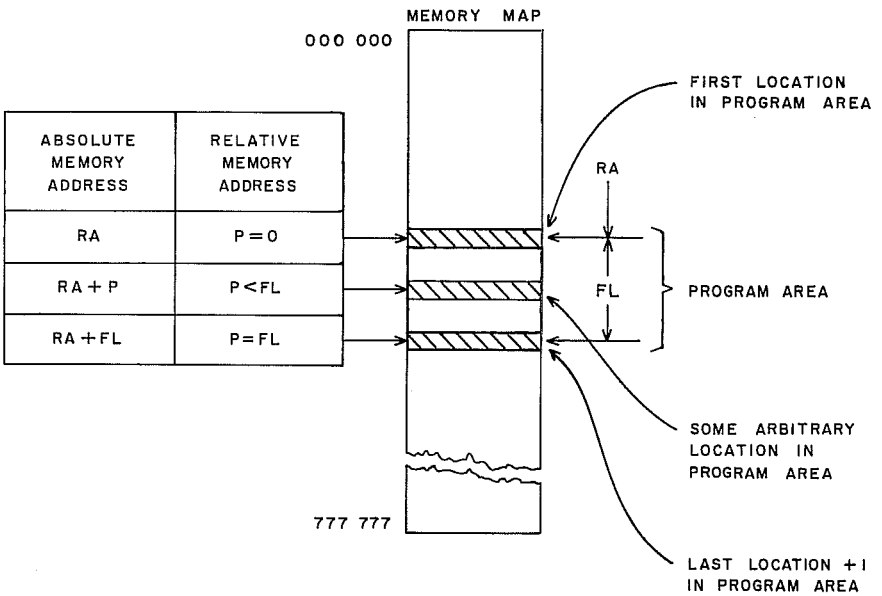


Figure 2-1. Memory Map

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

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

Note that FL is the number of 60-bit words comprising the program, not an address. Program restriction: Do not use a Field Length $< 10_8$.

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. When operating under SIPROS, however, addresses out of range always cause a halt.

3. CENTRAL PROCESSOR

ORGANIZATION

The Central Processor in the 6000 Series 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 and 6800 Central Processors in two important respects. These differences are tabulated below.

TABLE 3-1. CENTRAL PROCESSOR DIFFERENCES IN 6000 SERIES

SYSTEM	INSTRUCTION REGISTERS	ARITHMETIC SECTION
6400 Central Processor	Instruction Buffer Register; holds one 60-bit instruction word.	Unified Arithmetic Section; executes instructions in serial order. Requires no reservation control.
6600 and 6800 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 and 6800 systems. 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.

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 allocates 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 length
- 8 increment registers of 18 bits length
- 8 operand registers of 60 bits length

Eight 60-bit registers are provided to hold instructions (6600 and 6800), 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 and 6800 Computer Systems, 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, 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. Optionally (except when operating under SIPROS), 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 and Mass

Memory 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 and 6800 systems handle 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 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.

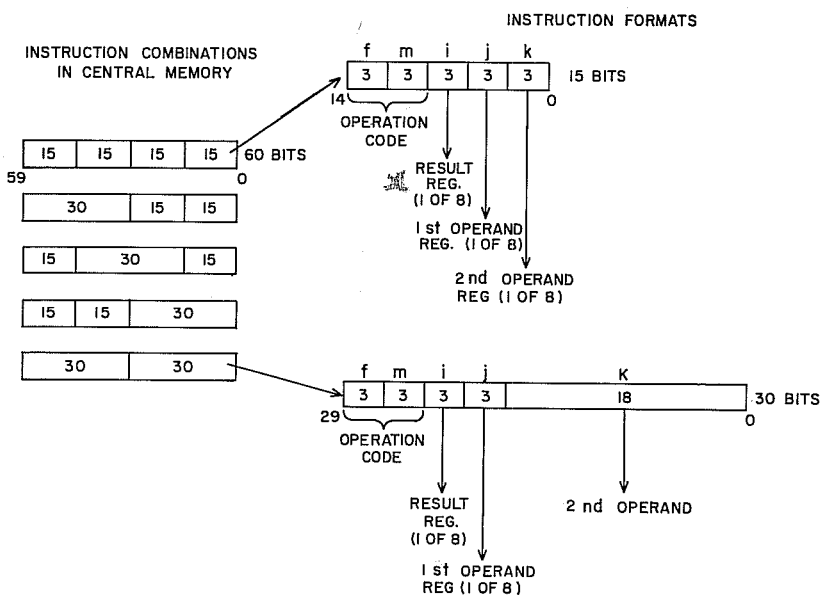


Figure 3-1. Central Processor Instruction Formats

The f and m digits identify the type of instruction and are the operation code. 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.

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.

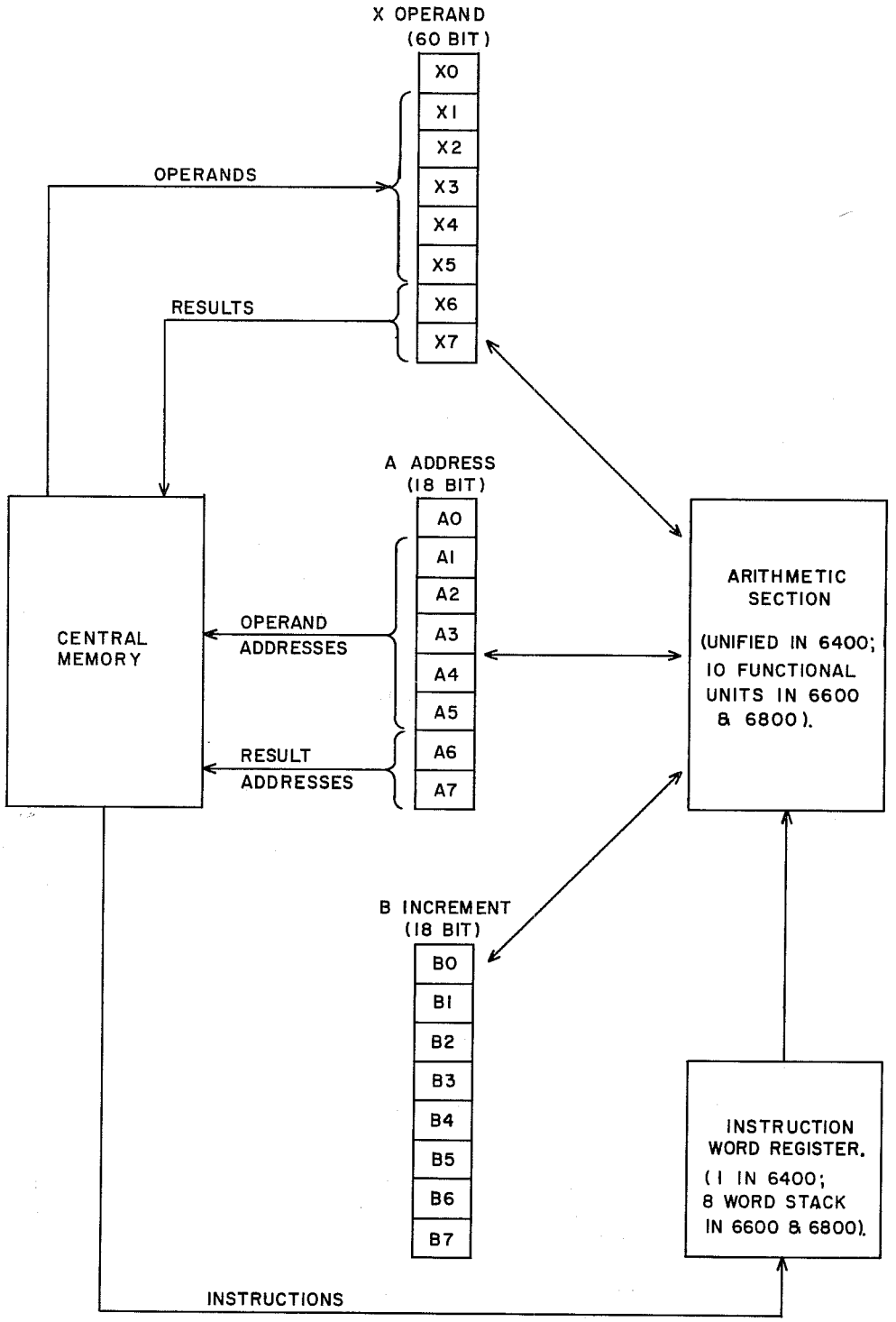


Figure 3-2. Central Processor Operating Registers

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 Mass Memory 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 it 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 Mass Memory (RA_{ECS})
5) Field length of program for Mass Memory (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)

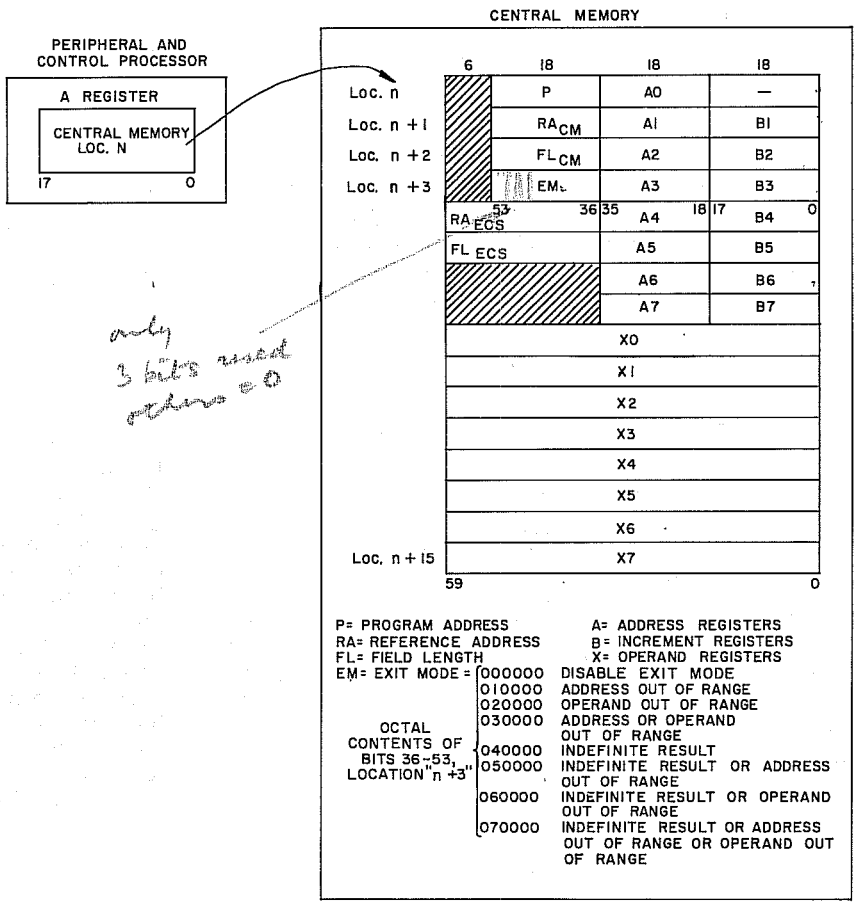


Figure 3-3. Exchange Jump Package

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

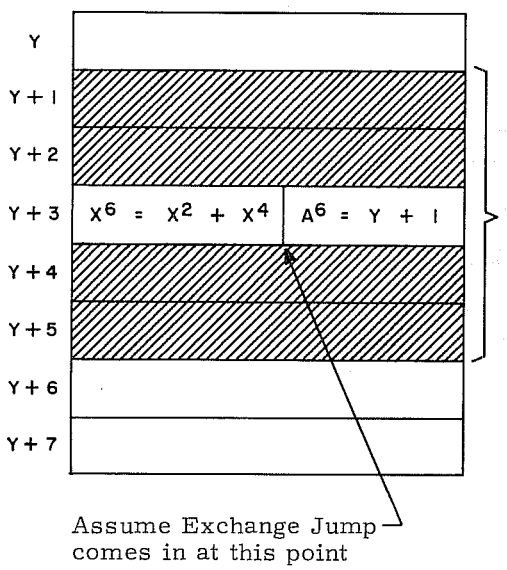
When an Exchange Jump interrupts the Central Processor, several steps occur to insure leaving the interrupted program in a usable state for re-entry:

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 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 optional Exit condition 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 is made to the address specified by P + RA. Hence program relocation is conveniently handled through a single change to RA.

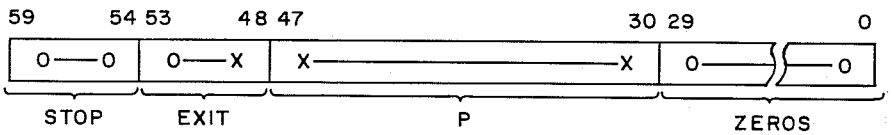
A P=0 condition specifies address zero and hence RA. This address is reserved for recording program exit (error) conditions.

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 - an attempt to reference either Central Memory or Extended Core Storage outside established limits, or the word count, $[(Bj)+K]$, in a Mass Memory Communication instruction, is negative. (For details on action when an address is out of range, refer to the Increment instructions.)
	= 020000	Operand out of range - floating point arithmetic unit received an infinite operand (see Range Definitions, page 3-14).
	= 030000	Address or operand out of range
	= 040000	Indefinite operand - floating point arithmetic unit (floating Add, Multiply, or Divide) attempted to use an indefinite operand (see Range Definitions, page 3-14)
	= 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

When an error exit is made, the Central Processor records at RA a Stop instruction, Exit condition (upper 2 octal digits only), and the Program Address at exit time in the format shown below and jumps to P = 0 (RA), thereby stopping.



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

For error stops the (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.

When operating under SIPROS, the Peripheral and Control Processor searches for an unchanging Central Processor P register (any value) to determine if the Central Processor has stopped. If P=0, an error stop has occurred; the contents of RA may then be examined to determine the nature of the error stop. A normal stop does not set P equal to zero, nor does it cause anything to be stored at RA. The Central Processor stops with P equal to the Program Address of the word containing the Stop instruction.

Floating Point Arithmetic

Format

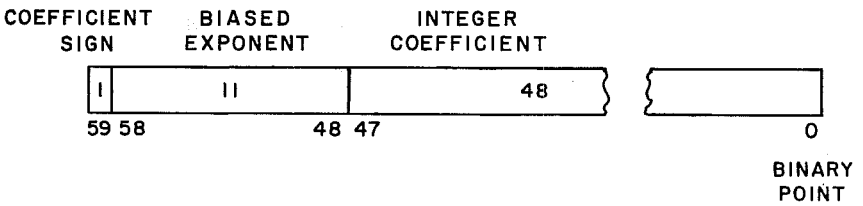
Floating point arithmetic takes advantage of the ability to express a number with the general expression kB^n, where:

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 15 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. The bias provides for a signed exponent within the following ranges:

$2^{1023} = 3777_8$

$2^0 = 2000_8$ (zero = 0000000000000000000_8)

$2^{-1023} = 0000_8$

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.

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 operands allows separate recovery of upper and lower half results with proper exponents; only upper half results can be obtained with rounded operands.

Range Definitions

A result with an exponent so large that it reaches or 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 is packed for this case. An optional exit is provided when an infinite operand is detected in the floating arithmetic units since its use may propagate an indefinite result as shown in Table 3-3. No error exit occurs when an infinite or indefinite result is generated in a functional unit.

TABLE 3-3. INDEFINITE FORMS

$\infty - \infty$	= INDEFINITE	$\infty \div N = \infty$	
$\infty \div \infty$	= 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.

Tests for infinite and indefinite operands are made only in the Floating Add, Multiply, and Divide units. Only the twelve 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 twelve 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 3-3 and Appendix F, 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 3-3 and Appendix F 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 3-33.)

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 but may be with a Normalize instruction.

Fixed Point Arithmetic

Fixed point addition and subtraction of 60-bit numbers are handled in the Long Add Unit (6600 and 6800). 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 one of the integer operands be converted (by program) to floating format to provide a biased exponent. 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 and 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 shift 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 (in the 6600 and 6800 systems) which execute them. Central Processor instructions as related to functional units are tabulated in Appendix E. Instruction Execution Times.

