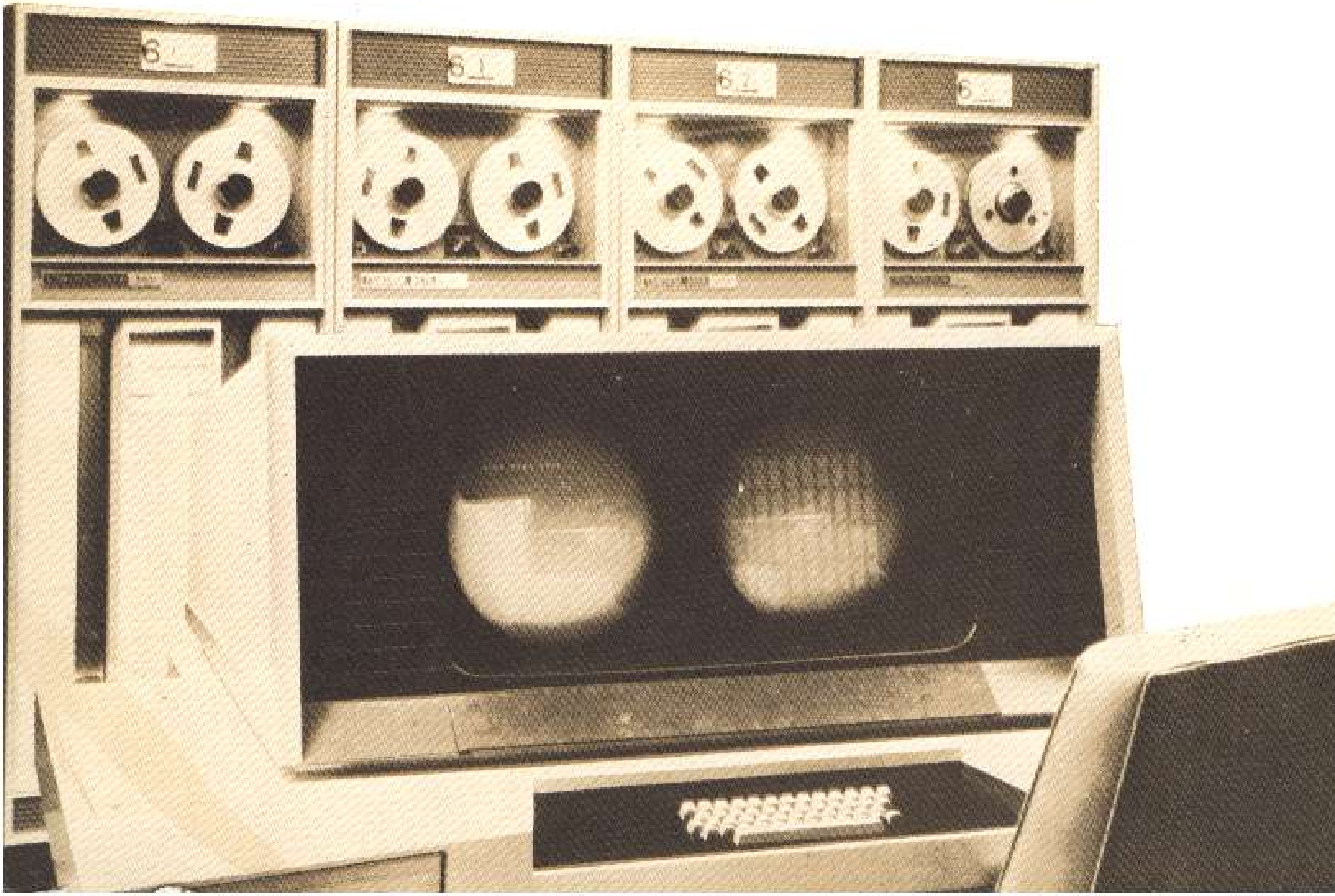
6400 CENTRAL PROCESSOR

CONTROL DATA INSTITUTE
CONTROL DATA CORPORATION



6400 CENTRAL PROCESSOR

TRAINING MANUAL

FOR TRAINING PURPOSES ONLY

This manual was compiled and written by instructional personnel of

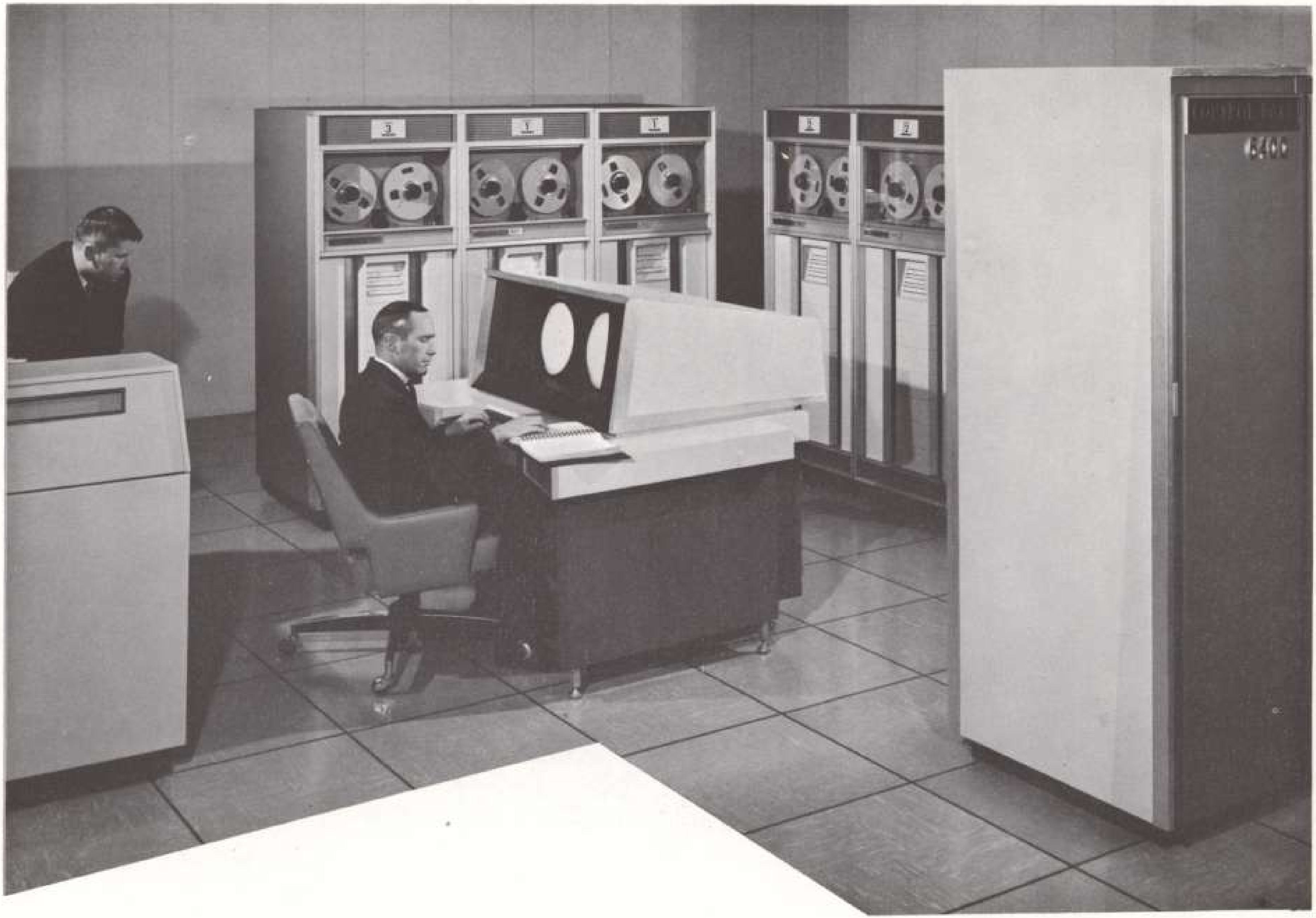
CONTROL DATA INSTITUTE
CONTROL DATA CORPORATION

Publication Number 60257200
(Formerly 011167)
February 1967

FOREWORD

The 6400 Central Processor Training Manual was written to assist the student in his study of the 6400 Computer System. This manual assumes that the reader is familiar with the contents of the 6400/6600 Introduction and Peripheral Processors Training Manual, Publications Number 020267.

In any technical writing effort, possibilities of errors are always present. Although Control Data Institute makes a conscious effort to minimize errors in its publications, errors are nevertheless inevitable. If you would like to make the existence of errors known, or would like to make comments or suggestions concerning the manual, you might find the Comments Sheet at the end of the manual to be of help. Forward your comments to the Educational Development Section, Control Data Institute, 3255 Hennepin Avenue South, Minneapolis, Minnesota, 55408.

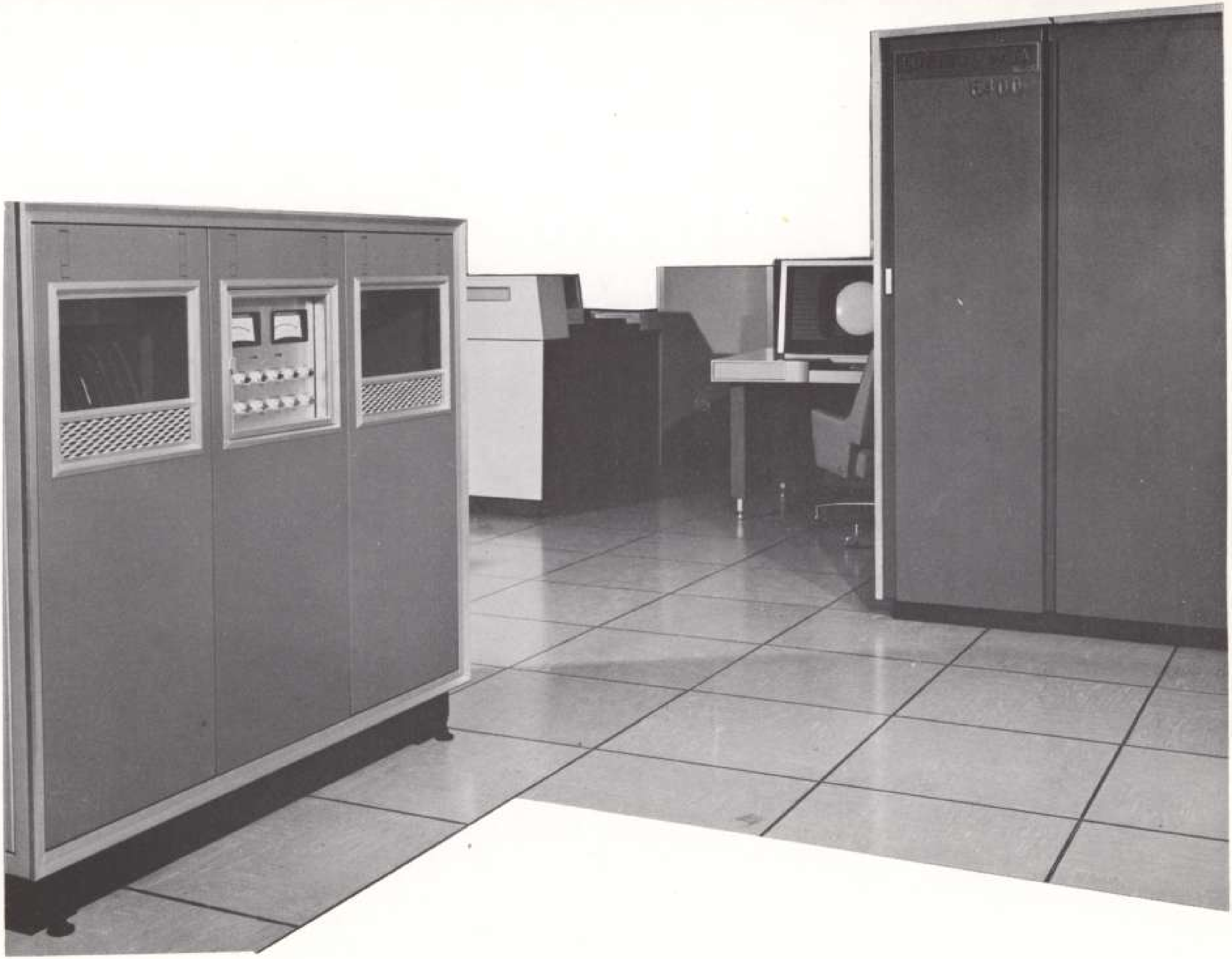


6400 COMPUTER SYSTEM

CONTENTS

CHAPTER I

AN INTRODUCTION TO THE 6400 COMPUTER SYSTEM



6400 COMPUTER SYSTEM

CHAPTER I

AN INTRODUCTION TO THE 6400 COMPUTER SYSTEM

INTRODUCTION

This chapter provides an introduction to the CONTROL DATA® 6400 Computer System.

MARKET POSITION

The 6400 is a large-scale, solid-state, automatic, parallel, digital computer aimed towards the general-purpose scientific market. Its input/output section is made up of 10 small general-purpose computers, with 12 data channels accessable to any of these computers, making the 6400 competetive in many areas.

6400 SYSTEM GENERAL BLOCK DIAGRAM

The general block diagram (Figure 1-1) will help review the overall functional sections and their interconnections. In the basic 6400 system, the only way to place data and instructions into the Central Memory is through the Peripheral Processors (PPU's). The Central Processor (CPU) is controlled by the PPU's. When the CPU stops, only a PPU can restart it. The PPU's do not have a Stop instruction.

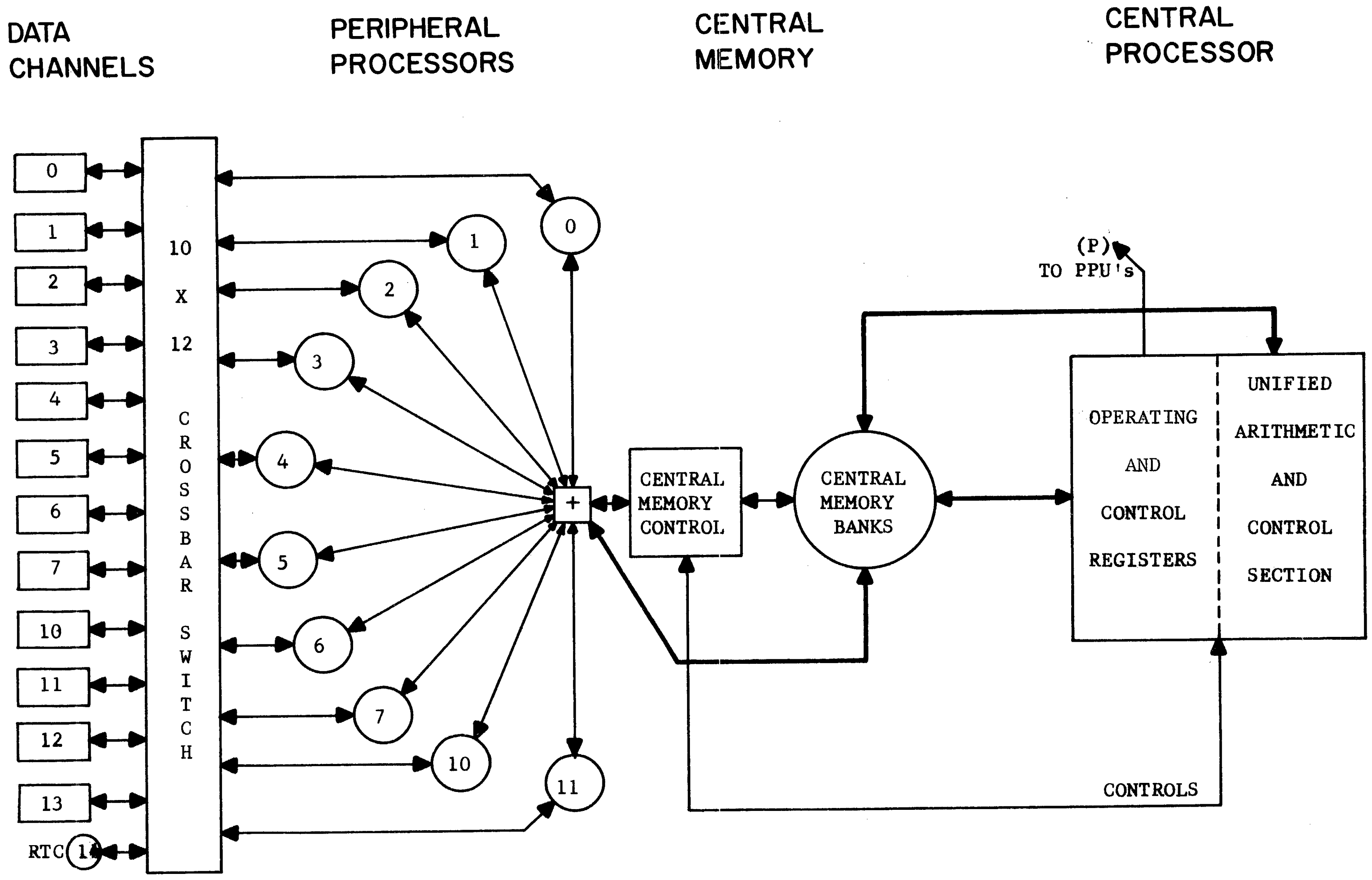


Figure 1-1. 6400 System General Block Diagram

SYSTEM CHARACTERISTICS

The 6400 system characteristics may be divided into four parts; Data Channels, PPU's, Central Memory, and Central Processor.

DATA CHANNELS

The ten 6400 Peripheral Processors are connected to thirteen independent data channels. Twelve of these data channels are bidirectional and can be connected to I/O equipment. The thirteenth is permanently connected to the Real Time Clock and is used specifically for monitoring the clock. The Real Time Clock is a 12-bit, 2's complement counter, advanced every 1 microsecond.

The method used to connect the data channels to the Peripheral Processors allows any processor to use any channel. This scheme allows flexibility in handling I/O equipment.

PERIPHERAL AND CONTROL PROCESSORS (PPU)

There are ten identical Peripheral and Control Processors (PPU) associated with the basic 6400 computer system. Each is independent of the other and each has its own 4K,12-bit memory. The basic cycle time is 1 microsecond. Each is capable of communicating with the central memory, the twelve input-output channels and the real time clock. Each processor is capable of executing a 64-instruction repertoire. The instructions include logical, branch, I/O, central memory access, direct, indirect, and indexing addressing modes. Average instruction execution time is 2 microseconds.

CENTRAL MEMORY

The central memory has 60-bit words, grouped into 4,096 word banks, and employs bank phasing. The 3 standard sizes available are 32,768 (8 banks), 65,576 words (16 banks), or 131,072 words (32 banks). The 60-bit word is obtained by arranging five PPU memory modules in parallel.

The basic memory cycle time is, therefore, the same as the PPU's; however, because of bank phasing, 60-bit words can be moved at a 100 ns rate. The Central circuits of Central Memory have a "Retry" feature which allows retrying a bank that was previously busy every 300 ns until it is free. The control circuit also has a "Scanner" to enable sorting of simultaneous requests from different sources. Central Memory can process a request every 100 nanoseconds. Data can flow into or out of memory at a rate of 60 bits every 100 nanoseconds.

CENTRAL PROCESSOR UNIT

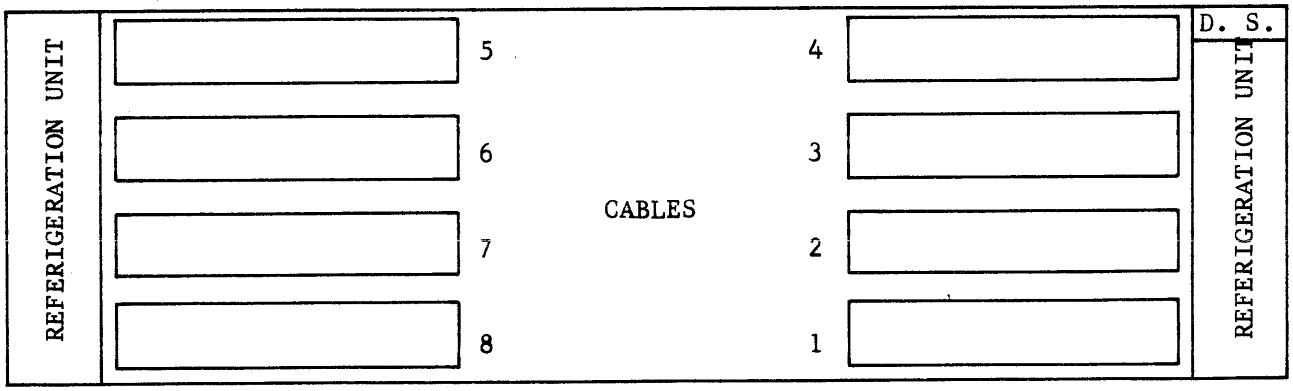
The 6400 Central Processor Unit is a high-speed, binary arithmetic and control section. Types of instructions are: logical, branch, shift, increment, floating point, pack, unpack, normalize, floating point add, subtract, multiply, divide (in single and double precision). All instructions and data come from Central Memory, and all new data (results of programs) are stored in Central Memory (the Central Processor has no I/O instruction). The instructions are executed in a serial fashion. Instructions are either 15 bits or 30 bits in size, 3-address type. The data word is always 60 bits. Data words are 59 bits plus 1 sign bit in a fixed point format; floating point has a 48-bit coefficient, 10-bit exponent, and 2 sign bits. Average instruction execution time is approximately 1 usec per instruction.

PHYSICAL CHARACTERISTICS

PHYSICAL LAYOUT

The 6400 is housed in an "in-line" frame, see Figure 1-2, or a "T" frame, see Figure 1-3. There is enough room in this layout for the PPU System, Central Memory (32K, 65K, or 131K), Central Processor, an optional Special-Purpose Central Processor. Figure 1-2 illustrates a top view of the 6400 main frame for the 32K and 65K memory sizes, Figure 1-3 illustrates the 131K memory size.

32K System			64K System		
Chassis	1	10 PPU's, 12 Data Channels, and real Time Clock	Chassis	1	10 PPU's, 12 Data Channels, and real Time Clock
	2	Central Processor		2	Central Processor
	3	16K Central Memory, CM Control, and Data Distributor		3	16K Central Memory, CM Control, and Data Distributor
	4	16K Central Memory, Disk Controller, and Data Distributor		4	16K Central Memory and Disk Controller
	5	Not used		5	16K Central Memory and Data Distributor
	6	Display Controller, Clock and Data Distributor		6	Display Controller and Clock
	7	Not used		7	16K Central Memory and Data Distributor
	8	Not used		8	Optional CPU

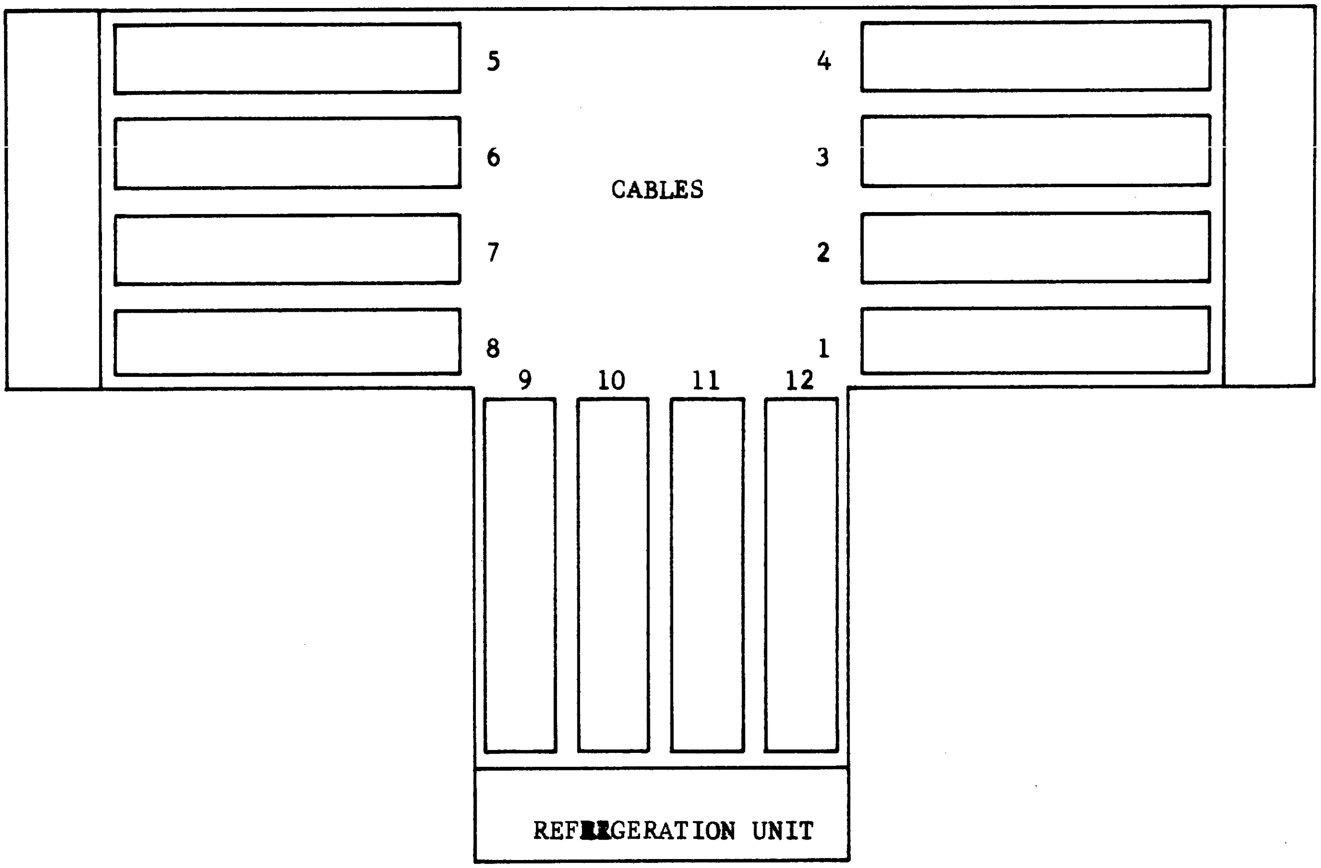


TOP VIEW

PHYSICAL LAYOUT: 6400 System - 32K or 64K Memory

Figure 1-2

Chassis	1	10 PPU's
	2	Central Processor
	3	16K Central Memory, Control and Data Distributor
	4	16K Central Memory, Sisk Controller and Data Distributor
	5	16K Central Memory and Data Distributor
	6	Display Controller
Chassis	7	16K Central Memory
	8	Optional CPU
	9	16K Central Memory and Data Distributor
	10	16K Central Memory and Data Distributor
	11	16K Central Memory and Data Distributor
	12	16K Central Memory and Data Distributor



