Computer Information Center

Computation Department

LTSS Livermore Time-Sharing System

Part I: OCTOPUS

Chapter 1: HARDWARE

John Fletcher, et al.

March 10, 1970 Edition - 2

Lawrence Radiation Laboratory
University of California/Livermore

LTSS-1
Edition 2

LIVERMORE TIME SHARING SYSTEM

Part 1: Octopus

Chapter 1: Hardware

University of California
Lawrence Radiation Laboratory
Livermore, California 94550

HARDWARE

CIC-LTSS-1-ED. 2

Authors: John G. Fletcher, J. Dennis Lawrence

March 10, 1970

If you wish to be placed on a mailing list for LTSS-related documents, please send your name and L-code to:

LTSS LIST

COMPUTER INFORMATION CENTER

L-60

Acknowledgment: My thanks to Harry L. Nelson for carefully verifying the technical accuracy of sections 1.2 and 1.3. -- J.D.L.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued) Page

ILLUSTRATIONS

1. HARDWARE

1.1. THE PDP-6 COMPUTER SYSTEM†

1.1.1. Hardware Components

The Octopus computer complex is centered on a "head" consisting of the following devices (illustrated in Figure 1.1):

A. Two Digital Equipment Corporation (DEC) Programmed Data Processor-6 (PDP-6) computers,[1] modified locally to include paged and segmented addressing. These are described more fully below, in Sections 1.1.2.-1.1.6.

B. 256 K (where K = 1024) 36-bit words of high-speed random-access magnetic core memory. This memory consists of 16 "boxes" of 16 K words each. Eight of these boxes are products of the Lockheed Electronics Company (LEC), six of Ampex and two of DEC. The DEC memories have a 1.75 microsecond cycle time; the others are one microsecond (see Section 1.1.7.).

C. One General Precision Librascope (GPL) Disc File with a capacity of 0.807 billion bits (22.4 million words), and a transfer rate of 20.35 megabits/second. This is a fixed head disc; all access delays are rotational in origin (averaging about 35 milliseconds).

D. One International Business Machines (IBM) Data Cell with a capacity of 3.24 billion bits (90 million words) and a transfer rate of 324 kilobits/second (reading) or 162 kilobits/second (writing). Random access averages half a second.

E. One IBM Photo-Digital Storage Device with a capacity of 1.02 trillion bits (28.3 billion words), and a transfer rate of 1.5 megabits/second (reading) or 255 kilobits/second (writing). Random access is about 5 seconds.

F. A number of line units, which are interfaces to devices which lie on the Octopus "tentacles", enabling those devides to directly read or write (under PDP-6 control) the core memory and to interrupt a processor. Currently, each CDC-6600 and CDC-7600 is connected to a line unit.

---

†Caution: The PDP-6's may be replaced with PDP-10's in the near future. This will render this entire discussion obsolete.

G. A pair of I/O busses enabling the processors to send or receive data or control information to or from the disc, data cell, photo store, line units, and other devices. These other devices include a clock, PDP-8 computers which multiplex teletypewriters, a switch which can rearrange I/O bus connections, two magnetic tape transports, an IBM 1401 computer, a card reader-punch, a paper tape reader, two console teletypewriters, and a television monitor display system (TMDS).

H. A video display which can examine memory and is used for maintenance and debugging.

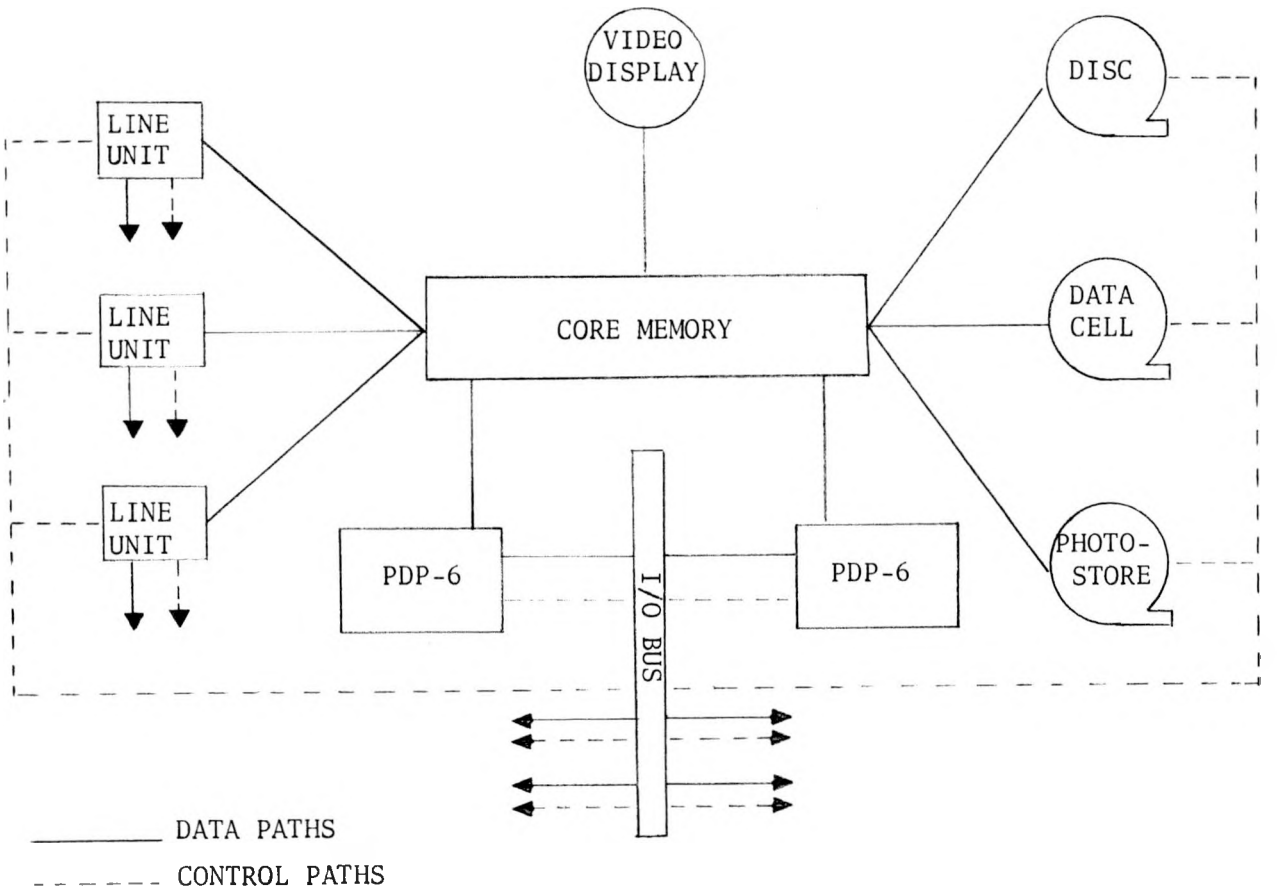


Figure 1.1. PDP-6 Hardware

1.1.2. Processor Registers

Each PDP-6 processor executes independently of the other. The only communication between them is through the shared core memory and by means of a mutual interrupt facility described in Section 1.1.6. A PDP-6 does not execute in synchronism with any clock; rather, each instruction proceeds as fast as circuitry and memory availability will allow. The typical instruction takes about 5 microseconds.

The following registers characterize the state of execution of a process:

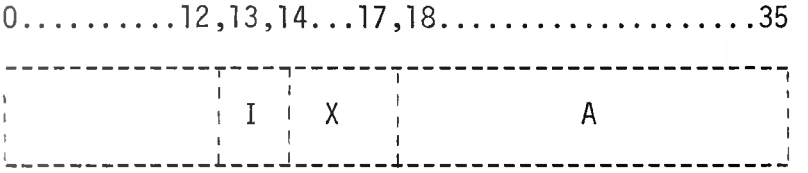
A. One program counter which holds the 18-bit effective address of the instruction next to be fetched.
B. Sixteen accumulators, each of which can hold a 36-bit word. These registers are addressable as core locations 0 through 15. Except for accumulator 0, their right halves are also used as index registers.
C. A number of registers used by the pagination-segmentation hardware, which are described more fully in Section 1.1.4. They include a descriptor base register, eight address base registers, two interrupt summary registers, and eight associative memory registers.
D. Nine flags, bits indicative of various processor events and states, including arithmetic overflow and carry (3 bits), jump or skip occurrence (1 bit) byte manipulation state (1 bit), processor mode (2 bits), and optional interrupt enables (2 bits). Further information of a similar character, but including enough information to diagnose the source of processor-generated interrupts, is obtained through the use of special I/O instructions.
E. Internal registers which retain the state of the interrupt channels, which are discussed in Section 1.1.5.

The PDP-6 repertoire of 393 instructions directly manipulates the contents of these registers (notably the accumulators) and the contents of memory words. The instructions are executed sequentially without significant overlap or look-ahead.

1.1.3. Word Formats

Certain uses of words by the processors (e.g., as operands of move, Boolean, and byte manipulation instructions) regard them as bit patterns and assume no special format. Other uses, however, interpret them according to the formats now summarized.

A. Under various circumstances a word may be used to generate an *effective address*. (The word "generate" is used to distinguish this situation from one in which a word is merely interpreted as holding an effective address in one or both halves.) The format is



The effective address is generated as follows. The A (address) field is added to the contents of the index register (right half of the accumulator) specified by the X-field (unless zero is specified, in which case A is used directly) to obtain an indirect address. This is also the effective address unless the I (indirect) bit is one, in which case the word specified by the indirect address is read and is used to generate a new indirect address; the effective address is finally obtained only when a word with the I bit equal to zero is encountered. If the chain of indirect addresses loops on itself, the processor will remain in the loop until interrupted. Bits 0-12 are never involved in effective address generation.