TABLE 3-4. 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.

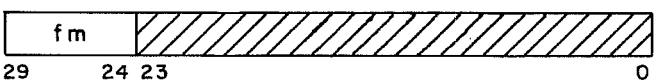
EXAMPLE:

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

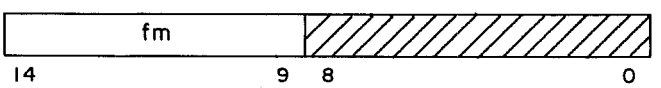
Program Stop and No Operation

00 PS Program Stop (30 Bits)



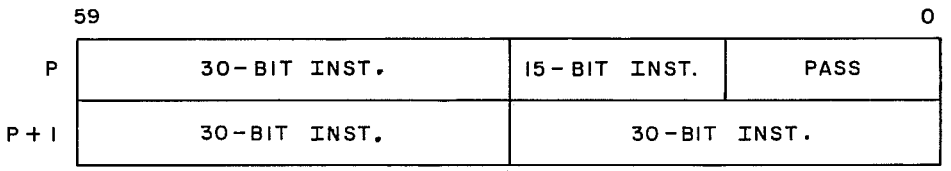
This instruction stops the Central Processor at the current step in the program. An exchange Jump is necessary to restart the Central Processor.

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.

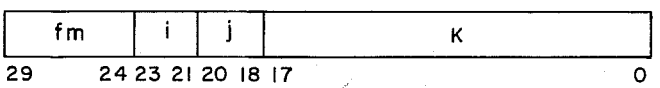
EXAMPLE:



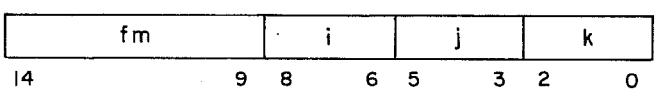
In this example, a Pass instruction is used to pad the remainder of the word at P. Since the next instruction is 30 bits, it cannot fit in P and must be placed in P + 1.

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 cleared to all zeros 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:

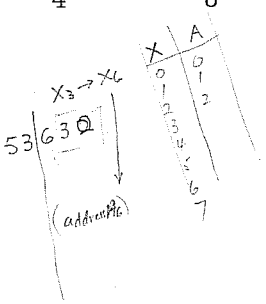
$K = 234567_8$

$A_4 = 321110_8$

$A_6 = 432100_8$

$X_4 = 00_8$

Storage location 666667_8 = 7...75342104600_8



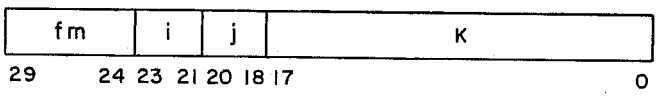
Final Quantities:

$A_4 = 666667_8$

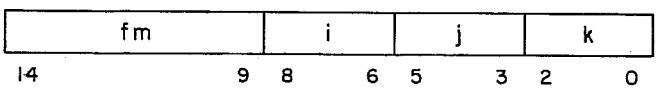
$A_6 = 432100_8$

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)*



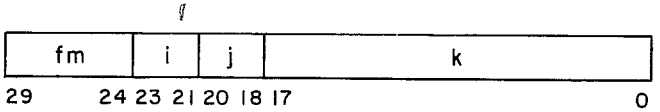
63	*SBi*	*Xj + Bk*	*Set Bi to Xj + Bk*	*(15 Bits)*
64	*SBi*	*Aj + Bk*	*Set Bi to Aj + Bk*	*(15 Bits)*
65	*SBi*	*Aj − Bk*	*Set Bi to Aj − Bk*	*(15 Bits)*
66	*SBi*	*Bj + Bk*	*Set Bi to Bj*	
66	*SBi*	*Bj + Bk*	*Set Bi to Bj + Bk*	*(15 Bits)*
67	*SBi*	*Bj − Bk*	*Set Bi to Bj − Bk*	*(15 Bits)*



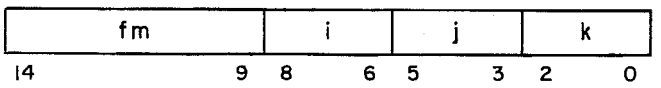
These instructions perform one's complement addition and subtraction of 18-bit operands and store an 18-bit result in increment register Bi. An overflow condition is ignored.

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, j = 3)

SX_2 $X_3 + B_1$

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

$SX_2 = 7...7777735555_8$

Initial Quantities:

$X_2 = 0...0745321402_8$

$X_3 = 0...0652224310_8$

$B_1 = 511245_8$

Final Quantities:

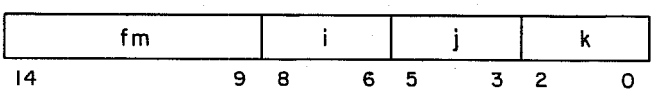
$X_2 = 7...7777735555_8$

$X_3 = 0...0652224310_8$

$B_1 = 511245_8$

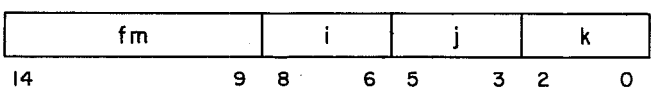
Fixed Point Arithmetic

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.

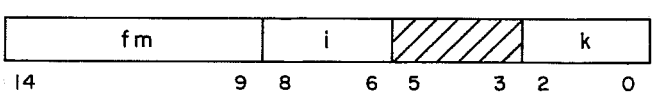
37 IXi Xj − Xk Integer difference of Xj and Xk to Xi (15 Bits)



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)

Population count



This instruction counts the number of "1's" in operand register Xk and stores the count in the lower order six bits of operand register Xi.

EXAMPLE:

```
47    CXi   Xk   i = 4
      CX4   X1   k = 1
      CX4 = 11(8)
```

Initial Quantities:

$X_1 = 0 \ldots 0543321_8$

$X_4 = 23420 \ldots 0005547_8$

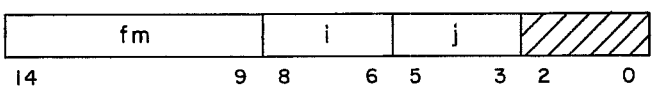
Final Quantities:

$X_1 = 0 \ldots 0543321_8$

$X_4 = 0 \ldots 0000011_8$

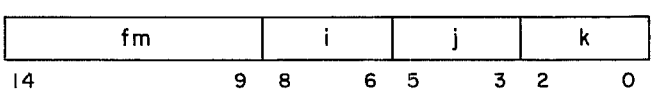
Logical

10 BXi Xj Transmit Xj to Xi (15 Bits)



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

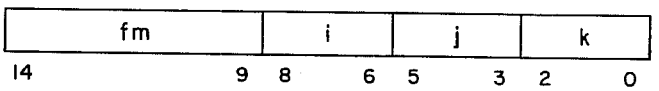
11 BXi Xj * Xk Logical Product of Xj and Xk to Xi (15 Bits)



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
```

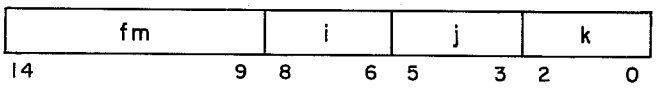
12 **BXi** ***Xj + Xk*** ***Logical sum of Xj and Xk to Xi*** ***(15 Bits)***



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

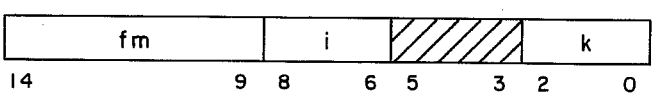
13 **BXi** ***Xj − Xk*** ***Logical difference of Xj and Xk to Xi*** ***(15 Bits)***



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:

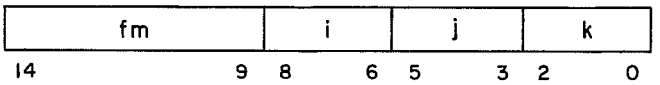
Xj = 0101
Xk = 1100
Xi = 1001

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.

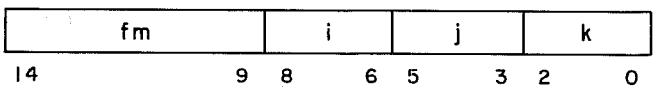
15 BXi −Xk * Xj Logical product of Xj and complement of Xk to Xi (15 Bits)



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

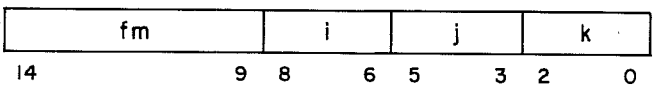
16 BXi −Xk+Xj Logical sum of Xj and complement of Xk to Xi (15 Bits)



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)***

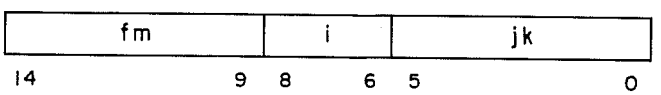


This instruction forms the logical difference (exclusive OR) of the quantity from operand register Xj and the complement of the 60-bit word from operand register Xk, and places the result in operand register Xi. Thus, bits of Xi are set to "1" if the corresponding bits of register Xj and the complement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