TOP VIEW

PHYSICAL LAYOUT: 6400 System - 131K

Figure 1-3

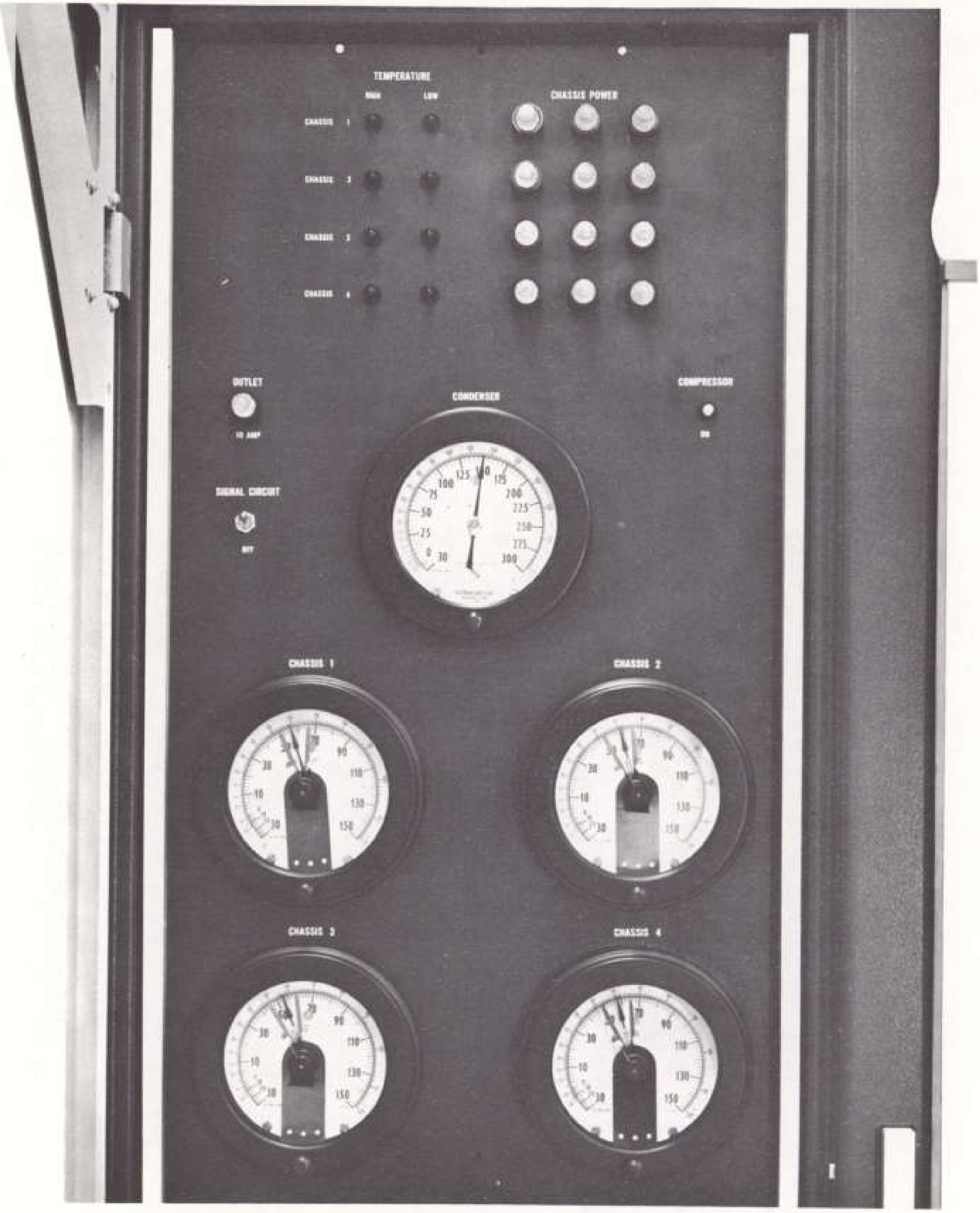
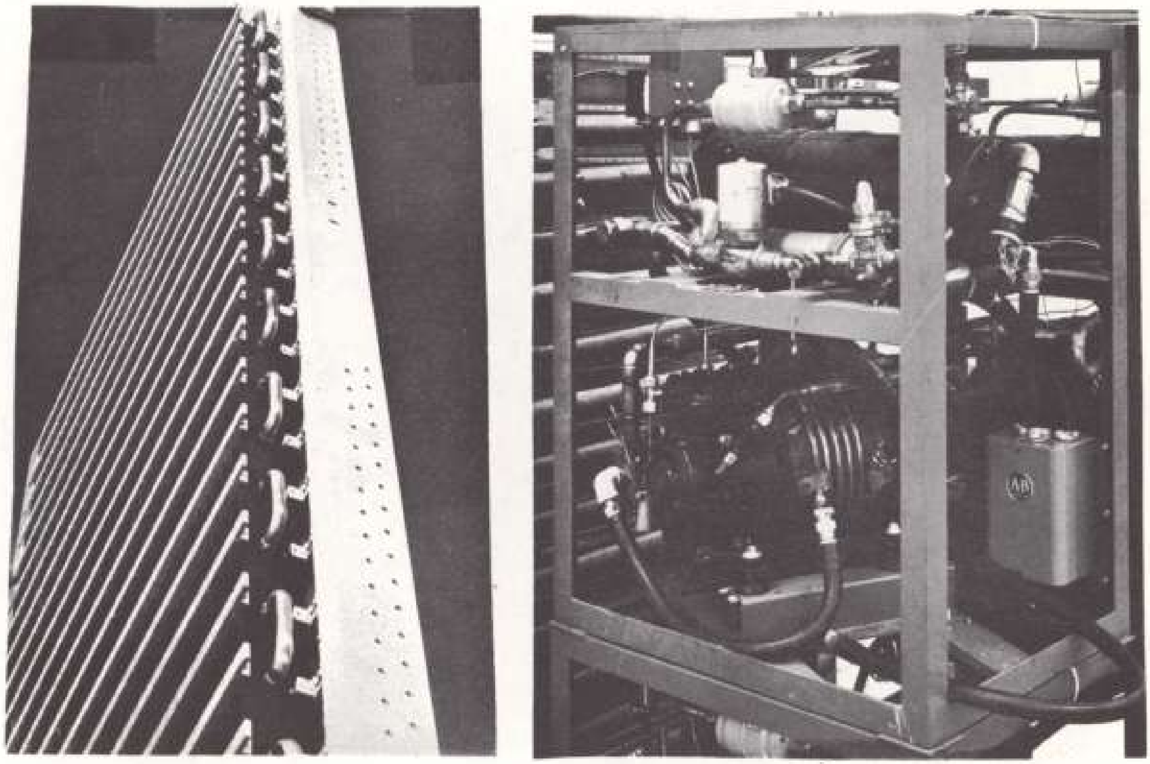


Figure 1-4. Control Panel - 6400 Cooling System

COOLING

Between each row of modules on a chassis is a bar that contains a copper tube that carries refrigerant. (See Figure 1-3) The refrigerant carries heat away from the bar to a closed cycle refrigeration unit located in each wing of the main frame. (See Figure 1-5A) The heat generated by the modules is transferred to the "cold bar" by the principles of convection and conduction. (see the Introduction to 6400/6600 Computer Systems Training Manual for details of the refrigeration system).



A. Refrigeration Tubes

B. Refrigeration Unit

Figure 1-5

LOGIC CIRCUITS

Modified transistor-coupled logic circuits operating in saturated mode are arranged in negative voltage and circuits. The transistors used are Silicon Planar NPN type. The circuits switch in 3-5 nanoseconds. A technique called "cordwood" packaging is used when building the individual modules that make up the 6400 logic circuits. The logic module will have approximately 64 transistors and has a 30-pin male connector for signal distribution and power connections (see Figure 1-6). The logical circuit contained in a module may be unique or may be a standard circuit that will be repeated many times within the overall computer. The individual modules are pluggable into a 30-pin female connector mounted on the chassis. The individual connectors are connected together using point-to-point wire techniques. This forms a thick wiring mat on the back of each chassis (see Figure 1-7). Specific modules can be located on the chassis by coordinates. The vertical edge of the chassis is labeled A-R and the horizontal bar is labeled 1-42. There are test points on the front plate of each module for ease of monitoring the modules internal signals. (see Figure 1-8).

Because of the extremely high speed of the 6400 logic circuits, a method of moving data is used that will not reduce the speed of the operating circuits. This method is transformer coupled coaxial cables arranged in groups called data trunks, which are located between points where data flow, such as the data trunk between central memory and the operating registers. The trunk consists of standard cables, transmitting circuits, and receiving circuits (catching registers, see Figure 1-9).

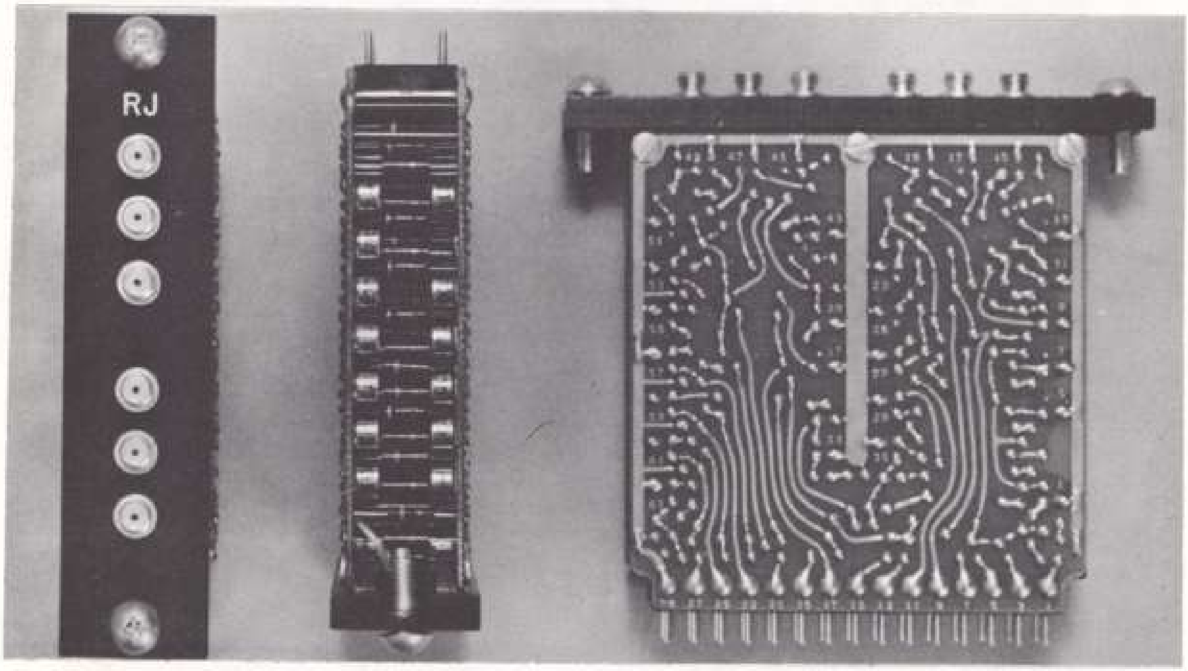


Figure 1-6. 6400 Logic Module

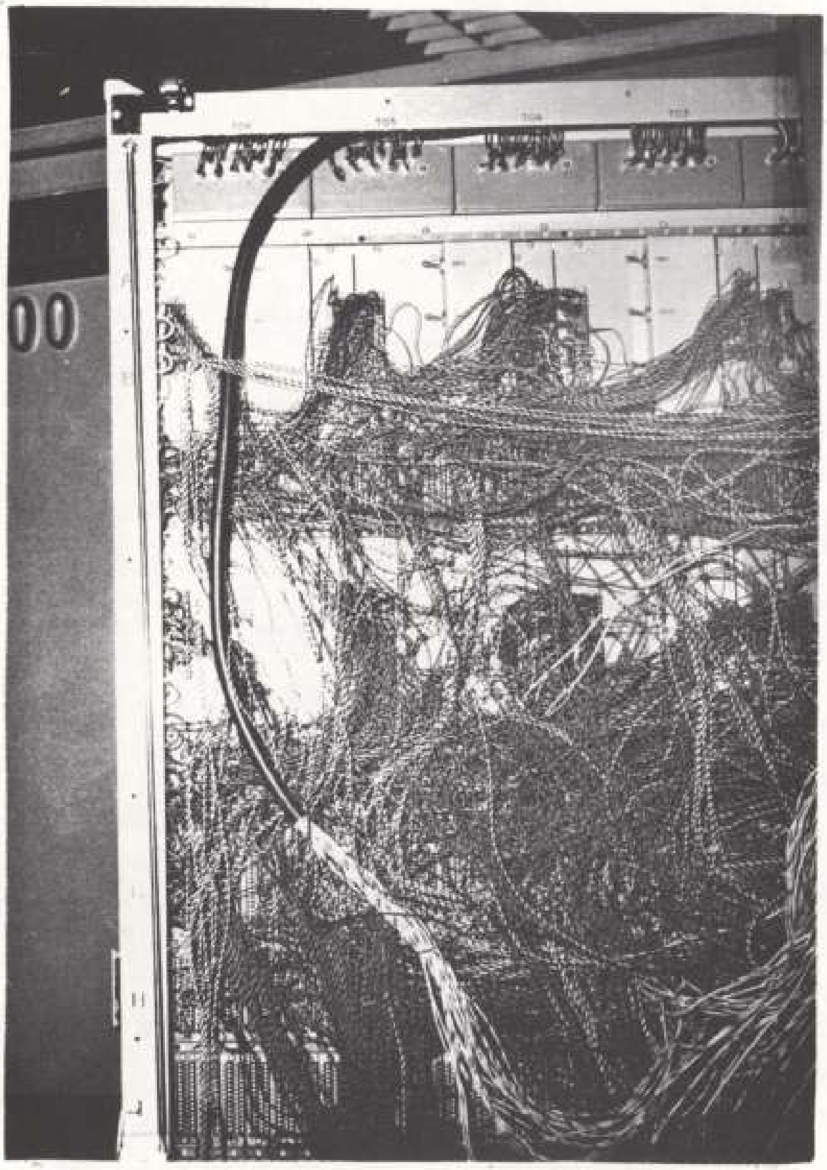


Figure 1-7. Back of Chassis Wiring

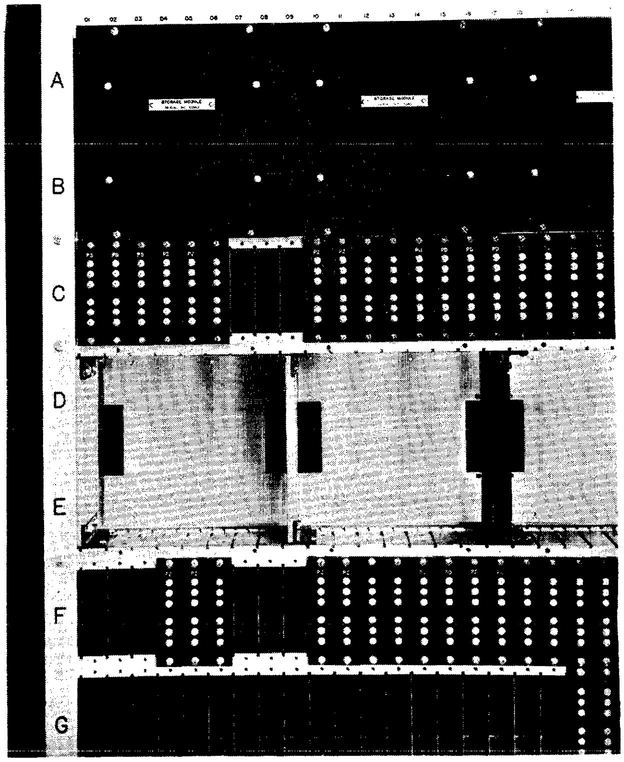


Figure 1-8 Typical 6400 Chassis

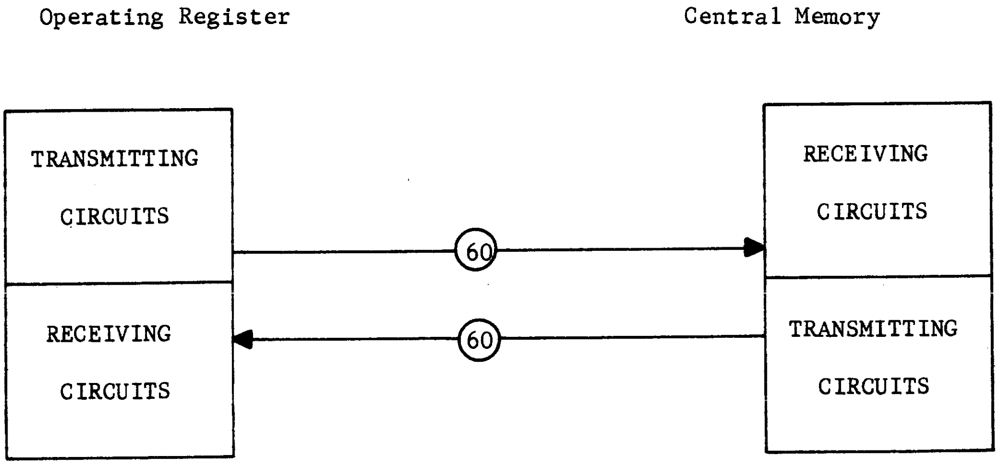


Figure 1-9

The transmission capability of this trunk is 600 million bits per second. This would be about the same as transmitting all the information in a large city telephone directory from memory to the operating register in one second. Transformer-coupled coaxial cable is also used for control signal transmission between chassis.

MAGNETIC CORE STORAGE

A unique package, the memory module (Figure 1-10), contains 49,152 magnetic cores used for storing bits of information. The size of each core is 0.012 inch inside diameter and 0.020 inch outside diameter. The cores are arranged on 12 planes, each plane having a 64x64 matrix. Each memory module has 4096 addressable words of 12 bits each. The memory module contains all the drive and inhibit circuits necessary to operate that module. Only an external Address Register, Restoration Register, and Sense Amplifiers are necessary to operate that module. The memory modules are plugged into the chassis in the same manner as the logic modules and are easily replaced if necessary. One module makes up each independent memory for the peripheral and control processors (see the 6400/6600 Introduction and Peripheral Processors Training Manual).

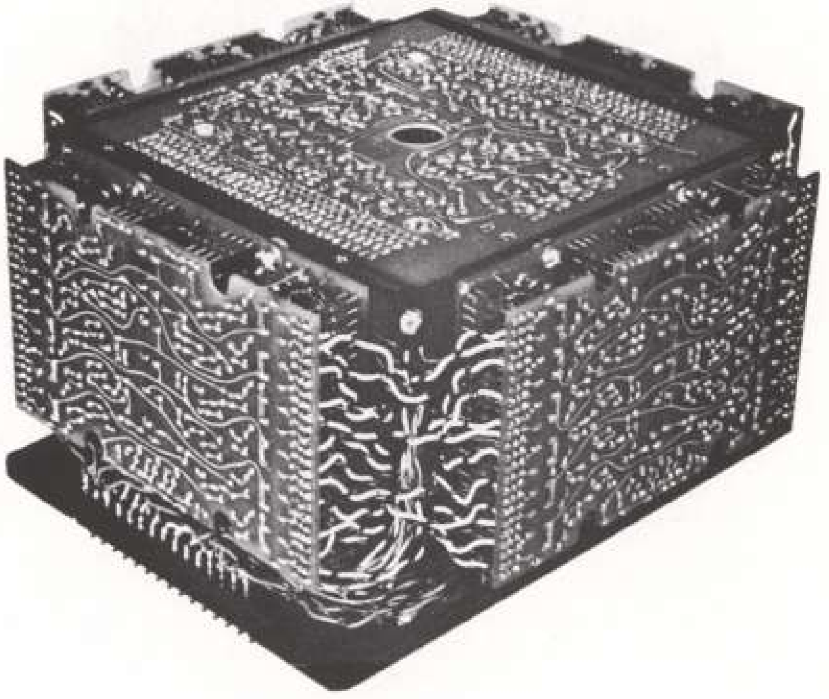


Figure 1-10. 6400 Memory Module

POWER SYSTEM

The 6400 computer uses a 30 KVA motor-generator set to supply power for the logic circuits. The input to the motor-generator is 208 vac, 3 phase, 60 cps; the output is regulated 208 vac, 3 phase, 400 cps which is stepped down, rectified and filtered to the +6 vdc used by the logic circuits. There are 3 other voltages used by the memory modules: +5.8 vdc for internal logic, +4.8 vdc for their inhibit circuits, and +15 vdc which goes through a current regulator and should be somewhere between 5.2 to 8.0 vdc for the drive circuits. (See the 6400/6600 Introduction and Peripheral Processors Training Manual for further details.)

SYSTEM ELEMENTS

Following is a description of the various possible elements within a 6400 system:

6415A - Central Computer: With 32,768 60-bit words of magnetic core storage, ten peripheral and control processors each with 4,096 12-bit words of magenetic core storage. Included with the peripheral processors are twelve bi-directional data channels and one data channel exclusively used for real time clock monitor.

6414A - Same as 6415A except with 65,536 60-bit words of magnetic core storage.

6413A - Same as 6414A except with 131,072 60-bit words of magnetic core storage.

SYSTEM BLOCK DIAGRAMS

There are several system configurations possible with a 6400 computer, each would have certain advantages depending on the needs of the user.

BASIC 6400 COMPUTER SYSTEM

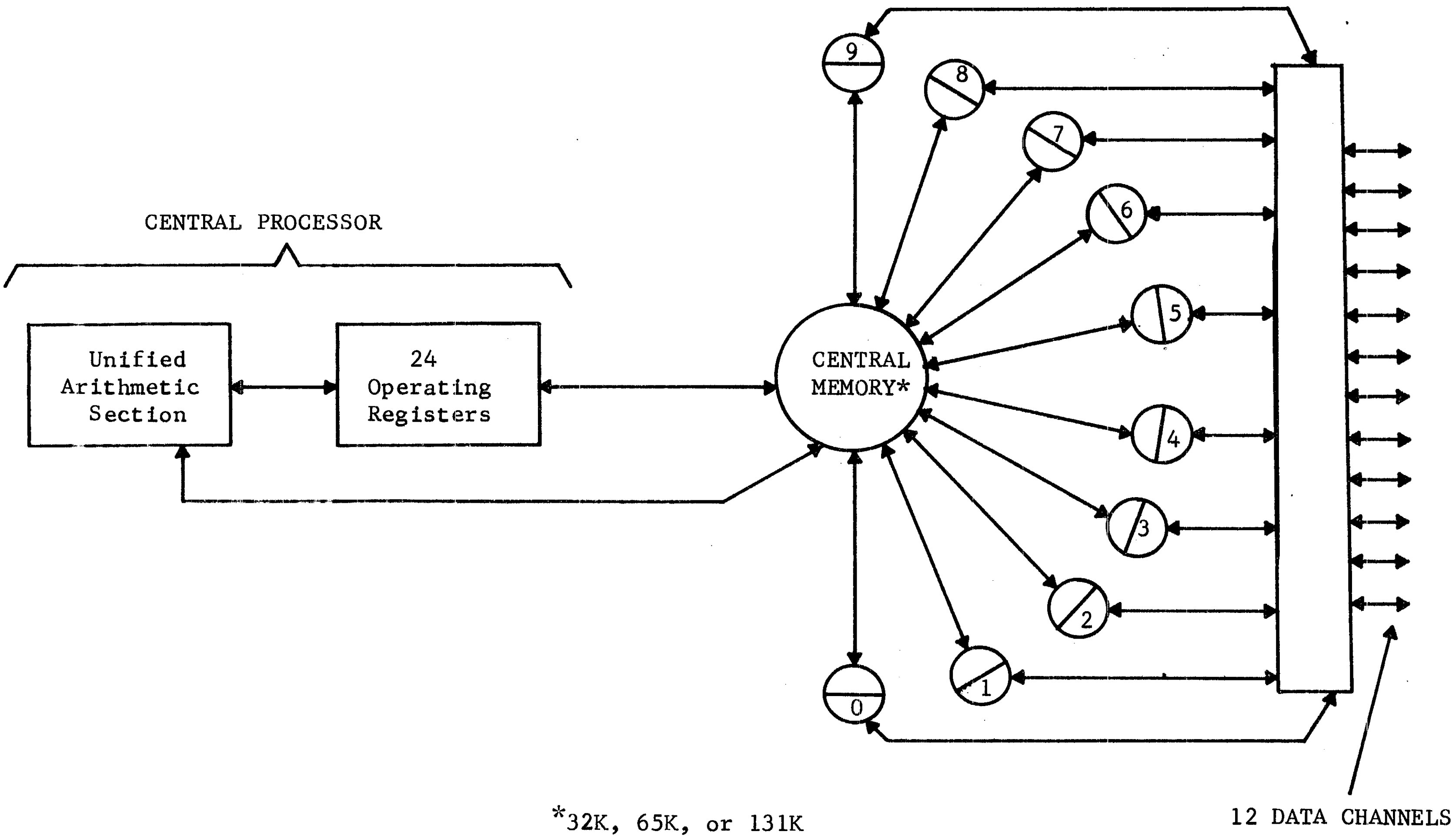
The basic 6400 system which includes a Central Processor, ten Peripheral Processors and twelve Data Channels. This basic system can have 3 memory variations (32K, 65K or 131K) of 60-bit Central Memory (Figure 1-11).

SPECIAL PURPOSE CENTRAL PROCESSOR

A Special Purpose Central Processor can be added that will improve the arithmetic capabilities of the basic system. This Special Purpose Central Processor can communicate with both the standard Peripheral Processors and the standard CPU through the Central Memory. The Special Central Processor shares the Central Memory on an equal basis with the standard CPU.