B. All instructions without exception generate an effective address as explained above; this address either is used to specify a word from which an operand is to be fetched and/or in which a result is to be stored, or is used itself as the right half of an operand having zeros in its left half. Bits 0-12 of the instruction are interpreted as follows.

1. If bits 0-2 are all zeros, the instruction is a *programmed operator*. This instruction stores bits 0-12 of itself and the effective address it generates into core location 32 and then

executes the instruction in location 33. It is intended that this be a jump to a subroutine which interpretively executes the information in 32, thus providing a means of extending the instruction repertoire.

2. If bits 0-2 are all ones, the instruction is one of eight I/O instructions as specified by bits 10-12. Bits 3-9 specify one of a possible 128 devices (connected to the I/O bus) to which the instruction may refer.
3. In all other cases, bits 0-8 specify one of 384 instructions. Usually the separate bits may be individually understood; for example, if bits 0-2 are 110, an accumulator mask and test instruction is indicated and bits 3-8 are respectively interpreted as complement masked bits, clear masked bits, mask with contents of effective address(rather than with the address itself), skip, reverse skip selection if masked bits are all zero, and swap halves of word before masking. Except for two jump instructions, for which they act as modifiers of the meaning of bits 0-8, bits 9-12 select an accumulator (or, in a few cases, a pagination-segmentation register) which contains an operand and/or is a repository for a result.

C. A *byte pointer* is a word used in connection with certain instructions in order to select a byte. The effective address generated by this word locates the word in which the byte is to be found, while bits 0-5 describe the position of the byte in the word and bits 6-11 give the byte size, which may range from 0 to 36. Bit 12 is unused.

D. All integer operands and results are two's complement numbers. That is, the 36-bit binary number implied by the bit pattern is interpreted directly if bit 0 is zero. Otherwise this number is interpreted as the negative number found by subtracting 2^{36}. It should be noted that there is, therefore, only one representation of the integer zero.

E. Floating point operands and results use bits 1-8 as a binary exponent plus 128, and bits 9-35 as a fraction. Bit 0 is zero for positive numbers. Negative numbers are the two's complement of the corresponding positive number. Both floating and fixed point zeros have

all bits zero. It follows that the same arithmetic comparison instructions may be used for fixed and floating point numbers.

1.1.4. Pagination - Segmentation

The PDP-6 uses two distinct schemes by which effective addresses (such as those generated by instructions) are translated into actual addresses (in core memory). The first scheme, *absolute addressing*, simply treats the effective address as the actual address. This scheme is used only when the processor is executing in *executive mode*, a mode intended to be reserved to the operating system itself. All computations (user problems) should be permitted to run only under the second scheme, *pagination-segmentation*.[2] The complete rationale behind this rather involved scheme, now described, will be made clear when its role in an operating system is discussed (see Section 2.1.).

A computation will have access to a *virtual memory* of 2^{22} words. The 22-bit *virtual address* of one of these words may be analyzed into a 7-bit (high-order) *segment number* specifying one of 128 *segments* into which the virtual memory is divided, and a 15-bit (low-order) *word number* specifying a word within the segment. The contents of each of the 22-bit *address base registers* (ABR) is interpreted as a virtual address. An (18-bit) effective address, when it is used to specify a memory word, is converted into a virtual address as follows (except for addresses 0-15 which always specify an accumulator). The high-order three bits of the effective address select an ABR. The word number in this register (called the *augment*) is added to the low-order fifteen bits of the effective address (called the *offset*) to produce a word number for the resultant virtual address. The segment number in the ABR becomes the segment number of the resultant address. The intention is that the ABR contain an entry point address (or pointer to an array) which serves as a reference point from which the offset is measured.

Address base register 0 is called the *procedure base register* (PBR), and always contains a zero augment. The program counter always has its high-order three bits equal to zero (except in executive mode), and thus selects this register to determine the segment containing instructions. Jump instructions reload the PBR if the jump is out of segment; a special pair of jump

instructions save and restore this register, thus permitting ready return from subroutines.

Every segment is divided into *pages* of a uniform size which may be 128, 512, 2 K, or 8 K words. A segment is brought into actual (core) memory a page at a time from the disc. A core table (the page table), associated with each segment that has at least one page in core, has one entry for each page of the segment. This entry contains the actual core address of the page or shows it to not be in core. A segment is limited to 32 pages; hence segments using the smaller page sizes cannot have the maximum 32 K words.

Every active computation is associated with a core table (the segment table), with an entry for each segment in the virtual memory of the computation. The entry contains the actual core address of the page table for that segment, or shows that the page table (and hence every page) for the segment is not in core. Several segment tables may point to the same page table; segments may be shared. The *descriptor base register* (DBR) contains the address of the segment table associated with the executing process.

Thus at each memory reference, the processor hardware, as well as converting the effective address into a virtual address as described above, goes to a segment table and a page table and finds the actual address of the desired word (see Figure 1.2.). If for any reason the actual address cannot be found (such as, if the page is not in core), the instruction execution is aborted and the processor interrupted (see Section 1.1.6.). The two *interrupt summary registers* then contain information which enables the operating system to diagnose and, if possible, correct the difficulty (such as by retrieving the page from disc — time shared execution of other processes meanwhile continuing).

The processor has eight *associative memory registers* which contain the segment and page numbers of eight pages together with the corresponding page table entry. At each memory reference, these registers are interrogated in parallel by the hardware to find whether they hold information about the referenced page; if so, the two memory references to the segment and page tables may be bypassed. Only because of these registers can paging be efficient. The associative memory registers are ranked by the hardware according to

recentness of use; every time segment and page tables are actually referenced, the page table information obtained replaces the information in the least recently used register. Clearly, programs which "jump around" will be less efficient than those which do not.

Paging has the advantages that programs need not occupy contiguous core and need not be fully loaded in order to execute, thus increasing the efficiency of core utilization.

1.1.5. Protection

In addition to executive mode, the processors have two other modes, in both of which pagination and segmentation are effective. One of these modes, the *IOT mode*, is intended to be used only by the operating system, since, except for the addressing scheme, it differs little from executive mode. The other mode, *user mode*, is intended for general use. In this mode the following things, possible in executive mode, are illegal and cause an interrupt (thus shifting control to the operating system, see Section 1.1.6.).

A. Memory references are limited to areas located by the chain of pointers: DBR to segment table to page table to page. Further, the kinds of references may be limited. Each page table entry contains three bits, the read (R), write (W), and execute (X) bits. If and only if the corresponding bit is set may the words of the page be referenced to fetch data (R), store results (W), or fetch instructions (X). (All references to accumulators are permitted.) Thus, shared segments will usually not have "W" access. It is assumed that segment and page tables and locations 32-47, if they are accessible at all in user mode, occur in pages with "W" access denied.

B. All I/O instructions are forbidden.

C. Certain other instructions are also forbidden, such as those which load the descriptor base register, change the processor mode, or halt the processor. It is assumed that the instruction in location 33 is carefully chosen.