```
          Xj = 0101
Complemented Xk = 0011
          Xi = 0110
```

Shift

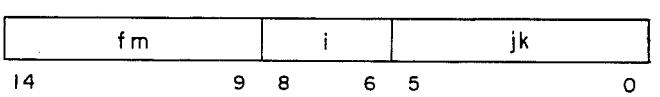
20 **LXi** ***jk*** ***Left shift Xi, jk places*** ***(15 Bits)***



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.

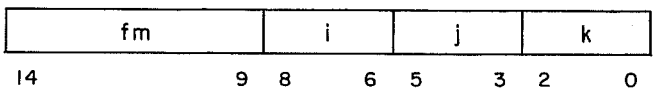
The 6-bit shift count jk allows a complete circular shift of register Xi.

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)

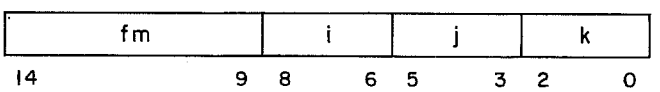


This instruction shifts the 60-bit quantity from operand register Xk the number of places specified by the quantity in increment register Bj and places the result in operand register Xi.

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

2) If Bj is negative, the quantity from Xk is shifted right (end off with sign extension). 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)



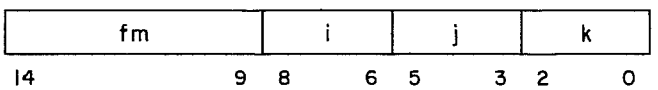
This instruction shifts the 60-bit quantity from operand register Xk the number of places specified by the quantity in increment register Bj and places the result in operand register Xi.

1) If Bj is positive, 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, the quantity from register Xk is shifted left circular. (The complement of the low order six bits of Bj specify the shift count.)

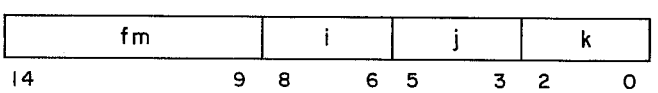
24 NXi Bj Xk Normalize Xk in Xi and Bj (15 Bits)



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)

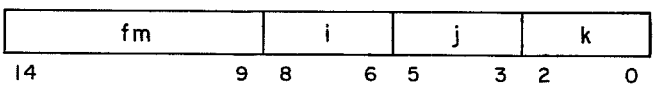


This instruction performs the same operation as instruction 24 except that the quantity

from operand register Xk is rounded before it is normalized. 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.

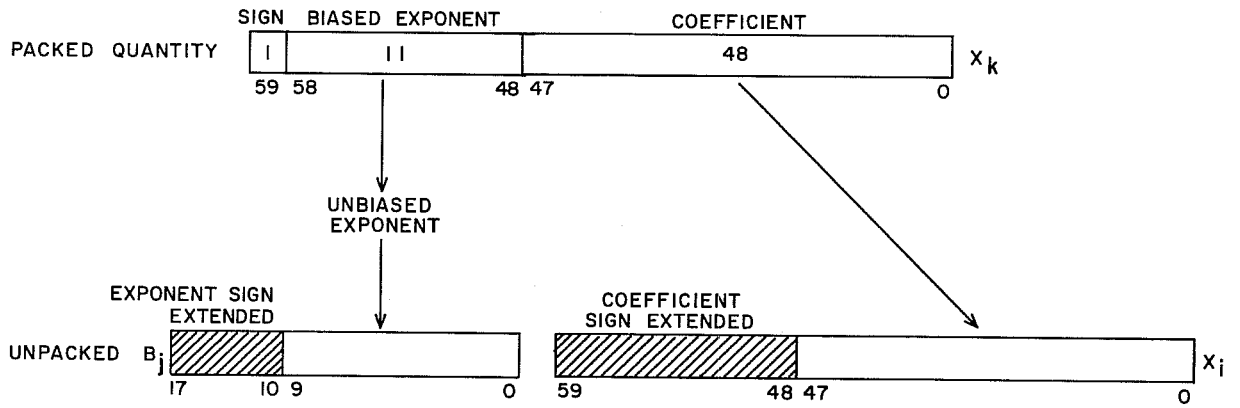
If Xk contains an infinite quantity (3777X...X or 4000X...X) or an indefinite quantity (1777X...X or 6000X...X), no shift takes place. The contents of Xk are copied into Xi and Bj is set equal to zero. Optional error exits do not occur.

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

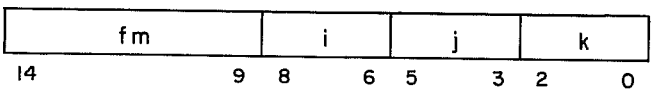


This instruction unpacks the floating point quantity from operand register Xk and sends the 48-bit coefficient to operand register Xi and the 11-bit exponent to increment register Bj. The exponent bias is removed during Unpack so that the quantity in Bj is the true one's complement representation of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

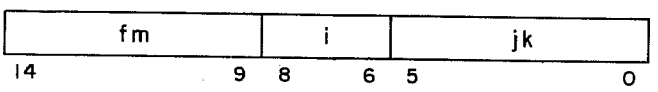


This instruction packs a floating point number in operand register Xi. The coefficient of the number is obtained from operand register Xk and the exponent from increment register Bj. Bias is added to the exponent during the Pack operation. The instruction does not normalize the coefficient.

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, the complement of this 11-bit quantity is packed.

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

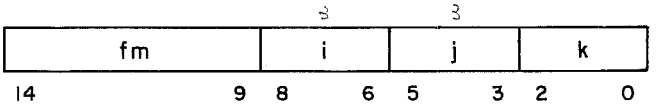


This instruction forms a mask in operand register Xi. The 6-bit quantity jk defines the number of "1's" in the mask as counted from the highest order bit in Xi.

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

Floating Point Arithmetic

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

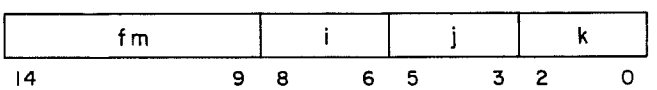


This instruction forms the sum of the floating point quantities from operand registers Xj and Xk and packs the result in operand register Xi. The packed result is the upper half of a double precision sum.

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 3-3 and Appendix F.

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

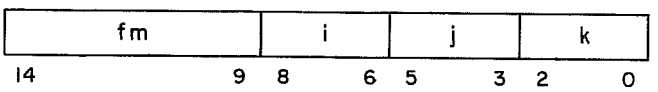


This instruction forms the difference of the floating point quantities from operand registers Xj and Xk packs the result in operand register Xi. Alignment and overflow operations are similar to the Floating Sum (30) instruction, and the difference is not necessarily normalized. The packed result is the 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 3-3 and Appendix F.

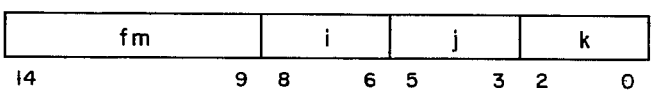
*A zero exponent is 2000_8.

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



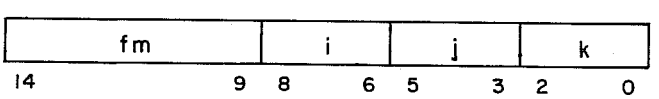
This instruction forms the sum of two floating point numbers as in the Floating Sum (30) instruction, but packs the lower half of the double precision sum with an exponent 48 less than the upper sum. For treatment of special operands and/or indefinite forms, refer to Table 3-3 and Appendix F.

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 3-3 and Appendix F.

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.

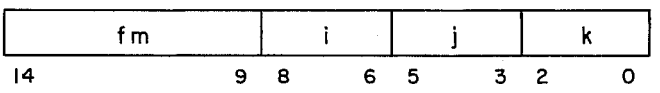
1) A round bit is attached at the right end of both operands if:

   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 3-3 and Appendix F.

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.

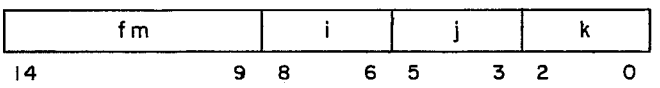
1) A round bit is attached at the right end of both operands if:

   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 3-3 and Appendix F.

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

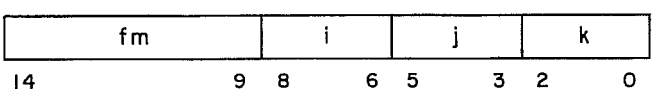


This instruction multiplies two floating point quantities obtained from operand registers Xj (multiplier) and Xk (multiplicand) and packs the upper product result in operand register Xi.

The 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 3-3 and Appendix F.

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

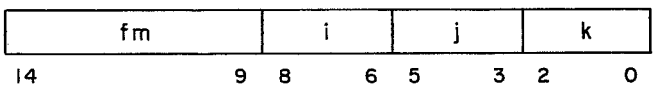


This instruction multiplies the floating point number from operand register Xk (multiplicand), by the floating point number from operand register Xj, and packs the upper product result in operand register Xi. (No lower product available.) During the first iteration of the multiply step, a partial carry is forced in bit 47. At the conclusion of the multiply step, rounding is accomplished by adding one to the upper product if the portion of the product to the right of the binary point is $\geq$ 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).

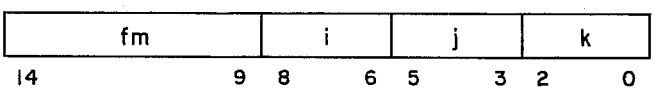
The result is unnormalized when either or both operands are unnormalized; the exponent in this case is the sum of the exponents plus 48. For treatment of special operands and/or indefinite forms, refer to Table 3-3 and Appendix F.

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 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 3-3 and Appendix F.

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



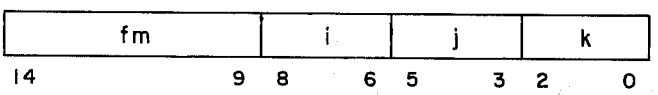
This instruction divides two normalized floating point quantities obtained from operand registers Xj (dividend) and Xk (divisor) and packs the quotient in operand register Xi.

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 dividend $\geq$ two times 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 3-3 and Appendix F.

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. A one-third round bit is added to the least significant bit of the dividend before division starts. (Rounding on a divide operation forces bit 0 of the original dividend Xj to a "1" and makes the portion of the dividend to the right of the binary point equal to one-third ($.2525...25_8$).

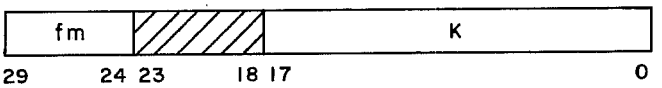
The result exponent in a no-overflow case is the difference of the dividend and divisor exponents minus 48.

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 dividend $\geq$ two times 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 3-3 and Appendix F.

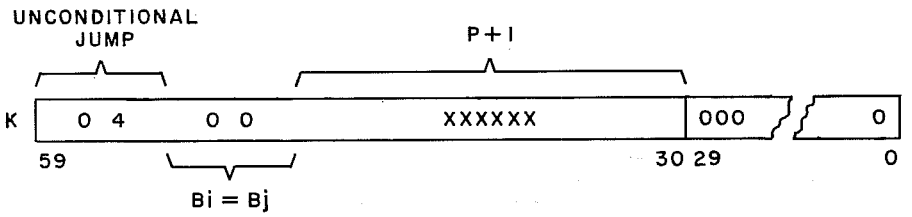
Branch

01 RJ K Return jump to K (30 Bits)



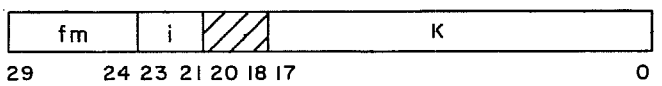
The instruction stores an 04 unconditional jump and the current address plus one (P + 1) in the upper half of address K, then branches to K + 1 for the next instruction. Note that this instruction is always out of the instruction stack, thus voiding the stack.

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.

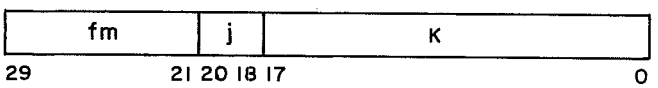
02 JP Bi + K Jump to Bi +K (30 Bits)



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 magnitude 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.

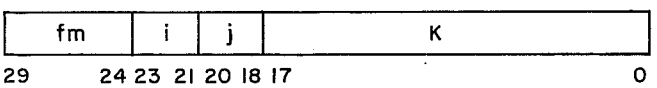
c) The 034 (IR) and 035 (OR) operations examine the upper-order 12 bits of Xj. Both plus and minus infinity are detected:

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 Bj ≠ 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

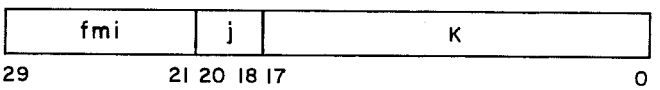
c) A positive number is recognized as greater than a negative number.

Mass Memory Communication

This category of instructions provides the ability to communicate with Extended Core Storage. To avoid cumbersome nomenclature, this description uses the term Mass Memory, rather than the term Extended Core Storage. Mass Memory communication instructions as related to the 6411 are described in Appendix A.

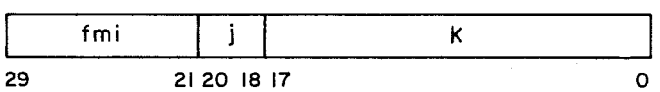
This section describes Mass Memory Communication instructions (and ramifications) only; information on Mass Memory itself is available in separate literature.

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



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

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 Mass Memory. The initial Central Memory address is $[(A0) + RA_{CM}]$; the initial Mass Memory address is $[(X0) + RA_{ECS}]$.

Address Formation: The starting address in Mass Memory 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 Mass Memory, and FL_{ECS} is the allotted Field Length within Mass Memory. 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.

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

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 absolute addresses in A0 and X0, however, must be placed there by the program prior to executing the Mass Memory Communication instructions.

Address Range Faults: Three address range fault conditions can arise when executing the Mass Memory Communication instructions:

- Word count fault
- Central Memory address out of range
- Mass Memory address out of range

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.

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.

c) Mass Memory Address

Mass Memory 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.

Note that address range checks are made on the entire block of both Mass 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 or Mass Memory, no data is transferred.

Exchange Jump During Mass Memory Communication: If an Exchange Jump occurs while a Mass Memory transfer is occurring, the transfer is truncated at the next record gap (the Extended Core Storage Coupler transmits a fake End of Transfer signal to the Central Processor) and the exchange takes place. The contents of P, the initial memory addresses, and associated information are stored into the Exchange Jump package in Central Memory. This permits a return Exchange Jump to begin with the Mass Memory Communication instruction and restart the Mass Memory transfer. Note that the transfer does not resume at the point it was truncated; the entire transfer must be repeated.

4. PERIPHERAL AND CONTROL PROCESSORS

ORGANIZATION

The ten Peripheral and Control Processors (for simplicity sometimes simply called Peripheral 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. 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.