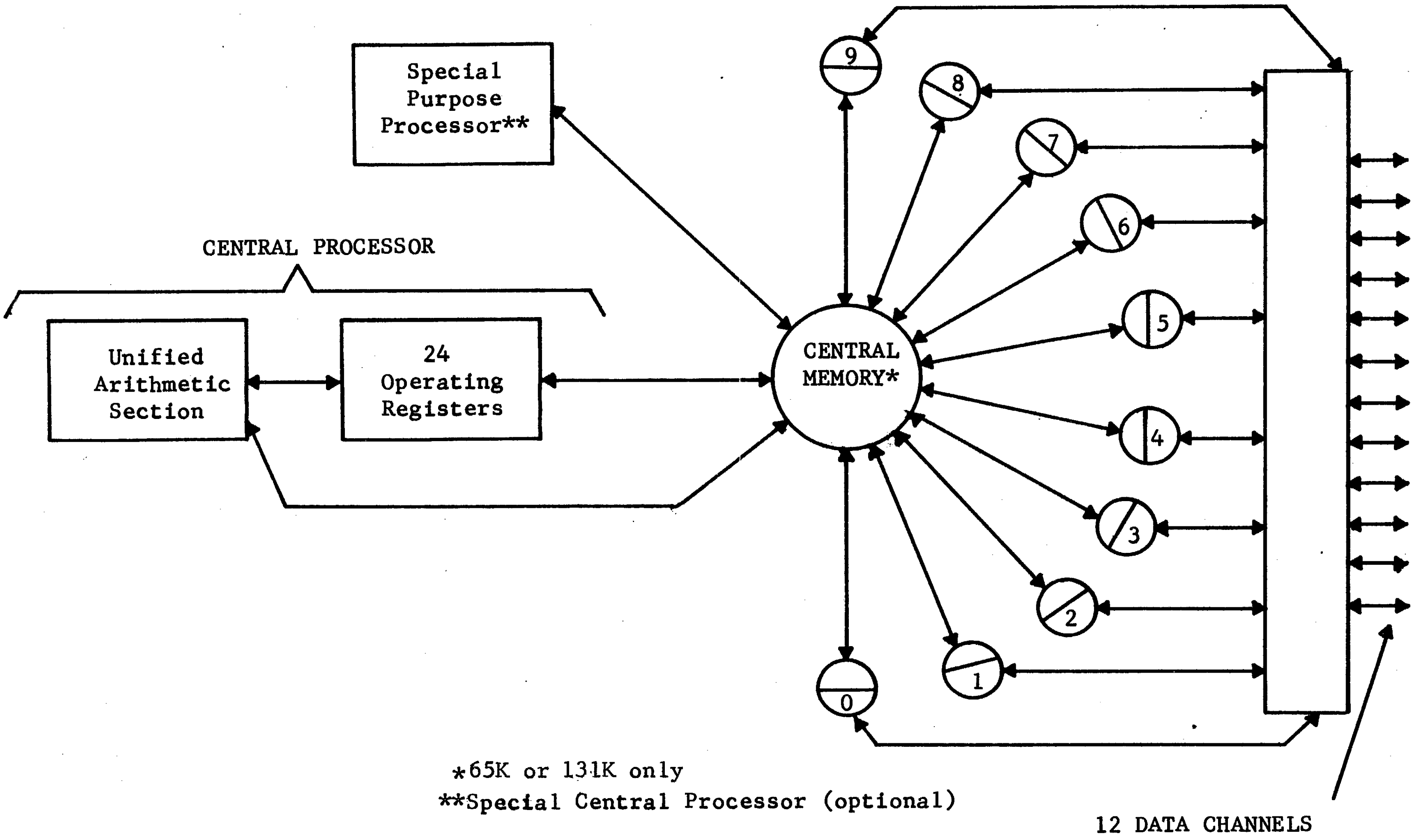
6681 DATA CHANNEL CONVERTER

The 6681 allows the 6400 peripheral processors to communicate with 3000 series I/O equipment. The converter acts as a signal interface and allows users who presently have 3000 series computers or equipment to use them with the 6400 system. The 6681 must be the first equipment on the channel and can control up to eight pieces of equipment. Figure 1-13 illustrates the relationship of a 6681 with other parts of a 6400 Computer system.



*32K, 65K, or 131K

Figure 1-11. Basic 6400 System



*65K or 131K only
**Special Central Processor (optional)

Figure 1-12 6400 System With a Special Purpose Processor

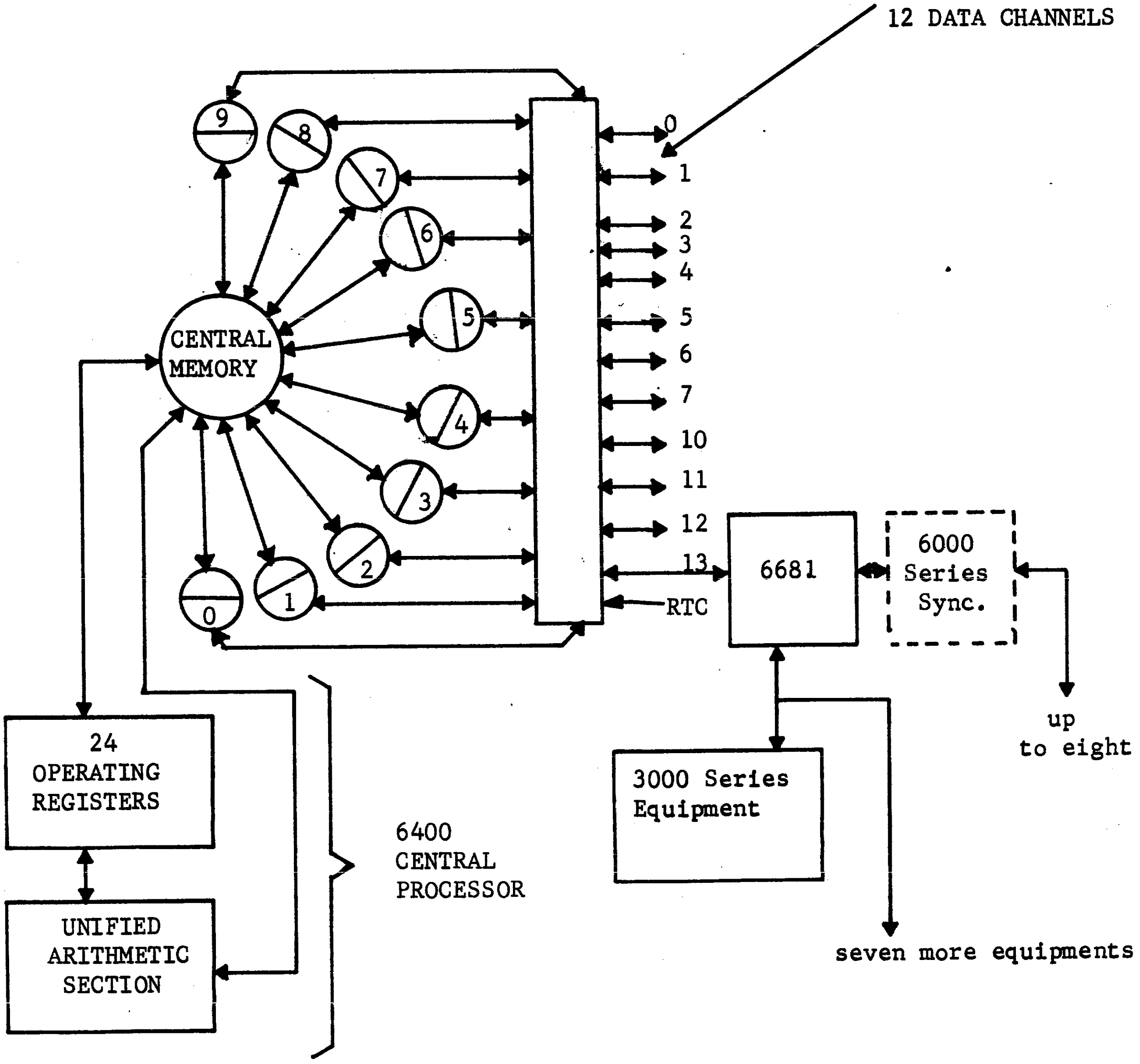


Figure 1-13 6681 Data Channel Converter

CHAPTER II

6400 PERIPHERAL PROCESSORS

CHAPTER II

6400 PERIPHERAL PROCESSORS

The 6400 Peripheral Processors are almost identical with the 6600 PPU's described in the Introduction to the 6400/6600 Training Manual. There are slight differences in the 26XX and 27XX instructions, the X registers, and a few module type changes.

26XX INSTRUCTION

The 260X instruction differs from its 6600 Computer counter-part in that the lowest bit is used to determine which CPU to Exchange Jump. The 2^0 bit equaling "0" means CPU-0 (Chassis 2); the 2^0 bit equaling a "1" means CPU-1, (Chassis 8) the Special Central Processor.

The 261X instruction has the 2^0 bit CPU definition and is conditioned upon the contents of a "FLAG" flip-flop in Central Memory Control. The instruction may or may not Exchange Jump CPU-X, the 261X would normally be used only with an ECS System. (See 6400/6600 ECS Training Manual.)

See the section covering Exchange Jumps for further details.

27XX INSTRUCTIONS

The 270X instruction differs from its 6600 Computer counter-part in that the 2^0 bit selects the CPU. The 2^0 bit equaling "0" means "Read P of CPU-0". The 2^0 bit equaling "1" means "Read P of CPU-1" (CPU-0 is the standard CPU located on Chassis 2, CPU-1 is the Special Purpose CPU located on Chassis 8).

ADDITIONAL X REGISTER

The possibility of two Central Processor Units necessitated two X registers, one for CPU-0 (X0, the original register) and one for CPU-1 (X1, the additional register).

MODULE CHANGES

Although a few slots were available, the addition of the X register, the selection circuit to choose either X register, the 2^0 and 2^3 bit translation, etc., necessitated changes in module types for more compactness.

The D5 transmitter modules were changed from PL's (6 transmitters) to JQ's (10 transmitters). This enabled reducing 10 modules down to 6, a gain of 4 slots.

The A register transmitters were changed from PL's to JQ's as well, providing a gain of one slot, plus a few extra transmitter circuits for other uses.

The X1 register was added (3 PI's) along with the 3 TE selection circuits.

These changes and the slots that were previously open allowed the 6600 Peripheral Processors to be converted to 6400's.

CHAPTER III

6400 CENTRAL MEMORY

CHAPTER III

6400 CENTRAL MEMORY

INTRODUCTION

Refer back to Figure 1-1, where the 6400 System General Block Diagram illustrated the Central Memory (CM) as the main storage section of the 6400. The 12-bit-word, 4K memories of the PPU's usually act only as small "batching" memories prior to writing into CM. Figure 3-1 illustrates the connections to and from CM.

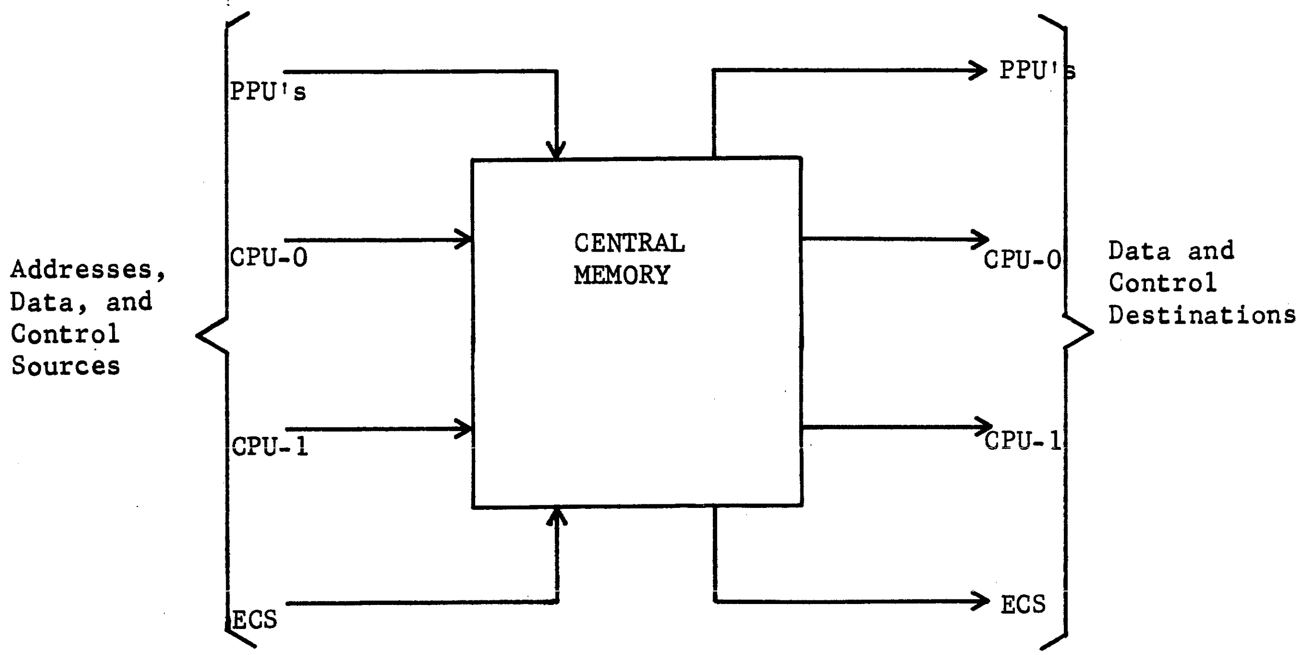

Addresses, Data, and Control Sources
PPU's
CPU-0
CPU-1
ECS
CENTRAL MEMORY
PPU's
CPU-0
CPU-1
ECS
Data and Control Destinations

Figure 3-1

There are four possible sources of Addresses, Data, and Control Signals: the PPU's for a CM Read, CM Write, Exchange Jumps, ECS Reads, or ECS Writes; the CPU-1 (Special Purpose Processor) used for CM Reads, CM Write, and Exchange Jumps; and the ECS source from the ECS Coupler which would be controlled by CPU-0 or CPU-1. The Data would always be 60 bits in size. The Addresses have 18 bits but, with 65K in the maximum size, only 16 bits are needed. The Control Signals are specialized for the particular operation.

The Central Memory can be divided into three parts: The Central Memory Control (CMC), Central Memory Banks (CMB), and the Data Distributor (Figure 3-2).

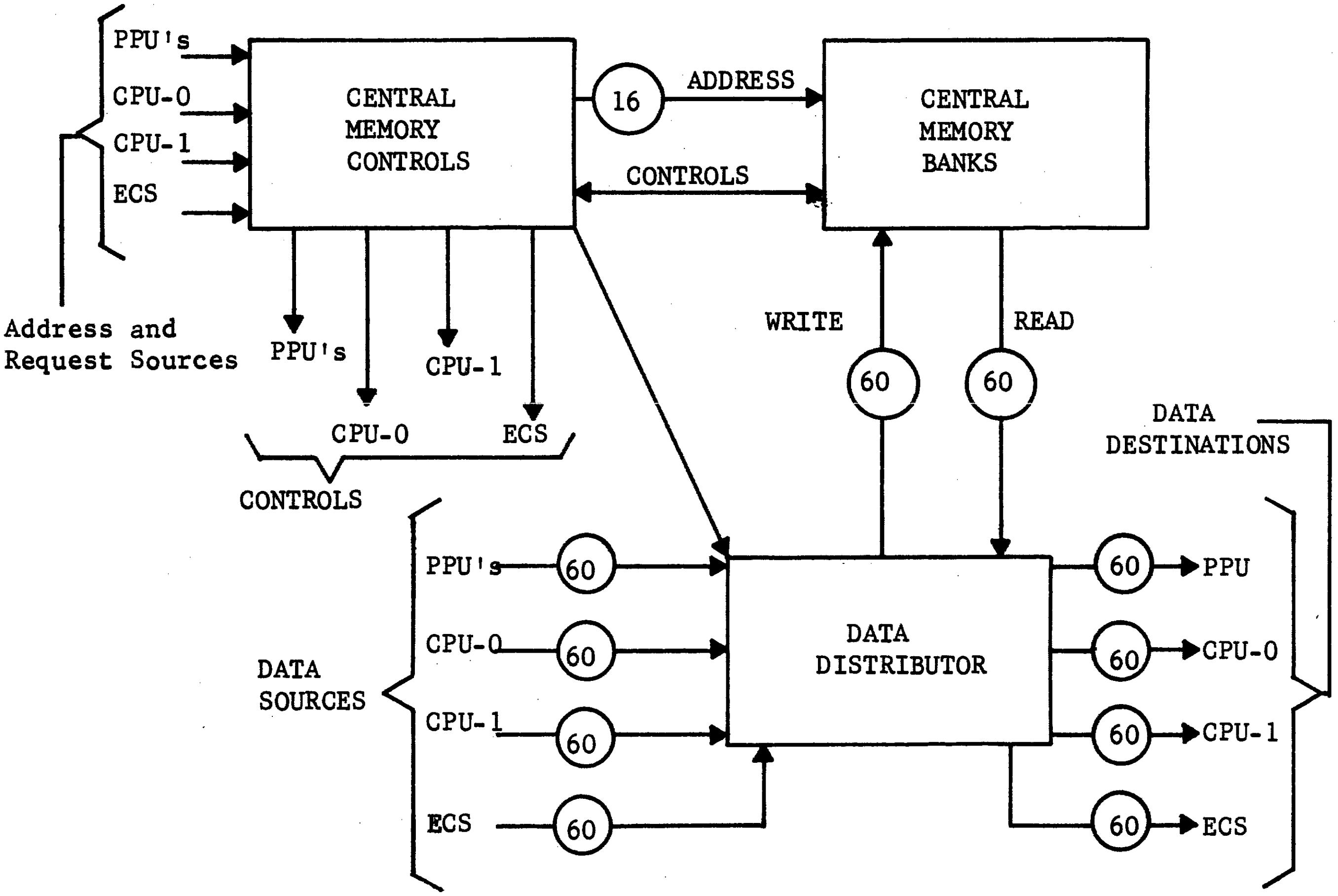


Figure 3-2

CENTRAL MEMORY BANKS

BANK LAYOUT

Central Memory is arranged in a number of independent banks (the number of banks depends on the size of the memory). Each bank has 4,096 60-bit words and is relatively independent. The banks are arranged to allow certain elements to be used commonly with other banks. Four banks are arranged on any one chassis. Common elements for those four banks are the Chassis Catching Register (CCR), the GO and ACCEPT Control, and the restore path. Each bank on a chassis has its own Storage Address Register (SAR), its own Storage Sequence Control (SSC), and its own Restore Register (Z).

Figure 3-3 is a block diagram of a typical Central Memory Chassis (6400 Chassis #3).

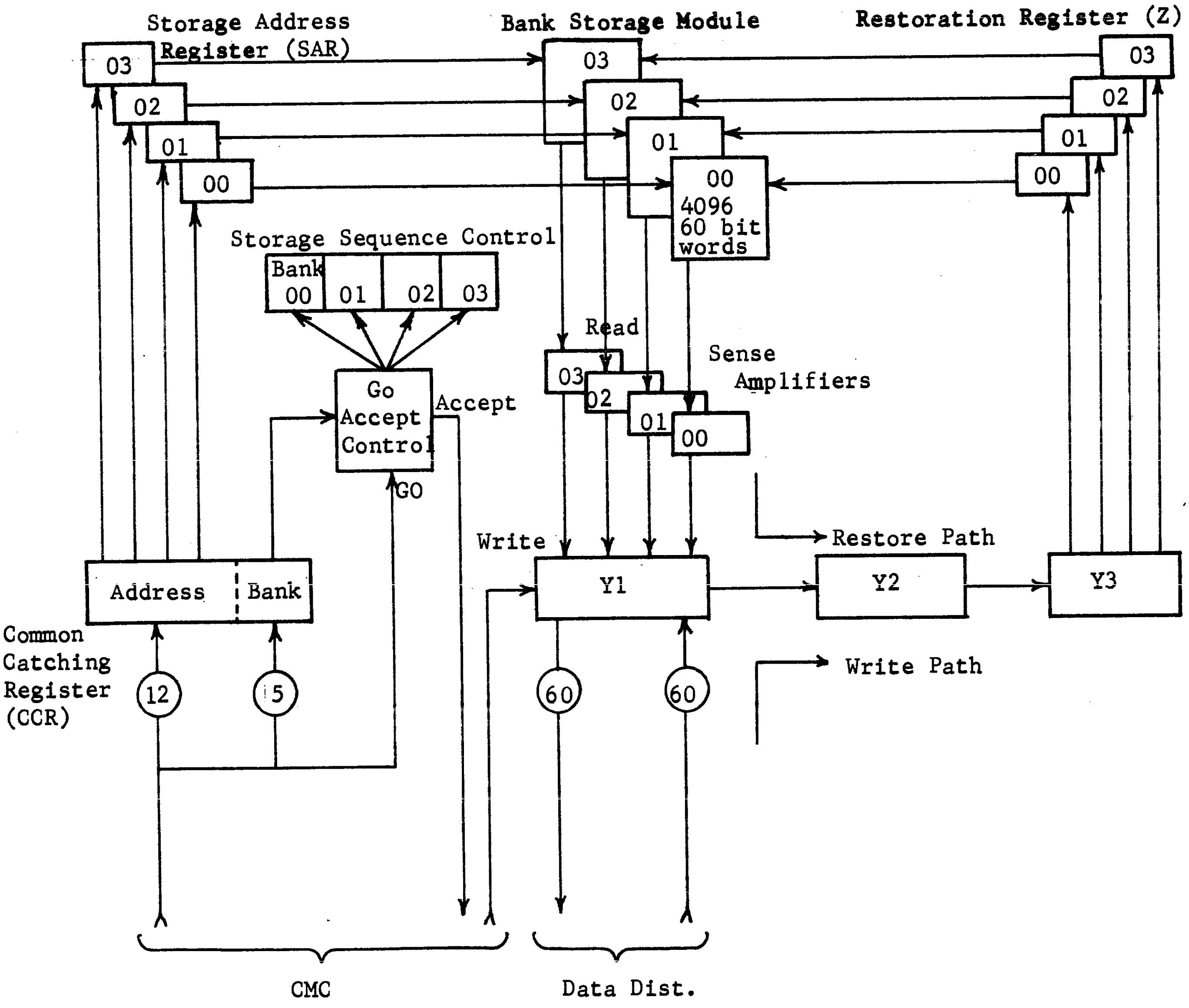


Figure 3-3. Central Memory Chassis

Chassis #4 contains Banks 04-07; Chassis #5 has banks 10-13; and Chassis #7 has banks 14-17, for a total of 16 banks which provide 65K 60-bit words. The memory modules are arranged physically such that the lower order bits are to the left when viewed from the test-point side of the chassis.

PHASING AND ADDRESS BREAKDOWN

The 6400 CM Bands are "phased" to further aid in speeding up the overall memory operation. The address sent to CMB from CMC is interpreted as follows:

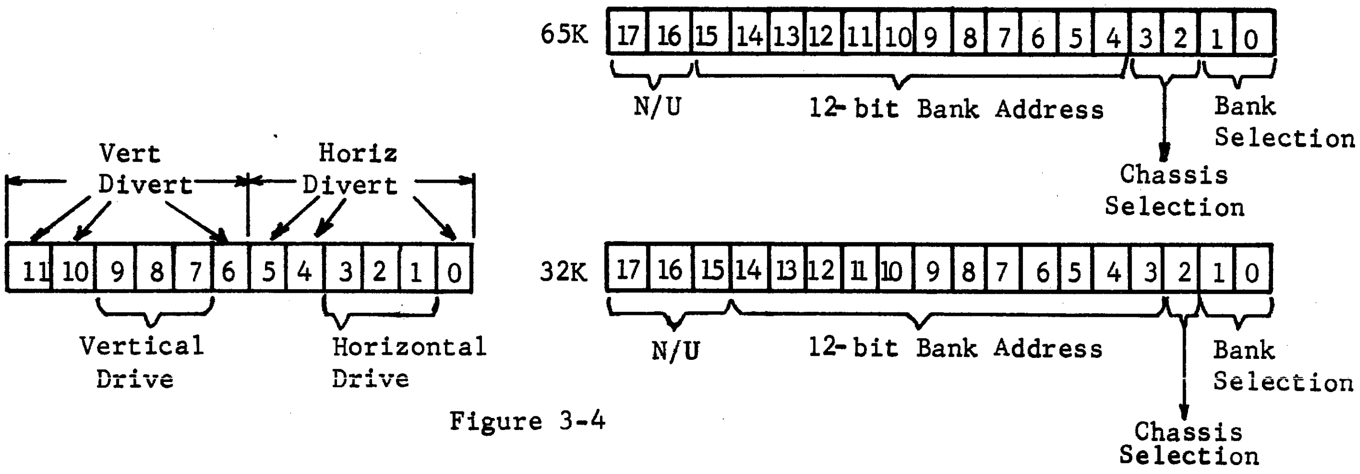


Figure 3-4

The lower two bits determine the bank--one of four on a chassis. The next one or two bits will determine which chassis the needed bank is on. The next twelve bits will be the bank address. The upper two or three bits will not be used (Figure 3-4). As addresses are referenced in sequential order, the banks are referenced in sequential manner. (Address 000000 is in bank 00, 000001 in 01, 000002 in 02, 000003 in 03, 000004 in 04, 000005 in 05, etc. When the highest numbered bank is reached, the next address would be in the lowest numbered bank.)

OPERATION

When CMB is to be referenced, CMC sends the address and a Go signal to all chasses. The Go and Accept Control circuits on each chassis check to see if that chassis has the address. The proper chassis will send an Accept signal back to CMC, (if that bank is not busy), and also start the Storage Sequence Control of the bank. Data flow from only the selected bank memory modules, to the Sense Amplifiers, to a fan-in of the 4 banks, to Y1. The Data then flow to the Data Distributor, on a Read operation and sampled; on a write operation,it would be egnored. During a write operation, a Write signal from CMC clears Y1 (erasing the Data from memory) and new Data flow in from the Data Distributor. The Data flow through Y1, Y2, and Y3, and to the selected bank's Z (Restoration) register, and into the memory modules.