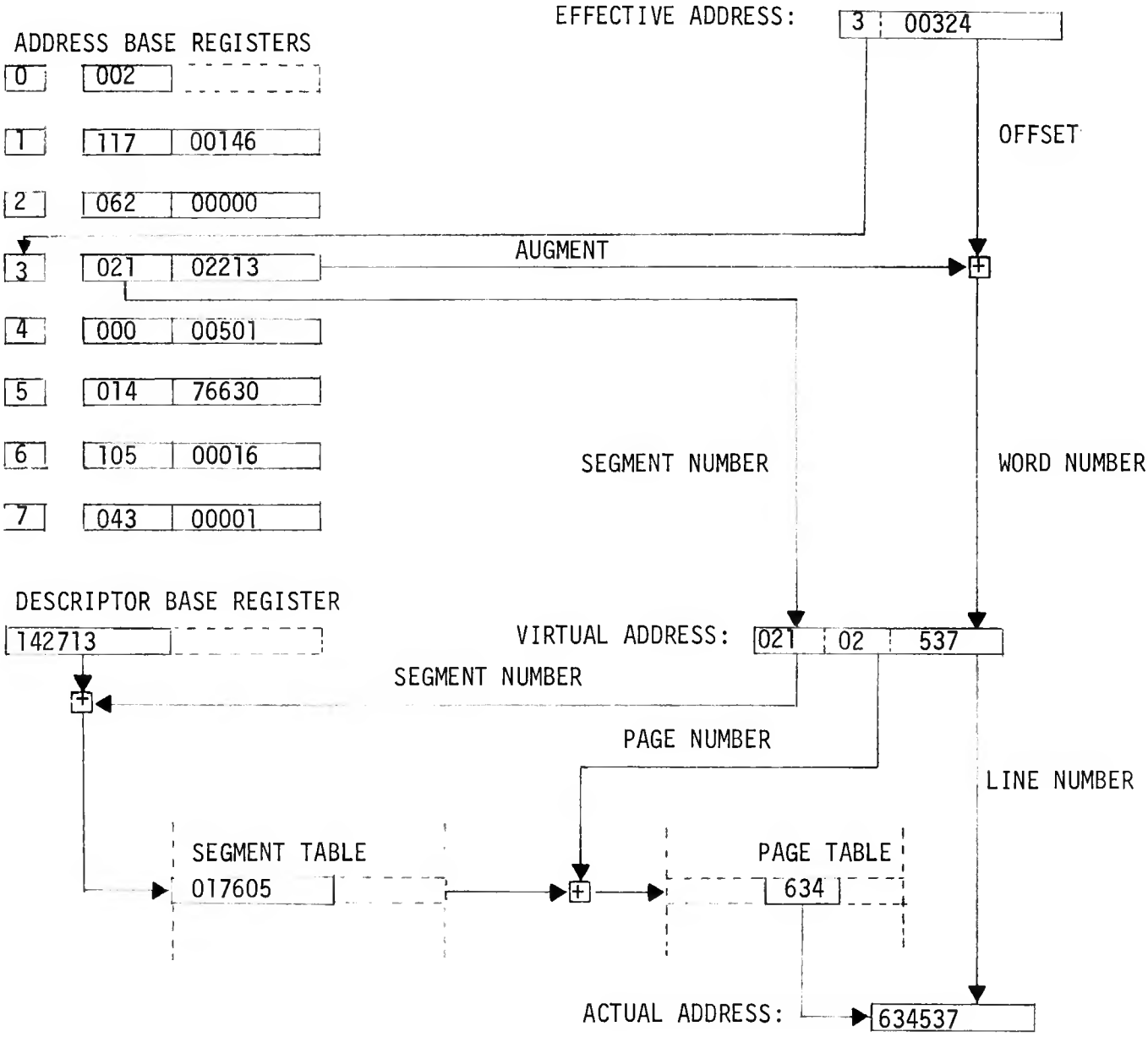


Figure 1.2. Pagination-Segmentation Hardware Operation
(For 512 Word Pages)

1.1.6. Interrupt Capability

Each processor has seven interrupt channels. Each device on the I/O bus may be connected to one of the channels by a suitable I/O instruction. The processor itself is considered to be device 0 and thus may (and should) be connected to a channel so that processor interrupts may be effective; these include push-down-list overflow, arithmetic overflow (if enabled), program counter skip or jump (if enabled for flow tracing), pagination-segmentation faults, nonexistent memory (because of faulty or missing box), and memory parity failure. The seven channels are ranked in priority from 1 (high) to 7 (low); any number of devices may be on a single channel. A suitable I/O instruction can clear, turn on, or turn off the entire interrupt system, can turn on or off any channel, and can request an interrupt on any channel. Also, a suitable I/O instruction to the I/O bus switch device will cause it to interrupt the other processor.

When an interrupt is requested on a channel (either by a device connected to that channel with the channel on or by the special I/O instruction just mentioned), the interrupt begins as soon as the current instruction reaches a suitable point and all higher priority channels are dismissed. This beginning of the interrupt consists of the execution of instruction at location (32+2n), where n is the channel number. This instruction should be carefully chosen. Usually it is a jump to subroutine, entering executive mode and saving the program counter (which may be that of a lower priority channel). This subroutine should determine the cause of the interrupt (by checking the status of all devices on that channel), service the needs indicated (saving and restoring any accumulators or other registers used), reset the interrupting devices to a noninterrupting status, and then return to the interrupted routine with a special jump instruction which dismisses the channel.

1.1.7. Core Memory

The core memories are accessible through six *ports*. Two of these ports are attached to the processors; the others to multiplexors to which several devices can be attached. Priority schemes in the porting structure and in the multiplexors select which device desiring access to a given box gets the

next memory cycle. Accesses to different boxes do not conflict. Clearly, the port and multiplexor connection of each device must be made with full consideration of its speed and its ability to survive delay without data loss. Consecutive addresses may be optionally placed in a single box or alternated between two interlaced boxes.

An important feature is that the processors may - on a given single access - not only read or write the memory, but may also read the memory and rewrite the same word with different data — no other processor or device being able to access the word in the interim. Many instructions exploit this feature. The result is vast simplifications in the programming required to handle certain "close call" situations when two processors share a common memory.

1.2. THE CDC 6600 COMPUTER SYSTEM

1.2.1. System Organization

The Control Data Corporation 6600 computer system[3] consists of eleven independent computers, a shared central memory, and twelve input/output data channels (Figure 1.3.). Ten of these eleven computers are called *peripheral and control processing units* (PPU); the eleventh is termed the *central processing unit* (CPU).

The CPU is a high speed arithmetic device with access to the central memory. Each PPU has its own memory, and can also access the central memory. A PPU may execute a program independently of the CPU and the other nine PPU's, may control and start the CPU, and may transfer data between the central memory and any of the twelve data channels.

The 6600 computer is capable of concurrent operations of three types: program execution, CPU functional unit operation, and memory access. Concurrent program execution may occur when the CPU and any PPU's are operating simultaneously, as described above. The other two types will be discussed below.

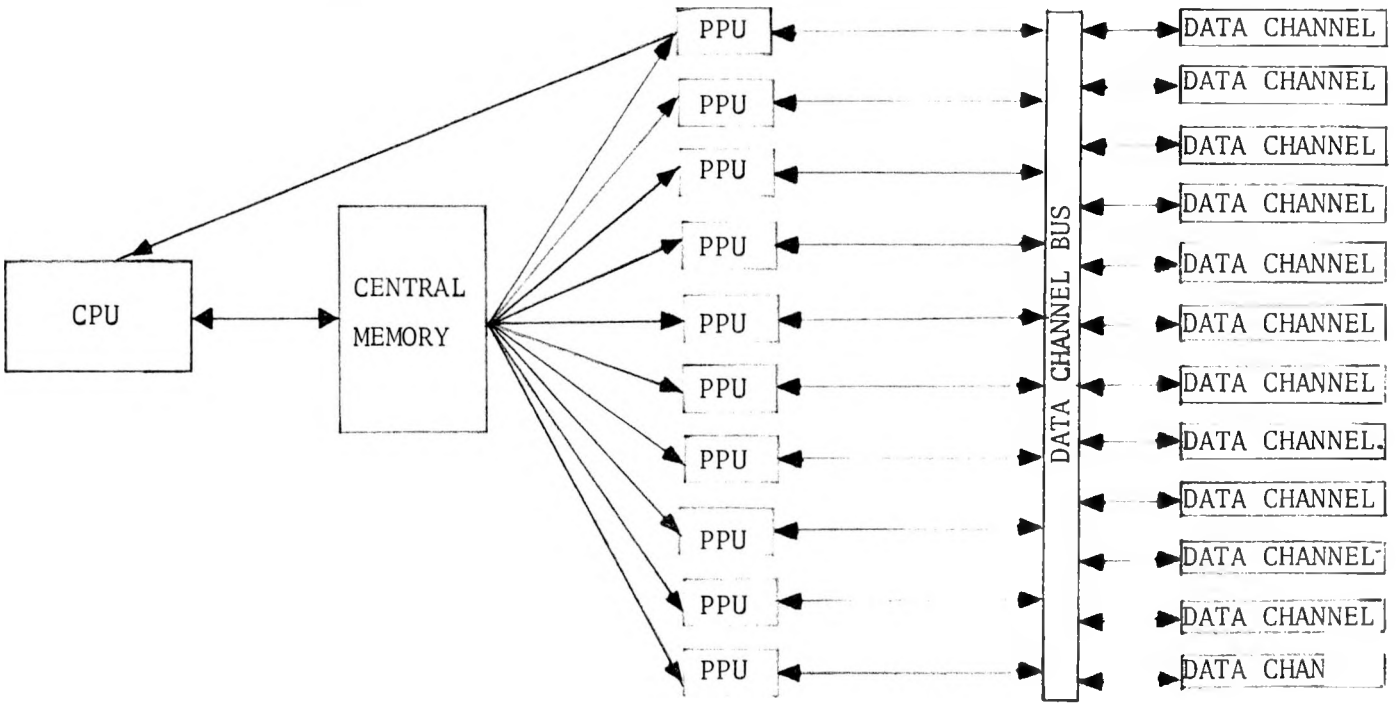


Figure 1.3. CDC 6600 Computer System

1.2.2. Hardware Components

The Lawrence Radiation Laboratory presently has three 6600 computing systems in the time sharing system. The equipment for each system is very similar to the other two; the components of a typical system are given below.

A. One CDC 6601 main frame, containing the CPU, ten PPU's, central memory, and some input/output synchronizers.
B. One CDC 6602 console display unit, with a manual keyboard and two cathode ray tube display units.
C. One CDC 405 high speed card reader.
D. One CDC 415 card punch.
E. One CDC 501 high speed printer.
F. One CDC 1612 high speed printer.
G. Eight CDC 607 one-half inch magnetic tape units.
H. Three CDC 6603 mass storage disc files.
I. One 280-C cathode ray tube display system, commonly known as the DD80-C.

1.2.3. Central Memory Characteristics

The 6600 *central memory* is a random access coincident-current magnetic core memory of 131,072 sixty-bit words, arranged in thirty-two logically independent banks of 4096 words each. The central memory is common to all eleven system computers. It requires one major cycle (1000 nanoseconds) to perform a read-write operation. Several banks may be in operation at any given time. The maximum memory reference rate is one address per minor cycle (100 nanoseconds). Similarly, the maximum rate of data flow to and from memory is one word per minor cycle.

The location of each word in central memory is specified by a seventeen bit address. Consecutive words are located in different banks to obtain the rapid access rates described above. Thus, each address is composed of two fields — a twelve-bit left-hand portion defining a location within a bank and a five-bit right-hand portion defining the bank. (The remaining bit of

the eighteen-bit address field, located to the left of the twelve-bit portion, is not used.)