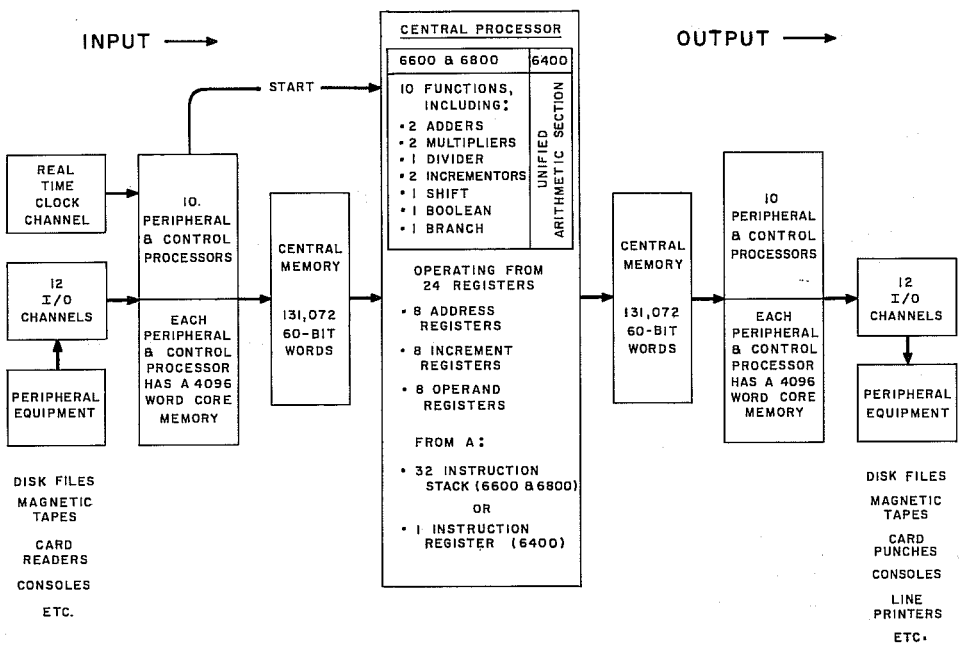


Figure 4-1. Flow Chart: 6000 Series System

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

* Major cycle = 1000 ns in 6400 and 6600; 250 ns in 6800

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. Programs operating under SIPROS can specify running time limits for both Central Processor and Peripheral and Control Processor use. The operating system then monitors the real-time clock to insure limits are not exceeded.

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 but call on the Central Processor for solutions. The processors 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 Mass Memory parameters with the same information for the new program. A later Exchange Jump may return to complete the interrupted program.

The Simultaneous Processing Operating System (SIPROS) provides an orderly scheme for supervising I/O and Central Processor activity. SIPROS uses one Peripheral Processor as an executive/monitor to direct channel assignments, provide file protection in Central Memory, handle Central Processor requests for all processors, assign specific I/O jobs to the processors, and assign other tasks as necessary.

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.

---

* in 6400 and 6600; 250 ns in 6800

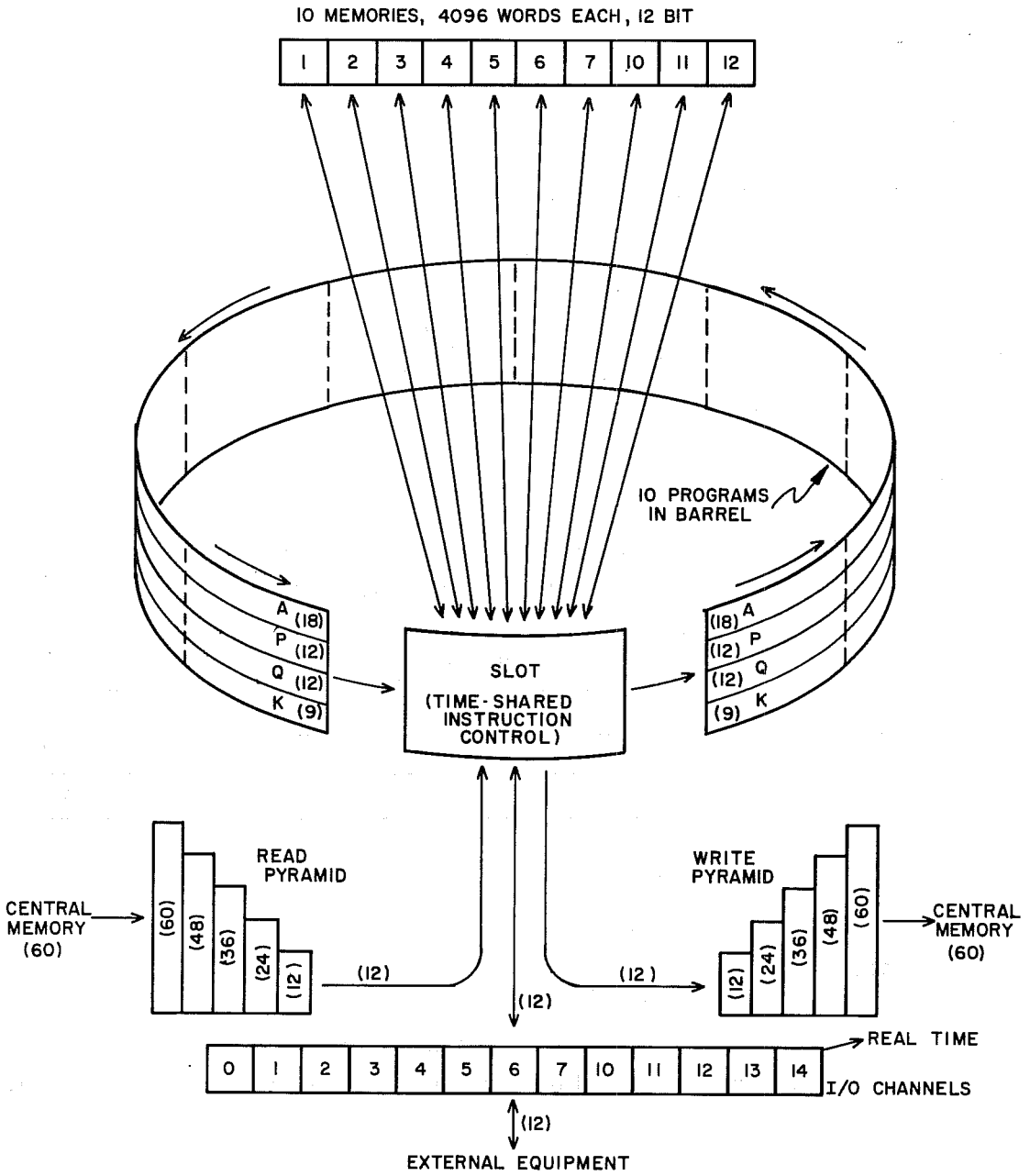
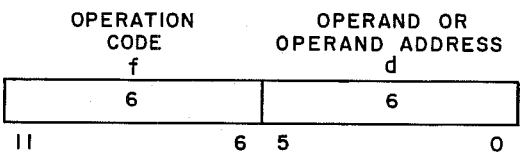


Figure 4-2. Peripheral and Control Processors

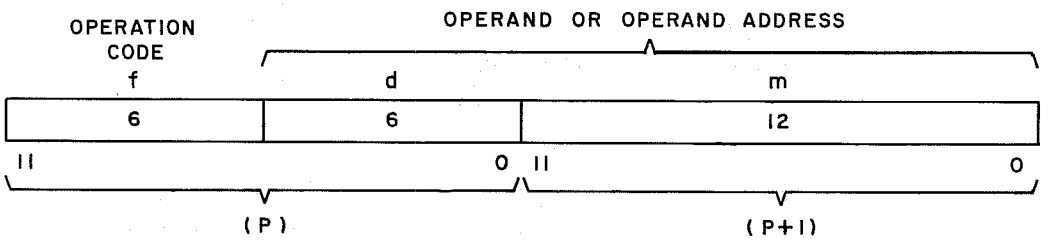
PERIPHERAL PROCESSOR PROGRAMMING

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 Processor instructions.

TABLE 4-1. ADDRESSING MODES FOR PERIPHERAL PROCESSOR INSTRUCTIONS

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

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 the number of times the instruction has been around the barrel and lends 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 Processor instructions. Table 4-2 lists designators used throughout the section.

TABLE 4-2. PERIPHERAL PROCESSOR INSTRUCTION DESIGNATORS

Designator	Use
A	The Peripheral Processor 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 Peripheral Processor Program Address register.
Q	The Peripheral Processor 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 Processor Assembly language.

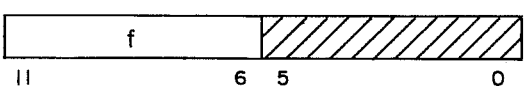
EXAMPLE:

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

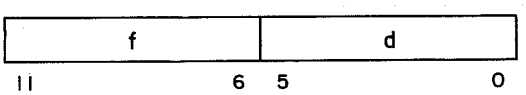
00	***PSN***	***Pass***	***(12 Bits)***
24	***PSN***	***Pass***	***(12 Bits)***
25	***PSN***	***Pass***	***(12 Bits)***



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

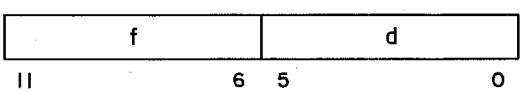
Data Transmission

14 LDN d Load d (12 Bits)



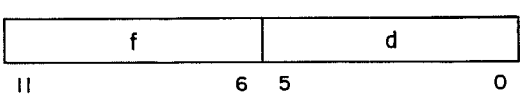
This instruction clears the A register and loads d. The upper 12 bits of A are zero.

15 LCN d Load Complement d (12 Bits)



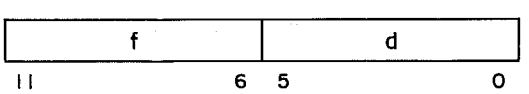
This instruction clears the A register and loads the complement of d. The upper 12 bits of A are set to one.

30 LDD d Load (d) (12 Bits)



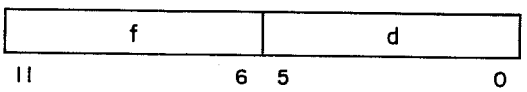
This instruction clears the A register and loads the contents of location d. The upper six bits of A are zero.

34 STD d Store (d) (12 Bits)



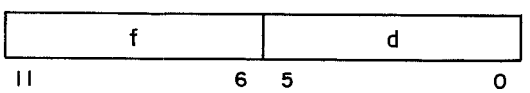
This instruction stores the lower 12 bits of A in location d.

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



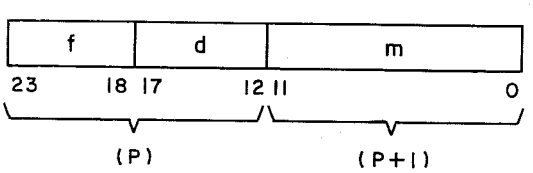
This instruction clears the A register and loads a 12-bit quantity that is obtained by indirect addressing. The upper six bits of A are zero. Location d is read out of memory, and the word obtained is used as the operand address.

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



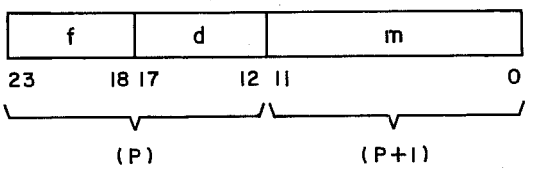
This instruction stores the lower 12 bits of A in the location specified by the contents of location d.

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



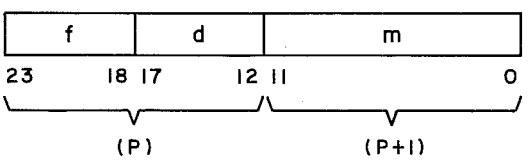
This instruction clears the A register and loads an 18-bit quantity consisting of d as the higher six bits and m as the lower 12 bits. The contents of the location following the present program address are read out to provide m.

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 simple 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.

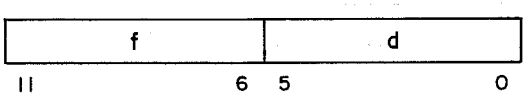
54 **STM** *m d* ***Store (m + (d))*** ***(24 Bits)***



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

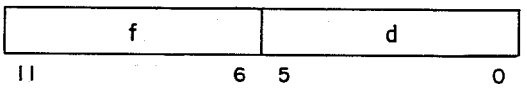
Arithmetic

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



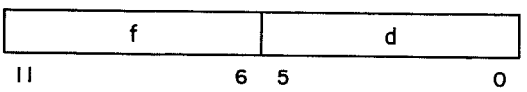
This instruction adds d (treated as a 6-bit positive quantity) to the content of the A register.

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



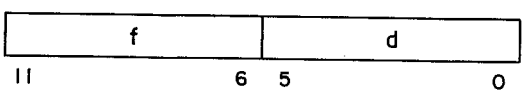
This instruction subtracts d (treated as a 6-bit positive quantity) from the content of the A register.

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



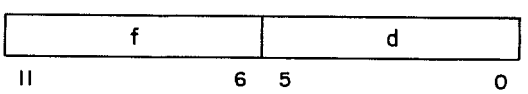
This instruction adds to the A register the contents of location d (treated as a 12-bit positive quantity).

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



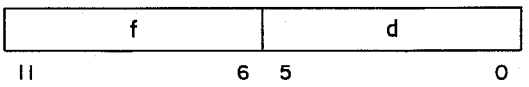
This instruction subtracts from the A register the contents of location d (treated as a 12-bit positive quantity).

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



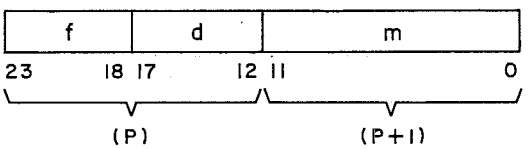
This instruction adds to the content of A a 12-bit operand (treated as a positive quantity) obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address.

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



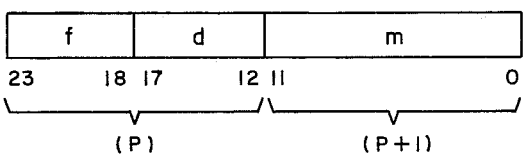
This instruction subtracts from the A register a 12-bit operand (treated as a positive quantity) obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address.

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



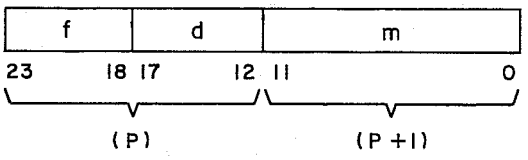
This instruction adds to the A register the 18-bit quantity consisting of d as the higher six bits and m as the lower 12 bits. The contents of the location following the present program address are read out to provide m.

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).

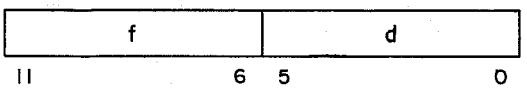
52 ***SBM*** ***m d*** ***Subtract (m + (d))*** ***(24 Bits)***



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

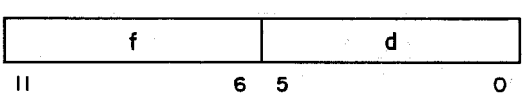
10 ***SHN*** ***d*** ***Shift d*** ***(12 Bits)***



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.

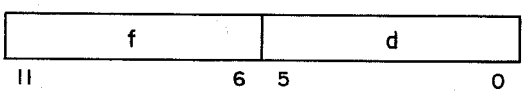
Logical

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



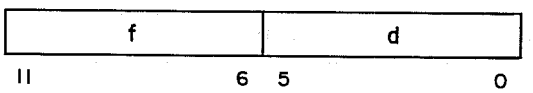
This instruction forms in A the bit-by-bit logical difference of d and the lower six bits of A. This is equivalent to complementing individual bits of A that correspond to bits of d that are one. The upper 12 bits of A are not altered.

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



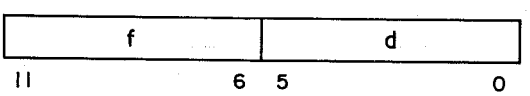
This instruction forms the bit-by-bit logical product of d and the lower six bits of the A register, and leaves this quantity in the lower 6 bits of A. The upper 12 bits of A are zero.

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



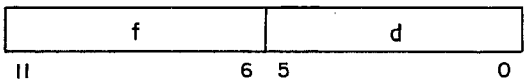
This instruction clears any of the lower six bits of the A register where there are corresponding bits of d that are one. The upper 12 bits of A are not altered.

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



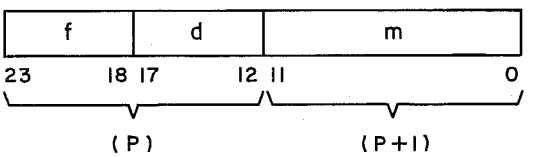
This instruction forms in A the bit-by-bit logical difference of the lower 12 bits of A and the contents of location d. This is equivalent to complementing individual bits of A which correspond to bits of (d) that are one. The upper six bits of A are not altered.

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



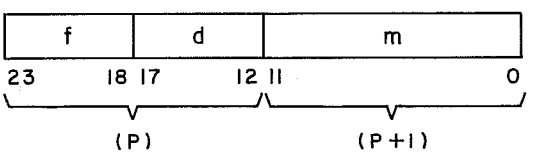
This instruction forms in A the bit-by-bit logical difference of the lower 12 bits of A and the 12-bit operand obtained by indirect addressing. Location d is read out of memory, and the word obtained is used as the operand address. The upper six bits of A are not altered.

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



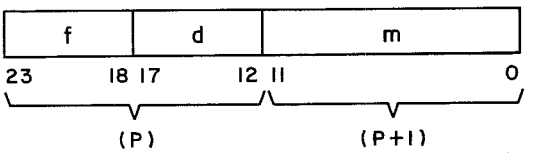
This instruction forms in the A register the bit-by-bit logical product of the contents of A and the 18-bit quantity dm. The upper six bits of this quantity consist of d and the lower 12 bits are the content of the location following the present program address.

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.

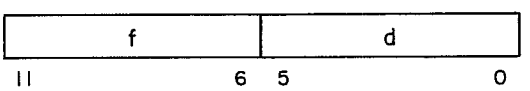
53 **LMM** *m d* ***Logical difference (m + (d))*** ***(24 Bits)***



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.

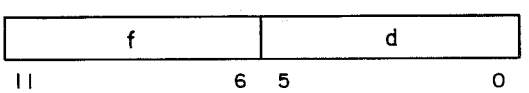
Replace

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



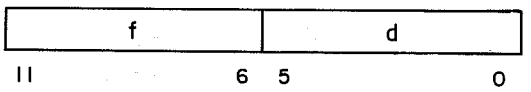
This instruction adds the quantity in location d to the contents of A and stores the lower 12 bits of the result at location d. The resultant sum is left in A at the end of the operation and the original contents of A are destroyed.

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



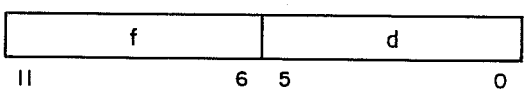
The quantity in location d is replaced by its original value plus one. The resultant sum is left in A at the end of the operation, and the original contents of A are destroyed.

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



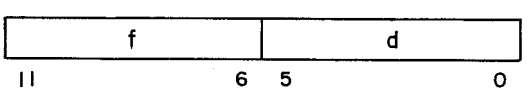
The quantity in location d is replaced by its original value minus one. The resultant difference is left in A at the end of the operation, and the original contents of A are destroyed.

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



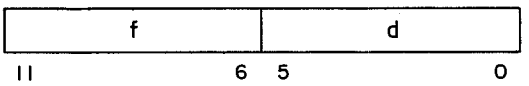
The operand which is obtained from the location specified by the contents of location d, is added to the contents of A, and the lower 12 bits of the sum replace the original operand. The resultant sum is also left in A at the end of the operation.

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



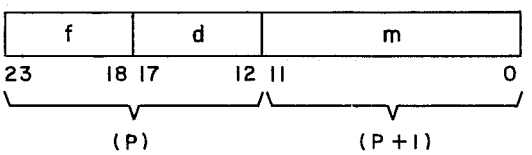
The operand, which is obtained from the location specified by the contents of location d, is replaced by its original value plus one. The resultant sum is also left in A at the end of the operation, and the original contents of A are destroyed.

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



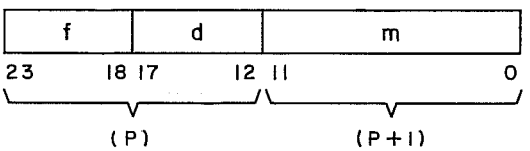
The operand, which is obtained from the location specified by the contents of location d, is replaced by its original value minus one. The resultant difference is also left in A at the end of the operation, and the original contents of A are destroyed.

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



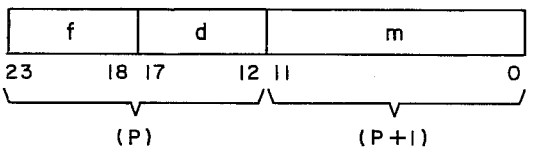
The operand, which is obtained from the location determined by indexed direct addressing, is added to the contents of A, and the lower 12 bits of the sum replace the original operand in memory. The resultant sum is also left in A at the end of the operation, and the original contents of A are destroyed.

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.

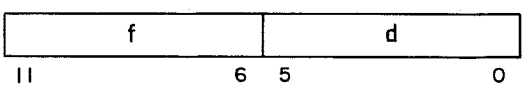
57 **SOM** *m d* ***Replace subtract one (m + (d))*** ***(24 Bits)***