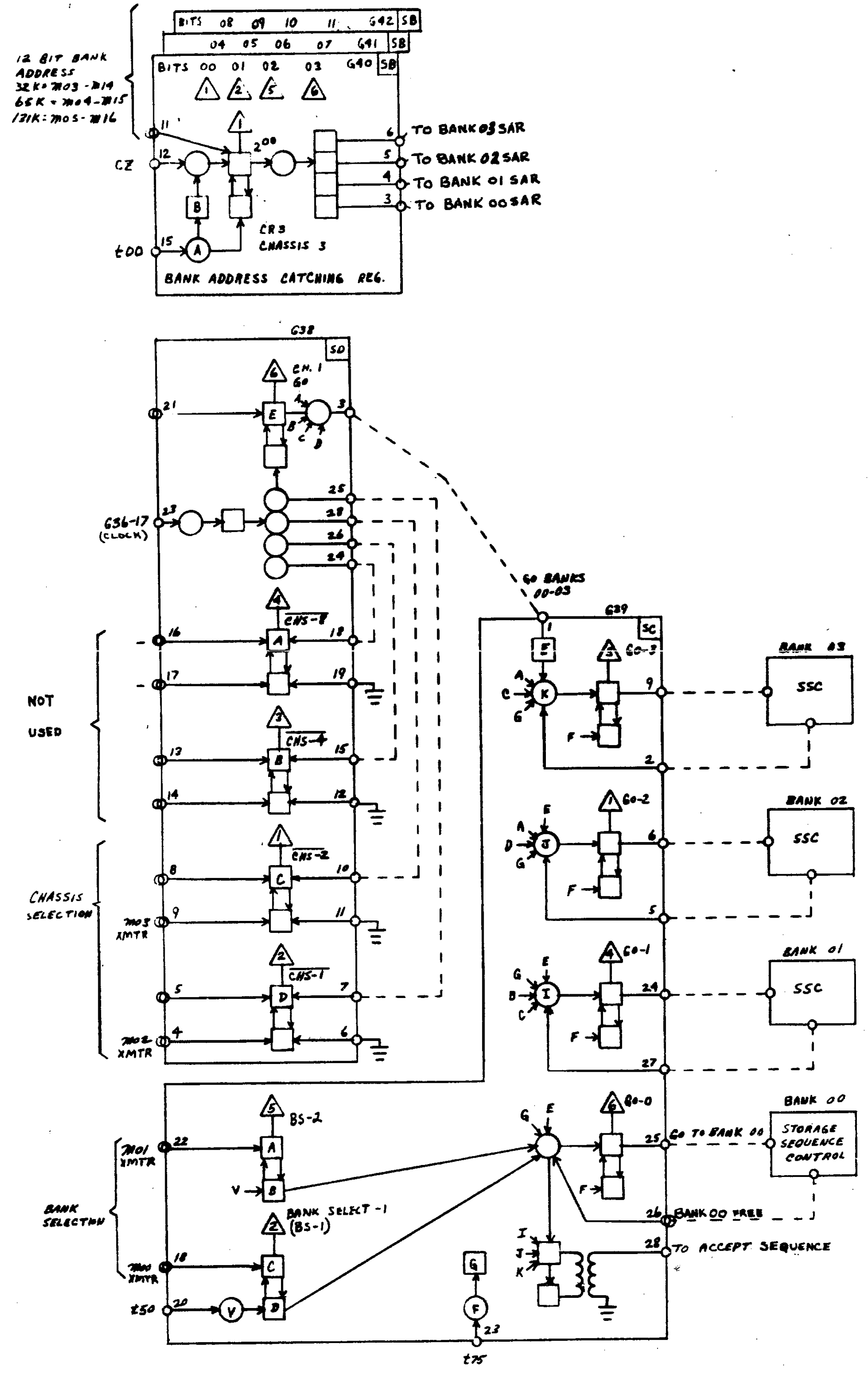
GO AND ACCEPT CONTROL

The GO and Accept Control circuits decide if the address sent from CMC is for one of the Banks on this chassis. If it is and that bank is free, (not busy), an Accept signal is sent back to CMC. The circuits for this operation on Chassis #3 are shown in Figure 3-5. The GO and the Address are received by the Catching FFs on the SB, SC and SD modules. The SB modules receive the Bank address (12 bits). The SD modules receive the Chassis Selection bits which, on a 65K memory, would be CMB address, bits 2^2 and 2^3.

A clock signal is uniquely connected so that the C and D FFs are set (or cleared) to the complement of the bit selection for that chassis. To help clarify this, notice that C and D in Figure 3-5 are set every 100 nanoseconds. Assume address 000300 is requested. The lowest two bits would go to the Bank Selection FFs. The next two bits would go to the Chassis Selection bits, FFs C and D. Notice that the signals are connected to the cleared side inputs. The signal to be sent would be a "0", meaning that the transmitter is not fired and, thus, C and D remain set. The set sides are used to enable the GO bit AND circuit to pin e. FFs A and B remain unchanged so that when the GO bit and Address arrive, the Bank-0 AND circuit at pin 3 is satisfied. This would cause pin 3 to go to a "0" and E (on the SC module) to become a "1". The Bank Select FFs would have a selection of B-D equaling "1's" for address 000300. With E enabling the gates, only the Bank Free signal remains to fulfill the gates. When a "1", this signal indicates that the Storage Sequence circuits of that bank are not functioning. If this is the case, the AND circuit is fulfilled, setting the GO FF for Bank D, and, at the same time, transmitting an ACCEPT back to CMC from the ACCEPT transmitter at pin 28. The GO-0 FF will start Bank D's Storage Sequence Controls (SSC) which would drop the Bank Free signal. The SSC will control and time the cycling of Bank 0.

If address 000302 were requested, the action would be the same as just explained except that the Bank Selection would have A-D as a combination and Bank 02 would be selected.

Chassis #4 would have its SD module wired differently than Chassis #3. The 2^2 input would be wired to input pin 5, the clock to pin 7, and pin 6 to ground. This means that only a Chassis Selection of 01 (binary) would leave C and D both "1's". The point to remember is that if the address is on the chassis, the AND circuit at pin 3 must be fulfilled.



LOGICAL SELECTION	CHASSIS NO. SELECTION
0	3
1	4
2	5
3	7
4	9
5	10
6	11
7	12

CHASSIS	BANKS
3	0-3
4	4-7
5	10-13
7	14-17

Figure 3-5

DATA DISTRIBUTOR

The Data Distributor can be divided into 3 parts: the Read Distributors, the Write Distributor, and the Control circuits for both enabled by CMC (Figure 3-6).

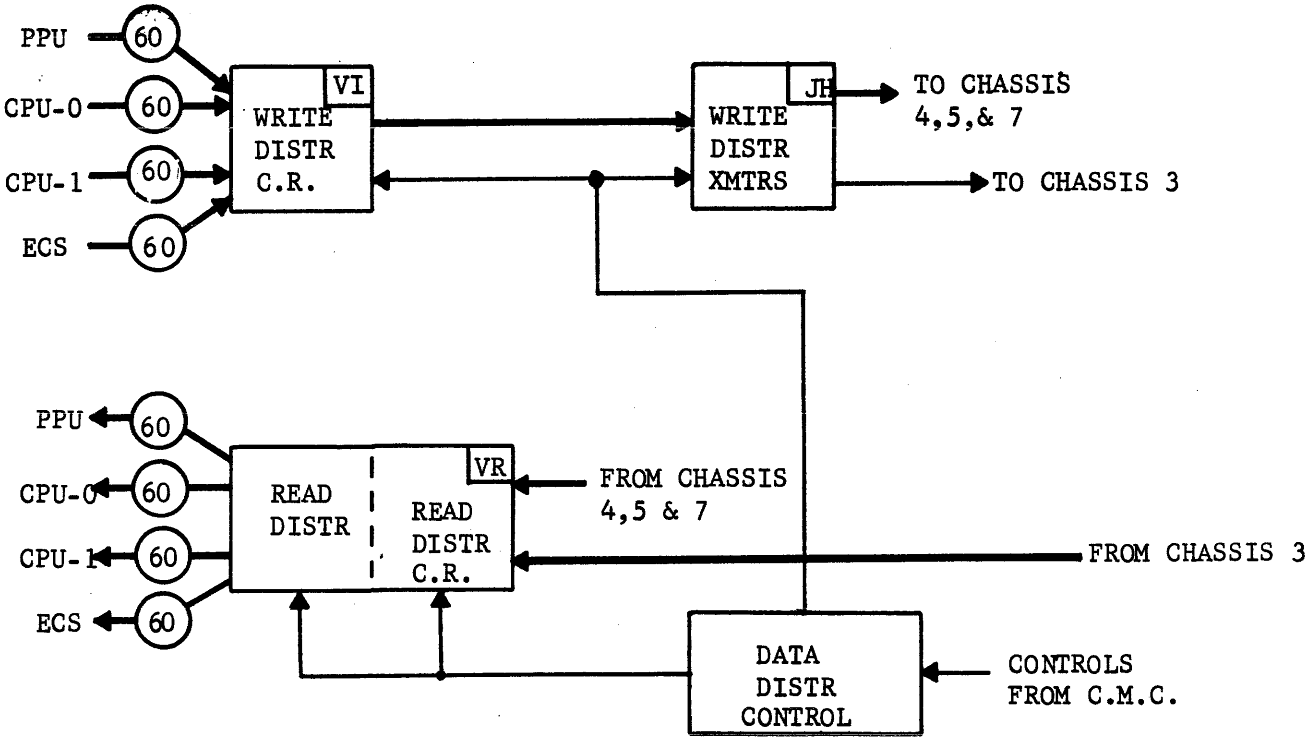


Figure 3-6. Data Distributor

The need for the Data Distributor may be seen in Figure 3-6. The Data Distributor is the final part of the fan-in/fan-out network of Data to or from the Central Memory Banks. On a Read operation (with 65K) the fan-in/fan-out arrangement would be from one of sixteen banks to one of four Read Transmitters to the Read Distributor Catching Register. The data are then retransmitted from the Read Distributor to one of the four possible requesting sources.

NOTE

> The data may actually be sent to more than one of the sources but only one of the four will use it at that particular time.

On a write operation, the data are sent from one of four possible sources to one of two possible Write Distributor Catching Registers; then they are retransmitted to one of four Y1 registers, and finally to one of sixteen possible Z registers.

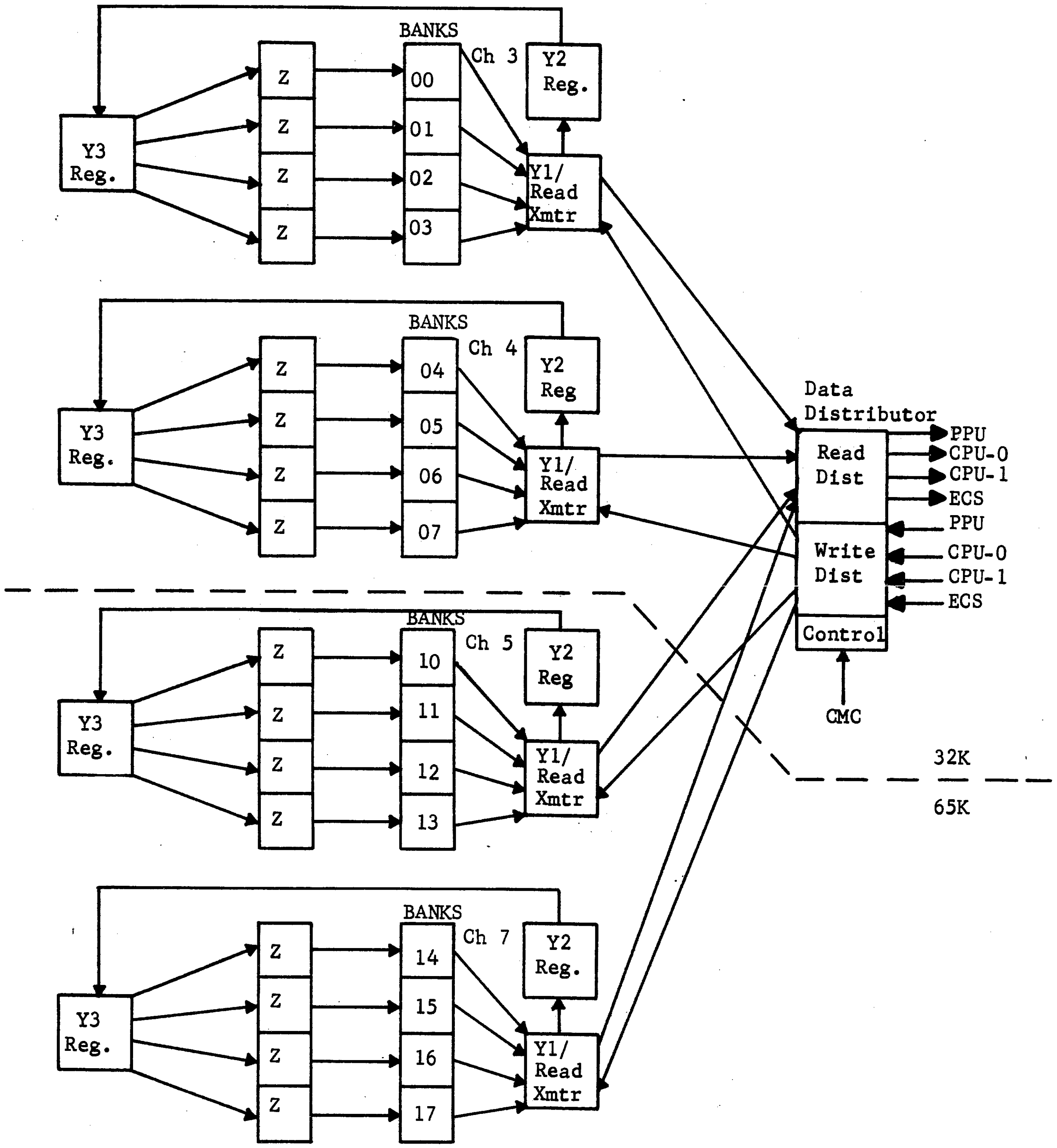


Figure 3-6 32K/65K CMB/Data Distributor

CENTRAL MEMORY CONTROLS

CMC is the overall control center of CM. It receives the address, requests, and other signals that may be necessary, then initiates and controls the operation within itself, the CMB, and the Data Distributor until the operation is complete. (Refer back to Figure 3-2, if necessary; also, refer to Figure 3-7.)

ELEMENTS OF CENTRAL MEMORY CONTROL

Figure 3-7 is a block diagram showing paths for addresses as they flow through CMC. Description of the various elements of CMC follows.

Catching Registers (CRO, CR1, CR2)

Catching registers are ungated flip flops that receive data from a coaxial cable. Since the input is not gated, there is an ability to de-skew data transmitted from chassis to chassis. The CMC has three such catching registers. They are:

1. Catching Register 0 (CRO)--for Central Processor 0 addresses
2. Catching Register 1 (CR1)--for Central Processor 1 addresses
3. Catching Register 2 (CR2)--Peripheral Processor addresses

CRO/1 will receive all addresses sent by the CPU's. The address may be for an operand (or an instruction) or may be the starting address for ECS transfer. CR2 will receive all addresses sent by any of the Peripheral Processors. The address may be for Data Read (or Data Write) or may be the starting address for an Exchange Jump. CRO and CR1 will also be used as holding registers. For example, CRO and CR1 will not be cleared on normal CPU requests until after the address has been accepted. This is necessary since there may be an attempt to re-try the address if no Accept is received. CRO and CR1 will also act as holding registers during ECS transfer. During this operation, the updated new address formed by the Address Incrementor will go to CRO/CR1 as part of the recirculation path for the counter.

PERIPHERAL ADDRESS REGISTER (PAR)

The PAR is a secondary register associated with CR2. This register is necessary since the PPU's are unconditionally sending their A register contents to CMC every 100 nanoseconds. Each word received by CR2 is automatically gated to PAR. When PPU's intend the word to be an address, they will send a request along with the address. This request will block the input gating term to PAR after the desired address is in PAR. The blocking input will protect the address until it can be accepted by the CMB (see Figure 3-8).

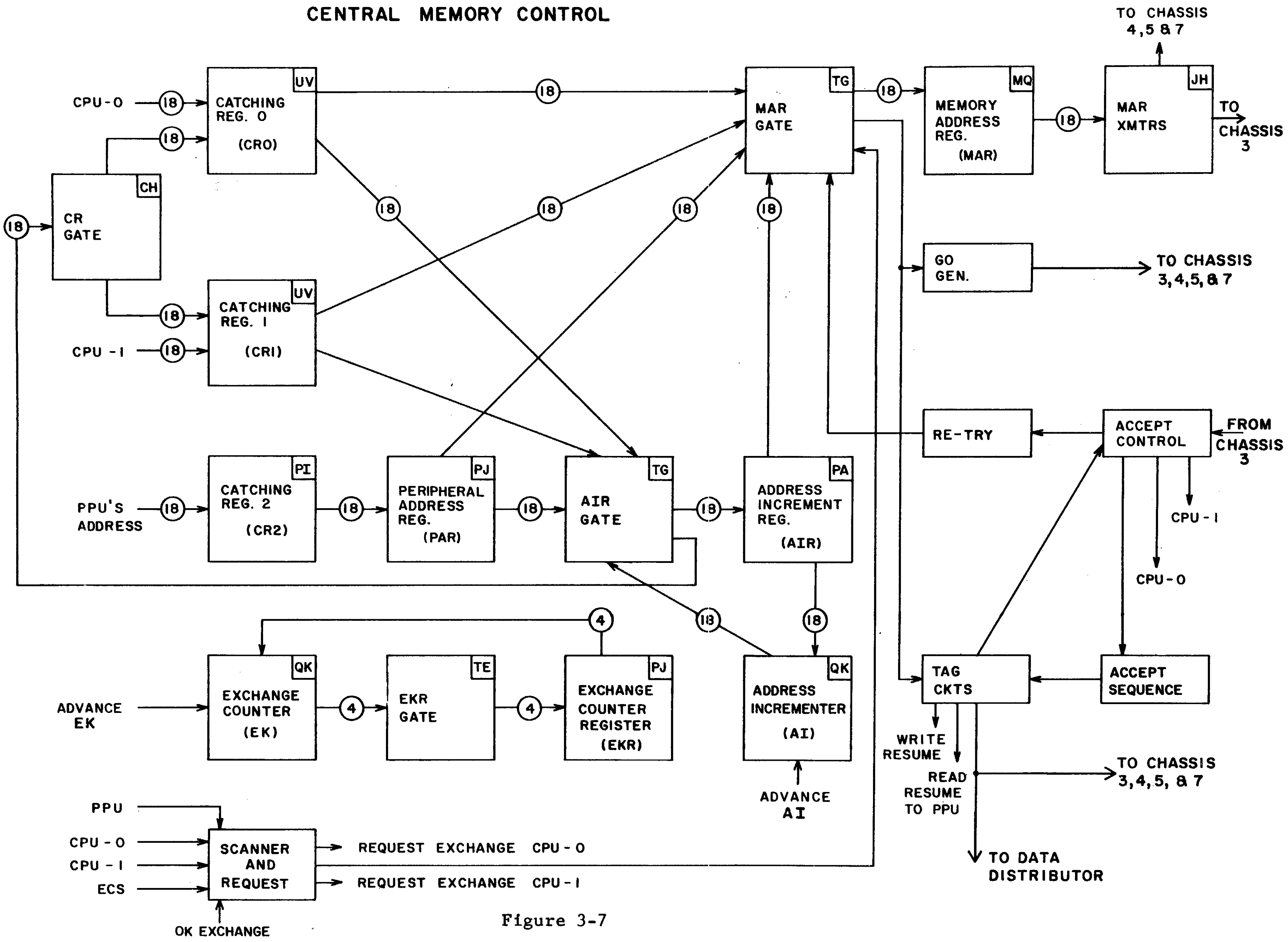

CENTRAL MEMORY CONTROL
CPU-O
18
CATCHING REG. O (CRO)
UV
CH
CR GATE
CATCHING REG. I (CRI)
CPU-I
PPU'S ADDRESS
CATCHING REG. 2 (CR2)
PI
PERIPHERAL ADDRESS REG. (PAR)
PJ
AIR GATE
TG
ADDRESS INCREMENT REG. (AIR)
PA
MAR GATE
MEMORY ADDRESS REG. (MAR)
MQ
MAR XMTRS
JH
TO CHASSIS 4,5 & 7
TO CHASSIS 3
GO GEN.
TO CHASSIS 3,4,5, & 7
RE-TRY
ACCEPT CONTROL
FROM CHASSIS 3
CPU-I
CPU-O
ACCEPT SEQUENCE
TAG CKTS
WRITE RESUME
READ RESUME TO PPU
TO CHASSIS 3,4,5, & 7
TO DATA DISTRIBUTOR
4
ADVANCE EK
EXCHANGE COUNTER (EK)
QK
EKR GATE
TE
EXCHANGE COUNTER REGISTER (EKR)
PJ
ADDRESS INCREMENTER (AI)
QK
ADVANCE AI
PPU
CPU-O
CPU-I
ECS
SCANNER AND REQUEST
OK EXCHANGE
REQUEST EXCHANGE CPU-O
REQUEST EXCHANGE CPU-I

Figure 3-7

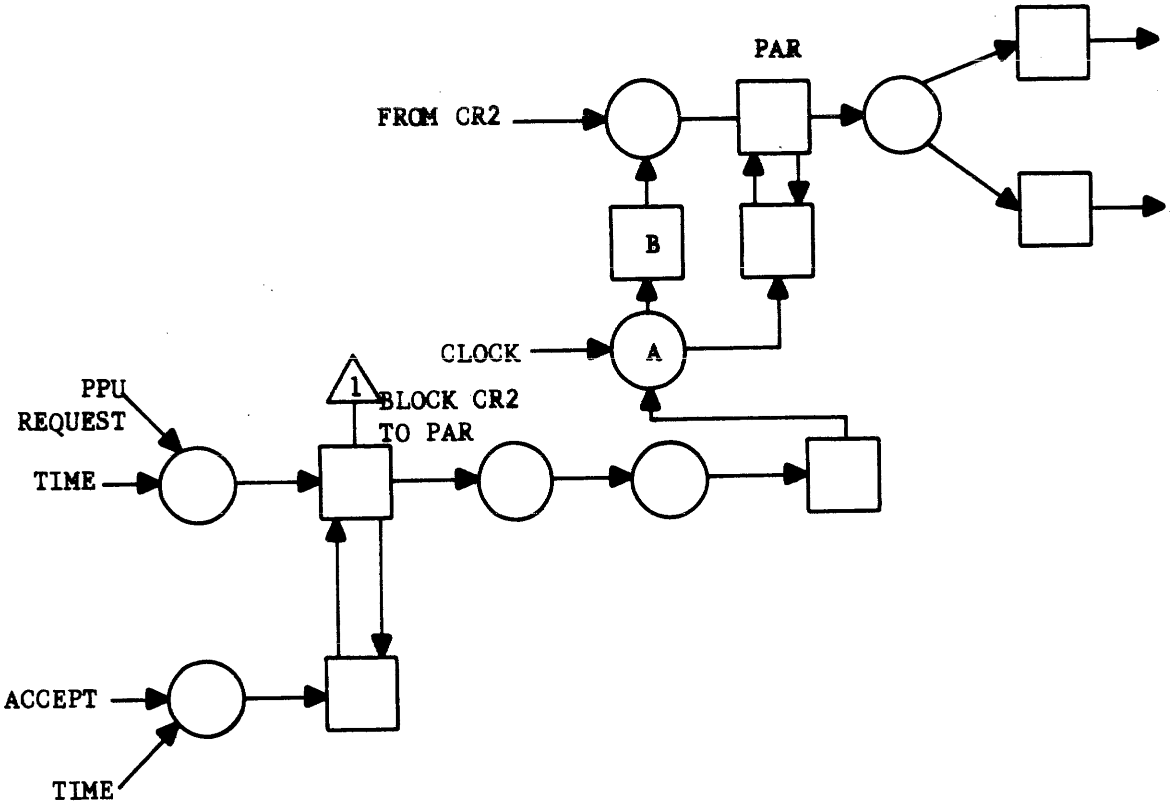


Figure 3-8

GATE Circuits

The GATE circuits allow the movement of the addresses from a register to the proper point depending on the operation. A typical GATE circuit is illustrated in Figure 3-9.

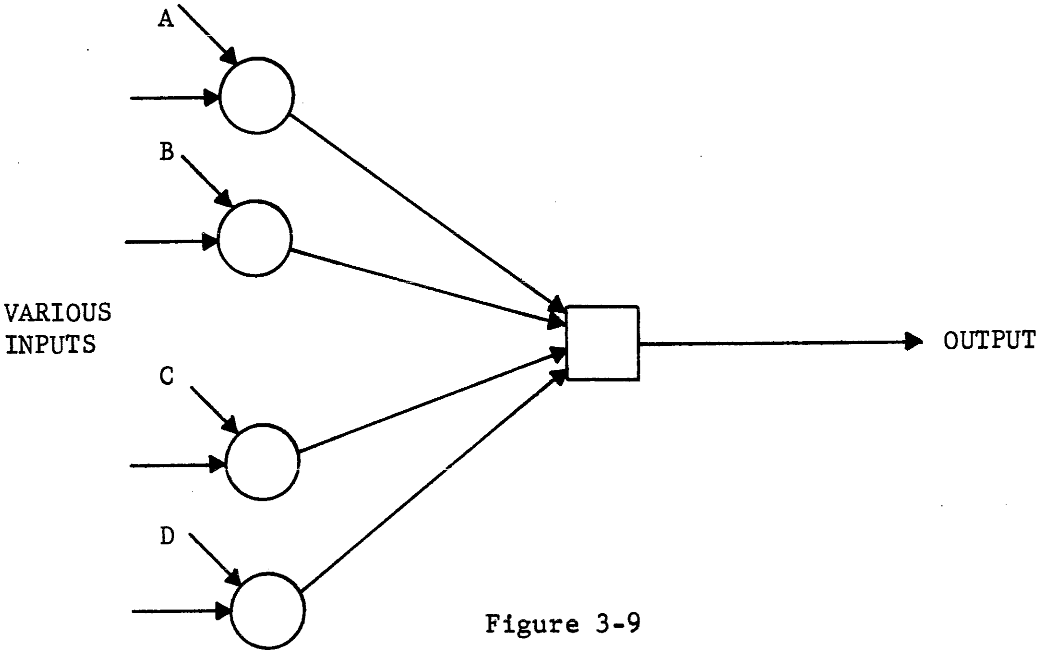


Figure 3-9

The Memory Address Register Gate (MAR GATE) gates one of the four possible inputs to the Memory Address Register; such gating is used on all CMC operations.

The Address Incrementor Register Gate (AIR GATE) gate one of the four possible inputs to the address Incrementer Regsiter, during Exchange Jumps and ECS operations.

The Catching Register Gates (CR GATES) gate the updated contents of the Exchange Counter Register (EKR) from the Exchange Counter to the EKR, during Exchange Jumps.

The Tag Bit Gate (TB Gate) is used to gate a Go, No-Go condition to the tag Bit Sequences. This is used on all CMC operations.

Memory Address and Address Incrementer Registers (MAR & AIR)

Various other registers within CMC, such as the MAR register and the AI register, are used as common registers during CMC operations. MAR receives all addresses that are to be transmitted to CMB. AIR serves as a holding register for AI. Such would be the case during Exchange Jump and ECS operations.

Address Incrementer (AI)

AI is an 18-bit 2's complement counter. It is used to update the original address during an Exchange Jump and to update the ECS operations Exchange Counter (EK).

The Exchange Counter is used during Exchange Jumps only. It counts the addresses during an Exchange Jump to determine when the Exchange Jump is completed.

Tag Sequences

The tag sequences are used to identify memory references. Since several addresses may be operating in CMC Accept and which address are associated. The tag sequences are started when the address is transmitted to CMB. The sequence determines the time period before an Accept is expected. If no Accept is present, the Re-try circuit is initiated.

There are several tag sequences. They include:

Peripheral Tag Bit (PTB) - present on PPU Read or Write Operations

Write Tag Bit (WTB) - present any time there is a Write Operation intended from PPU's or CPU's (including Exchange Jump and ECS Operation)

X Tag Bit (XTB - present during an exchange jump or an ECS transfer

CPU 0 Tag Bit (COT) - present during CPU-0 request memory read and write and during peripheral exchange to CPU-0

CPU 1 Tab Bit (CIT) - present during CPU-1 request memory read and write and during peripheral exchange to CPU-1

A Tag Sequence cycle time is 325 nanoseconds and is initiated by the Tag Bit Gate circuit.

The GO to CMB will be sent when the tag sequences are started. If the Accept has not returned before near the end of the tag sequence, a Re-try will be initiated.