OAA AAAAA AAAAA BBBBB

where A = address within bank, and B = bank.

Certain precautions are required so that independent programs may time share a computer. In the 6600, every program has a *reference address* (RA) and a *field length* (FL) attached to it by the PPU that initiates the program. All CPU references to central memory (for instructions or data) are therefore made relative to the reference address and are checked by the hardware to insure that they do not exceed the field length. This allows easy relocation of the program in central memory, and insures the integrity of each program in central memory. As an example, suppose P is an address in a program (relative to the program's first word address of zero). The program is loaded into central memory beginning at location RA (that is, program address zero corresponds to memory address RA). Any reference by the program to location P results first in a check to insure that

$$0 \leq P \leq FL \quad .$$

Then, the sum P + RA is formed, and this word of central memory is accessed.

1.2.4. Peripheral Processor Characteristics

There are ten identical *peripheral and control processors* (PPU), each with a twelve-bit 4096-word random access coincident-current magnetic core memory. There are twelve input/output data channels in the 6600; all PPU's have access to all twelve channels. In addition, all PPU's have access to central memory, and may cause the CPU to begin execution of a program in central memory. Each PPU may operate independently of and simultaneously with the other nine. The PPU's act as system control computers, performing input, output, and supervisory tasks while the CPU carries out high-speed arithmetic computations.

The basic time units for the 6600 are a minor cycle of 100 nanoseconds and a major cycle of 1000 nanoseconds. A PPU may access its own 4096 word memory in one major cycle, and may transmit or receive data through an I/O channel at a maximum rate of one twelve-bit word per major cycle. There is a real time clock, available on a channel of its own (not one of the twelve I/O channels), which counts major cycles. The clock period is 4096 major cycles (4.096 milliseconds); that is, the twelve-bit word holding the clock time overflows every 4.096 ms.

An important feature of a PPU is its ability to control the CPU, by issuing an *exchange jump* instruction. This instruction sends an eighteen-bit address to the CPU, and causes the CPU to cease executing instructions. The address is the starting location of a sixteen word *exchange jump package* containing various pieces of information about a CPU program to be executed. Hardware in the CPU replaces the exchange jump package with similar data from the interrupted program.

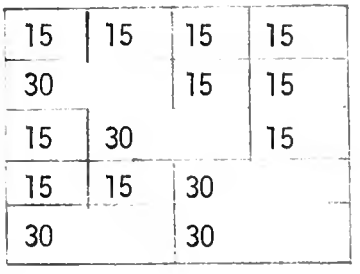
A PPU can also monitor a CPU program by obtaining the current program address.

1.2.5. Central Processor Characteristics

The 6600 CPU is a high speed arithmetic processor that has access only to the central memory; it is incapable of any input or output functions. It is composed of several functional units, to carry out arithmetic and logical operations, and a control unit, to direct the functional units, initiate instruction fetching, and perform fetching and storing of data. The high speed of the CPU is obtained by minimizing memory references for both instructions and data, by the bank interlacing of the central memory, and by the concurrent functional unit operation on unrelated instructions.

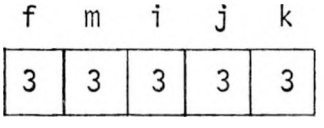
1.2.6. Instruction Format

The 6600 instructions may occupy fifteen bits or thirty bits; one instruction word may contain any of five different instruction combinations, as indicated below:



(bits)

Groups of bits in an instruction are commonly identified by the letters f, m, i, j, k (all three-bit groups), and K (eighteen-bit constant). A typical fifteen-bit instruction has five three-bit fields:

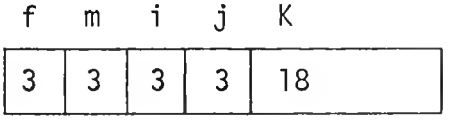


(total of 15 bits),

where

-fm- represents the operation code.
-i- normally represents a result register.
-j,k- normally represent operand registers.

The typical thirty-bit instruction has four three-bit fields and one eighteen-bit field:



(total of 30 bits),

where

-fm- represents the operation code.
-i- normally represents a result register.
-j- normally represents an operand register.
-K- is the (literal) second operand.

Details of the CPU instructions are summarized in the appendix.

1.2.7. CPU Instruction Registers

The CPU[4] contains eight sixty-bit *instruction registers* (commonly called the *stack*), designated by I0, I1, ..., I7. During the execution of a program, instruction words are transferred from central memory to register I0, one at a time from (usually) sequential locations. The instruction word is transferred by the control unit from I0 to a device called the *U-register*, where it is decoded into the component two, three, or four instructions. These are then issued, sequentially, to the functional units for execution.

If the instruction word does not contain a branch instruction, a new instruction word is requested for I0 as soon as the current instruction word has been transferred to the U-register. At this point, all instruction words in registers I0 - I6 are transferred to the next higher instruction register; I6 → I7, I5 → I6, ..., I0 → I1. The contents of I7 are lost from the stack.

If the instruction word contains a branch instruction, this process changes somewhat. First, the branch is tested to see if the branch condition is satisfied; if not, processing continues as described in the above paragraph. If the branch is to be taken, however, the branch address is examined to see if the next instruction word is already in a stack register. If not, the new word is fetched into I0 and the foregoing procedure continues. (There is a considerable delay while this fetch from memory to I0 takes place.) If the new instruction word is already in the stack (say, in register I5), it is fetched into the U-registers from that instruction register. Succeeding instructions will be taken from successive stack registers as long as possible at a more rapid rate (that is, from I4, ..., I0). No more instruction words are brought to I0 until an instruction word is requested that is not in the stack. A loop of seven words or less (at most 27 instructions) can be executed very rapidly, since no waiting is necessary for instruction words to be brought from memory. Efficient 6600 programming may require maximal use of stack loops and minimal use of out-of-stack branch instructions.

1.2.8. CPU Operating Registers

References to central memory for fetching and storing data are minimized by the use of twenty-four operating registers, divided into three sets of eight registers:

Eight 18-bit address registers (A-registers) A0, ..., A7
Eight 60-bit operand registers (X-registers) X0, ... X7
Eight 18-bit increment registers (B-registers) B0, ..., B7 .

All X-registers are used to hold operands for and operation results from the functional units. The five registers X1, ..., X5 can hold operands read from central memory; the two registers X6 and X7 similarly can hold results to be sent to central memory. As an illustration of the use of these registers, consider the following instruction sequence:

Transfer [A] to B[†]
Set D equal to [A] + [A]*[C] .

This may be accomplished by the following sequence of operations:

1. Fetch [A] into register X2.
2. Fetch [C] into register X4.
3. Multiply the contents of X2 and X4 together; send the result to X0.
4. Move the contents of X2 to X6.
5. Add the contents of X0 and X2 together and send the result to X7.
6. Store [X6] into B.
7. Store [X7] into D.

The address registers are used to fetch operands from memory and store results into memory. Placing a number P in address register A1, ..., A5 will cause the contents of memory word P to be placed into the corresponding X-register. Similarly, placing a number P in the address register A6 or A7 will cause the contents of the corresponding X-register to be placed into memory location P. Registers A0 and X0 are independent and have no connection with central memory. As an illustration, consider the sequence given above. It may more accurately be given by changing certain steps as follows:

[†]"[A]" means the contents of the variable named "A".

1. Place (A) into register A2, causing [A] to be placed into X2.†
2. Place (C) into A4, causing [C] to be placed in X4.
3. Multiply the contents of X2 and X4 together; send the result to X0.
4. Move the contents of X2 to X6.
5. Add the contents of X0 and X2 together and send the result to X7.
6. Place (B) into A6, causing [X6] to be transferred to B.
7. Place (D) into A7, causing [X7] to be transferred to D.

The B-registers have no direct connection to central memory; registers B1, ..., B7 are used to provide program indexing. Register B0 is eternally fixed as an eighteen-bit zero. In the example above, suppose we wish to perform the instructions

Set B(I) = A(I)
SET D(I) = A(I) + C*A(I)

for I = 0, 1, ..., 10. Instructions to do this, using two B-registers, might be the following:

1. Initialize the B-registers by setting B1 = 0 and B3 = 11.
2. Place (A) + [B1] into A2 (that is, the location of the first word of array A, plus the increment given in B1), resulting in [A(I)] in X2.
3. Place (C) into A4, resulting in [C] in X4.
4. Multiply [X2] and [X4] together, and send the result to X0.
5. Move [X2] to X6.
6. Add [X0] and [X2] together, sending the result to X7.
7. Place (B) + [B1] into A6, resulting in B(I) = A(I).
8. Place (D) + [B1] into A7, resulting in D(I) = A(I) + C*A(I).
9. Place [B1] + 1 into B1.
10. If [B1] < [B3], jump to step 2 for the next iteration of this loop.

1.2.9. CPU Functional Units

Additional operating speed is obtained by the use of ten *functional units* which may operate simultaneously on unrelated instructions, as long as no conflicts are present. The multiply and increment units are duplexed, so

---

†"(A)" means the location of the variable named "A".

that an instruction may be sent to the second unit whenever the first one is busy.

Branch unit.	Handles all jumps or branches within a program.
Increment unit.	Performs one's complement addition and subtraction of eighteen-bit numbers. There are two of these units.
Long add unit.	Performs one's complement addition and subtraction of sixty-bit fixed point numbers.
Add unit.	Performs floating point addition and subtraction.
Multiply unit.	Performs floating point multiplication. There are two of these units.
Divide unit.	Performs floating point division. This unit can also sum the number of ones in a sixty-bit word.
Boolean unit.	Performs the logical operations: transfer, intersection, union, and complement.
Shift unit.	Performs all operations of shifting, normalizing, packing and unpacking, and mask generation.