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.

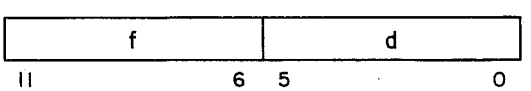
Branch

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



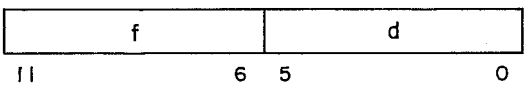
This instruction provides an unconditional jump to any instruction up to 31 steps forward or backward from the current program address. The value of d is added to the current program address. If d is positive (01 - 37), then 0001 (+1)→0037 (+31) is added and the jump is forward. If d is negative (40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

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



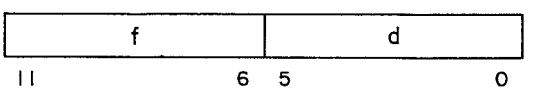
This instruction provides a conditional jump to any instruction up to 31_{10} steps forward or backward from the current program address. If the content of the A register is zero, the jump is taken. If the content of A is non-zero, the next instruction is executed. Negative zero (777777) is treated as non-zero. For interpretation of d see instruction 03.

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



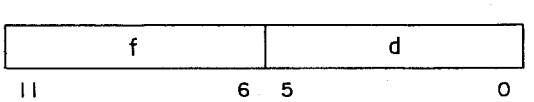
This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is nonzero, the jump is taken. If A is zero, the next instruction is executed. Negative zero (777777) is treated as nonzero. For interpretation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

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



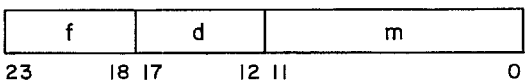
This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is positive, the jump is taken. If A is negative, the next instruction is executed. Positive zero is treated as a positive quantity; negative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

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



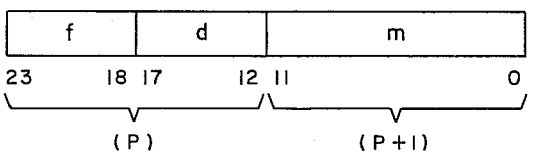
This instruction provides a conditional jump to any instruction up to 31 steps forward or backward from the current program address. If the content of the A register is negative, the jump is taken. If A is positive, the next instruction is executed. Positive zero is treated as a positive quantity; negative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

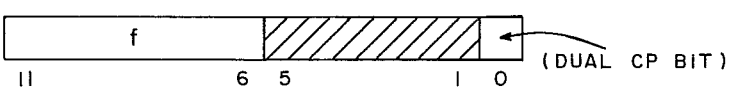
02 **RJM** *m d* ***Return jump to m + (d)*** ***(24 Bits)***



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

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

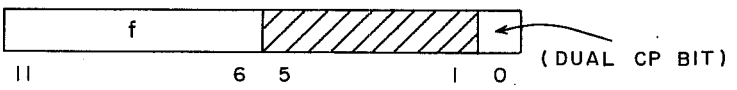


This instruction transmits an 18-bit address (only 17 bits are used) from the A register to the Central Processor with a signal which tells the Central Processor to perform an Exchange Jump, with the address in A as the starting location of a file of 16 words containing information about the Central 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.

26.00 Regular Ex. Jump to CPU 0
26 01 " " " CPU 1
26 10 Monitor Ex. Jump CPU 0
26 11 " " " CPU 1

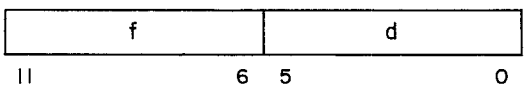
In a 6400 system with dual Central Processors, the lowest order bit of the instruction format specifies which Central Processor the Exchange Jump will interrupt. In 6600 and 6800 systems, this bit is not interpreted.

27 RPN Read program address (12 Bits)



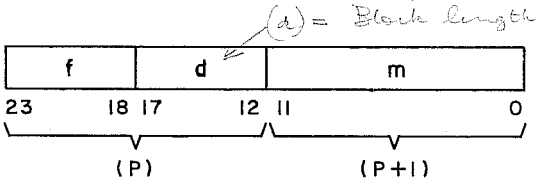
This instruction transfers the content of the Central Processor Program Address register, P, to the Peripheral Processor A register; this allows the Peripheral Processor to determine whether the Central Processor is running. In a 6400 system with dual Central Processors, the lowest order bit of the instruction format specifies which Central Processor P register is to be examined. In 6600 and 6800 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. 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)



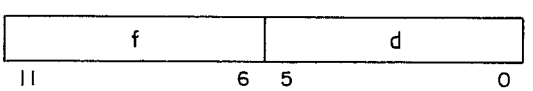
This instruction reads a block of 60-bit words from Central Memory. The contents of location d gives the block length. The 18-bit address of the first central word must be

loaded into A prior to executing this instruction. 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.

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.

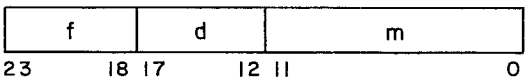
62 CWD d Central write to (A) from d (12 Bits)



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.

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 contents of location d gives the number of 60-bit words. The content of the A register gives the beginning Central Memory 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.

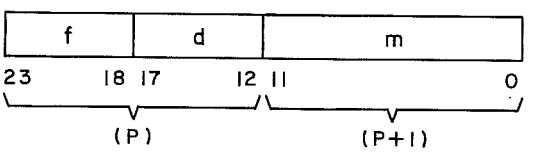
The content of P (the m portion of the instruction) gives the address of the first word to be read out of the processor memory. This word appears as the higher order 12 bits of the first 60-bit word to be stored in Central Memory.

The content of P is advanced by one to provide the next address in the processor memory as each 12-bit word is read.

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.

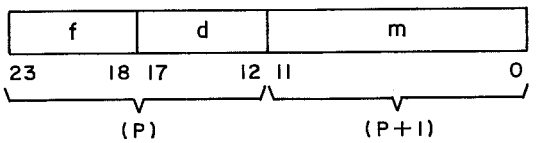
Input/Output

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



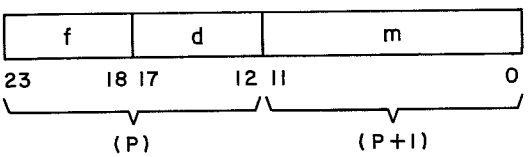
This instruction provides a conditional jump to a new program sequence beginning at an address given by the contents of m. The jump is taken if the channel specified by d is active. The current program sequence continues if the channel is inactive.

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.

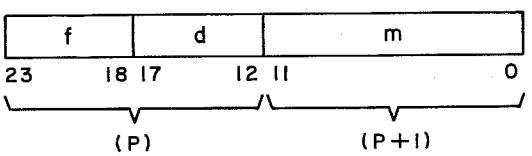
66 **FJM** *m d* ***Jump to m if channel d full*** ***(24 Bits)***



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.

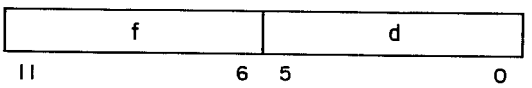
An input channel is full when the input equipment has placed a word on the channel and that word has not yet been sampled by a processor. The channel is empty when a word has been accepted. An output channel is full when a processor places a word on the channel. The channel is empty when the output equipment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

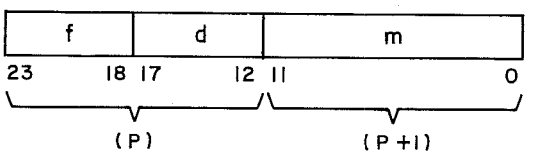


This instruction transfers a word from input channel d to the lower 12 bits of the A register.

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.

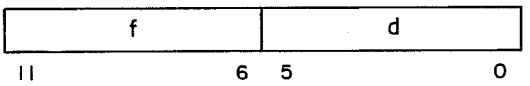
During this instruction address 0000 temporarily holds P, while m is held in the P register. The content of P advances by one to give the address for the next word as each word is stored.

NOTE

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

If block length is > record length then one extra word (12 bits) of zeros is stored equipment disconnects.

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

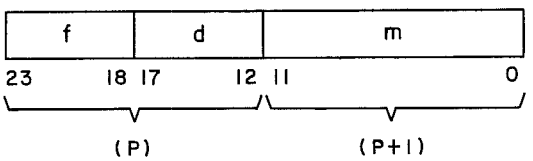


This instruction transfers a word from A (lower 12 bits) to output channel d.

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)***



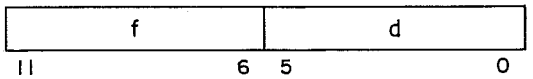
This instruction transfers a block of words from the processor memory to channel d. The first word comes from the address specified by m. The content of A specifies the number of words to be sent out. The content of A is reduced by one as each word is read out. The output operation is complete when A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

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)***

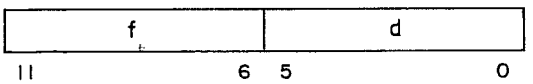


This instruction activates the channel specified by d. Activating a channel (must precede a 70 - 73 instruction) alerts and prepares the I/O equipment for the exchange of data.

NOTE

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

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

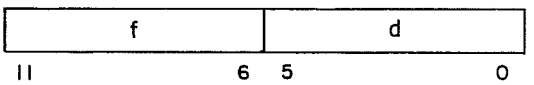


This instruction deactivates the channel specified by d. As a result, the I/O equipment stops and the buffer terminates.

NOTE

1) Do not deactivate an already inactive 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 causes an exit to address 0001 and execution of program.

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

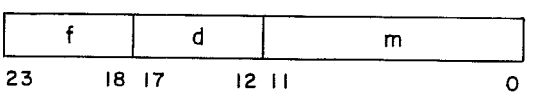


The external function code in the lower 12 bits of A is sent out on channel d.

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.

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 26 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 Processors 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 $\neq$ 0, the Central Processor is running a program or may have come to a normal (instruction) stop. If A = 0, the Central Processor has stopped in an Exit mode; 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.

*6400 and 6600; 6800 is one word each 250 ns.

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.

*6400 and 6600; its period in the 6800 is 1.024 ms.

5. SYSTEM INTERRUPT

INTRODUCTION

Essentially, detecting and handling interruptible conditions in the 6000 Series computer systems involves both hardware and software. This section describes hardware provisions for detecting and handling interrupt, and outlines the salient features of the operating system (SIPROS) for implementing interrupt handling in the software.

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/Ouput 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 3-10.)

SOFTWARE IMPLEMENTATION

The Simultaneous Processing Operating System (SIPROS), through the Peripheral Processor which it has designated as an executive/monitor, provides for implementing interrupt handling. Following is an examination of software implementation of hardware provisions for interrupt:

Exchange Jump

The executive and monitor Peripheral Processor is permanently assigned the duties of activities-director and operations-monitor. In scheduling the activity of the Central Processor, the Executive sets up all jobs in Central Memory in a priority sequence and prepares a list for the Monitor. As the Monitor cycles through the list of jobs and finds Wait conditions, for example, it exchange-jumps to the next job in the priority sequence. Thus, the executive/monitor Peripheral Processor uses the Exchange Jump facility to interrupt operating Central Processor programs and to begin new programs.

Exit Mode

The 6000 Series Operating System (SIPROS) sets Error Halt (Exit mode) selections in the job to be executed and periodically checks for an error halt.

The address out of range selection is monitored throughout job execution and an error halt always occurs when an address lies outside the defined parameters for the job. As

an aid to debugging programs, SIPROS allows the programmer to ignore two of these conditions, operand out of range and indefinite result, through the use of control cards. When one or both of these error halt conditions occurs, and an IGNORE (Control card) entry has been specified, SIPROS continues to process the job. (Refer to Control Card Specifications, Operating System Reference Manual Pub. No. 60101800.)

Whether or not the error halt condition is ignored, the occurrence of the condition is logged in a job log on the disk along with other information. The job accounting information for each program is automatically printed out from the disk on the last page of output for the job. (Job control cards provide for printing out other information as well, e.g., memory dump, memory map.) Operating personnel may also make requests through the console keyboard for all or portions of the job accounting log to be displayed.