RE-TRY

The Re-try Sequence determines the time between GO signals for an address requesting CMB. If an address cannot be accepted because of a bank conflict, this address will again be sent to the CMB each 300 nanoseconds until accepted. The Re-try Sequence is initiated at the time the GO is sent to CMB. The Re-try Sequence is made up of three FFs: Re-try 1, Re-try 2 and Re-try 3. Re-try 3 will set only if there is no Accept received for the address that started the sequence. If Re-try 3 does set, the address and GO signal will be sent to CMB. This will also restart the Re-try Sequence.

Bank conflicts arise when an address is sent to a bank that is busy with a previous request. A bank is considered busy for the complete 1 usec cycle time of that memory. The situation that will cause the longest delay of a bank conflict occurs when four addresses try to access the same bank. The first address will make the bank busy and CMC will receive an Accept within the first 100 nanoseconds after issue. The second address issued to the bank will have to re-try three times before it will be accepted. The third address, issued 100 nanoseconds later, will retry six times before it is accepted. The fourth address will encounter the longest possible delay for an address because of bank confliction (see Figure 3-10 for an illustration of this confliction).

ACCEPT SEQUENCE

The Accept Sequence is initiated by an Accept signal returning from CMB. The Accept signals from all banks come into two ungated flip-flops (Accept 1 and Accept 2). The output of Accept 1 and Accept 2 are strobed into Accept 3 every minor cycle. If the Accept is for a Central Processor address, the Accept signal will be retransmitted back to the proper CPU determined by the Tag Sequence. This will allow the CPU to communicate with the Data Distributor at the proper time. The output of Accept 3 is sent to the Re-try circuit and will inhibit the setting of Re-try 3. This will stop the sending of successive GO signals for that address. The output of Accept 3 is also sent to a circuit that will examine it with respect to the Peripheral Tag Bit Sequence for the purpose of sending back the Resumes to the PPU's. The output of this circuit will also start the PPU Read or Write Sequences that communicate with the CMB Data Distributor, which times the placing or removing of the data in the Distributor for PPU operations. The remaining flip-flops in the Accept Sequence (Accept 4 through Accept 12) are used for the "All Quiet" network. The All Quiet is used to determine the starting of an Exchange Jump.

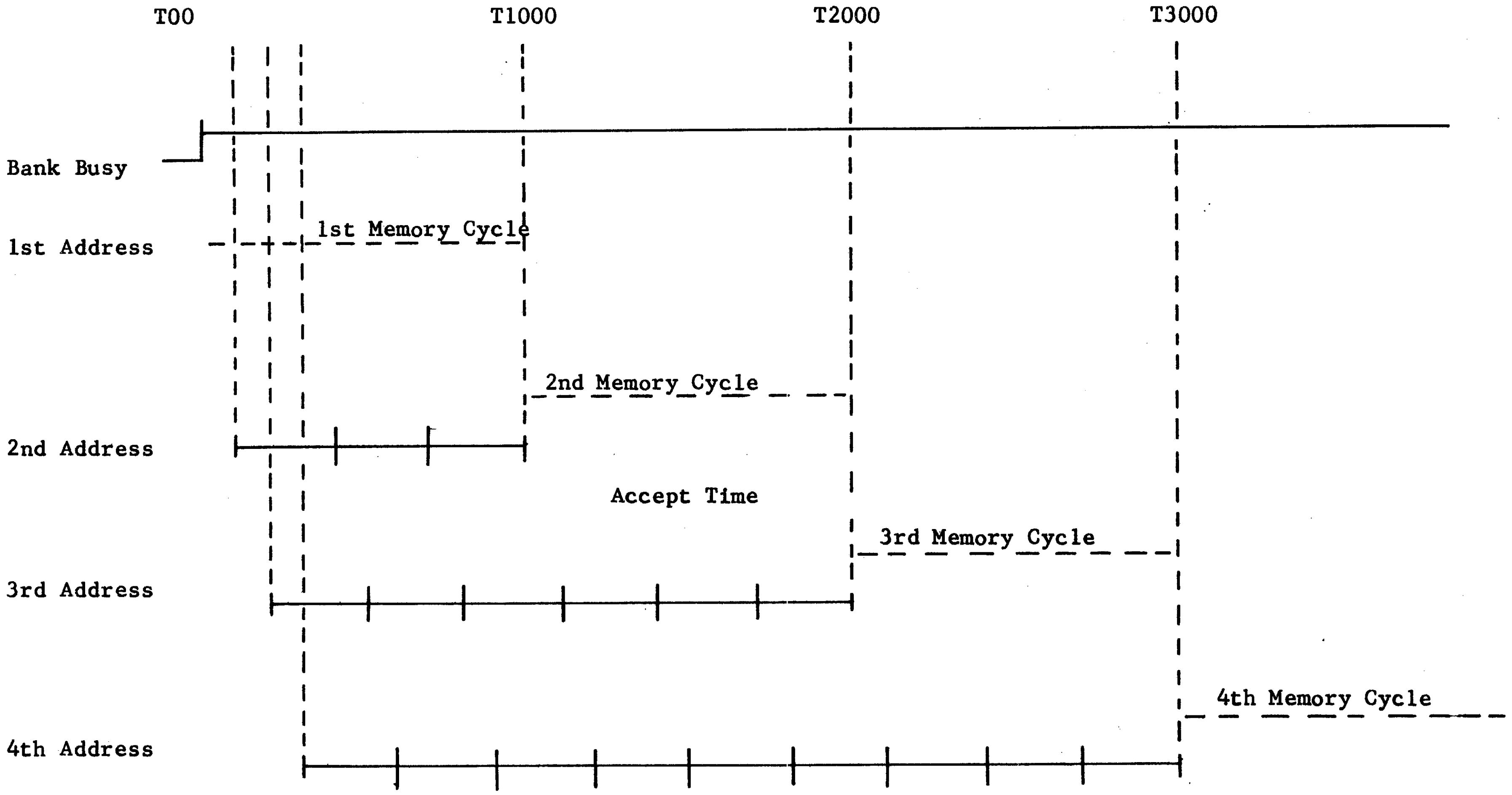


Figure 3-10 Memory Conflicts

SCANNER

Within CMC, some consideration must be given to the possibility of the PPU's, CPU-0, CPU-1, and ECS making simultaneous Requests. These simultaneous Requests must be recognized and sorted out to utilize the Central Memory in the most efficient manner possible. This sorting operation is done by the Scanner.

The Scanner contains two flip-flops interconnected to form a closed loop. At first glance, with no Requests, the circuit appears to be free-running. (The 1st slip-flop sets the 2nd, the 2nd clears the 1st, which then clears the 2nd, which then re-sets the 1st flip-flop, etc.) However, in actual operation, the Scanner locks up in a position with the set and cleared sides of both flip-flops equal to a "1" output. This method is faster than a free-running Scanner. Assuming no Requests are present, and then a Request occurs, the time for it to be recognized and the enable to leave the module is less than 20 ns (see Figure 3-11).

Central Memory Control recognizes Requests from either the PPU's, CPU-0, CPU-1, or ECS and generates a Stop Scanner signal corresponding to the device which made the request. Examining Figure 3-11, and assuming a PPU Request is present, it can be seen that the Scanner will stop with both bits of the counter cleared (00_2). This would give a counter translation of "Scanner = PPU", which allows the PPU Request to actually start a CMB reference. At this time, any other Requests would have to wait until the Scanner was released. The next device to have its Request honored would be CPU-0, then ECS, and finally CPU-1. Normally, the Scanner will only remain locked up for approximately 75 nanoseconds. However, when an Exchange Jump is being executed, the Scanner is stopped until the exchange is completed (approximately 1.6 usec). This ensures the rapid completion of the Exchange Jump by eliminating memory conflicts.

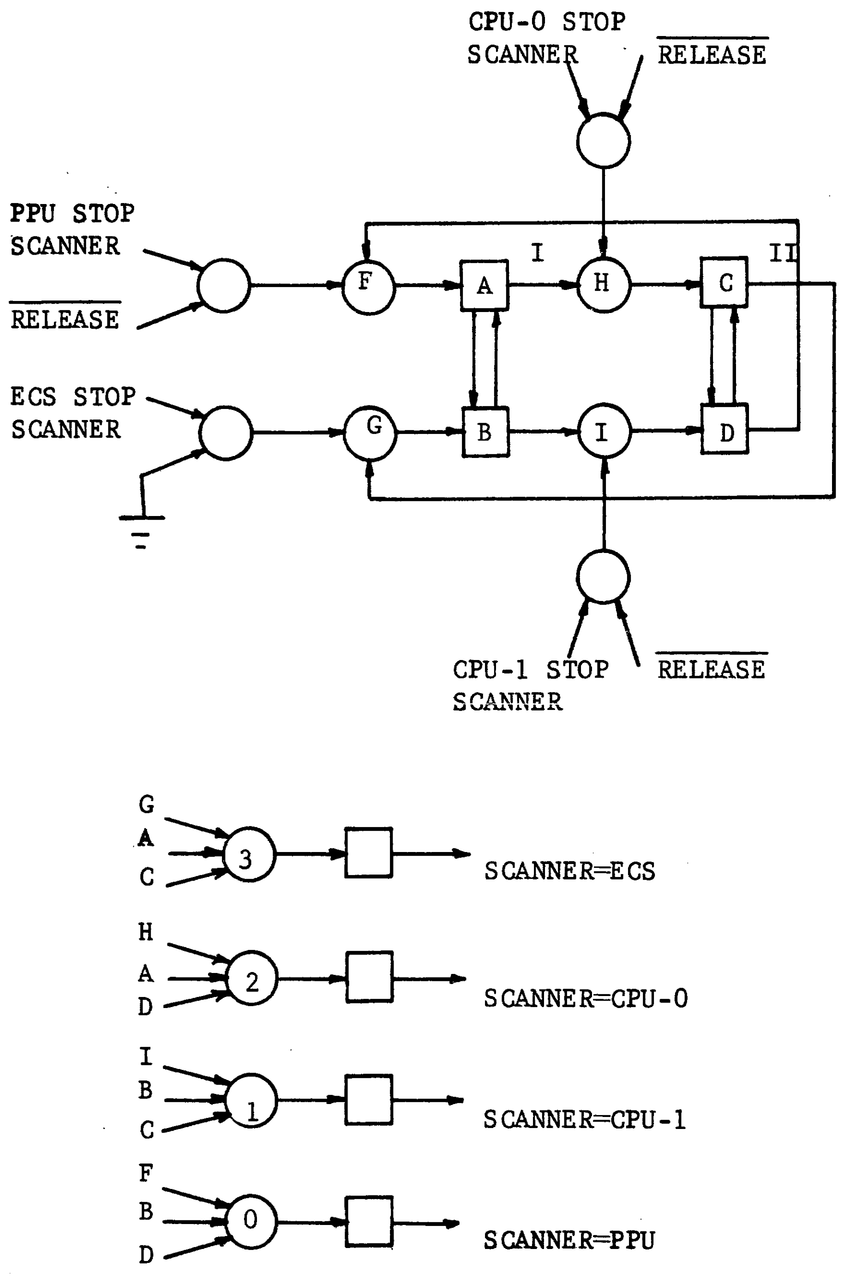


Figure 3-11. Scanner

CONTROL SIGNALS

Various signals flow between CMC, CMB, Data Distributors, the PPU's, the CPU's, and ECS. Table 3-1 lists and explains the source, destination, and function of each signal.

CENTRAL MEMORY CONTROL SIGNALS

NAME	FROM	TO	FUNCTION
1. CPU-0 REQUEST	CPU-0	CMC	Requests the use of CMC by CPU-0. Waits for an ACCEPT CPU-0 from CM.
2. CPU-1 REQUEST	CPU-1	CMC	Same as #1 only for CPU-1
3. GO	CMC	CMB	Banks will examine address sent to them. Will start the memory cycle of the proper bank if it it isn't busy. Enables sending back an ACCEPT.
4. ACCEPT	CMB	CMC	The address has been accepted and a memory cycle started. Starts the Accept Sequence in CMC.
5. ACCEPT CPU-0	CMC	CPU-0	CM has accepted address, and started memory cycle.
6. ACCEPT CPU-1	CMC	CPU-1	Same as #5, only for CPU-1.
7. WRITE	ECS/CPU's or PPU's	CMC to CMB	Enables clearing Y1 register when the old data is present. Allows new data from the Distributor to enter.
8. ECS REQUEST	ECS	CMC	ECS wants control of CM
9. ECS DEFINE CPU-1	ECS	CMC	If "1", = ECS is controlled by CPU-1. "0" = CPU-0.
10. ECS ACCEPT	CMB	CMC to ECS	Address examined, accepted and memory cycle started.
11. CENTRAL READ	PPU's	CMC	PPU wants control of CM for a Central read instruction.
12. CENTRAL WRITE	PPU's	CMC	PPU wants control of CM for a Central write instruction.
13. READ RESUME	CMC	PPU	Clear Central busy FF.

Table 3-1

CENTRAL MEMORY CONTROL SIGNALS (Continued)

	NAME	FROM	TO	FUNCTION
14.	C^5 Full	CMC	PPU	Data from CM is in PPU catching register
15.	PPU WRITE RESUME	CMC	PPU	Clear "CM busy" F/F in PPU.
16.	PPU WRITE	CMC	CMB	Enable data from proper CMB to Write Distributor, to PPU.
17.	EXCHANGE JUMP	PPU	CMC	PPU wants CM for an Exchange Jump instruction.
18.	EXCHANGE CPU-0	CMC	CPU-0	PPU wants to execute Exchange Jump with CPU-0.
19.	EXCHANGE CPU-1	CMC	CPU-1	PPU wants to execute exchange with CPU-1.
20.	OK EXCHANGE CPU-0	CPU-0	CMC	CPU-0 is now ready to execute an exchange.
21.	OK EXCHANGE CPU-1	CPU-1	CMC	CPU-1 is Now ready to execute an exchange.
22.	EXCHANGE RESUME	CMC	PPU	Clears "CM busy" F/F in PPU. Indicate PPI's can continue to next instruction.
23.	MONITOR	PPU	CMC	If a "1" it means do a MONITOR EXCHANGE JUMP.
24.	CPU-0 EXCHANGE REQUEST	CPU-0	CMC	CPU-0 wants to initiate an internal EXCHANGE JUMP.
25.	#0 EXCHANGE #1	CPU-0	CMC	If a "1", it means CPU-0 wants to internally exchange CPU-1
26.	CPU # Bit	PPU	CMC	If a "0", it means CPU-0 wants to internally exchange itself. A "1" implies PPU defines CPU-1, A "0" implies CPU-0.

Table 3-1

Generalization of memory request and the ensuing reactions are shown in **Figure** 3-12.

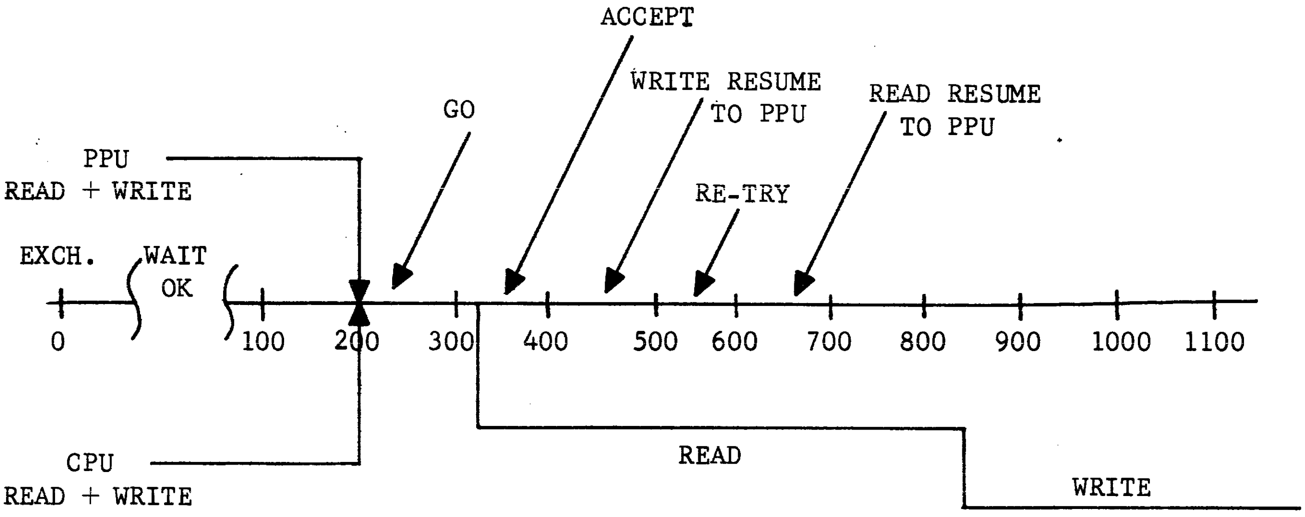


Figure 3-12 General Timing

Particular points to keep in mind are: Re-try is always 300 nanoseconds after the GO to CMB, if there is no Accept for that address. The Accept will be expected to return within 100 nanoseconds after sending the GO.

OVERALL CENTRAL MEMORY BLOCK DIAGRAM

Figure 3-13 ties the Central Memory Controls, Central Memory Banks, and Data Distributor together.

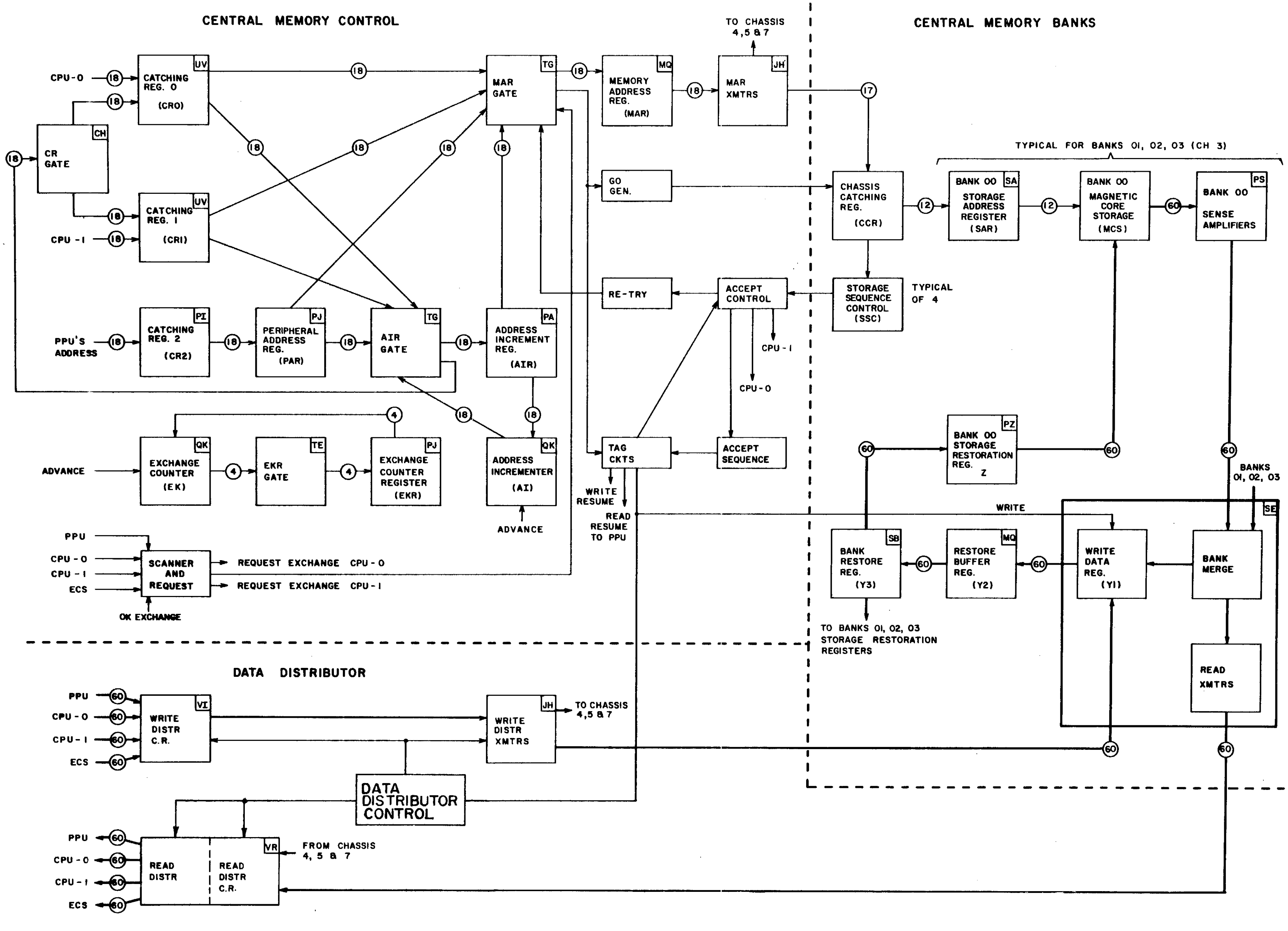

CENTRAL MEMORY CONTROL
CENTRAL MEMORY BANKS
DATA DISTRIBUTOR
CPU-0
CPU-1
CATCHING REG. 0 (CRO) UV
CATCHING REG. 1 (CR1) UV
CR GATE CH
PPU'S ADDRESS
CATCHING REG. 2 (CR2) PI
PERIPHERAL ADDRESS REG. (PAR) PJ
AIR GATE TG
ADDRESS INCREMENT REG. (AIR) PA
MAR GATE TG
MEMORY ADDRESS REG. (MAR) MQ
MAR XMTRS JH
TO CHASSIS 4,5 & 7
GO GEN.
RE-TRY
ACCEPT CONTROL
TAG CKTS
ACCEPT SEQUENCE
CPU-1
CPU-0
WRITE RESUME
READ RESUME TO PPU
ADVANCE
EXCHANGE COUNTER (EK) QK
EKR GATE TE
EXCHANGE COUNTER REGISTER (EKR) PJ
ADDRESS INCREMENTER (AI) QK
ADVANCE
PPU
CPU-0
CPU-1
ECS
SCANNER AND REQUEST
REQUEST EXCHANGE CPU-0
REQUEST EXCHANGE CPU-1
OK EXCHANGE
18
17
12
60
4
TYPICAL FOR BANKS 01, 02, 03 (CH 3)
CHASSIS CATCHING REG. (CCR)
BANK 00 STORAGE ADDRESS REGISTER (SAR) SA
BANK 00 MAGNETIC CORE STORAGE (MCS)
BANK 00 SENSE AMPLIFIERS PS
STORAGE SEQUENCE CONTROL (SSC)
TYPICAL OF 4
BANK 00 STORAGE RESTORATION REG. Z PZ
BANKS 01, 02, 03
WRITE
BANK RESTORE REG. (Y3) SB
RESTORE BUFFER REG. (Y2) MQ
WRITE DATA REG. (Y1)
BANK MERGE SE
READ XMTRS
TO BANKS 01, 02, 03 STORAGE RESTORATION REGISTERS
PPU
CPU-0
CPU-1
ECS
WRITE DISTR C.R. VI
WRITE DISTR XMTRS JH
TO CHASSIS 4,5 & 7
DATA DISTRIBUTOR CONTROL
READ DISTR
READ DISTR C.R. VR
FROM CHASSIS 4, 5 & 7
DATA
CONTROL

Figure 3-13

PERIPHERAL PROCESSOR READ/WRITE OF CENTRAL MEMORY

INTRODUCTION

When a Peripheral Processor executes a 60-63 instruction, communication becomes established between Central Memory and the processor. The read and write sequences of Central Memory control are nearly identical. The few differences will be pointed out during the following discussion of a Control Memory Request by a Peripheral Processor.

PPU OPERATION

Whenever the PPU initiates a Central Memory Read/Write, it sends an address from its A register to Central Memory. During a write operation, the processor sends a Central Write signal and the Data Word to Central Memory. After the Accept signal is received from CMB, Central Memory issues a Resume to the PPU. During a read operation, the processor sends a Central Read signal to Central Memory. The Resume from Central Memory is delayed so that it arrives at the PPU about the same time as the Data Word.

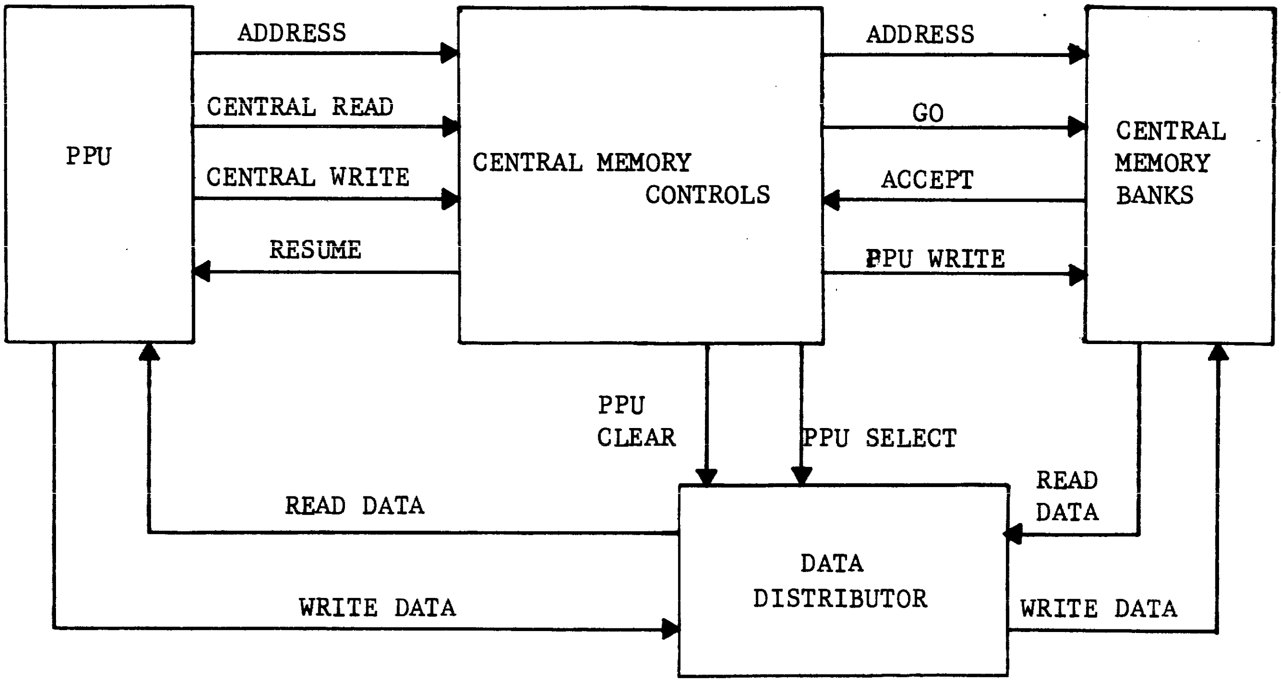


Figure 3-14

The address in CR2 will be sent to the Peripheral Address Register (PAR) on a timed pulse that comes every 100 nanoseconds. Once the address has been sent to PAR, it must be able to remain there because it may be re-issued every 300 nanoseconds until the Accept is received. To protect PAR, therefore, the minor cycle pulse on PAR's input must be disabled until the Accept is received from CMB. To disable PAR's input, the PPU Request flip-flop, set by Central Read or Central Write, will set the Block CR2 to PAR flip-flop, keeping the Store CR2 in PAR signal disabled. (Review Figure 3-8.)