1.2.10. Exchange Jump

An eighteen-bit *P-register* is used to hold the address of each program instruction word as it is being executed. P is advanced in the following ways:

A. P is advanced by one when all instructions in a sixty-bit word have been extracted and sent to the instruction registers.
B. P is set to the address specified by a branch instruction.
C. P is set to the address specified in the exchange jump package.

A program is begun in the CPU by means of an *exchange jump* instruction from a PPU. This causes initial values to be entered into all operating registers and the P-register from a 16-word *exchange jump package* located in central memory. The PPU actually provides the CPU with the first word address of this package, and the CPU exchanges the current contents of the program's registers with the new contents as given in the exchange jump package. Thus,

the controlling data for two programs is interchanged; another exchange jump may later cause control to go back to the interrupted program.

The exchange jump package provides the following items of information for a program to be executed:

A. Program address (P).
B. Reference address for central memory (RA).
C. Field length for central memory (FL).
D. Program exit mode (EM).
E. Initial contents of operating registers.

These quantities are located as indicated in Figure 1.4.

bits 59...54,53.....36,35.....18,17.....0

location	6	18	18	18
0	///////	P	A0	//////////////////
1	///////	RA	A1	B1
2	///////	FL	A2	B2
3	///////	EM	A3	B3
4	//////////////////////////		A4	B4
5	//////////////////////////		A5	B5
6	//////////////////////////		A6	B6
7	//////////////////////////		A7	B7
10	X0			
11	X1			
12	X2			
13	X3			
14	X4			
15	X5			
16	X6			
17	X7			

Location is relative to the first word of the exchange jump package.
Bits are numbered 0 to 59, reading right to left.

Figure 1.4. CDC 6600 Exchange Jump Package

1.2.11. CPU Exit Mode

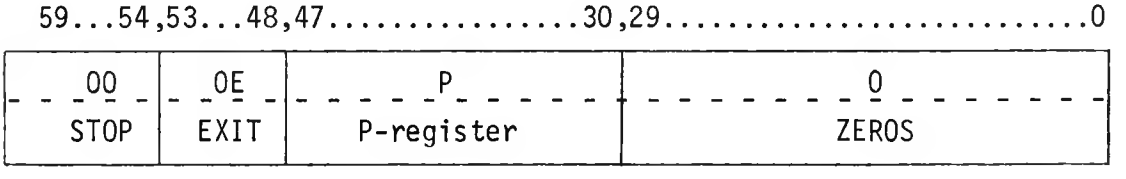
Execution of a program by the CPU will continue until a PPU causes an exchange jump to take place, or until an error occurs. The *exit mode* feature allows the programmer to control the three error conditions that may occur. These are:

Address out of range —	An attempt was made to reference central memory with an address greater than FL.
Operand out of range —	A floating point arithmetic unit received an infinite operand.
Indefinite operand —	A floating point arithmetic unit received an indefinite operand.

To select the exit conditions that he wishes to cause execution to stop, the programmer must preset the EM bits of the exchange jump package as follows:

000000	No exit selections are made.
010000	Address out of range.
020000	Operand out of range.
030000	Address or operand out of range.
040000	Indefinite operand.
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 is detected for which the exit mode is set, the CPU generates a halt instruction at location zero of the program (location RA of central memory), containing the upper six bits of the exit condition and the contents of the P-register.



The CPU then sets the P-register to zero, causing a jump to this generated halt instruction. When an error is detected for which the exit mode is not set, it is ignored.

1.2.12. Floating Point Arithmetic

All arithmetic in the 6600 is performed using one's complement numbers. This means that a K-bit number is interpreted directly if the sign bit (bit K-1) is zero, and is interpreted by complementing the entire number if the sign bit is one. Note that there are therefore two representations of zero (00...0 and 77...7, octal).

Since the 6600 is intended to be used primarily for large scientific problems, floating point arithmetic is used for most calculations. A floating point data word on the 6600 computer system occupies an entire sixty-bit word, and contains three fields:

Length	Bits	Field
1	59	Coefficient sign.
11	48 - 58	Biased exponent (characteristic).
48	0 - 47	Integer Coefficient.

The binary point is considered to be to the right of the coefficient. The sign bit is zero for plus and one for minus; negative numbers are represented in one's complement notation. The exponent is biased by 2000 octal (2^{10}). This means that the characteristic is formed by adding 2000 to the true exponent if it is positive, or by adding 3777 to the true exponent if it is negative:

True exponent 274 becomes biased exponent 2274.
True exponent -36 becomes biased exponent 1741.

For example, floating point numbers 1.0 and 0.75 would be given by

20000 00000 00000 00001
17760 00000 00000 00003

if unnormalized, and by

17204 00000 00000 00000
17176 00000 00000 00000

if normalized.

Note that exponent arithmetic uses the one's complement notation. Floating point numbers may appear with exponents from 0000 to 3776 (thus ranging from -1023 to +1022, decimal). This allows floating point numbers in the range 10^{-293} to 10^{322}, approximately.

Floating point numbers may be normalized or unnormalized, and performing an arithmetic operation on two normalized floating point numbers need not produce a normalized result. The 6600 also has capability for operating on double precision floating point numbers, each with its own exponent and coefficient.

An arithmetic operation that results in an exponent larger than 3777 is treated as an *infinite* quantity; a coefficient of all zeros and an exponent of 3777 is created for such a result. Use of infinity (and, in some cases, zero) as operands may produce an *indefinite* result; a coefficient of all zeros and an exponent of 1777 is created for such a result. Note that no error occurs when an infinite or indefinite result is generated. An optional exit is available when such results are used as operands. For more information on these topics, please refer to reference 3.

1.3. THE CDC 7600 COMPUTER SYSTEM

1.3.1. Hardware Components

The Control Data Corporation 7600 computer system[5,6] actually consists of eleven independent computers, much like the CDC 6600. The machine was, in fact, designed to be upward compatible with the 6600 for user programs. The input-output section has been greatly changed, however, and core memory now comes in two types.

The *central processing unit* (CPU) contains a high speed computation section, with access to both types of central memory. The *peripheral processor units* (PPU) operate independently of the CPU and of each other, and control the input-output functions of the system. The *small core memory* (SCM) is a very fast coincident current multibank device, and is used to contain executing programs, the resident system monitor program, and some I/O buffer areas. The *large core memory* (LCM) is a much larger and slower linear selection type of memory, and provides the basic working storage for the CPU. The system is illustrated in Figure 1.5.

1.3.2. Small Core Memory Characteristics

The 7600 CPU contains two memories. The *small core memory* (SCM) is a random access coincident-current magnetic core memory of 65,536 sixty-bit words, arranged in 32 logically independent banks of 2048 words each. Up to ten banks may be in operation at any given time. The maximum memory reference rate is one word per clock period (27.5 nanoseconds); the read-write cycle time is ten clock periods (275 ns).

The location of each word in SCM is specified by a sixteen-bit address. Consecutive words are located in different banks to obtain the rapid access rates. Thus, each address is composed of two fields — an eleven bit left-hand portion defining a location within a bank, and a five-bit right-hand portion selecting the bank. (The remaining portion of the eighteen-bit address field, located to the left of the eleven-bit portion, is not used.)