Channel and Equipment Status

System macro instructions to the operating system provide communication links between a Peripheral Processor or Central Processor program and system Peripheral Processors. While most of these macros direct the operating system to perform input/output operations, others request equipment assignment, check the status of external operations, use system Peripheral Processors in conjunction with Peripheral or Central Processor programs, etc. Further, whenever applicable, system macros provide a buffered and a non-buffered mode. (Refer to System Macros section, Operating System Reference Manual.)

The system macros provide a means for obtaining equipment and channel status information. For example, a system macro request for magnetic tape operations provides return to the Central Processor program full status information as to the success in carrying out the request. Examples of status information (returned to an address specified in the macro address field) are:

- End of file
- Read length error
- Device not ready
- Request aborted

The Central Processor program can then examine the address holding this status information and respond as desired for a particular condition.

REAL-TIME INTERRUPT FACILITY

Typically, a real-time program to be executed in the Central Processor might use a Peripheral Processor for controlling and transferring real-time information for this program. For example, a Peripheral Processor might monitor some external equipment. On occurrence of a specified external condition, the Central Processor program is to be interrupted and an interrupt subroutine executed. Upon leaving the interrupt subroutine, control is to return to the interrupted Central Processor program.

Within the 6000 Series Operating System, (SIPROS), two methods currently exist for handling real-time interrupt situations. Both methods of providing a real-time interrupt facility employ special user programs written in the ASPER (Peripheral Processor assembler) language. One method uses the Executive/Monitor Peripheral Processor for transferring control to and from the interrupt subroutine, the other method uses more direct means.

Prior to loading the job, the necessary control cards for the job are prepared. These control cards specify, for example:

a) job name
b) external equipment requirements
c) Central Memory estimate
d) Peripheral Processor assignment
e) priorities assigned to the job and to I/O operations

For additional information on control card specifications, refer to the Operating System Reference Manual.

Included in the job deck is a system macro assigning the special ASPER program to the assigned Peripheral Processor. When the job is loaded, the Central Processor program and the ASPER program are placed in Central Memory. During execution of the job, the system macro assigning the ASPER program to the designated Peripheral Processor is interpreted. The ASPER program is then transferred to the Peripheral Processor and execution begins with the first ASPER instruction. The operating system provides the ability to communicate between the ASCENT and ASPER programs.

The Executive/Monitor Peripheral Processor, through a system scan cycle, examines the status of operations throughout the system. This scan cycle occurs approximately once every 200 microseconds, with an "average wait for a scan" time of approximately 100 microseconds. When the special Peripheral Processor detects an interruptible condition, it sets a flag in a control portion of Central Memory. The Executive/Monitor, in its scan cycle, examines this control area for flags. Upon interpretation of the flag, it exchange jumps (i.e., interrupts) the running Central Processor program and transfers control to an interrupt subroutine. Upon completion of the interrupt subroutine, control is returned to the interrupted Central Processor program.

The second method for handling real-time interrupt situations also employs a special ASPER program as described. However, action upon receiving an Interrupt signal is not dependent on Executive/Monitor action; hence, entry into the interrupt subroutine is faster. Figure 5-1, with the accompanying text, illustrates this method.

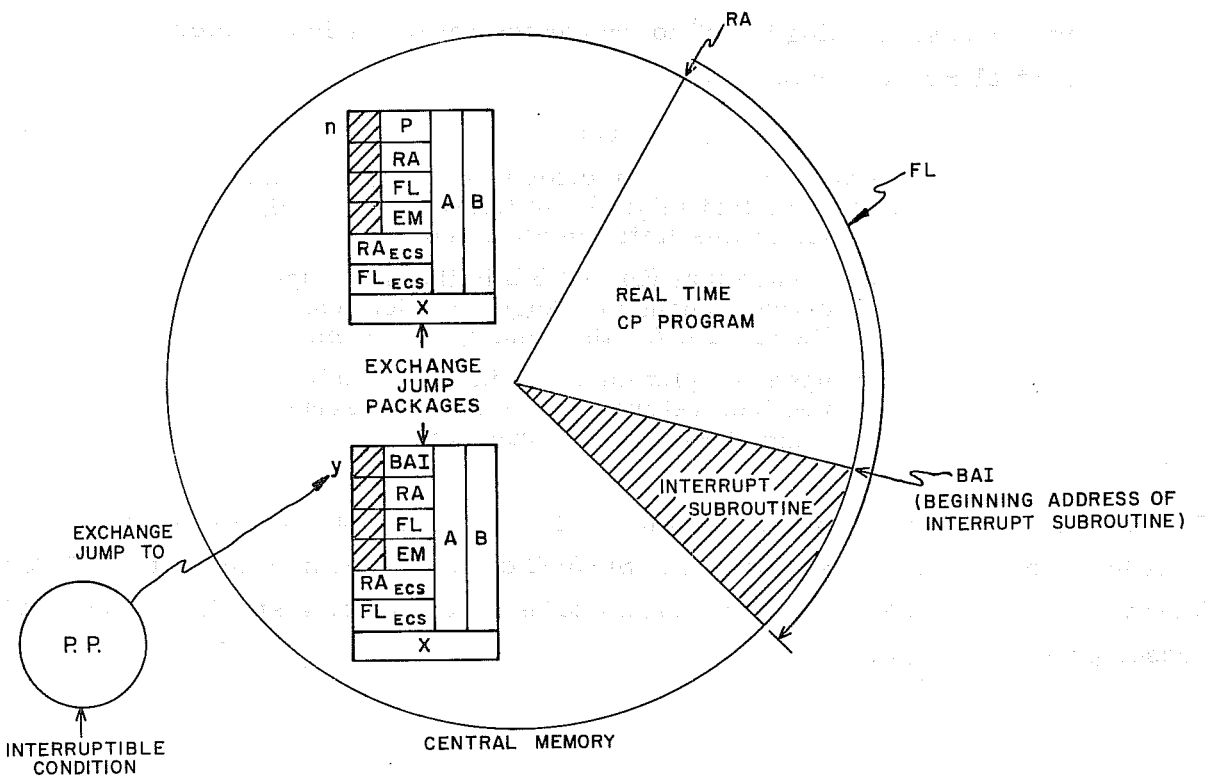


Figure 5-1. Real-Time Interrupt (ASPER Program Controlled)

When an interruptible condition is detected by the specially assigned Peripheral Processor (with its special ASPER program), the ASPER program itself initiates an Exchange Jump. The Exchange Jump is to a special (pre-stored) Exchange Jump package beginning at address "y". Within this package, RA and FL are identical to RA and FL for the running Central Processor program: X, B, and A are set to values desired upon entry into the interrupt subroutine. The portion of the package normally holding a P value holds BAI (the beginning address of the interrupt subroutine). Thus, upon completion of the Exchange Jump, control is transferred from the Central Processor program to an interrupt subroutine beginning at BAI. The average time required to enter an interrupt subroutine using this method is approximately 5-10 microseconds.

Using this method, the Peripheral Processor which interrupted the Central Processor program also returns control to that program by the following means:

a) the interrupt subroutine indicates that it is complete to a common control area in Central Memory, and

b) the Peripheral Processor examines this control area and responds by exchange jumping to "y" to return control to the interrupted Central Processor program.

NOTE

To preserve Executive/Monitor integrity (and other jobs that might be in Central Memory), the user of this latter method should:

1) ensure that RA and FL in the interrupt entry exchange package are identical to RA and FL for the running program.

2) upon completion of the interrupt subroutine, return control to the interrupted Central Processor program.

The Central Processor program, in this real-time interrupt processing case, is also afforded protection from reallocation. SIPROS does not permit reallocation of Central Memory if the Central Processor program to be relocated has a special Peripheral Processor program in operation.

6. MANUAL CONTROL

INTRODUCTION

Manual control of 6000 Series 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 which may be set manually and read by processor 0 as twelve 12-bit words. With the MODE switch in LOAD position, turning on the DEAD START switch* initiates the Dead Start operations:

1) Load the 12 words from the toggle switches into memory locations 0001 - 0014_8 of processor 0.
2) Assign processors 0 - 11_8 to corresponding data channels.
3) Set all processors to input instruction 71.
4) Set all channels to active and empty (ready for input).

After the program is read from the dead start panel, the panel is automatically disconnected and processor 0 begins executing the program. The program from the dead start panel is normally a load routine used to load a larger program from an input device such as a disk file or magnetic tape.

CONSOLE

The display console (Figure 6-2) consists of two cathode ray tube displays and a keyboard for manual entry of data. A typical 6000 Series system may have several display consoles for controlling independent programs simultaneously.

*The DEAD START switch is turned on momentarily, then off.

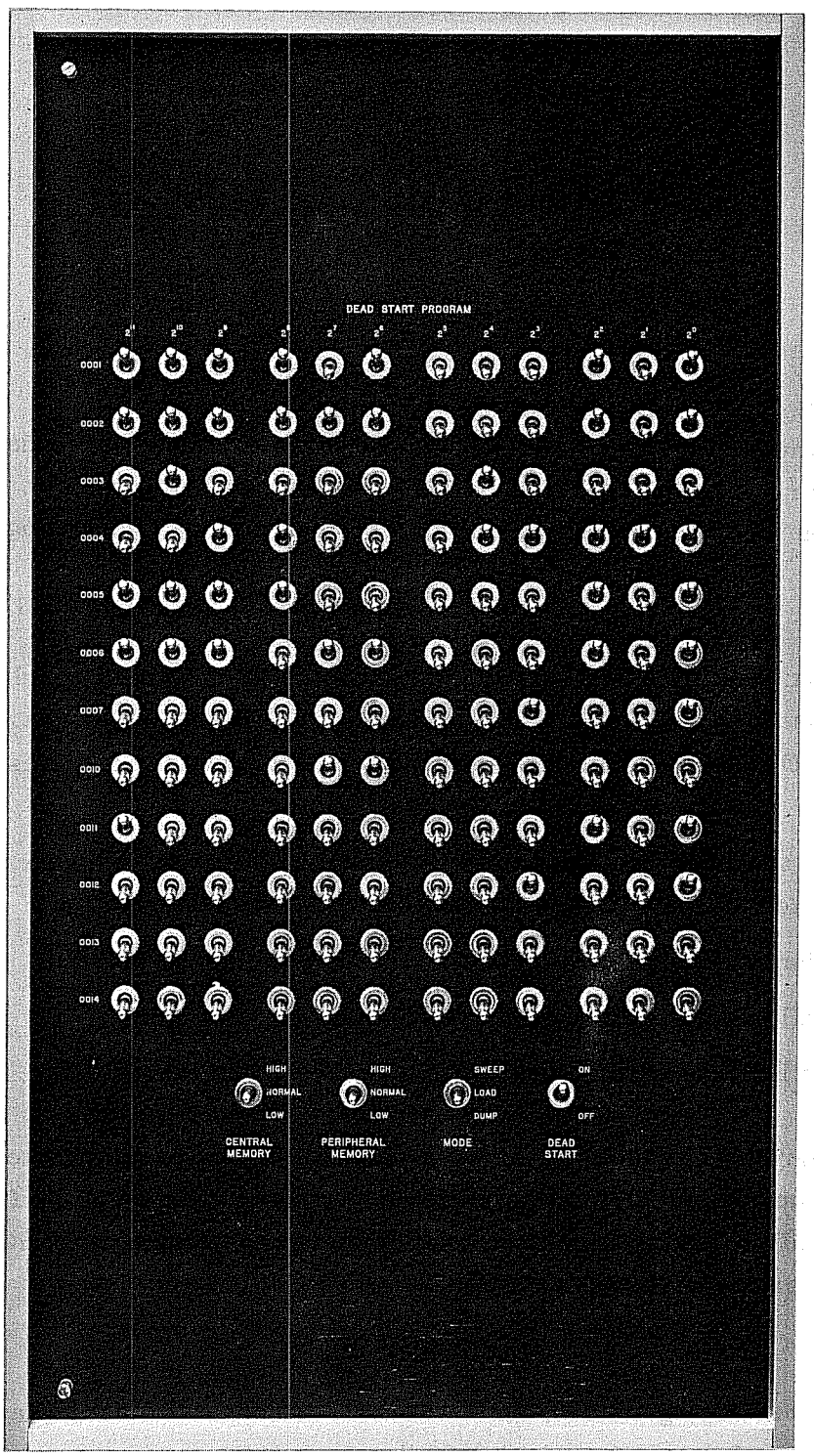
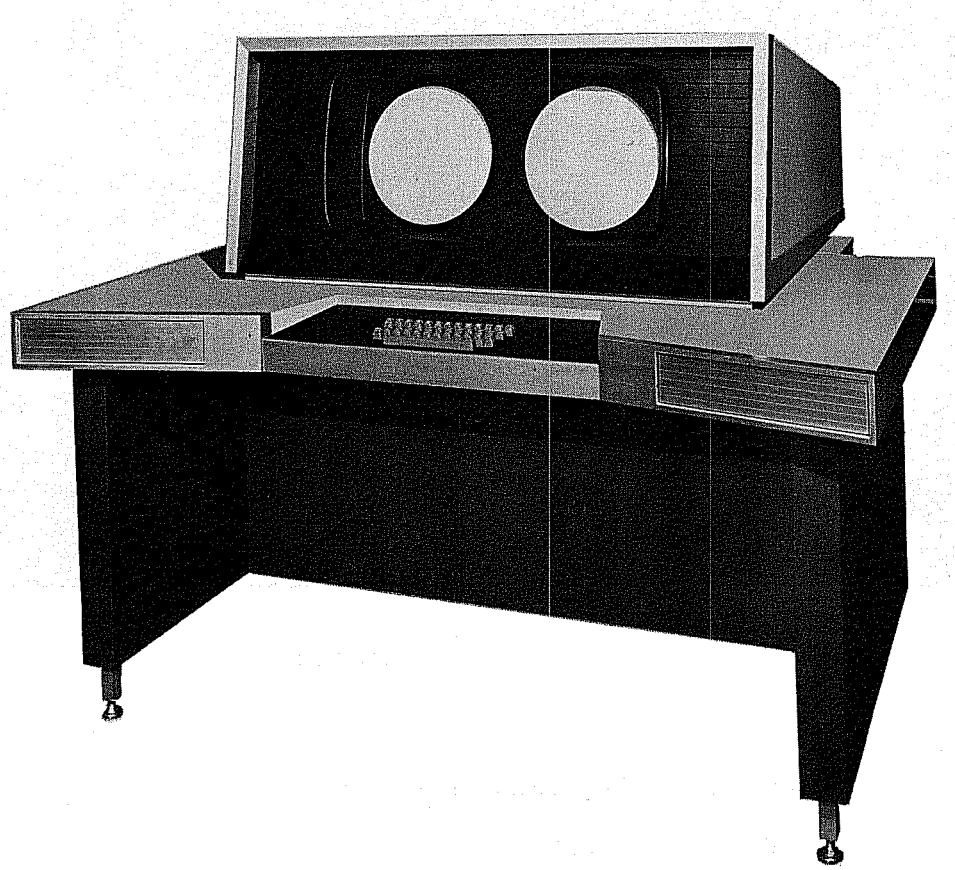


Figure 6-1. Dead Start Panel



1371

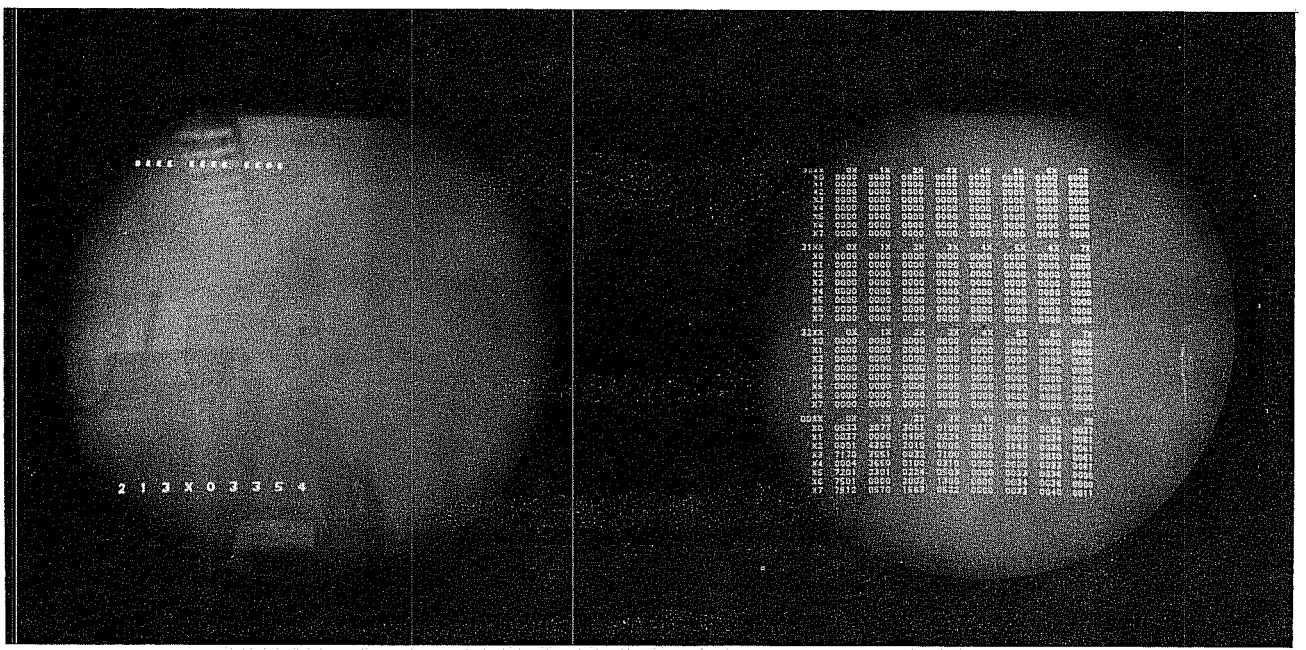
Figure 6-2. Display Console

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



1586

Figure 6-3. Sample Display

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.

Appendix A

AUGMENTED I/O BUFFER AND CONTROL (6411)

CONTROL DATA 6411

AUGMENTED I/O BUFFER AND CONTROL

The CONTROL DATA 6411 Augmented I/O Buffer and Control unit is a large-scale, solid state device for communication with the Central Processor of 6400 and 6600 Computer Systems.

DESCRIPTION

The 6411 is comprised of ten Peripheral and Control Processors and a Central Memory. A summary of characteristics for the 6411 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 (Mass Memory) 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 6411 has no Central Processor; otherwise, the 6411 is identical to the 6400 and 6600 Computer Systems. The discussion which follows assumes use of the 6411 in a 6000 Series system; the 6411, however, is a computer capable of operating alone.

SYSTEMS CONFIGURATIONS

The 6411, in typical systems configurations, provides an extremely useful and powerful 6000 Series system expansion. For installations with multiple on-line users, the 6411 provides additional data channels facilitating additional external equipments. The 10 Peripheral and Control Processors, each capable of independently executing programs, and the 16, 384 60-bit Central Memory significantly increase the multiprogramming and batch job processing capabilities of the 6400 and 6600 Computer Systems.

A typical configuration diagrammed in Figure A-1 illustrates the orientation of a 6411 with a 6400 or 6600 Computer System. The 6411 is attached to the 6400 or 6600 system via one of the Peripheral Processor Data Channels.

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

In this configuration, the 6411 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.

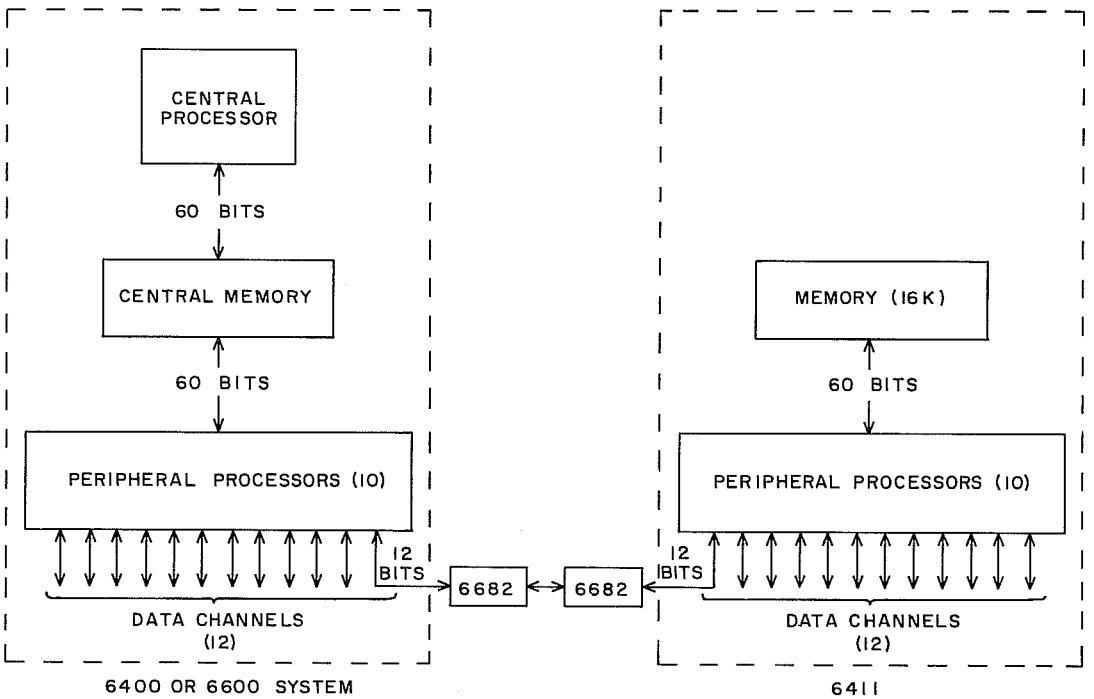


Figure A-1. Typical Configuration: 6411 with 6400 or 6600 System

Another possible systems configuration (Figure A-2) incorporates a Mass Memory between the 6400 or 6600 Central Memory and the 6411 16K memory. This configuration implies a hierarchy of memories as follows:

1) Mass Memory as a system Central Memory
2) 6400 or 6600 Central Memory as a system Central Processor memory
3) 6411 16K memory as a distributive memory

Communication with Mass Memory (Figure A-2) is accomplished as follows:

1) Read and Write instructions in the 6400/6600 Central Processor initiate transfers between Mass Memory and the 6400/6600 Central Memory.
2) An Exchange Jump instruction in the 6411 Peripheral Processor initiates Read and Write operations between Mass Memory and the 6411 16K memory. (Refer to the instruction descriptions which follow.)

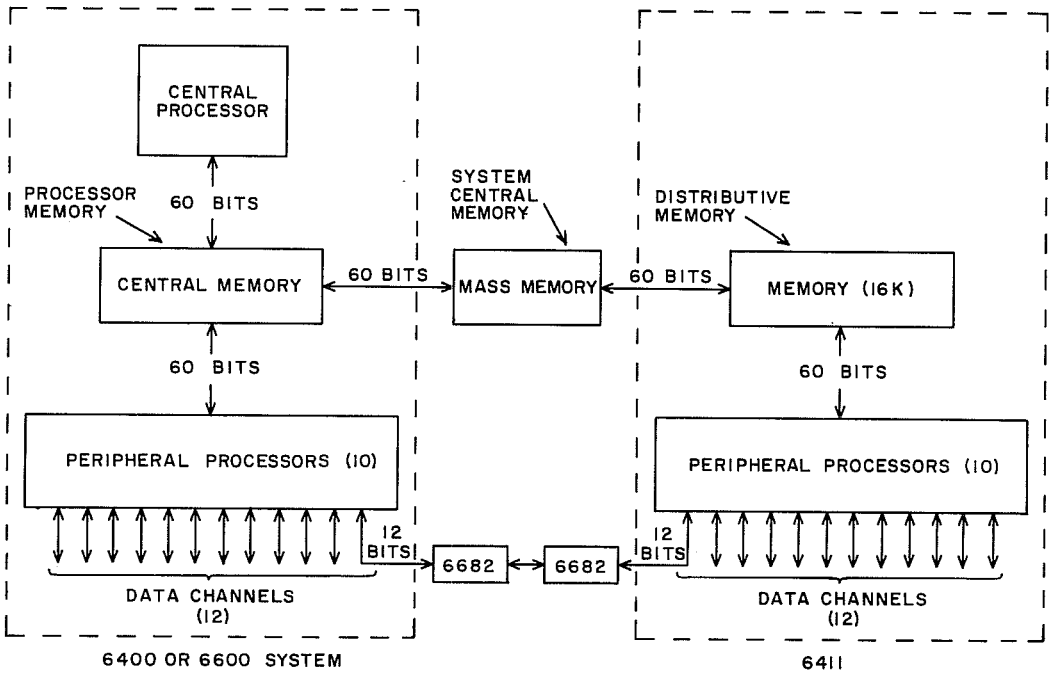
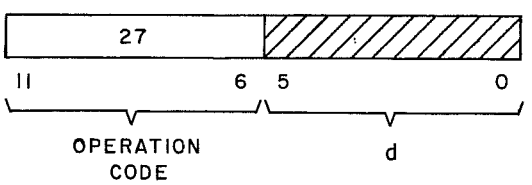


Figure A-2. Typical Configuration with Mass Memory.

6411 INSTRUCTIONS

Within the 6411, Peripheral Processor instructions are identical to those of the 6400 and 6600 systems with two exceptions. Note that these two instructions (the exceptions) are meaningful only when Mass Memory is attached to the system.

27 d Read Program Address



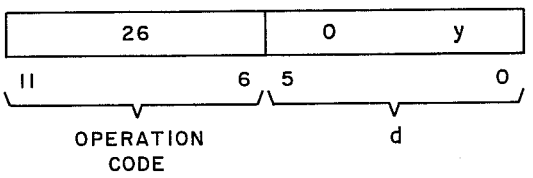
This instruction examines the status of the data trunk between the 6411 16K memory and Mass Memory. If this data trunk is busy (a Read or a Write is in progress), a "1" (Busy) flag is placed in the Peripheral Processor A register. If the trunk is free (Not Busy), the A register remains cleared. The "d" portion of this instruction is ignored.

NOTE

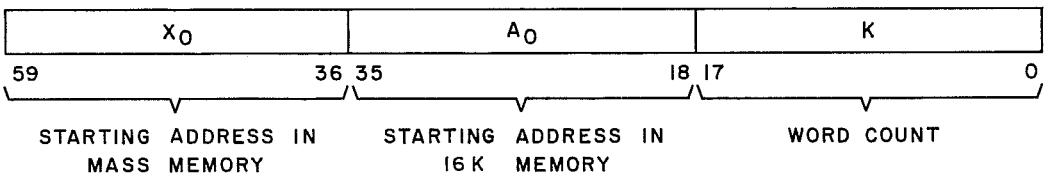
If this instruction is executed without mass memory in the system configuration, it acts as a Pass (Do-Nothing) instruction.