The PPU Request will interrogate the Scanner with a Stop Scanner signal. When the Scanner recognizes the PPU, the translators will start the sequence of events necessary for the address and a GO signal to be sent to CMB.

Figure 3-16 illustrates the path of the address during a PPU Request of Central Memory. Once the scanner has started the sequence of events, the address in PAR will be transferred to the MAR register via the MAR Gate circuit. From MAR the address will be transmitted to CMB. At the same time as the address is sent to CMB, a GO signal is sent to CMB and the Peripheral Tag Sequence is started to time the wait for the Accept signal.

If the Accept does not return, the Re-try circuit will send the same address and another GO to CMB in 300 nanoseconds. This process will be continued until the address is accepted. Once the CMB starts, the data will flow from the Read Distributor or to the Write Distributor, whichever the situation warrants. The reception of an Accept in CMC also sends the necessary Resume back to the PPU's for control purposes. The PPU Select signal, during a PPU Write, will enable the data to flow from the Write Distributor Catching register to the CMB Y1 register. The Write signal to CMB clears Y1 just before the data word arrives. The PPU Clear signal clears the Write Distributor Catching register after it was sent to Y1 in preparation for the next PPU Write operation.

CENTRAL MEMORY CONTROL OPERATION

The address is received by the Catching Register #2 (CR2) of the Central Memory Control (CMC) at time zero. At the same time, the PPU Request flip-flop sets. The following flow chart (Figure 3-15) illustrates the path of the address and should allow a better understanding of the control sequence.

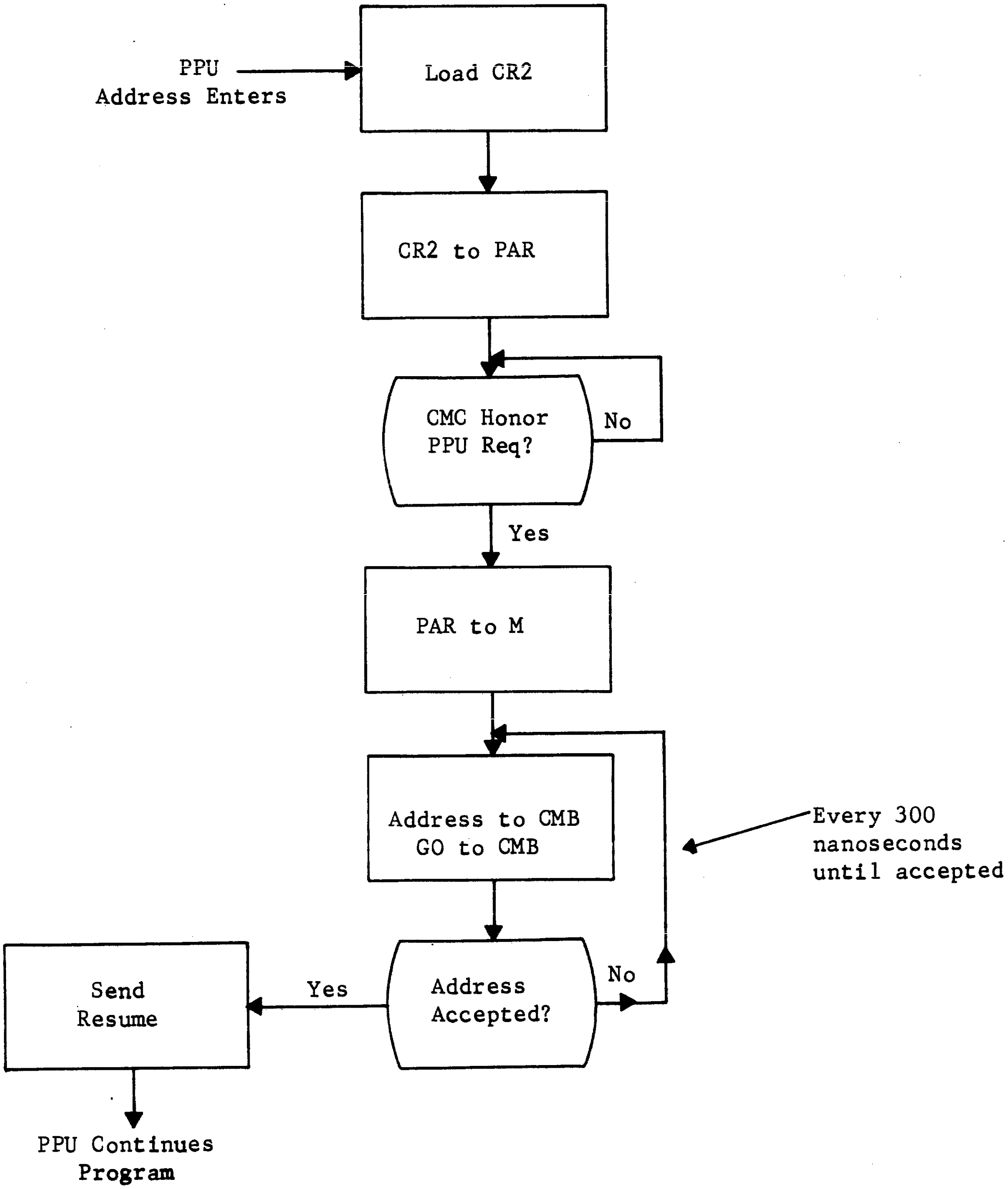


Figure 3-15

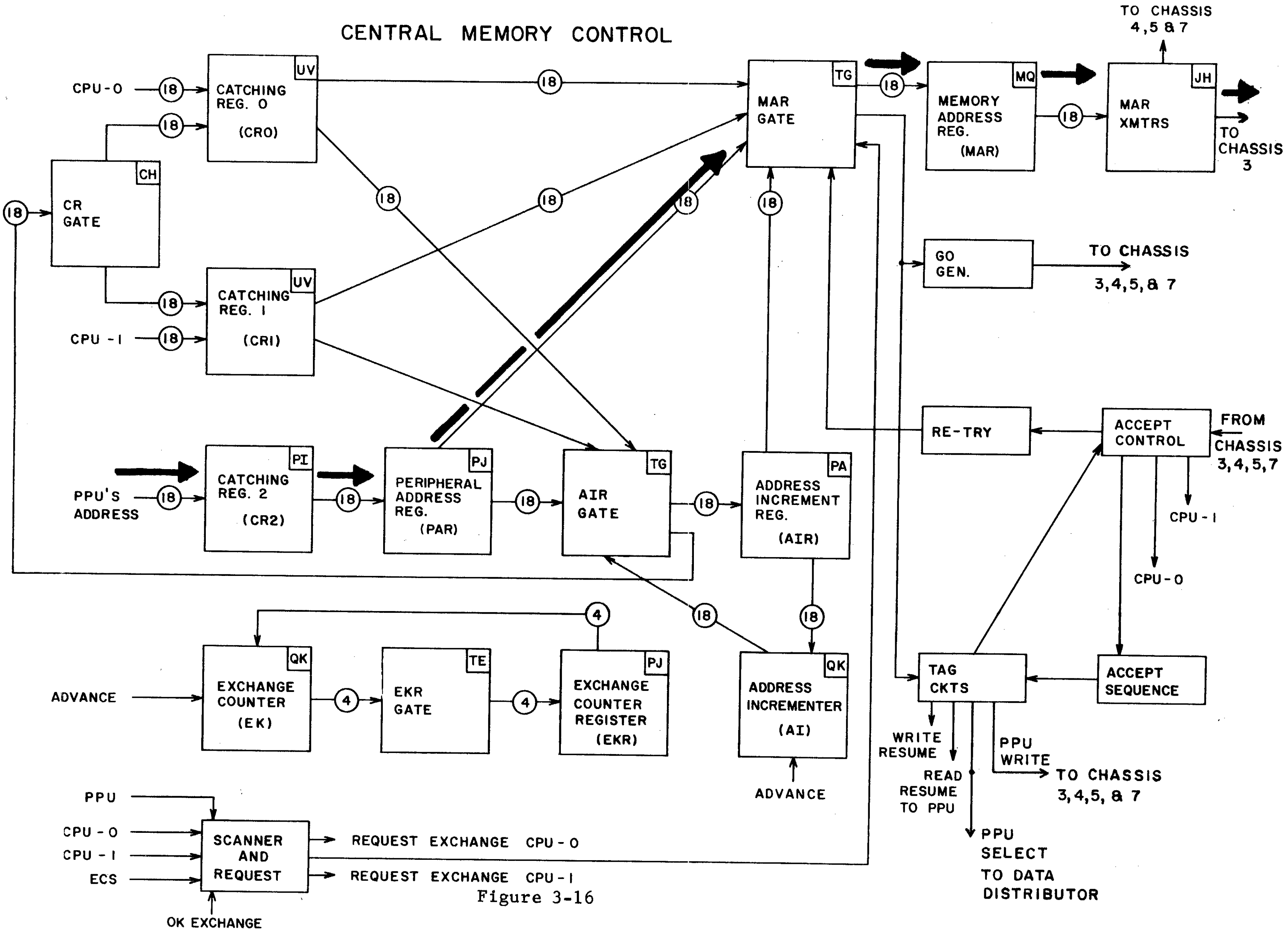

CENTRAL MEMORY CONTROL
CPU-O
18
CATCHING REG. O (CRO)
UV
CH
CR GATE
CATCHING REG. I (CRI)
CPU -I
PPU'S ADDRESS
CATCHING REG. 2 (CR2)
PI
PERIPHERAL ADDRESS REG. (PAR)
PJ
AIR GATE
TG
ADDRESS INCREMENT REG. (AIR)
PA
MAR GATE
MEMORY ADDRESS REG. (MAR)
MQ
MAR XMTRS
JH
TO CHASSIS 4,5 & 7
TO CHASSIS 3
GO GEN.
TO CHASSIS 3,4,5,& 7
RE-TRY
ACCEPT CONTROL
FROM CHASSIS 3,4,5,7
CPU - I
CPU-O
4
ADVANCE
EXCHANGE COUNTER (EK)
QK
EKR GATE
TE
EXCHANGE COUNTER REGISTER (EKR)
ADDRESS INCREMENTER (AI)
ADVANCE
TAG CKTS
ACCEPT SEQUENCE
WRITE RESUME
READ RESUME TO PPU
PPU WRITE
TO CHASSIS 3,4,5, & 7
PPU SELECT TO DATA DISTRIBUTOR
PPU
CPU - O
CPU - I
ECS
SCANNER AND REQUEST
OK EXCHANGE
REQUEST EXCHANGE CPU - O
REQUEST EXCHANGE CPU - I

Figure 3-16

CENTRAL PROCESSOR READ/WRITE OF CENTRAL MEMORY

A Central Processor Request to Central Memory is initiated for one of two reasons: either the CPU wants a new instruction word (RNI), or it wants to reference memory to store or read an operand. However, CMC knows only that the CPU is requesting memory and that it is a Read or a Write.

When the Central Processor sends a Request and an address, the address is received in CR0 or CR1 depending on which processor is requesting. The action of the controls is much the same for CPU-0/CPU-1 Request as it was for PPU Request. Figure 3-17 shows the path of the address during a CPU Request. When the Accept is returned to the CMC, it is retransmitted to the requesting processor so that it can either send the write word to the Data Distributor or accept the read word from the distributor.

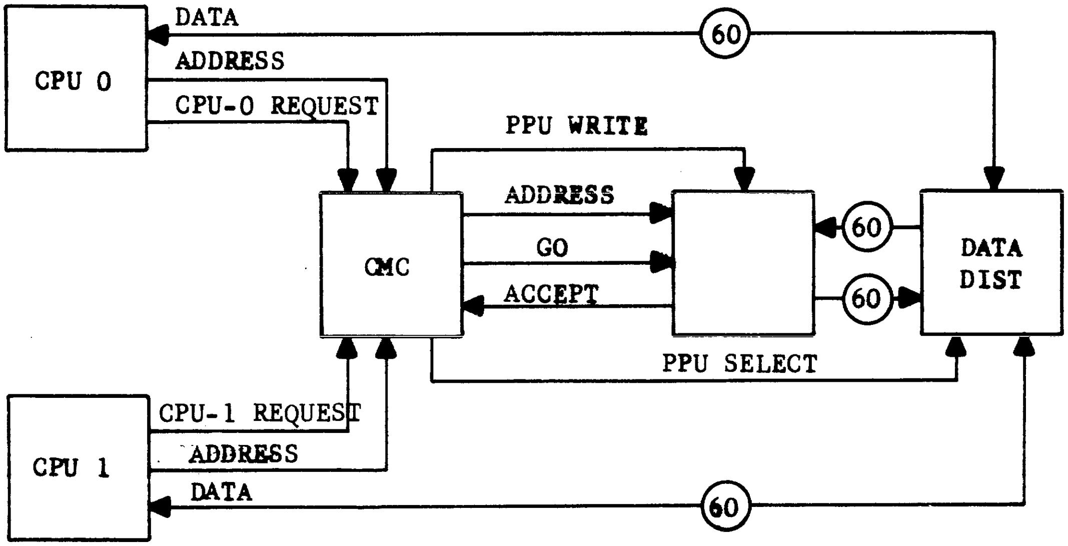


Figure 3-17

6400 EXCHANGE JUMPS

INTRODUCTION

The purpose of the Exchange Jump is to establish conditions for starting a new program in the CPU. In order to cause this change in program, all Operational and Control register contents in the CPU must be changed. The new contents for these registers are stored in 16 consecutive CM addresses, called an Exchange Jump Package. Figure 3-18 shows the contents of this package which is loaded into CM prior to the initiation of the Exchange Jump.

Address n →	P	A0	--
n + 1 →	RA	A1	B1
.	FL	A2	B2
.	EM	A3	B3
. *	RA ECS	A4	B4
. *	FL ECS	A5	B5
. *	MA	A6	B6
.		A7	B7
.	X0		
.	X1		
.	X2		
.	X3		
.	X4		
..	X5		
.	X6		
n + 15 →	X7		

Figure 3-18. Exchange Package

The 6400 has two types of Exchange Jumps. The Regular Exchange Jumps and the Monitor Exchange Jumps. The Monitor Exchange Jumps are available only as an option.

The Regular Exchange Jumps have only two variations. The PPU exchanges CPU-0 via a 2600 instruction, or PPu exchanges CPU-1 via a 2601 instruction. In both instructions, address "n" (Figure 3-18) is from the PPU A register, and the Exchange Jumps are unconditional.

The Monitor Exchange Jumps have several variations, all of which are conditioned by a "Flag bit" located in CMC. Address "n" comes from the CPU MA register. (See the 6400/6600 ECS Training Manual for further details.)

* RAecs, FLecs, and MA values are only for ECS options.

Figure 3-19 illustrates the connections between PPU, CPU, and CM. The numbers indicate the order of operation.

Operations #6 through #10 are cycled 16 times (starting #6 every 100 ns), before #11 occurs. The operations of all Exchange Jumps are identical after #3. Before #3, the various Requests are being conditioned, etc., for each particular operation.

PPU REGULAR EXCHANGE JUMP (see Figure 3-19)

$\boxed{1}$ The 2600 instruction sends to CMC a "0" CPU # bit (indicating CPU-0), a "0" Monitor Bit (indicating not a Monitor Exchange Jump), the PPU Exchange Request, and the address "n" (from its A register).

$\boxed{2}$ CMC checks to see if there are any other exchanges in programs--so, the request would wait; if not, an Exchange Request will be sent to CPU-0.

$\boxed{3}$ If it is stopped, the CPU-0 will honor the request immediately; if running, the request is honored at the end of a 60-bit instruction w word. An OK Exchange signal is sent back to CMC, and the CPU enables the register contents for address "n" to the Data Transmitters. The contents are not sent to the Write Distributor until later. (All exchanges become identical from here on.)

NOTE

See ECS volume for ECS transfer, Exchange Jump Conflicts, and steps $\boxed{1}$ thru $\boxed{3}$ of Monitor Exchange Jumps.

$\boxed{4}$ The OK Exchange signal enables requesting of the Scanner in CMC. If the Scanner is honoring other requests, the exchange will wait. (The special purpose CPU-1 could be requesting a CM read or CM Write.) Scanner hangs up for the duration of an Exchange Jump.

$\boxed{5}$ CMC now waits until CMB's are quiet (all Banks not busy). This allows an Exchange Jump to run at the maximum CMB speed.

$\boxed{6}$ Address "n" and a GO sent to CMB.

$\boxed{7}$ Accept returns from CMB.

$\boxed{8}$ Accept retransmitted from CMC to CPU-0 and steps the Exchange Jump Sequence in CPU-0.

9 The Exchange Jump Sequence in CPU-0 gates the old contents of registers to the Write Distributor. At this time, CMB is sending the new data to the Read Distributor.

10 Write Distributor retransmits old register data to CMB. Read Distributor retransmits new data to CPU-0 where the data are gated into the registers from which came the old data.

NOTE

> The operations #6 through #10 occur 16 times for the 16 addresses in the Exchange Jump Package.

11 When the last operation is near completion, the Scanner is released, control flip-flops are cleared, and an Exchange Resume is sent to the PPU. Other Requests which may have waited in CMC now are honored.

12 CPU-0 gates in the last new data, then starts the new program.

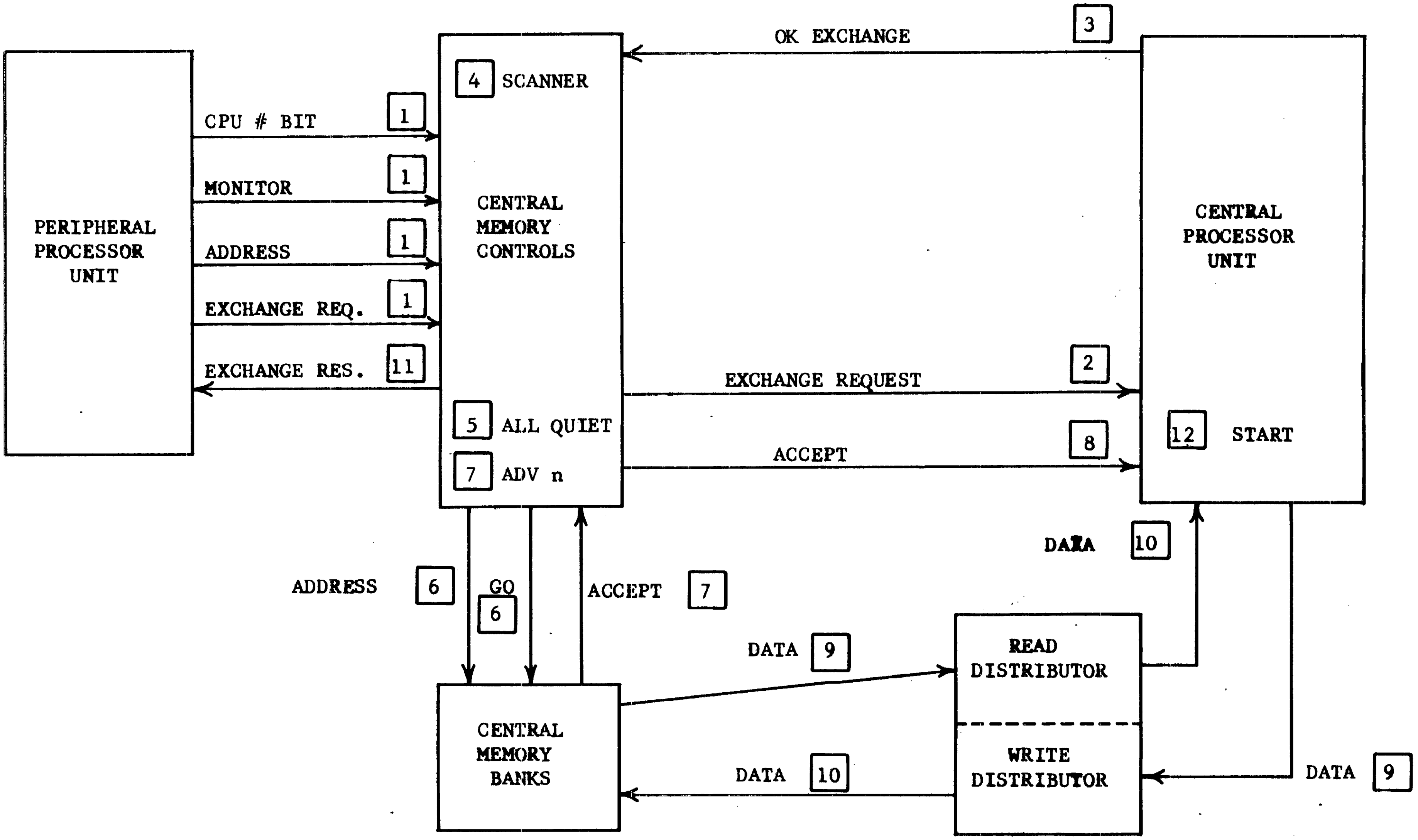


Figure 3-19 PPU EXCHANGE JUMPS

DETAILED CMC OPERATION (See Figure 3-20)

In CMC, the initial Exchange Jump operations are handled by two Regular Exchange Jump (RXJ) flip-flops. RXJ-1 receives the Exchange Jump Request signal from the catching flip-flop (PPU Exchange) and is responsible for setting RXJ-2. Setting RXJ-2 provides a lockout for additional exchange jumps through the Exchange in Progress flip-flop and generates the Request Exchange signal to the designated CPU. Setting of the Block CR2 to PAR flip-flop provides a lockout for incoming PPU addresses by blocking a CR2 to PAR transfer. Refer to Figure 3-21 for address paths. The OK Exchange signal returned by the CPU sets an OK Exchange flip-flop. The output of RXJ-2 and OK Exchange are ANDed to allow the Exchange Jump to be completed by stopping the Scanner and setting PPU Control-1 and PPU Control-2. Although PPU Control-1 was used to give a PAR to MAR on PPU Read and PPU Write, it is now used for a PAR to AIR. PPU Control-2 sets the PPU Start Exchange flip-flop. Since the First Word Address for the Exchange Jump is in AIR, the Block CR2 to PAR flip-flop is now cleared. This, along with the previous clearing of RXJ-1, allows another PPU to store an address in PAR while this Exchange Jump is in progress. The Start Exchange flip-flop disables PAR to AIR and enables the Address Incrementer output to AIR (Figure 3-20). The transfer from AI to the AI Register is a minor cycle timing pulse that is unconditional and the transfer from the AI register to AI is constant. The first word address can circulate through the counter without being incremented until conditions are met for the transfer. The remaining condition to be met is Central Memory All Quiet (all banks not busy).

CMC must complete all references in progress before it can begin to process an Exchange Jump. When CMB is quiet, the Exchange Started flip-flop sets, taking care of the actual transfer by allowing an AIR to MAR transfer and conditioning the advance AIR count and the advance Exchange Counter (EK). The EKR counts the number of outputs so that the jump can be terminated when 16 words have been sent. Once the Exchange Started flip-flop has been set, transfers will occur at a minor cycle rate, with the exception of a delay for a bank conflict on a 32K machine. Every Accept returned by CMB is channelled back to the proper CPU by CMC. The End Exchange output from EK clears Exchange Started, Exchange in Progress, and OK Exchange flip-flop as well as generating an Exchange Resume to the PPU. An Exchange Resume flip-flop clears RXJ-2, PPU Control (which clears PPU Control-2), and PPU Start Exchange flip-flops leaving CMC in its original condition.

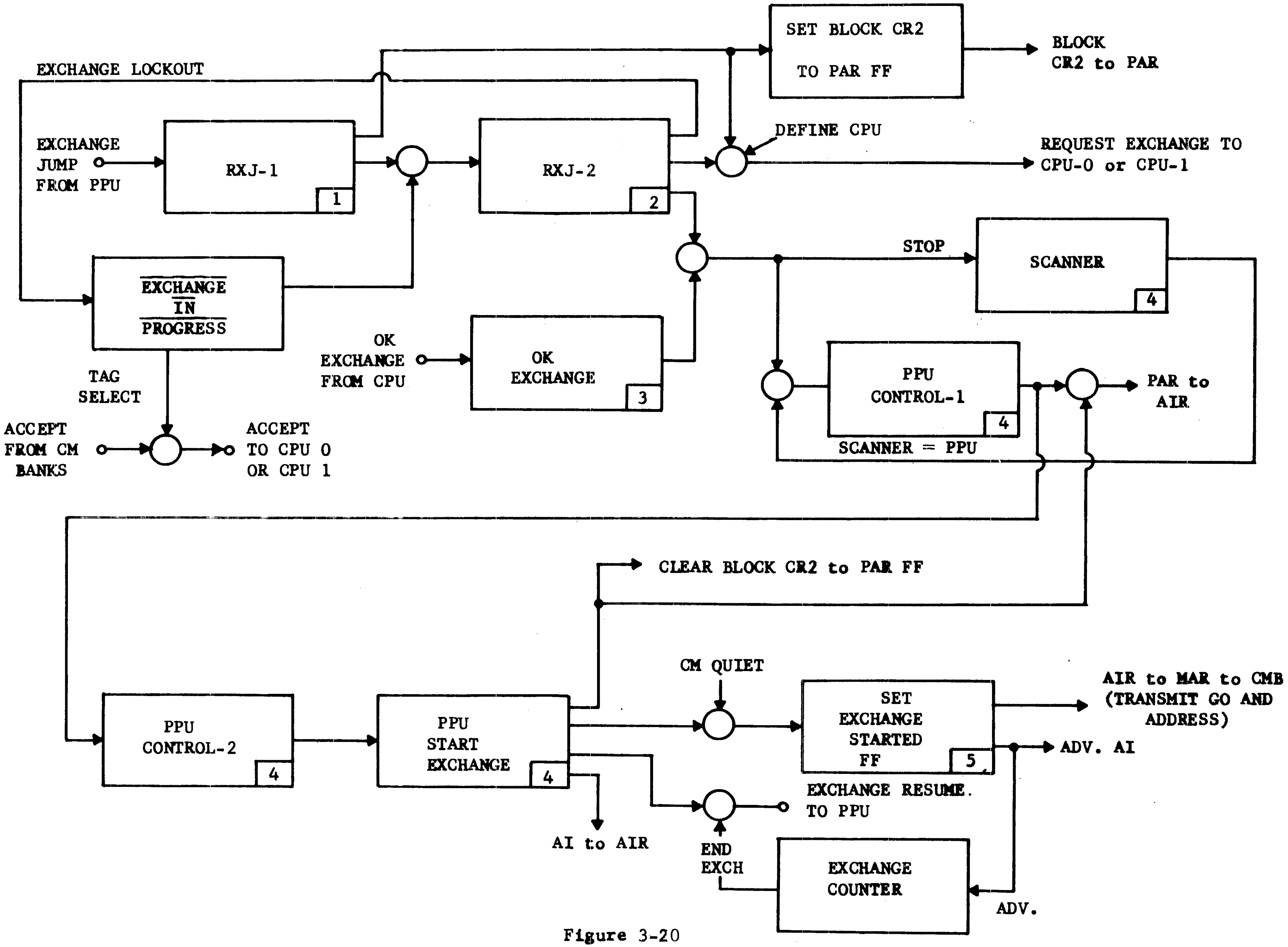

SET BLOCK CR2 TO PAR FF
BLOCK CR2 to PAR
EXCHANGE LOCKOUT
DEFINE CPU
EXCHANGE JUMP FROM PPU
RXJ-1
1
RXJ-2
2
REQUEST EXCHANGE TO CPU-0 or CPU-1
STOP
SCANNER
4
EXCHANGE IN PROGRESS
OK EXCHANGE FROM CPU
OK EXCHANGE
3
PPU CONTROL-1
4
PAR to AIR
TAG SELECT
SCANNER = PPU
ACCEPT FROM CM BANKS
ACCEPT TO CPU 0 OR CPU 1
CLEAR BLOCK CR2 to PAR FF
CM QUIET
AIR to MAR to CMB (TRANSMIT GO AND ADDRESS)
PPU CONTROL-2
4
PPU START EXCHANGE
4
SET EXCHANGE STARTED FF
5
ADV. AI
EXCHANGE RESUME. TO PPU
AI to AIR
END EXCH
EXCHANGE COUNTER
ADV.