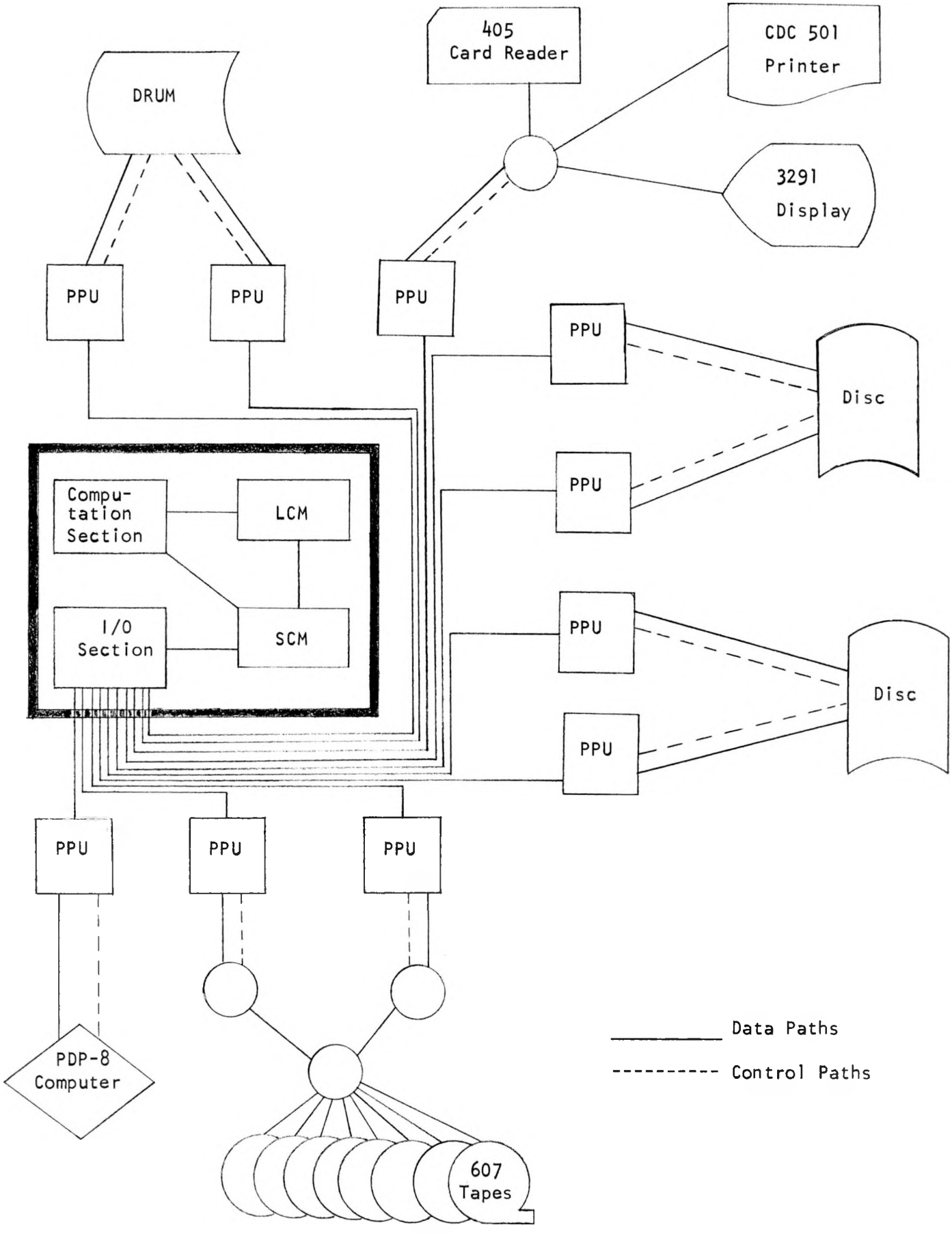


Figure 1.5. CDC 7600 Hardware

00A AAAAA AAAAA BBBBB ,

where A = address within bank, and B = bank.

The 7600 memory protection scheme is similar to the 6600 process. Every program in SCM has a reference address (RAS) and a field length (FLS). All CPU references to SCM are made relative to RAS, and are checked by the hardware to insure that they do not exceed FLS.

The first 4096 addresses of SCM are reserved for input-output and control buffers. All data entering the CPU from a PPU goes through these buffers; when a buffer is full, the CPU normally transfers the data to LCM.

1.3.3. Large Core Memory Characteristics

The 7600 *large core memory* (LCM) is a linear selection type memory of 512,000 sixty-bit words, arranged in eight independent banks of 64,000 words each. A reference to a word in LCM results in eight sixty-bit words being read simultaneously into a 480-bit register; there is one such register per bank. The memory cycle time is 1.76 microseconds (64 clock periods).

The large core memory is intended to provide basic working storage for a CPU program. Instructions cannot be executed directly from LCM — they must be read into the SCM and executed from there. Maximum data transfer rates between LCM and SCM occur when many consecutive words are moved. One sixty-bit word per clock period can be transferred during such block copy instructions. However, single words may be read from LCM. A contiguous group of eight words is brought to a 480-bit register in 1.76 μs. The other seven words in this packet may then be referenced directly from this register in only three clock periods each (82.5 ns).

Memory protection is similar to that described for SCM, using a relative address (RAL) and a field length (FLL).

1.3.4. Peripheral Processor Characteristics

Each of the ten *peripheral processor units* (PPU) is an independent computer with a twelve-bit 4096 word random access coincident current magnetic core memory. Each PPU has provision for eight fully duplex input/output channels, one of which leads to the small core memory buffer area. The PPU's are used to perform input and output at the request of the CPU system monitor program.

The basic time unit for a 7600 PPU is the clock period of 27.5 ns. A PPU may access its own memory in 275 ns, and may transmit or receive data through an I/O channel at a maximum rate of one twelve-bit word per nine clock periods (247.5 ns).

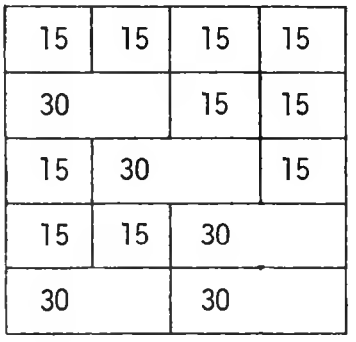
1.3.5. Central Processor Characteristics

The 7600 *Central Processing Unit* (CPU) consists of a computation section, both memories, and an input/output multiplexor. The *computation section* is a high speed arithmetic processor that has access only to the central memories; it is incapable of any input or output function. It has a sixty-bit internal word, nine independent functional units to carry out arithmetic and logical operations, and a twelve-word instruction stack. The high speed of the CPU is obtained, as in the 6600, by minimizing memory references for instructions and data, by the interlacing of the SCM, and by the concurrent operation of the functional units. In addition, the functional units are segmented (Section 1.3.9).

The CPU *I/O Multiplexor* (MUX) is used to communicate between the PPU's and the SCM, by a data-buffering mechanism. Communication between CPU and PPU is over a twelve-bit fully duplex channel. The MUX has 15 of these channels, each with separate SCM buffer areas for input and for output.

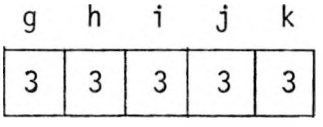
1.3.6. Instruction Format

The CDC 7600 instruction format is identical to that of the 6600. Each instruction may occupy fifteen or thirty bits; one instruction word may contain any of five different instruction combinations, as indicated below:



(bits)

Groups of bits in an instruction are commonly identified by the letters g, h, i, j, k (three-bit groups), and K (eighteen-bit constant). A typical fifteen-bit instruction has five three-bit fields:

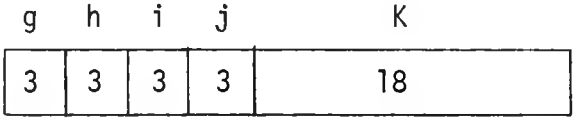


(total of 15 bits),

where

-gh- represents the operation code.
-i- normally represents a result register.
-j,k- normally represent operand registers.

The typical thirty-bit instruction has four three-bit fields and one eighteen-bit field:



(total of 30 bits),

where

-gh- represents the operation code.
-i- normally represents a result register.
-j- normally represents an operand register.
-K- is the (literal) second operand.

The g bits generally identify the type of instruction and the functional unit. The h bits usually specify the functional unit mode. Details of the instructions are summarized in the appendix.

1.3.7. CPU Instruction Registers

The CPU contains twelve sixty-bit *instruction registers*, commonly called the *stack*. During the execution of a program, instruction words are transferred to the stack from the SCM, two words ahead of the instruction currently being executed. As each new instruction is fetched into the stack, the ones already there are shifted one register. The oldest instruction is lost.

The stack is used most efficiently for small programs that can be contained entirely within the stack. Only ten of the stack registers may be used in such a loop, so it may contain at most 40 instructions.

An 18-bit *P-register* is used to hold the address of each program instruction word as it is being executed. P is advanced in the following ways:

A. P is advanced by one when all instructions in a sixty-bit stack register have been extracted and sent to the *current instruction word* register.
B. P is set to the address specified by a branch instruction.
C. P is set to the address specified in the exchange package.

1.3.8. CPU Operating Registers

The twenty-four operating registers are nearly identical in arrangement, number, and function to those of the 6600. (See Section 1.2.8.) There are three sets of eight registers each:

Eight 60-bit operand registers (X-registers) X0, ..., X7,
Eight 18-bit address registers (A-registers) A0, ..., A7,
Eight 18-bit increment registers (B-registers) B0, ..., B7.

All X-registers are used to hold operands for and operation results from the functional units and LCM. The five registers X1, X2, ..., X5 can hold operands read from SCM, and the two registers X6 and X7 can hold results to be sent to SCM.

The A-registers are used to fetch operands from SCM and store results into SCM. Placing a number in address register A1, A2, ..., A5 will cause the

contents of the corresponding SCM word to be transferred to the corresponding X-register. Similarly, placing a number in A6 or A7 causes the contents of X6 or X7 to be transferred to SCM.

Registers A0 and X0 have no connection with SCM. They are used to hold addresses for the block copy instructions that transfer data between LCM and SCM.

Any X-register may be used to hold the 21-bit address of a single word in LCM. This word may then be brought to an X-register from LCM, or sent to LCM from an X-register, by means of the read and write LCM instructions (014 and 015 — see Appendix).

The B-registers are used primarily to provide program indexing and loop counting. Register B0 is eternally fixed as an eighteen-bit zero.

1.3.9. CPU Functional Units

The *functional units* of the 7600 are considerably different from the 6600. Except for the multiply and divide units, all functional units have *one clock period segmentation*. This means that the operands for the unit move through the unit in a "pipe-line" fashion, freeing previous portions each clock period. Thus, a new set of operands may be entered into the functional unit each clock period. The multiply unit has a two clock period segment, and the divide unit has no segmentation at all (it uses an iterative algorithm, and requires eighteen clock periods before a new divide instruction may begin).

Increment unit.	Performs one's complement addition and subtraction of eighteen bit numbers.
Long add unit.	Performs integer addition and subtraction of sixty-bit numbers.
Floating add unit.	Performs floating point addition and subtraction.
Floating multiply unit.	Performs floating point multiplication.
Floating divide unit.	Performs floating point division.
Population count unit.	Counts the number of one bits in a sixty-bit word.

Boolean unit.	Performs the logical operations of transfer, intersection, union, and complement. Performs pack and unpack operations.
Shift unit.	Performs all operations of shifting.
Normalize unit.	Performs the normalize operations.

Functional unit times are given in Figure 1.6.