After executing this instruction, the program typically tests the A register for zero and transfers control to an instruction which initiates memory operations.

26 d Exchange Jump



Execution of the Exchange Jump instruction initiates memory operations by transmitting an 18-bit address, "n", from the Peripheral Processor A register to the 6411 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 "y" = 0, Read "K" words from Mass Memory into 16K memory.
If "y" = 1, Write "K" words from 16K memory into Mass Memory.

NOTE

If this instruction is executed without mass memory 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 such as SIPROS 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 Mass Memory.) The 6411 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.

SOFTWARE

The 6411, in a 6400 or 6600 system configuration, operates under control of the Simultaneous Processing Operating System (SIPROS).

Under SIPROS control, one of the 6411 Peripheral Processors is designated as an Executive/Monitor which cooperates with (and is subservient to) the 6400/6600 system Executive/Monitor in assigning and monitoring systems tasks. Thus, I/O functions (for example) in support of operational and system programs can be assigned to 6400/6600 Peripheral Processors or to 6411 Peripheral Processors.

For additional, more definitive information on systems software, consult the software manuals.

Appendix B

POWERS OF TWO

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625

Appendix C

OCTAL-DECIMAL INTEGER CONVERSION TABLE

OCTAL-DECIMAL INTEGER CONVERSION TABLE

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

0000	0000
to	to
0777	0511
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

1000	0512
to	to
1777	1023
(Octal)	(Decimal)

OCTAL-DECIMAL INTEGER CONVERSION TABLE (Cont'd)

2000 to 2777 (Octal) — 1024 to 1535 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2100	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1519	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

3000 to 3777 (Octal) — 1536 to 2047 (Decimal)

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

OCTAL-DECIMAL INTEGER CONVERSION TABLE (Cont'd)

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

4000	2048
to	to
4777	2559
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

5000	2560
to	to
5777	3071
(Octal)	(Decimal)

OCTAL-DECIMAL INTEGER CONVERSION TABLE (Cont'd)

6000	3072
to	to
6777	3583
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

7000	3584
to	to
7777	4095
(Octal)	(Decimal)

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3496	3497	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998.	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

Appendix D

OCTAL-DECIMAL FRACTION CONVERSION TABLE

OCTAL-DECIMAL FRACTION CONVERSION TABLE

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

OCTAL-DECIMAL FRACTION CONVERSION TABLE (Cont'd)

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

OCTAL-DECIMAL FRACTION CONVERSION TABLE (Cont'd)

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

Appendix E

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 and 6800 SYSTEMS)

The execution time of Central Processor instructions is given in minor cycles, * and instructions are grouped under the functional unit (6600 and 6800) 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.

*A minor cycle is 100 nanoseconds in the 6400 and 6600 systems; 25 nanoseconds in the 6800 system.

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 SYSTEM)

The 6400 system Central Processor has a unified Arithmetic unit, rather than separate

functional units as in the 6600 and 6800 systems. Instructions in the 6400 Central Processor, therefore, are executed in sequential fashion with little concurrency.

All execution times for instructions listed in Table E-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 E-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 E-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 the example in Figure E-1.)

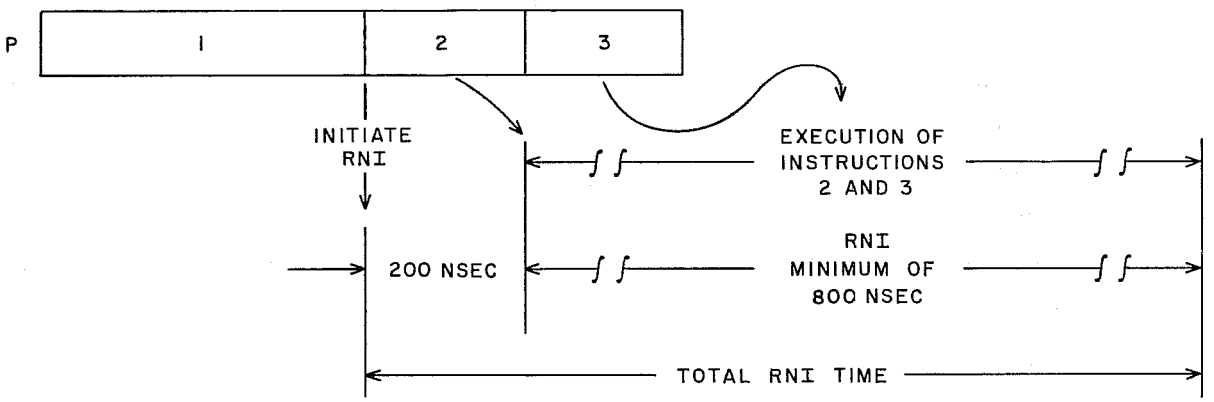


Figure E-1. RNI Timing Example

In the example diagrammed in Figure E-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 E-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 E-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 Central Processor:

- 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, 6600 and 6800 systems are tabulated in Table E-1. 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 system; its Central Processor has a unified arithmetic section.

Instruction execution times are listed in minor cycles. A minor cycle is 100 nanoseconds in the 6400 and 6600 systems and 25 nanoseconds in the 6800 system. A time listed as 4 minor cycles for the 6600 and 6800 systems means, therefore, an execution time of 400 nanoseconds (6600) and 100 nanoseconds (6800).

TABLE E-1. INSTRUCTION EXECUTION TIMES: CENTRAL PROCESSOR

Octal Code	BRANCH UNIT		6400	6600 and 6800
00	STOP		-	-
01	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 E-1. (Cont'd)

Octal Code	BOOLEAN UNIT	6400	6600 6800
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	6600 6800
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	6600 6800
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	6600 6800
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	6600 6800
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 E-1. (Cont'd)

Octal Code	DIVIDE UNIT	6400	6600 6800
44	FLOATING DIVIDE Xj by Xk to Xi	56	29
45	ROUND FLOATING DIVIDE Xj by Xk to Xi	56	29
47	SUM of 1's in Xk to Xi	68	8
46	PASS	3	1
Octal Code	INCREMENT UNIT*	6400	6600 6800
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

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 E-2. PERIPHERAL AND CONTROL PROCESSOR INSTRUCTION EXECUTION TIMES

(6400, 6600, and 6800)

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 tnat a major cycle is 1000 nanoseconds in the 6400 and 6600 systems, and 250 nanoseconds in the 6800 system.

TABLE E-2. (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
26	Exchange jump	min. 2
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
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

*Note that a major cycle is 1000 nanoseconds in the 6400 and 6600 systems, and 250 nanoseconds in the 6800 system. Note also that the shorter time is taken in certain instructions when d = 0.

TABLE E-2. (Cont'd)

OCTAL CODE	NAME	TIME* (MAJOR CYCLES
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 a major cycle is 1000 nanoseconds in the 6400 and 6600 systems, and 250 nanoseconds in the 6800 systems. Note also that the shorter time is taken in certain instructions when d = 0.

Appendix F

INDEFINITE FORMS

INDEFINITE FORMS

FLOATING ADD

$(+\infty) + (+\infty) = 377700\ldots00$

$(+\infty) + (-\infty) = 177700\ldots00$

$(-\infty) + (-\infty) = 400000\ldots00$

$(-\infty) + (+\infty) = 177700\ldots00$

$(+\infty) - (+\infty) = 177700\ldots00$

$(+\infty) - (-\infty) = 377700\ldots00$

$(-\infty) - (+\infty) = 400000\ldots00$

$(-\infty) - (-\infty) = 177700\ldots00$

$(+\infty) \pm (\pm N) = 377700\ldots00$

$(-\infty) \pm (\pm N) = 400000\ldots00$

$(\pm \text{Indef.}) \pm (\pm N) = 177700\ldots00$

$(\pm \text{Indef.}) \pm (\pm \infty) = 177700\ldots00$

$(\pm \text{Indef.}) \pm (\pm 0) = 177700\ldots00$

Underflow = 0000 (coefficient = coefficient $X_j \pm$ coefficient X_k)

Overflow on right shift one = 3777XXX...XX (coefficient positive)

4000XXX...XX (coefficient negative)

MULTIPLY

$(+\infty) \cdot (+\infty) = 377700\ldots00$

$(+\infty) \cdot (-\infty) = 400000\ldots00$

$(\pm \infty) \cdot (\pm 0) = 177700\ldots00$

$(\pm 0) \cdot (\pm 0) = 000000\ldots00$

$(\pm 0) \cdot (\pm N) = 000000\ldots00$

$(\pm \text{Indef.}) \cdot (\pm N) = 177700\ldots00$

$(\pm \text{Indef.}) \cdot (\pm \infty) = 177700\ldots00$

$(\pm \text{Indef.}) \cdot (\pm 0) = 177700\ldots00$

Underflow: (no left shift one) = 000000...00

(left shift one & sign record) = 7777 (coefficient = coefficient X_j coefficient X_k)

(left shift one & no sign record) = 0000 (coefficient = coefficient X_j coefficient X_k)

Overflow:# (sign record) = 40000...00

(no sign record) = 37700...00

Left shift one does not take the exponent out of overflow

DIVIDE

(± ∞) ÷ (± ∞) = 177700...00
(∞) ÷ (N) = 377700...00
(∞) ÷ (-N) = 400000...00
(- ∞) ÷ (N) = 400000...00
(± 0) ÷ (± ∞) = 000000...00
(± 0) ÷ (± N) = 000000...00
(± N) ÷ (± ∞) = 000000...00
(N) ÷ (0) = 377700...00
(-N) ÷ (0) = 400000...00
(N) ÷ (- 0) = 400000...00
(-N) ÷ (- 0) = 377700...00
(± Indef.) ÷ (± N) = 177700...00
(± Indef.) ÷ (± ∞) = 177700...00
(± Indef.) ÷ (± 0) = 177700...00

Underflow: # = 000000...00

Overflow: (right shift & sign record) = 4000 (coefficient = coefficient X_j coefficient X_k)

(right shift & sign record) = 3777 (coefficient = coefficient X_j coefficient X_k)

Right shift one does not take the exponent out of underflow

NORMALIZE

(+ ∞) = 3777XX...XX B_j = 000000
(- ∞) = 4000XX...XX B_j = 000000
(± Indef.) = 1777XX...XX B_j = 000000

Underflow = 0000...00 B_j = Shift count

SUPPLEMENT TO TABLE OF INDEFINITE FORMS
(Coefficient Fields for Indefinite Operands in X_j and/or X_k May Be Any Value in Any Flt. Pt. Unit)

FLOATING ADD UNIT USING 30, 31, 34 or 35 INSTRUCTION

X_j		X_k		X_i
3777000000000000000	+	3777000000000000000	=	3777000000000000000
3777000000000000000	+	4000000000000000000	=	1777000000000000000
4000000000000000000	+	4000000000000000000	=	4000000000000000000
4000000000000000000	+	3777000000000000000	=	1777000000000000000
3777000000000000000	-	3777000000000000000	=	1777000000000000000
3777000000000000000	-	4000000000000000000	=	3777000000000000000
4000000000000000000	-	3777000000000000000	=	4000000000000000000
4000000000000000000	-	4000000000000000000	=	1777000000000000000
3777000000000000000	+	1720600000000000000	=	3777000000000000000
3777000000000000000	+	6057177777777777777	=	3777000000000000000
3777000000000000000	-	1720600000000000000	=	3777000000000000000
3777000000000000000	-	6057177777777777777	=	3777000000000000000
4000000000000000000	+	1725700000000000000	=	4000000000000000000
4000000000000000000	+	6052077777777777777	=	4000000000000000000
4000000000000000000	-	1725700000000000000	=	4000000000000000000
4000000000000000000	-	6052077777777777777	=	4000000000000000000
1777000000000000000	+	1620450000000000000	=	1777000000000000000
1777000000000000000	+	6157327777777777777	=	1777000000000000000
6000000000000000000	+	1620450000000000000	=	1777000000000000000
6000000000000000000	+	6157327777777777777	=	1777000000000000000
1777000000000000000	-	1620450000000000000	=	1777000000000000000
1777000000000000000	-	6157327777777777777	=	1777000000000000000
6000000000000000000	-	1620450000000000000	=	1777000000000000000
6000000000000000000	-	6157327777777777777	=	1777000000000000000
1777000000000000000	+	3777000000000000000	=	1777000000000000000
1777000000000000000	+	4000000000000000000	=	1777000000000000000
6000000000000000000	+	3777000000000000000	=	1777000000000000000
6000000000000000000	+	4000000000000000000	=	1777000000000000000
1777000000000000000	-	3777000000000000000	=	1777000000000000000
1777000000000000000	-	4000000000000000000	=	1777000000000000000

FLOATING ADD (Cont'd)

X_j		X_k		X_i
6000000000000000000	-	3777000000000000000	=	1777000000000000000
6000000000000000000	-	4000000000000000000	=	1777000000000000000
3776540000000000000	+	3776400000000000000	=	3777460000000000000
4001237777777777777	+	4001377777777777777	=	4000317777777777777