Figure 3-20

PPU Regular Exchange Jump Address Path

Operation	ENABLE
CR2 to PAR	Block CR2 to PAR
PAR to AIR	PPU Control-1
AIR to COUNTER	Unconditional
AIR to MAR	Exchange Started
AI to AIR	Start Exchange

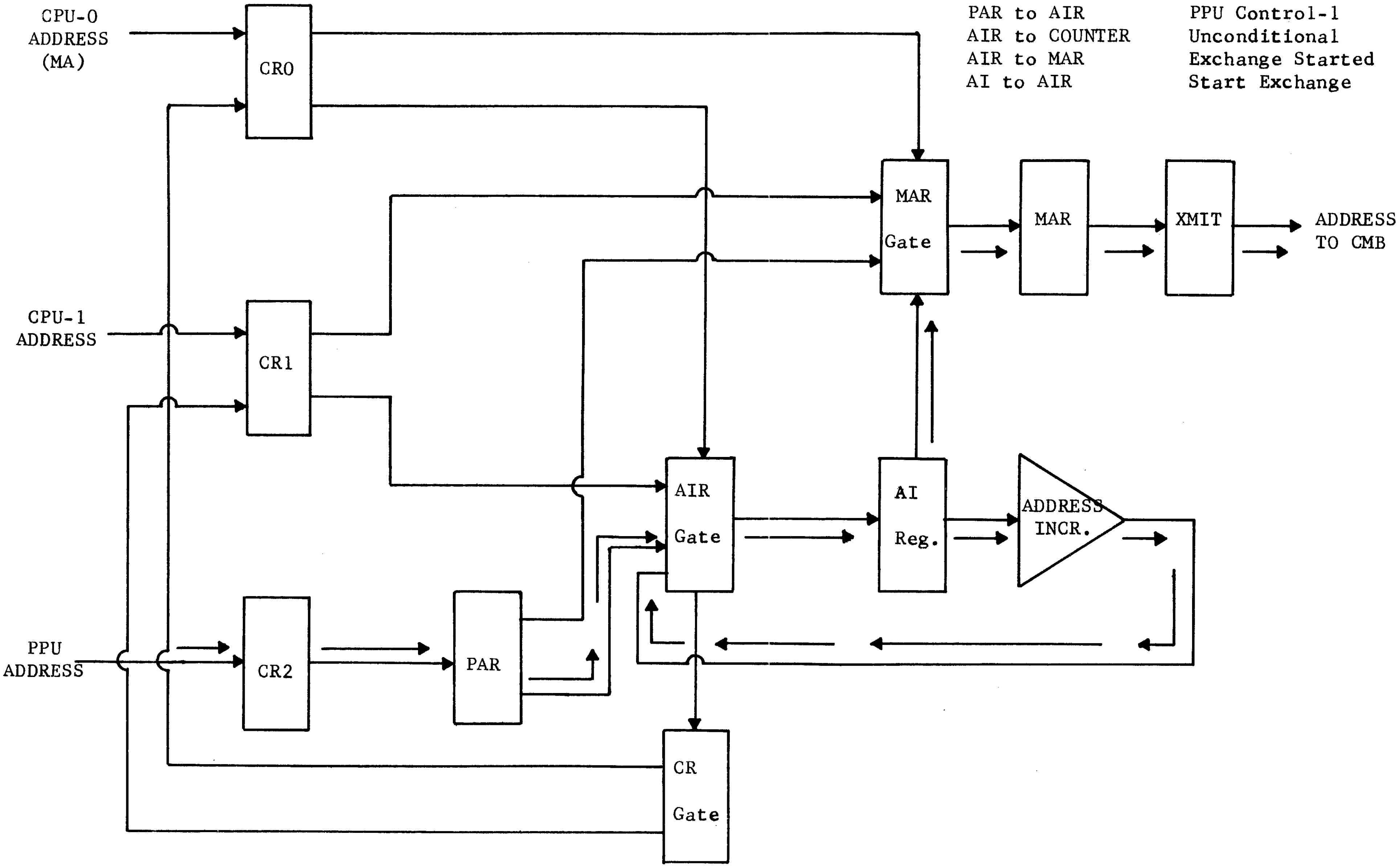


Figure 3-21 CENTRAL MEMORY ADDRESS

CHAPTER IV

CENTRAL PROCESSOR

CHAPTER IV

CENTRAL PROCESSOR

The 6400 Central Processor is a very high speed arithmetic device. It is capable of performing many arithmetic and logical operations and can communicate with the Central Memory and the Peripheral Processors.

UNIFIED CENTRAL PROCESSOR

Instructions are acquired from Central Memory, decoded, and executed in a serial manner. Since instructions are executed one after the other, rather than in a parallel fashion using functional units, the Central Processor is considered to be a unified arithmetic device.

Operating Registers

In order to facilitate the execution of each program step and to reduce memory access times, operands used during the execution of an instruction will come from 24 flip-flop operating registers that will be loaded and stored under program control.

The Central Processor's 24 operating registers are divided as follows:

Eight 60-bit X registers that will hold operands used for computation. The X registers have a direct access path from Central Memory.

Eight 18-bit A registers. These registers will be used for loading and storing operands in the X registers. These registers are known as the address registers.

Changing an A register will cause a memory reference for an associated X register; that is to say, changing A1-A5 will cause a memory read and X1-X5 will receive the operand. Changing A6 or A7 will cause a memory write and the operand in X6 and X7 will be stored. A0 and X0 are not considered in this memory scheme.

Eight 18-bit B registers will be used as index registers.

All of the previously-mentioned registers are unique to the Central Processor and are not associated with the Peripheral Processors in any way. Figure 4-1 illustrates the interconnections discussed.

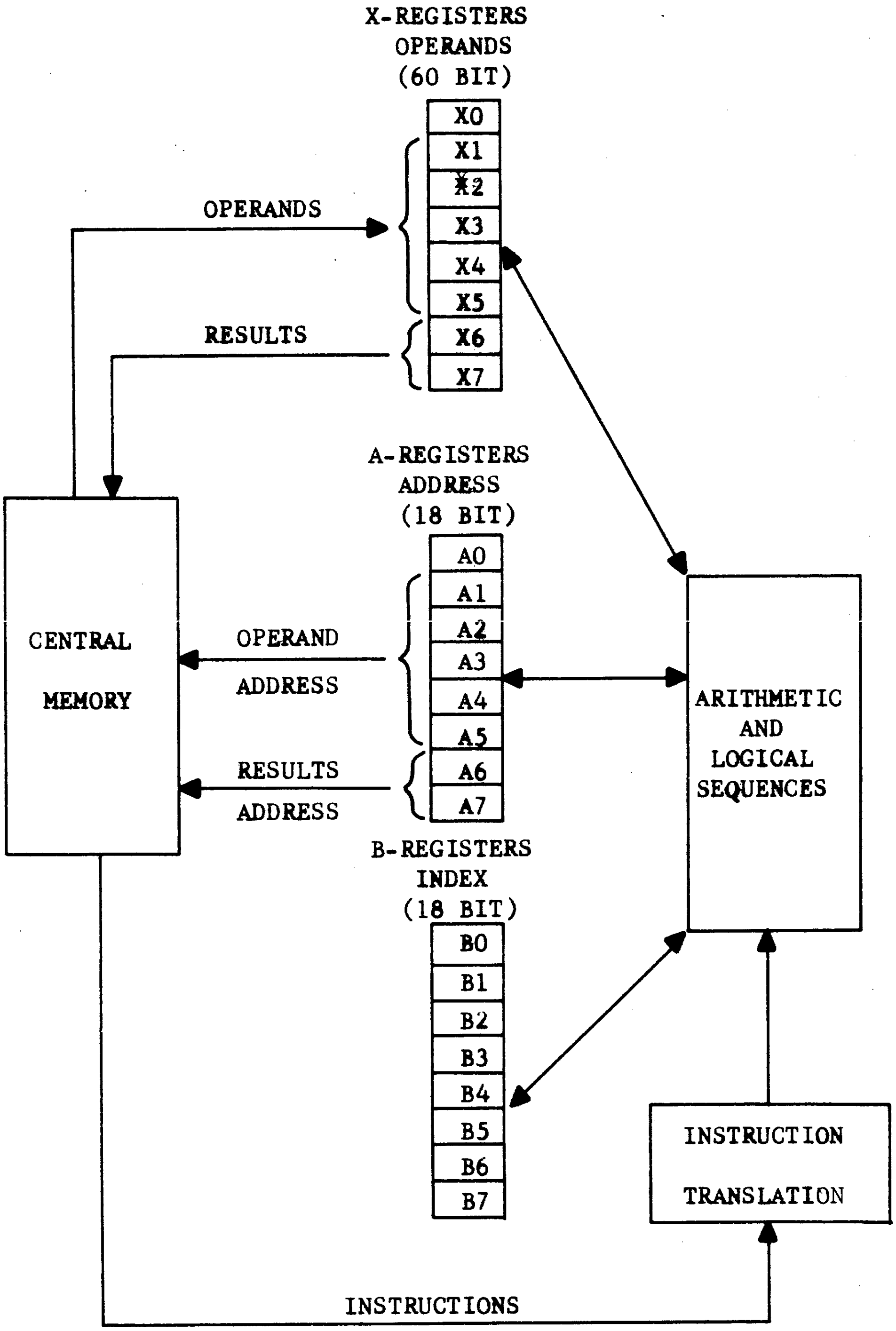


Figure 4-1. Central Processor Block Diagram

INSTRUCTION FORMAT

The basic format of a 6400 instruction is such as to facilitate easy understanding of its operation. Figure 4-2 illustrates the format of the instructions and also shows all combinations of 15- and 30-bit instructions possible in a 60-bit word. In examing the instruction format, it should be noted that the upper 6 bits are the operation code and the next 3 bits form the "i" portion (designating which operating register receives the results of a program step). The "j" and "k" portions designate those operating registers holding source operands used to execute the program step. If an instruction is a 30-bit type, an 18-bit constant "K" is part of the instruction and serves as one operand. In such a case, "j" and "K" serve as the two operands necessary to execute the program step. If it is remembered that "i" is the results register and "j" and "k" (or K) are source operands, code interpretation becomes quite easy; for example, the multiply instruction reads:

40 Floating product of Xj and Xk to Xi

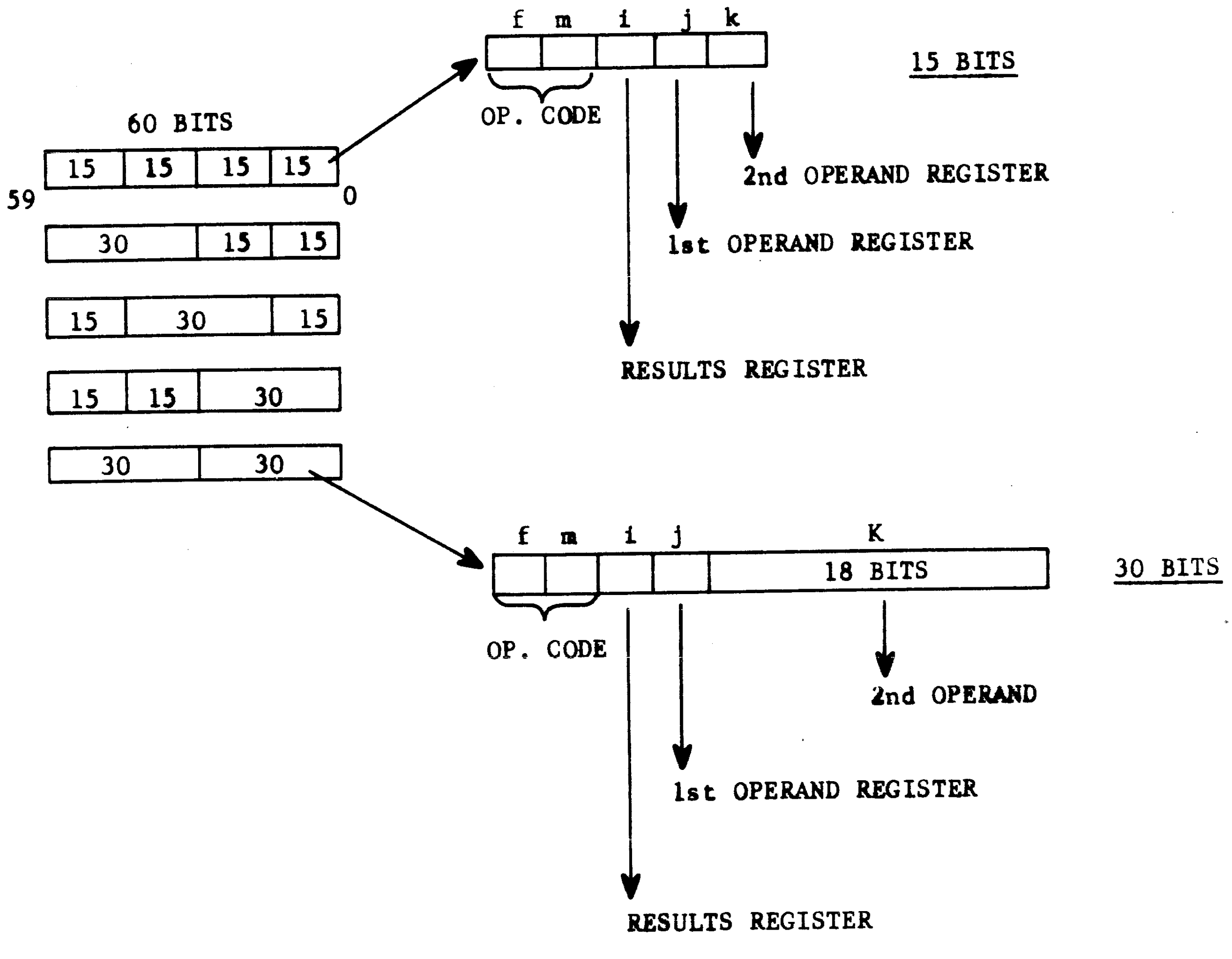


Figure 4-2. Instruction Formats

The 40 is the operation code. The instruction result goes to one of eight X registers. Both source operands come from X registers. In coded form, the instruction could look like:

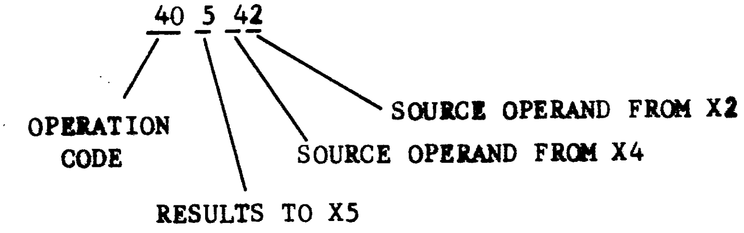


Figure 4-3 illustrates a short program using symbolic form of coding. This program also gives an example of operand manipulation with the operating registers.

OPERATION	RESULT
A1 = B1 + K	Load X1 from memory location B1 + K
A2 = B2 + B0	Load X2 from memory location B2 + B0
X6 = X1 x X2	Multiply X1 and X2, store results in X6
A6 = B5 + K	Store X6 in memory location B5 + K

Figure 4-3

Table 4-1 lists all the central processor instructions.

EXCHANGE JUMP

A peripheral and control processor starts the first Central Processor program and any new program running in the central processor by executing an Exchange Jump instruction. This scheme is necessary since the Central Processor is like an "island" in that there are no logical external connections to any part of the central processor controls that can affect it. Only through programming can the Central Processor be reached. This exchange-jump method of starting programs in the Central Processor is ideal since it allows the interruption of a program currently running in the Central Processor and the recovery of the same program at a later time.

One situation where temporary program interruption is desirable occurs when the program presently residing in the Central Processor has reached an idle period because it requires some input or output process. At this time, an exchange jump would allow a new program to start in the Central

CENTRAL PROCESSOR INSTRUCTIONS AND EXECUTION TIMES
(TIMES LISTED IN MINOR CYCLES)

TABLE 4-1

BRANCH

00	STOP	12
01	RETURN JUMP to K	12
02	GO TO K + Bi	12
030	GO TO K if Xj = zero	12
031	GO TO K if Xj ≠ zero	12
032	GO TO K if Xj = positive	12
033	GO TO K if Xj = negative	12
034	GO TO K if Xj is in range	12
035	GO TO K if Xj is out of range	12
036	GO TO K if Xj is definite	12
037	GO TO K if Xj is indefinite	12
04	GO TO K if Bi = Bj	12
05	GO TO K if Bi ≠ Bj	12
06	GO TO K if Bi ≥ Bj	12
07	GO TO K if Bi < Bj	12

BOOLEAN

10	TRANSMIT Xj to Xi	4
11	LOGICAL PRODUCT of Xj and Xk to Xi	4
12	LOGICAL SUM of Xj and Xk to Xi	4
13	LOGICAL DIFFERENCE of Xj and Xk to Xi	4
14	TRANSMIT Xk COMP. to Xi	4
15	LOGICAL PRODUCT of Xj and Xk COMP. to Xi	4
16	LOGICAL SUM of Xj and Xk COMP. to Xi	4
17	LOGICAL DIFFERENCE of Xj and Xk COMP. to Xi	4

SHIFT

20	SHIFT Xi LEFT jk places	5
21	SHIFT Xi RIGHT jk places	5
22	SHIFT Xk NOMINALLY LEFT Bj places to Xi	5
23	SHIFT Xk NOMINALLY RIGHT Bj places to Xi	5
24	NORMALIZE Xk in Xi and Bj	6
25	ROUND AND NORMALIZE Xk in Xi and Bj	6
26	UNPACK Xk to Xi and Bj	6
27	PACK Xi from Xk and Bj	6
43	FORM jk MASK in Xi	5

ADD

30	FLOATING SUM of Xj and Xk to Xi	11
31	FLOATING DIFFERENCE of Xj and Xk to Xi	11
32	FLOATING DP SUM of Xj and Xk to Xi	11
33	FLOATING DP DIFFERENCE of Xj and Xk to Xi	11
34	ROUND FLOATING SUM of Xj and Xk to Xi	11
35	ROUND FLOATING DIFFERENCE of Xj and Xk to Xi	11

LONG ADD

36	INTEGER SUM of Xj and Xk to Xi	6
37	INTEGER DIFFERENCE of Xj and Xk to Xi	6

MULTIPLY

40	FLOATING PRODUCT of Xj and Xk to Xi	56
41	ROUND FLOATING PRODUCT of Xj and Xk to Xi	56
42	FLOATING DP PRODUCT of Xj and Xk to Xi	56

DIVIDE

44	FLOATING DIVIDE Xj by Xk to Xi	56
45	ROUND FLOATING DIVIDE Xj by Xk to Xi	56
47	SUM of 1's in Xk to Xi	68

INCREMENT

50	SUM of Aj and K to Ai	5
51	SUM of Bj and K to Ai	5
52	SUM of Xj and K to Ai	5
53	SUM of Xj and Bk to Ai	5
54	SUM of Aj and Bk to Ai	5
55	DIFFERENCE of Aj and Bk to Ai	5
56	SUM of Bj and Bk to Ai	5
57	DIFFERENCE of Bj and Bk to Ai	5
60	SUM of Aj and K to Bi	5
61	SUM of Bj and K to Bi	5
62	SUM of Xj and K to Bi	5
63	SUM of Xj and Bk to Bi	5
64	SUM of Aj and Bk to Bi	5
65	DIFFERENCE of Aj and Bk to Bi	5
66	SUM of Bj and Bk to Bi	5
67	DIFFERENCE of Bj and Bk to Bi	5
70	SUM of Aj and K to Xi	5
71	SUM of Bj and K to Xi	5
72	SUM of Xj and K to Xi	5
73	SUM of Xj and Bk to Xi	5
74	SUM of Aj and Bk to Xi	5
75	DIFFERENCE of Aj and Bk to Xi	5
76	SUM of Bj and Bk to Xi	5
77	DIFFERENCE of Bj and Bk to Xi	5
46	Pass	

Octal Code at left of instruction

Comp. - Complement

DP - Double Precision

Processor and would also record the necessary information needed to restart the original program when the input or output process has completed.

For a basic understanding of how the exchange jump works, remember that any program running in the Central Processor is under control of the operating and control registers. For example, the Program Address register (P) keeps track of the instruction sequence. The X, B, and A operating registers keep track of operands used during execution of program steps; and the Reference Address register (RA) keeps track of the relocation point of the present program. There are other important registers, of course, but these serve as examples of what is meant.

A program running in the Central Processor can be stopped and restarted without difficulty if a record can be kept of the control and operating register contents at the time of interruption. The new program must supply its own values to the control and operating registers used during the program. A peripheral processor stores into the central memory new values for the control and operating registers that will be used by the interrupting program. This area of central memory is called the exchange jump package (Figure 4-4). Of course, the instructions for the new program will also be located in the central memory before exchange jump execution (Figure 4-5).

Once the exchange jump package has been stored in central memory, the PPU executes the exchange jump. This causes the operating and control register contents for the present program to exchange with that of the new program. These new values come from the exchange jump package and represent the operating conditions for the new program. Meanwhile, the old values of the operating and control registers are safely stored in the exchange jump package so that another exchange jump will restore these values whenever the origianl program is to continue.

CENTRAL PROCESSOR OVERVIEW

Figure 4-6 illustrates the Central Processor in a simplified manner. An explanation of each major part will aid in the understanding of Central Processor operations.

Input

All inputs to the Central Processor pass through this area, including instructions, operands, and control register contents.

Instruction Controls

This area includes all instruction translation logic and the control sequences logic necessary to carry out the execution.