Functional Unit	Instructions Affected	Segment Time (clock periods)	Execution Time (clock periods)
Increment	50-57 (A-reg.)	1	2 to set Aj
			8 to read to Xj
			1 to store from Xj
	60-77 (B, X-reg.)	1	2
Long Add	36-37	1	2
Floating Add	30-35	1	4
Floating Multiply	40-42	2	5
Floating Divide	44-45	18	20
Population Count	47	1	2
Boolean	10-17, 26-27	1	2
Shift	20-23, 43	1	2
Normalize	24-25	1	2
Branch	02-07	---	2 - no branch
			3 - branch into stack
			11 - branch out of stack
-----	46 (pass)	---	1

Figure 1.6. CDC 7600 Timing

1.3.10. Exchange Jump

The *exchange jump* instruction is a special instruction to allow the CPU to switch execution from one program to another. It causes initial values to be entered into all operating registers and the program address register from a sixteen word *exchange package*, and previous values of those registers to be stored in the same package. The following quantities are involved:

A. Program address (P).
B. Reference address for small core (RAS) and large core (RAL) memories.
C. Field length for small core (FLS) and large core (FLL) memories.
D. Program status designation register (PSD).
E. Normal exit address (NEA).
F. Error exit address (EEA).
G. Breakpoint address (BPA).
H. The operating registers.

These quantities are diagrammed in Figure 1.7.

An exchange jump may be initiated under several different conditions.

A. An exchange exit instruction is issued by the system monitor program.
B. An error exit instruction is issued by a user program or by a CPU error condition.
C. An input or output interrupt occurs.
D. A real time interrupt occurs by an overflow of the clock period counter (every 3.6 ms).
E. A program breakpoint is reached.
F. The program is operating in step mode.

The 7600 has two exit addresses in the exchange package. These designate absolute SCM addresses, assumed to reside within the system monitor program. NEA is used by a user program issuing an exchange exit instruction, to request monitor services (such as an input or output request). EEA is used by a user program issuing an error exit instruction, or if a CPU error is detected (such as arithmetic overflow, indefinite results, hardware failure, or address out of range). In the latter case, the type of error will be specified in the PSD register. (See Section 1.3.11.)

bits 59...54,53.....36,35.....18,17.....0

location	6	18	18	18
0	///////	P	A0	BPA
1	///////	RAS	A1	B1
2	///////	FLS	A2	B2
3	///////	PSD	A3	B3
4	RAL		A4	B4
5	FLL		A5	B5
6	NEA		A6	B6
7	EEA		A7	B7
10	X0			
11	X1			
12	X2			
13	X3			
14	X4			
15	X5			
16	X6			
17	X7			

Location is relative to the first word of the exchange package. Bits are numbered 0 to 59, reading right to left.

Figure 1.7. CDC 7600 Exchange Package

A real time interrupt occurs after a user program has run for about 3.6 milliseconds. This is used to segment program execution, and to enable the system monitor program to perform required periodic "house-keeping": initiate input or output processing, update the clock, update the charge to the user, and possibly initiate a new user program.

During debugging, a program may be executed in small sections by using the *breakpoint address* (BPA) register. Whenever the program reaches the breakpoint (P = BPA), then execution ceases with a jump to EEA. Normally, no instructions are executed at BPA. A program executing in *step mode* will cease execution, with a jump to EEA, at the end of each instruction word.

1.3.11. Program Status Designators

Execution of a program by the CPU will continue until an exchange jump occurs, as discussed in the previous section. The reason for the exit may be determined by examining the bits of the *Program Status Designator* (PSD) in the exchange package.

Bit	Function	Bit	Function
17	Exit Mode Flag	8	SCM Block Range Condition Flag
16	Monitor Mode Flag	7	LCM Direct Range Condition Flag
15	Step Mode Flag	6	SCM Direct Range Condition Flag
14	Indefinite Mode Flag	5	Program Range Condition Flag
13	Overflow Mode Flag	4	Breakpoint Condition Flag
12	Underflow Mode Flag	3	Step Condition Flag
11	LCM Parity Condition Flag	2	Indefinite Condition Flag
10	SCM Parity Condition Flag	1	Overflow Condition Flag
9	LCM Block Range Condition Flag	0	Underflow Condition Flag

The six *mode flags* are set before the program is initiated, and govern its execution. The *exit mode flag* controls the source of the exchange package address for the execution of an exchange exit instruction, and the *monitor mode flag* determines whether the program can be interrupted by an input or output request. These two flags are never set for user programs.

The *step mode flag* causes the program to be interrupted at the end of each instruction word. The *indefinite mode flag* will cause a program interrupt whenever an indefinite floating point result is detected. The *overflow mode flag* and *underflow mode flag* cause a program interrupt whenever a floating point overflow or underflow is detected in a floating point result. The interrupt will occur only after the current instruction word is completed, and control is sent to EEA in all four cases.

The twelve *condition flags* are set by the hardware whenever the specified conditions occur; control is sent to EEA at the end of the current instruction word. The *LCM parity condition flag* and the *SCM parity condition flag* are set whenever a parity error is detected in a memory reference. (The SCM has one parity bit per 12 bits, and the LCM has one parity bit per 15 bits.) The *LCM block range condition flag* and the *SCM block range condition flag* are set whenever a block copy instruction (instructions 011 and 012) requests a reference to an address greater than or equal to FLL or FLS. The *LCM direct range condition flag* is set whenever a direct read or write (instructions 014 and 015) requests a reference to an address greater than or equal to FLL. The *SCM direct range condition flag* is set whenever any SCM reference (other than block copy) is greater than or equal to FLS, and whenever the program register is greater than or equal to FLS.

The *breakpoint condition flag* is set whenever P = BPA, and the *step condition flag* is set whenever an instruction issues and the step mode flag is set. The *indefinite condition flag, overflow condition flag,* and *underflow condition flag* are set whenever the corresponding conditions are detected by a floating point functional unit. If the corresponding mode flag is also set, execution will terminate at the end of the current instruction word. This word probably does not contain the instruction that caused the condition, because of the overlapping of the functional unit processing. Indefinite and overflow conditions may occur during the execution of instructions 30-35, 40-42, and 44-45; underflow may occur during the execution of instructions 32-33, 40-42, and 44-45.

1.3.12. Floating Point Arithmetic

All arithmetic in the 7600 is performed using one's complement numbers. This means that a k-bit number is interpreted directly if the sign bit is zero, and is interpreted by complementing the entire number if the sign bit is one. A floating point data word occupies sixty bits, including a one-bit sign (the leftmost bit), an eleven-bit biased exponent, and a 48-bit integer coefficient. The binary point is considered to be to the right of the coefficient. The exponent is biased by 2000 octal (2^{10}). This means that the biased exponent is formed by adding 2000 to the true exponent if it is positive, and adding 3777 to the true exponent if it is negative.

Floating point numbers need not be normalized, and arithmetic operations performed on normalized operands need not produce a normalized result. The 7600 can operate with double precision numbers, each with its own exponent and coefficient.

It is sometimes necessary to convert a floating point number into decimal notation. One way to do this is described here.

A. Examine the sign bit.
   1. If the sign bit is zero, the number is positive, so let S = +1.
   2. If the sign bit is one, the number is negative. Let S = -1, and complement the number.

B. Separate the exponent, E, and the coefficient, C. The exponent is the leftmost four octal digits.

C. Examine the exponent.
   1. If $E \geq 2000$ (octal), let R = E - 2000. (R is the true exponent.)
   2. If $E < 2000$ (octal), let R = -(1777 - E).

D. Convert the coefficient to a decimal integer. Suppose the coefficient has octal digits $C_{16}\ C_{15}\ C_{14} \ldots C_2\ C_1$. Then, this is converted to a decimal integer D by the formula

$$D = C_{16} \cdot 8^{15} + C_{15} \cdot 8^{14} + \ldots + C_2 \cdot 8 + C_1 \quad .$$

E. The final result is formed by attaching the true exponent and the sign: $Y = S \cdot 2^R \cdot D$.

This can be clarified by some examples.

1. The number is X = 17204 00000 00000 00000.
   A. S = +1.
   B. E = 1720, C = 4 00000 00000 00000.
   C. $R = -57_8 = -47_{10}$.
   D. $D = 4 \cdot 8^{15} = 4 \cdot 2^{45} = 2^{47}$.
   E. $Y = (+1)(2^{-47})(2^{47}) = +1$.
2. The number is X = 60511 57777 77777 77777.
   A. S = -1. X = 17266 20000 00000 00000.
   B. E = 1726, C = 6 20000 00000 00000.
   C. $R = -51_8 = -41_{10}$.
   D. $D = 6 \cdot 8^{15} + 2 \cdot 8^{14} = 25 \cdot 2^{43}$.
   E. $Y = (-1)(2^{-41})(25 \cdot 2^{43}) = -25 \cdot 2^2 = -100$.
3. The number is 16744 14336 75013 27554.
   A. S = +1.
   B. E = 1674, C = 4 14336 75013 27554.
   C. $R = -103_8 = -67_{10}$.
   D. $D = 4 \cdot 8^{15} + 8^{14} + 4 \cdot 8^{13} + 3 \cdot 8^{12} + \ldots \simeq 2147_{10} \cdot 2^{36}$.
   E. $Y = (+1)(2^{-67})(2147)(2^{36}) = 2147 \cdot 2^{-31}$
      $= (2147)(.465)(10^{-9}) = (.998)(10^{-6}) \simeq 10^{-6}$.