FLOATING ADD UNIT USING 32 or 33 INSTRUCTION

0057432000000000000	+	0057540000000000000	=	0000475000000000000
7720345777777777777	+	7720237777777777777	=	7777302777777777777
0056432000000000000	+	0055540000000000000	=	0000000000000000000
7721345777777777777	+	7722237777777777777	=	0000000000000000000

MULTIPLY UNIT USING 40 or 41 INSTRUCTION

X_j		X_k		X_i
3777000000000000000	·	5777317777777777777	=	4000000000000000000
3777000000000000000	·	2000460000000000000	=	3777000000000000000
4000000000000000000	·	2000460000000000000	=	4000000000000000000
4000000000000000000	·	5777317777777777777	=	3777000000000000000
3777000000000000000	·	3777000000000000000	=	3777000000000000000
3777000000000000000	·	4000000000000000000	=	4000000000000000000
3777000000000000000	·	0000000000000000000	=	1777000000000000000
3777000000000000000	·	7777777777777777777	=	1777000000000000000
4000000000000000000	·	0000000000000000000	=	1777000000000000000
4000000000000000000	·	7777777777777777777	=	1777000000000000000
0000000000000000000	·	1715437000000000000	=	0000000000000000000
7777777777777777777	·	1715437000000000000	=	0000000000000000000
0000000000000000000	·	6062340777777777777	=	0000000000000000000
7777777777777777777	·	6062340777777777777	=	0000000000000000000
1777000000000000000	·	2060654300000000000	=	1777000000000000000
1777000000000000000	·	5717123477777777777	=	1777000000000000000
6000000000000000000	·	2060654300000000000	=	1777000000000000000
6000000000000000000	·	5717123477777777777	=	1777000000000000000
1777000000000000000	·	3777000000000000000	=	1777000000000000000
1777000000000000000	·	4000000000000000000	=	1777000000000000000
6000000000000000000	·	3777000000000000000	=	1777000000000000000
6000000000000000000	·	4000000000000000000	=	1777000000000000000
0030500000000000000	·	1627700000000000000	=	0000000000000000000
0030500000000000000	·	6150077777777777777	=	0000000000000000000
7747277777777777777	·	1627700000000000000	=	0000000000000000000
7747277777777777777	·	6150077777777777777	=	0000000000000000000
0721400000000000000	·	0777700000000000000	=	0000700000000000000
7056377777777777777	·	0777700000000000000	=	7777077777777777777
3000700000000000000	·	2717400000000000000	=	3777000000000000000
3000700000000000000	·	5060377777777777777	=	4000000000000000000

DIVIDE UNIT USING 44 OR 45 INSTRUCTION

X_j		X_k		X_i
0000000000000000000	/	0000000000000000000	=	1777000000000000000
0000000000000000000	/	7777777777777777777	=	1777000000000000000
7777777777777777777	/	0000000000000000000	=	1777000000000000000
7777777777777777777	/	7777777777777777777	=	1777000000000000000
3777000000000000000	/	3777000000000000000	=	1777000000000000000
3777000000000000000	/	4000000000000000000	=	1777000000000000000
4000000000000000000	/	3777000000000000000	=	1777000000000000000
4000000000000000000	/	4000000000000000000	=	1777000000000000000
3777000000000000000	/	20424321000000000000	=	3777000000000000000
3777000000000000000	/	573534567777777777777	=	4000000000000000000
4000000000000000000	/	20424321000000000000	=	4000000000000000000
4000000000000000000	/	5735345677777777777	=	3777000000000000000
0000000000000000000	/	3777000000000000000	=	0000000000000000000
0000000000000000000	/	4000000000000000000	=	0000000000000000000
7777777777777777777	/	3777000000000000000	=	0000000000000000000
7777777777777777777	/	4000000000000000000	=	0000000000000000000
0000000000000000000	/	17347560000000000000	=	0000000000000000000
0000000000000000000	/	60430217777777777777	=	0000000000000000000
7777777777777777777	/	17347560000000000000	=	0000000000000000000
7777777777777777777	/	60430217777777777777	=	0000000000000000000
16717400000000000000	/	3777000000000000000	=	0000000000000000000
16717400000000000000	/	4000000000000000000	=	0000000000000000000
61060377777777777777	/	3777000000000000000	=	0000000000000000000
61060377777777777777	/	4000000000000000000	=	0000000000000000000
32044540000000000000	/	0000000000000000000	=	3777000000000000000
45733237777777777777	/	0000000000000000000	=	4000000000000000000
20615567000000000000	/	7777777777777777777	=	4000000000000000000
57162210777777777777	/	7777777777777777777	=	3777000000000000000
17770000000000000000	/	17367540000000000000	=	1777000000000000000
17770000000000000000	/	60410237000000000000	=	1777000000000000000
60000000000000000000	/	17756677000000000000	=	1777000000000000000
60000000000000000000	/	60021100777777777777	=	1777000000000000000
17770000000000000000	/	3777000000000000000	=	1777000000000000000

DIVIDE (Cont'd)

X_j		X_k		X_i
1777000000000000000	/	4000000000000000000	=	1777000000000000000
6000000000000000000	/	3777000000000000000	=	1777000000000000000
6000000000000000000	/	4000000000000000000	=	1777000000000000000
0777600000000000000	/	2720400000000000000	=	0000000000000000000
3000600000000000000	/	0721400000000000000	=	3777600000000000000
4777177777777777777	/	0721400000000000000	=	4000177777777777777

NORMALIZE

X_k	B_j	X_i
3777004320000000000	000000	3777004320000000000
4000773457777777777	000000	4000773457777777777
1777000210000000000	000000	1777000210000000000
6000777567777777777	000000	6000777567777777777
0000000000000000000	000060	0000000000000000000
* 0000000000000000000	000060	0000000000000000000
0004000600000000000	000011	0000000000000000000
7777777777777777777	000060	0000000000000000000
* 7777777777777777777	000060	0000000000000000000
7773777777777777777	000011	0000000000000000000
2000000000000000000	000060	0000000000000000000
* 2000000000000000000	000060	1717400000000000000
5777777777777777777	000060	0000000000000000000
* 5777777777777777777	000060	6060377777777777777

* Results due to rounded normalize

Appendix G

DECIMAL/BINARY POSITION TABLE

DECIMAL/BINARY POSITION TABLE

Largest Decimal Integer	Decimal Digits Req'd*	Number of Binary Digits	Largest Decimal Fraction
1		1	.5
3		2	.75
7		3	.875
15	1	4	.937 5
31		5	.968 75
63		6	.984 375
127	2	7	.992 187 5
255		8	.996 093 75
511		9	.998 046 875
1 023	3	10	.999 023 437 5
2 047		11	.999 511 718 75
4 095		12	.999 755 859 375
8 191		13	.999 877 929 687 5
16 383	4	14	.999 938 964 843 75
32 767		15	.999 969 482 421 875
65 535		16	.999 984 741 210 937 5
131 071	5	17	.999 992 370 605 468 75
262 143		18	.999 996 185 302 734 375
524 287		19	.999 998 092 651 367 187 5
1 048 575	6	20	.999 999 046 325 683 593 75
2 097 151		21	.999 999 523 162 841 796 875
4 194 303		22	.999 999 761 581 420 898 437 5
8 388 607		23	.999 999 880 790 710 449 218 75
16 777 215	7	24	.999 999 940 395 355 244 609 375
33 554 431		25	.999 999 970 197 677 612 304 687 5
67 108 863		26	.999 999 985 098 838 806 152 343 75
134 217 727	8	27	.999 999 992 549 419 403 076 171 875
268 435 455		28	.999 999 996 274 709 701 538 085 937 5
536 870 911		29	.999 999 998 137 354 850 769 042 968 75
1 073 741 823	9	30	.999 999 999 068 677 425 384 521 484 375
2 147 483 647		31	.999 999 999 534 338 712 692 260 742 187 5
4 294 967 295		32	.999 999 999 767 169 356 346 130 371 093 75
8 589 934 591		33	.999 999 999 883 584 678 173 065 185 546 875
17 179 869 183	10	34	.999 999 999 941 792 339 086 532 592 773 437 5
34 359 738 367		35	.999 999 999 970 896 169 543 266 296 386 718 75
68 719 476 735		36	.999 999 999 985 448 034 771 633 148 193 359 375
137 438 953 471	11	37	.999 999 999 992 724 042 385 816 574 096 679 687 5
274 877 906 943		38	.999 999 999 996 362 021 192 908 287 048 339 843 75
549 755 813 887		39	.999 999 999 998 181 010 596 454 143 524 169 921 875
1 099 511 627 775	12	40	.999 999 999 999 090 505 298 227 071 762 084 960 937 5
2 199 023 255 551		41	.999 999 999 999 545 252 649 113 535 881 042 480 468 75
4 398 046 511 103		42	.999 999 999 999 772 626 324 556 767 940 521 240 234 375
8 796 093 022 207		43	.999 999 999 999 886 313 162 278 383 970 260 620 117 187 5
17 592 186 044 415	13	44	.999 999 999 999 943 156 581 139 191 985 130 310 058 593 75
35 184 372 088 831		45	.999 999 999 999 971 578 290 569 595 992 565 155 029 296 875
70 368 744 177 663		46	.999 999 999 999 985 789 145 284 797 996 282 577 514 648 437 5
140 737 488 355 327	14	47	.999 999 999 999 992 894 572 642 398 998 141 288 757 324 218 75

*Larger numbers within a digit group should be checked for exact number of decimal digits required.

Examples of use:

Q. What is the largest decimal value that can be expressed by 36 binary digits?
A. 68,719,476,735.

Q. How many decimal digits will be required to express a 22-bit number?
A. 7 decimal digits.

Appendix H

CONSTANTS

CONSTANTS

π	=	3.14159 26535 89793 23846 26433 83279 50
$\sqrt{3}$	=	1.732 050 807 569
$\sqrt{10}$	=	3.162 277 660 1683
e	=	2.71828 18284 59045 23536
ln 2	=	0.69314 71805 599453
ln 10	=	2.30258 50929 94045 68402
$\log_{10} 2$	=	0.30102 99956 63981
$\log_{10} e$	=	0.43429 44819 03251 82765
$\log_{10} \log_{10} e$	=	9.63778 43113 00537
$\log_{10} \pi$	=	0.49714 98726 94133 85435
1 degree	=	0.01745 32925 11943 radians
1 radian	=	57.29577 95131 degrees
$\log_{10}(5)$	=	0.69897 00043 36019
7!	=	5040
8!	=	40320
9!	=	362,880
10!	=	3,628,800
11!	=	39,916,800
12!	=	479,001,600
13!	=	6,227,020,800
14!	=	87,178,291,200
15!	=	1,307,674,368,000
16!	=	20,922,789,888,000
$\frac{\pi}{180}$	=	0.01745 32925 19943 29576 92369 07684 9
$\left(\frac{\pi}{2}\right)^2$	=	2.4674 01100 27233 96
$\left(\frac{\pi}{2}\right)^3$	=	3.8757 84585 03747 74
$\left(\frac{\pi}{2}\right)^4$	=	6.0880 68189 62515 20
$\left(\frac{\pi}{2}\right)^5$	=	9.5631 15149 54004 49
$\left(\frac{\pi}{2}\right)^6$	=	15.0217 06149 61413 07
$\left(\frac{\pi}{2}\right)^7$	=	23.5960 40842 00618 62
$\left(\frac{\pi}{2}\right)^8$	=	37.0645 72481 52567 57
$\left(\frac{\pi}{2}\right)^9$	=	58.2208 97135 63712 59
$\left(\frac{\pi}{2}\right)^{10}$	=	91.4531 71363 36231 53
$\left(\frac{\pi}{2}\right)^{11}$	=	143.6543 05651 31374 95
$\left(\frac{\pi}{2}\right)^{12}$	=	225.6516 55645 350
$\left(\frac{\pi}{2}\right)^{13}$	=	354.4527 91822 91051 47
$\left(\frac{\pi}{2}\right)^{14}$	=	556.7731 43417 624

CONSTANTS (Cont'd)

π^2	=	9.86960 44010 89358 61883 43909 9988
$2\pi^2$	=	19.73920 88021 78717 23766 87819 9976
$3\pi^2$	=	29.60881 32032 68075 85680 31729 9964
$4\pi^2$	=	39.47841 76043 57434 47533 75639 9952
$5\pi^2$	=	49.34802 20054 46793 09417 19549 9940
$6\pi^2$	=	59.21762 64065 36151 71300 63459 9928
$7\pi^2$	=	69.08723 08076 25510 33184 07369 9916
$8\pi^2$	=	78.95683 52087 14868 95067 51279 9904
$9\pi^2$	=	88.82643 96098 04227 56950 95189 9892

$\sqrt{2}$	=	1.414 213 562 373 095 048 801 688
$1 + \sqrt{2}$	=	2.414 213 562 373 095 048 801 688
$(1 + \sqrt{2})^2$	=	5.828 427 124 746 18
$(1 + \sqrt{2})^4$	=	33.970 562 748 477 08
$(1 + \sqrt{2})^6$	=	197.994 949 366 116 30
$(1 + \sqrt{2})^8$	=	1153.999 133 448 220 72
$(1 + \sqrt{2})^{10}$	=	6725.999 851 323 208 02
$(1 + \sqrt{2})^{12}$	=	39201.999 974 491 027 40
$(1 + \sqrt{2})^{14}$	=	228485.999 995 622 956 38
$(1 + \sqrt{2})^{16}$	=	1331713.999 999 246 711
$(1 + \sqrt{2})^{18}$	=	7761797.999 999 884 751

Sin .5	=	0.47942 55386 04203
Cos .5	=	0.87758 25618 90373
Tan .5	=	0.54630 24898 43790
Sin 1	=	0.84147 09848 07896
Cos 1	=	0.54030 23058 68140
Tan 1	=	1.55740 77246 5490
Sin 1.5	=	0.99749 49866 04054
Cos 1.5	=	0.07073 72016 67708
Tan 1.5	=	14.10141 99471 707

CUT ALONG LINE

COMMENT SHEET

6000 SERIES COMPUTER SYSTEM

Reference Manual

Pub. No. 60100000

FROM NAME : _____

BUSINESS ADDRESS : _____

COMMENTS: (DESCRIBE ERRORS, SUGGESTED ADDITION OR DELETION AND INCLUDE PAGE NUMBER, ETC.)

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