Biased floating point exponents may range from 2000 to 3777 (octal) for positive exponents, and from 1776 to 0000 for negative exponents. If a number is generated with a biased exponent greater than 3777, an overflow condition occurs, and the overflow condition flag will be set.† The exponent will be forced to 3777, and the coefficient (or its complement) will be forced to zeros. If a number is generated with a biased exponent less than 0000, an underflow condition occurs. A word of all zeros or all ones is produced.

---

†Some exceptions to these remarks may occasionally occur; see reference 6.

An indefinite condition occurs whenever a floating point functional unit cannot complete a calculation (such as dividing zero by zero). An exponent of 1777 and a zero characteristic will be generated.

The standard cases are as follows:

Positive overflow (+∞)	37770 00000 00000 00000
Negative overflow (-∞)	40007 77777 77777 77777
Positive indefinite (+IND)	17770 00000 00000 00000
Negative indefinite (-IND)†	60007 77777 77777 77777
Positive underflow (+0)	00000 00000 00000 00000
Negative underflow (-0)	77777 77777 77777 77777

Overflow and indefinite conditions are unlikely to occur during error-free computations, since the permissible exponents allow numbers in the approximate range 10^{-293} to 10^{+322}.

†Can only occur from packing or Boolean operations.

REFERENCES

1. *Programmed Data Processor-6 Handbook*, Digital Equipment Corporation, Maynard, Massachusetts, DEC Publication Number F-65 and F-65 Change Notice number 3 (July, 1965).

2. David L. Pehrson, *Pagination and Segmentation of the PDP-6 and Other Problems in Memory Control*, Lawrence Radiation Laboratory, Livermore, California, UCRL-70084 (August 15, 1966).

3. *Control Data 6000 Series Computer Systems: Reference Manual*, second edition, Control Data Corporation, St. Paul, Minnesota, CDC Publication Number 60100000 (July, 1965).

4. Harry L. Nelson, *Program Optimizing Techniques for the CDC 6600 Central Processor*, Lawrence Radiation Laboratory, Livermore, California, UCRL-12489 (April 6, 1965).

5. *Control Data 7600 Computer System: Preliminary System Description*, Control Data Corporation, St. Paul, Minnesota, CDC Publication Number 60258400 (September, 1969).

6. *Control Data 7600 Computer System: Preliminary Reference Manual*, second edition, Control Data Corporation, St. Paul, Minnesota, CDC Publication Number 60258200 (1969).

7. John E. Ranelletti, *CPU76, A 6600/7600 Central Assembler*, Lawrence Radiation Laboratory, Livermore, California, Computer Information Center, CIC Report L1-004 (March 5, 1970).

APPENDIX

CDC 6600/7600 Operation Codes

The instruction sets for the CDC 6600 and CDC 7600 computers are nearly identical, so they can be discussed together. The following conventions are used.

A. Each character in the octal column represents three bits.
B. Lower-case letters (i, j, k) designate the three-bit portions of an instruction (usually operating registers). Upper-case letters (KKKKKK) designate the eighteen-bit portion of some instructions.
C. A dash indicates an unused portion of an instruction.
D. The mneumonics are those used by the CPU76 Assembler.[7]
E. Operating registers are designated by an upper case/lower case pair (Ai, Bj, Xk).
F. The contents of an operating register are indicated by brackets ([Ai], [Bj], [Xk]).
G. Instruction 00 has a different function on the two machines. Instructions 01i, for $i \neq 0$, are available on the 7600 only.

octal	mnemonic	function
00---	HALT	Stop (6600).
00---	ERJP	Error exit to EEA (7600).
01--KKKKKK	RTJ	Return jump to K.
011jKKKKKK	BCLS	Block copy K + [Bj] words from LCM to SCM (7600).
012jKKKKKK	BCSL	Block copy K + [Bj] words from SCM to LCM (7600).
01300	EXNEA	Exchange exit to NEA if exit flag is clear (7600).
013jKKKKKK	EXK	Exchange exit to K + [Bj] if exit flag is set (7600).
014jk	RLCM	Read LCM at [Xk] to Xj (7600).
015jk	WLCM	Write [Xj] into LCM at Xk (7600).
0160k	RCIB	Reset channel [Bk] input buffer (7600).
016jk	RCIS	Read channel [Bk] input status to Bj (7600).
0170k	RCOB	Reset channel [Bk] output buffer (7600).
017jk	RCOS	Read channel [Bk] output status to Bj (7600).

octal	mnemonic	function
02i-KKKKKK	J	Jump to K + [Bi].
030jKKKKKK	JZ	Jump to K if [Xj] = 0.
031jKKKKKK	JNZ	Jump to K if [Xj] ≠ 0.
032jKKKKKK	JP	Jump to K if [Xj] ≥ 0.
033jKKKKKK	JM	Jump to K if [Xj] < 0.
034jKKKKKK	JINR	Jump to K if [Xj] is in range.
035jKKKKKK	JOUTR	Jump to K if [Xj] is out of range.
036jKKKKKK	JDEF	Jump to K if [Xj] is definite.
037jKKKKKK	JIDEF	Jump to K if [Xj] is indefinite.
04ijKKKKKK	JBE	Jump to K if [Bi] = [Bj].
05ijKKKKKK	JBNE	Jump to K if [Bi] ≠ [Bj].
06ijKKKKKK	JBGE	Jump to K if [Bi] ≥ [Bj].
07ijKKKKKK	JBLT	Jump to K if [Bi] < [Bj].
10ij-	REP	Copy [Xj] to Xi.
11ijk	AND	Logical product of [Xj] and [Xk] to Xi.
12ijk	OR	Logical sum of [Xj] and [Xk] to Xi.
13ijk	XOR	Logical difference of [Xj] and [Xk] to Xi.
14i-k	NOTK	Complement of [Xk] to Xi.
15ijk	ANDC	Logical product of [Xj] and comp. ([Xk]) to Xi.
16ijk	ORC	Logical sum of [Xj] and comp. ([Xk]) to Xi.
17ijk	XORC	Logical difference of [Xj] and comp. ([Xk]) to Xi.
20ijk	LS	Left shift [Xi] by jk bits, circular.
21ijk	RS	Right shift [Xi] by jk bits, sign extended.
22ijk	LSB	Left shift [Xk] by [Bj] bits to Xi, circular.
23ijk	RSB	Right shift [Xk] by [Bj] bits to Xi, sign extended.
24ijk	N	Normalize [Xk] to Xi and Bj.
25ijk	NR	Round and normalize [Xk] to Xi and Bj.
26ijk	UNPAK	Unpack [Xk] to Xi and Bj.
27ijk	PAK	Pack [Xk] and [Bj] to Xi.

octal	mnemonic	function
30ijk	ADD	Floating sum of [Xj] and [Xk] to Xi.
31ijk	SUB	Floating difference of [Xj] and [Xk] to Xi.
32ijk	ADDLO	Floating lower sum of [Xj] and [Xk] to Xi.
33ijk	SUBLO	Floating lower difference of [Xj] and [Xk] to Xi.
34ijk	ADDR	Rounded floating sum of [Xj] and [Xk] to Xi.
35ijk	SUBR	Rounded floating difference of [Xj] and [Xk] to Xi.
36ijk	ADD6Ø	Integer sum of [Xj] and [Xk] to Xi.
37ijk	SUB6Ø	Integer difference of [Xj] and [Xk] to Xi.
40ijk	MPY	Floating product of [Xj] and [Xk] to Xi.
41ijk	MPYR	Rounded floating product of [Xj] and [Xk] to Xi.
42ijk	MPYLO	Floating lower product of [Xj] and [Xk] to Xi.
43ijk	MOM	Form mask of jk bits in Xi.
44ijk	DVI	Floating divide of [Xj] by [Xk] to Xi.
45ijk	DVIR	Rounded floating divide of [Xj] by [Xk] to Xi.
46---	PASS	Pass.
47i-k	POP	Population count of [Xk] to Xi.
50ijKKKKKK	AR	Set Ai to [Aj] + K.
51ijKKKKKK	AI/STI	Set Ai to [Bj] + K.
52ijKKKKKK	AO	Set Ai to [Xj] + K.
53ijk	AOI	Set Ai to [Xj] + [Bk].
54ijk	ARI	Set Ai to [Aj] + [Bk].
55ijk	ARD	Set Ai to [Aj] - [Bk].
56ijk	AII	Set Ai to [Bj] + [Bk].
57ijk	AID	Set Ai to [Bj] - [Bk].
60ijKKKKKK	BR	Set Bi to [Aj] + K.
61ijKKKKKK	BI	Set Bi to [Bj] + K.
62ijKKKKKK	BO	Set Bi to [Xj] + K.
63ijk	BOI	Set Bi to [Xj] + [Bk].
64ijk	BRI	Set Bi to [Aj] + [Bk].
65ijk	BRD	Set Bi to [Aj] - [Bk].
66ijk	BII	Set Bi to [Bi] + [Bk].
67ijk	BID	Set Bi to [Bj] - [Bk].

octal	mnemonic	function
70ijKKKKKK	FR	Set Xi to [Aj] + K.
71ijKKKKKK	FI	Set Xi to [Bj] + K.
72ijKKKKKK	FO	Set Xi to [Xj] + K.
73ijk	FOI	Set Xi to [Xj] + [Bk].
74ijk	FRI	Set Xi to [Aj] + [Bk].
75ijk	FRD	Set Xi to [Aj] - [Bk].
76ijk	FII	Set Xi to [Bj] + [Bk].
77ijk	FID	Set Xi to [Bj] - [Bk].