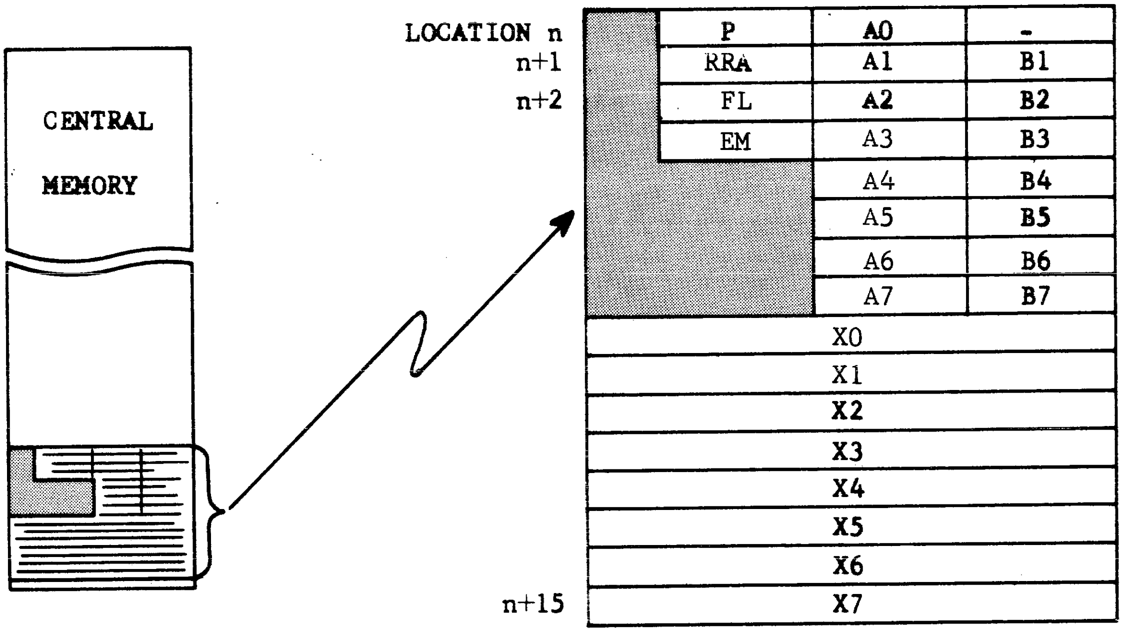


Figure 4-4. Exchange Jump Package

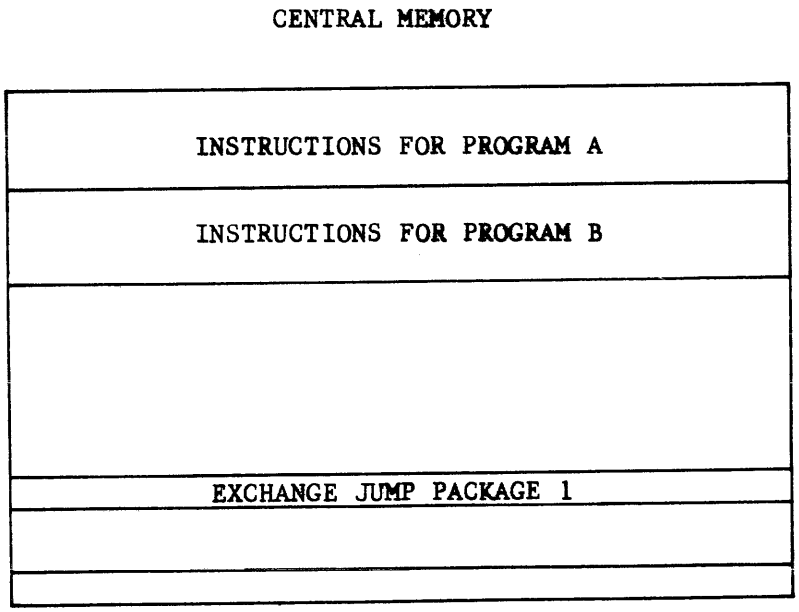


Figure 4-5. Program Distribution in Central Memory

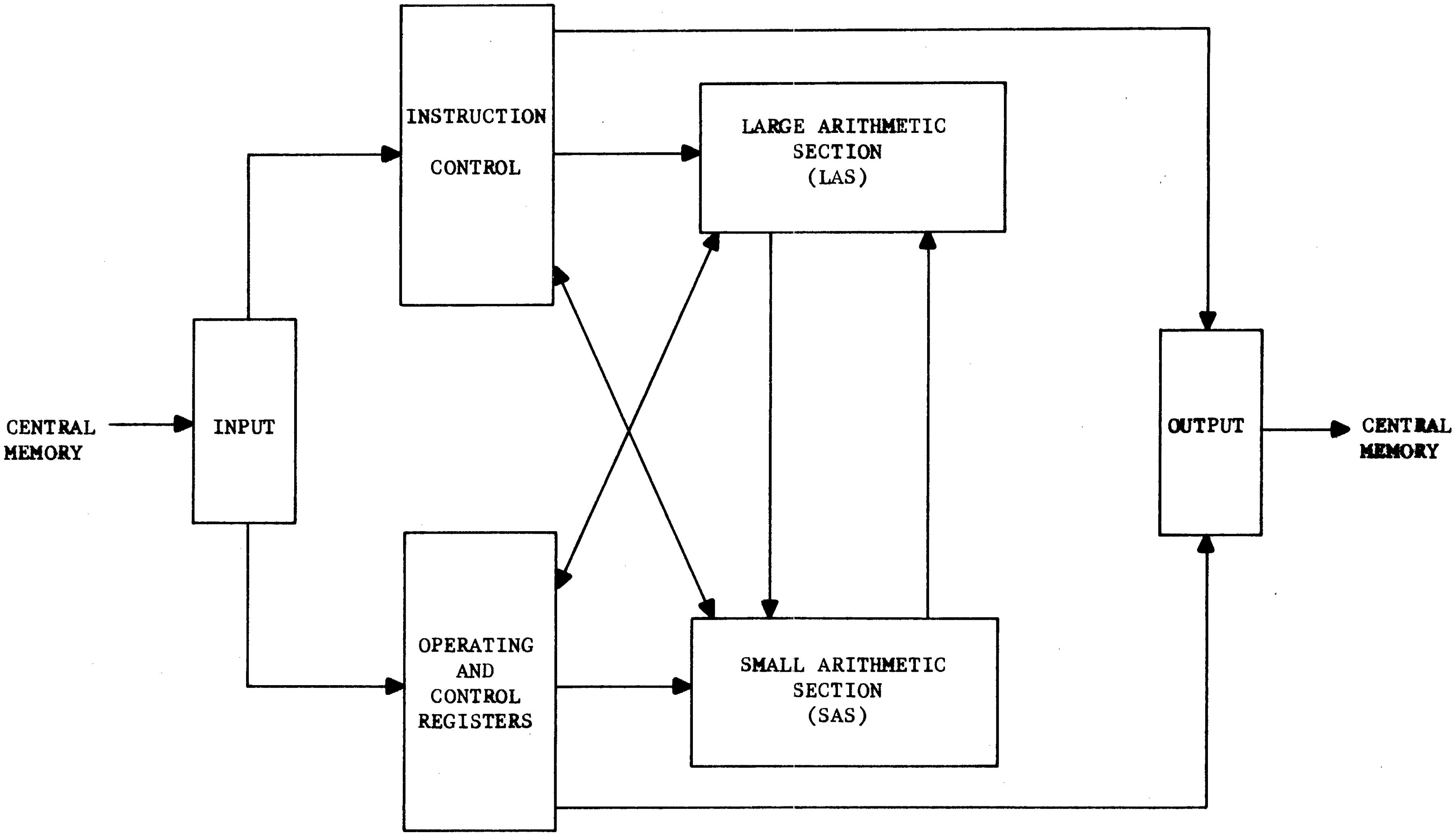

INSTRUCTION CONTROL
LARGE ARITHMETIC SECTION (LAS)
CENTRAL MEMORY
INPUT
OUTPUT
CENTRAL MEMORY
OPERATING AND CONTROL REGISTERS
SMALL ARITHMETIC SECTION (SAS)

Figure 4-6. CUP Simplified Block Diagram

Operating and Control Registers

The control registers include the Program Address register (P), the Reference Address register (RA), the Exit Mode register (EM), the Field Length register (FL) and the RAecs, FLecs, and MA registers for the ECS option. All of these registers are used in conjunction with the controls during the execution and progression of instructions.

The operating registers hold the operands used during the execution of instructions. There are 24 operating registers divided into three groups of eight each: the address registers (A), 18 bits in size; the index registers (B), 18 bits in size; and the operand registers (X), 60 bits in size. These registers will be specified by the i, j, and k designators of each instruction and will serve as the source and destination of operands for each executed instruction. This eliminates the need for a memory access to execute an instruction, which greatly increases the speed of execution. The initialization of the operating registers is done by either the exchange jump or by execution of an instruction.

The X registers can also be loaded directly from memory and are also the focal points for operands that must be written into memory. This loading and storing of the X registers with memory makes use of the A registers. Five of the X registers, X1 - X5, are designated as being receivers of operands; the associated A registers, A1 - A5, serve as the address for the memory reference. The X6 and X7 registers can only store operands whereas A6 and A7 are the associated address registers. Whenever A1 - A7 changes because of an instruction designating these destination registers, a memory reference is automatically initiated. Whether a load or a store takes place depends upon which A register is changed.

Large Arithmetic Section (LAS)

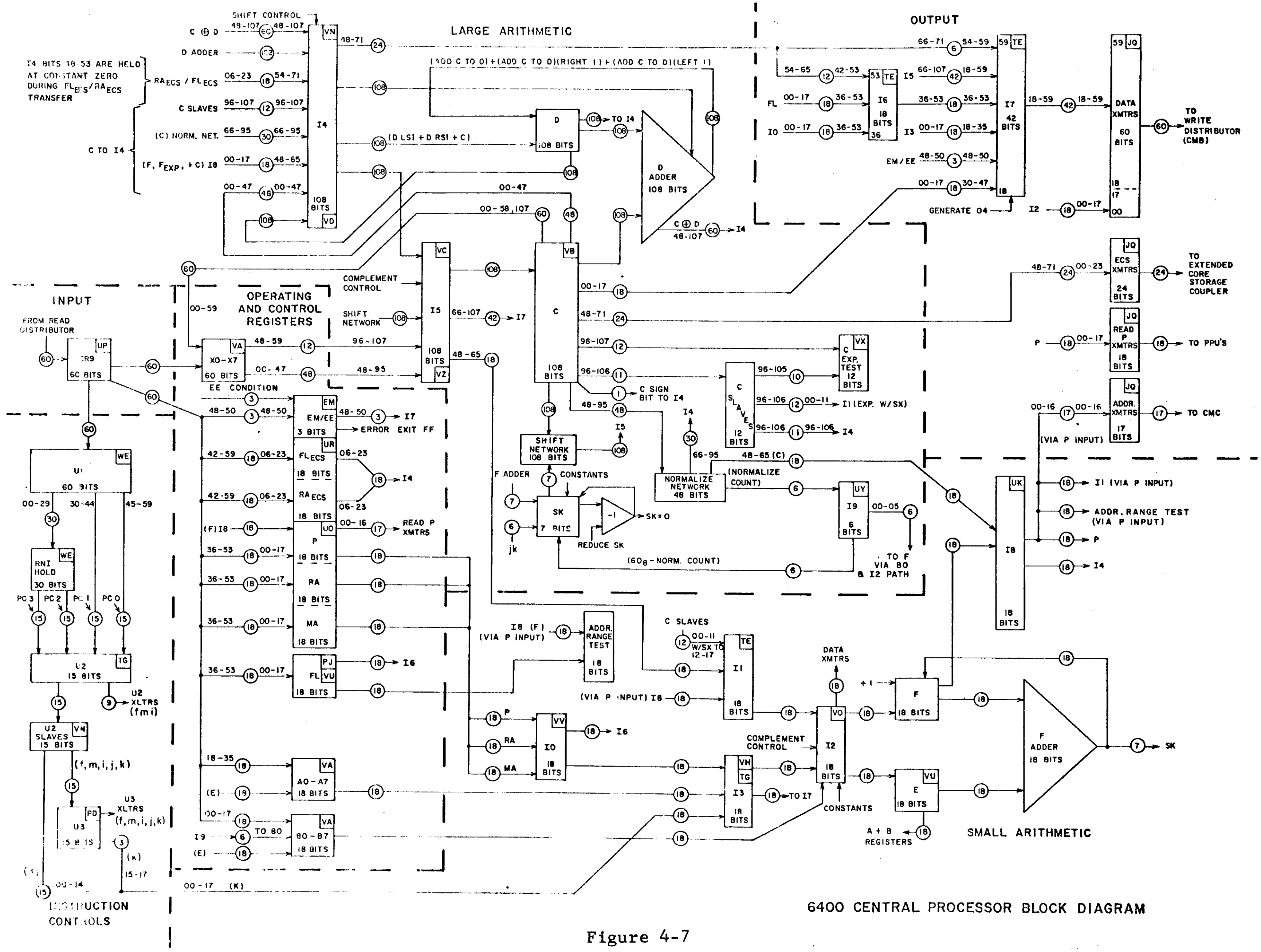
The large arithmetic section is used during the execution of instructions using 60-bit operands, usually of floating point format, and includes all multiplies, divides, logical, add, and shift instructions. The LAS includes two 108-bit registers, an adder, a shift network, a normalize network, a shift and iteration counter, and all the necessary select circuits.

Small Arithmetic Section (SAS)

The small arithmetic section will handle instructions using 18-bit operands, which includes increment instructions, jump instructions, and the exponent manipulation for floating point instructions. SAS includes two 18-bit registers, an adder, an address range tester, and all necessary select circuits.

Output Section

The output section is where all data and control must pass on their way to central memory and ECS. This section includes operands, operating and control registers for exchange jumps, all addresses developed by the P



6400 CENTRAL PROCESSOR BLOCK DIAGRAM

Figure 4-7

register or central addresses for operands, and extended core storage addresses.

CENTRAL PROCESSOR OPERATION

The first program is started in the CPU by the execution of an Exchange Jump from the PPU's. At this time all operating and control registers are initialized and the CPU is started. Starting of the CPU will cause the first RNI (Read Next Instruction). RNI acquires the first set of instructions from CM and causes the decoding and execution of these instructions. The CPU continues to run until a stop is encountered. If the CPU stops by encountering a Stop instruction, it can be restarted only by an Exchange Jump. The CPU also stops during execution of an ECS instruction if an Exchange Request is received from the PPU's. When an Exchange Request is received, the present 60-bit instruction word is completely executed before the CPU stops.

The following section of this manual explains the CPU. Each major area is described along with its operations. Figure 4-7 is a detailed block diagram of the CPU which is referred to throughout this section.

INPUT

All inputs to the CPU must go through Catching Register 9 (CR9). CR9 is 60 bits in size and consists of ungated flip-flops receiving data from coaxial cable inputs. Each flip-flop has a three-way fan-out associated with it (Figure 4-8). Two of the outputs are gated; the third is not. Bits 48, 49, and 50 have an additional output (shown in phantom) for the Exit Mode register. CR9 is unconditionally cleared each minor cycle.

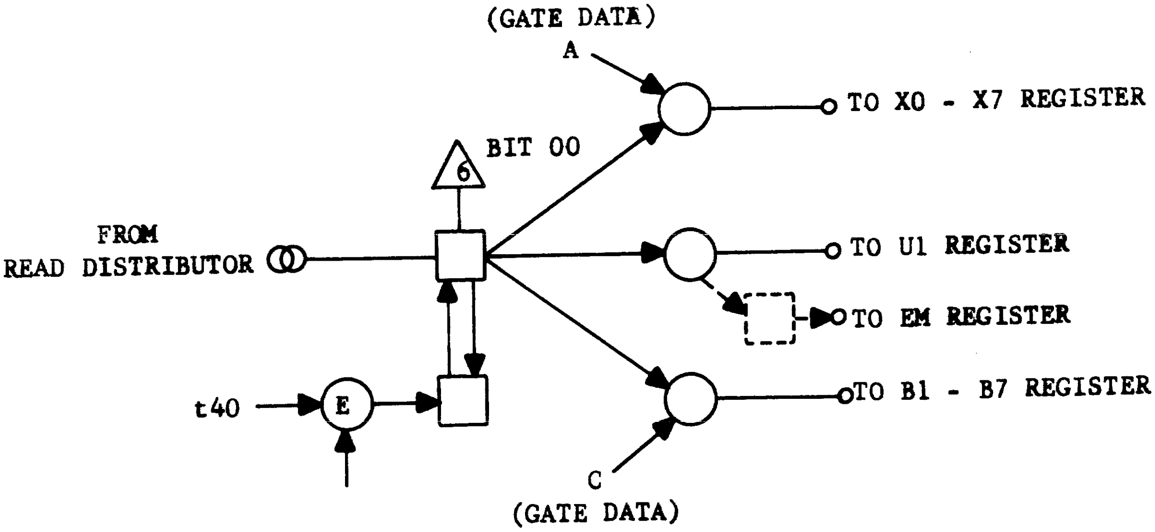
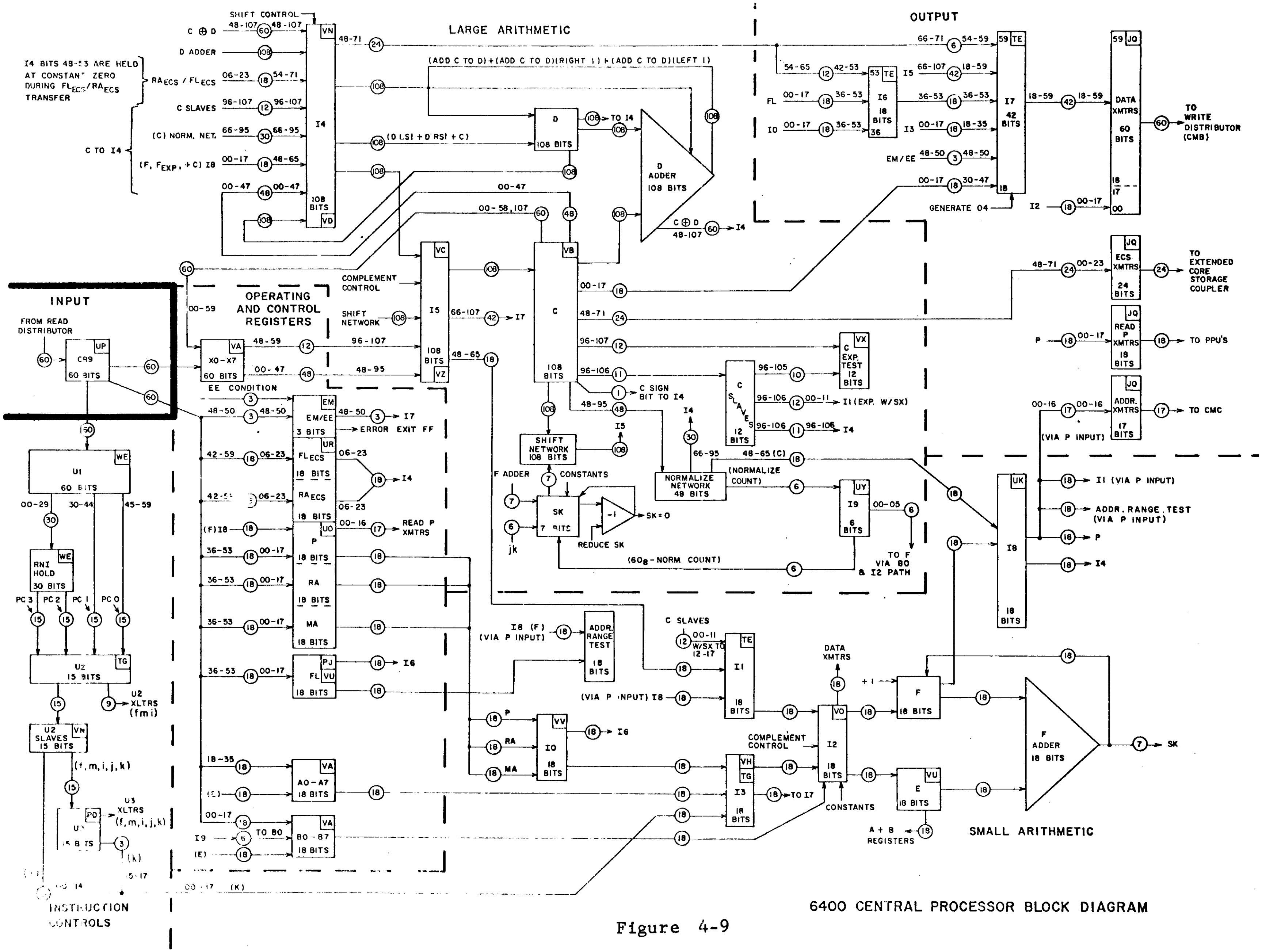


Figure 4-8. Typical Input F.F.



6400 CENTRAL PROCESSOR BLOCK DIAGRAM

Figure 4-9

OPERATING AND CONTROL REGISTERS

The operating and control registers are flip-flop registers used during the execution and control of instructions. Each register, or group of registers, (operating registers) has a definite purpose in the orderly execution and progression of instructions. This topic concerns the manner in which these registers are loaded, how their contents are extracted, and some basic information about their relationship to operations.

CONTROL REGISTERS

The Control registers are concerned with the execution of an instruction, as contrasted with the operating registers which hold addresses, hold index quotations, or hold operands used by the instructions.

P Register (18 bits)

The Program Address register contains the relative memory location of the present instruction word. The 60-bit central memory word can contain up to four instructions. In such circumstances, P will represent the location of all four of these instructions.

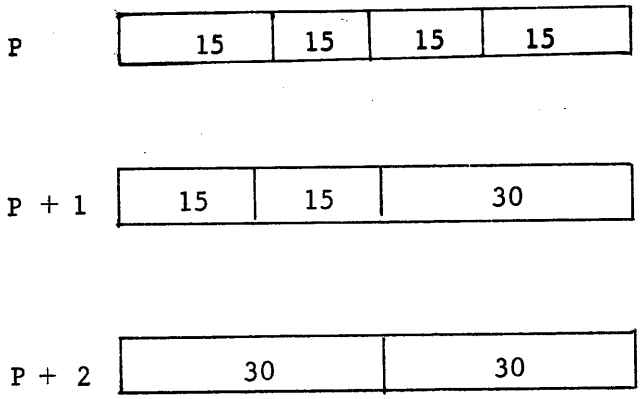


Figure 4-10

P is not the exact central memory location since it will always be added to the reference address before being used to access memory. P is advanced each time a new instruction word is required from memory. The advancing of P is handled by the 18-bit adder in the small arithmetic section. (P) is sent to the adder and the constant, +1, is used as the second operand. The result of the addition (P + 1) is sent back to the P register and, at the same time, is sent to the adder for the addition with RA before being sent to memory.

The P register can be loaded in two separate manners (see Figure 4-12). One method is during an exchange jump, when, the contents of P come from memory through CR9. The second method is from the F register through the I8 inverter group.

The P register flip-flops have two outputs. One output goes to the I0 inverter group used during advancing of P and to store P during an exchange jump. The second output goes directly to the Read P transmitters and then to the peripheral processors every minor cycle. To be able to send the newly-developed contents of P after advancing, to the address transmitters and the FL check circuit, a parallel path of inverters (fed by the same source that feeds the P register) is used.

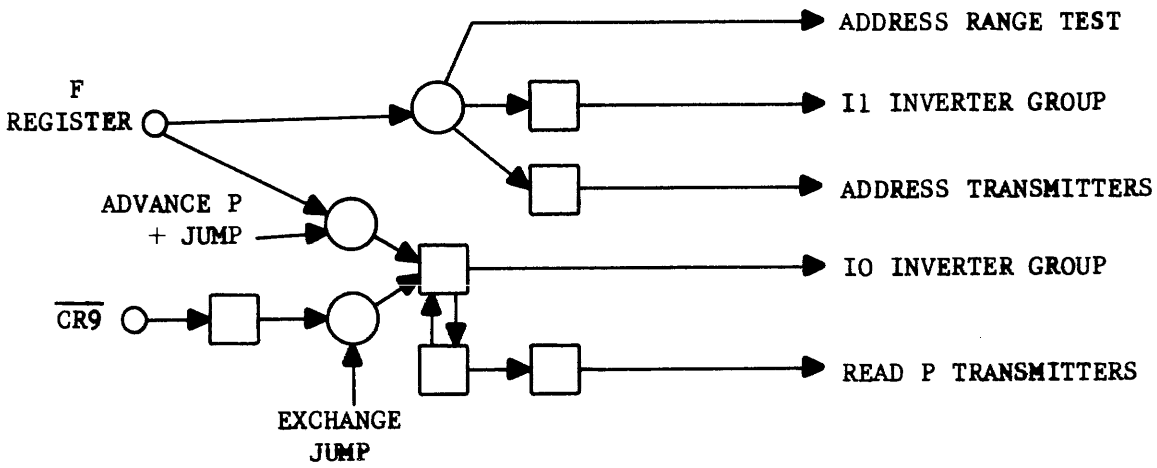


Figure 4-12. Typical P Stage

RA and FL Registers (18 bits each)

RA and FL are loaded during an exchange jump and will remain loaded until the next exchange jump. The RA (reference address) and the FL (field length) registers are used to define the upper and lower limits of any program presently running from CM. No CPU memory access of any kind is allowed outside of these limits. RA is added to every address developed in the CPU before the address is sent to CM. The address could originate from the P register or from an increment instruction, to access memory for an operand. Every address is checked against FL, before RA is added, to determine if the upper limit of the program is exceeded. The RA register feeds only inverter group I0 (see Figure 4-13). The FL register feeds the I6 inverter group and the FL checker (see Figure 4-11).

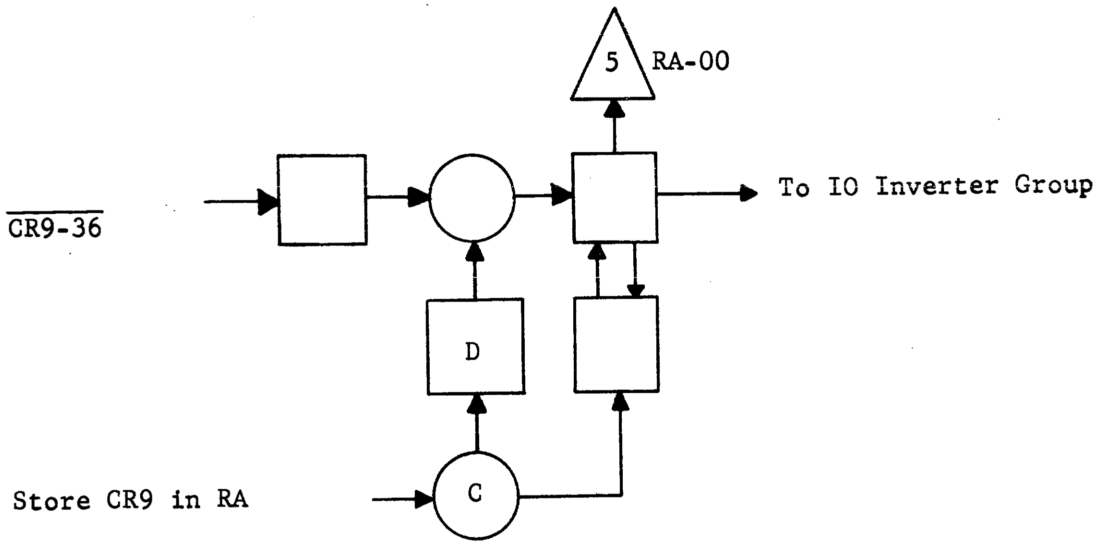


Figure 4-13. Typical RA Stage

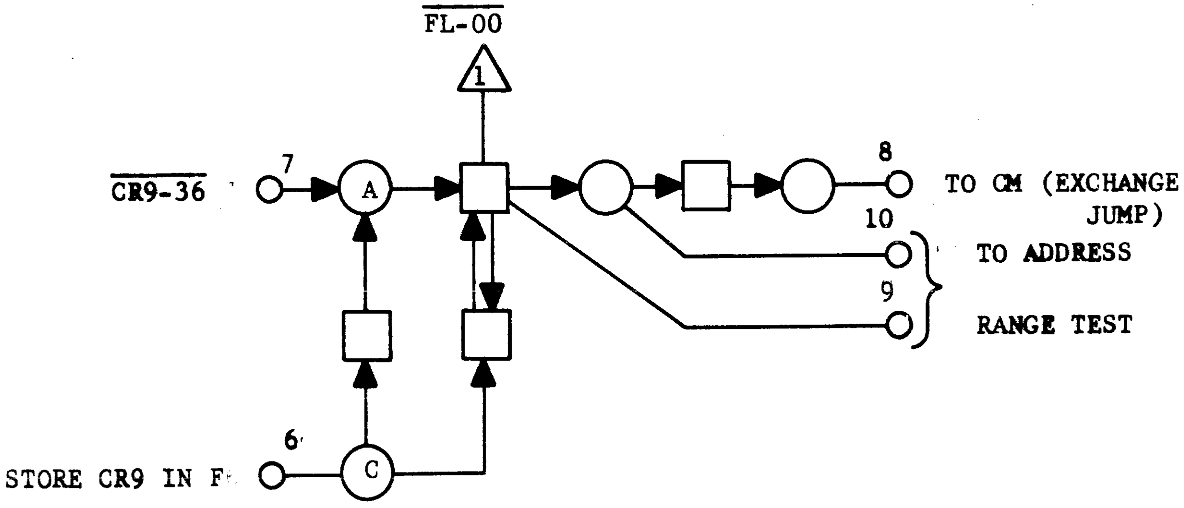


Figure 4-14. Typical FL Stage

EM Register (3 bits)

The Exit Mode register contains a code that will decide whether the Central Processor will stop on certain error conditions.

EM = Exit Mode =	000000	Normal Stop
	010000	Address Out of Range
	020000	Operand Out of Range
	030000	Address or Operand Out of Range
	040000	Indefinite Result
	050000	Indefinite Result or Address Out of Range
	060000	Indefinite Result or Operand Out of Range
	070000	Indifinite Result or Address Out of Range or Operand Out of Range

Table 4-1

The Exit Mode register is loaded during the exchange jump. The EM register is three bits in size but is effectively 18 bits during operations and is illustrated this way in the exchange package (Figure 4-15).

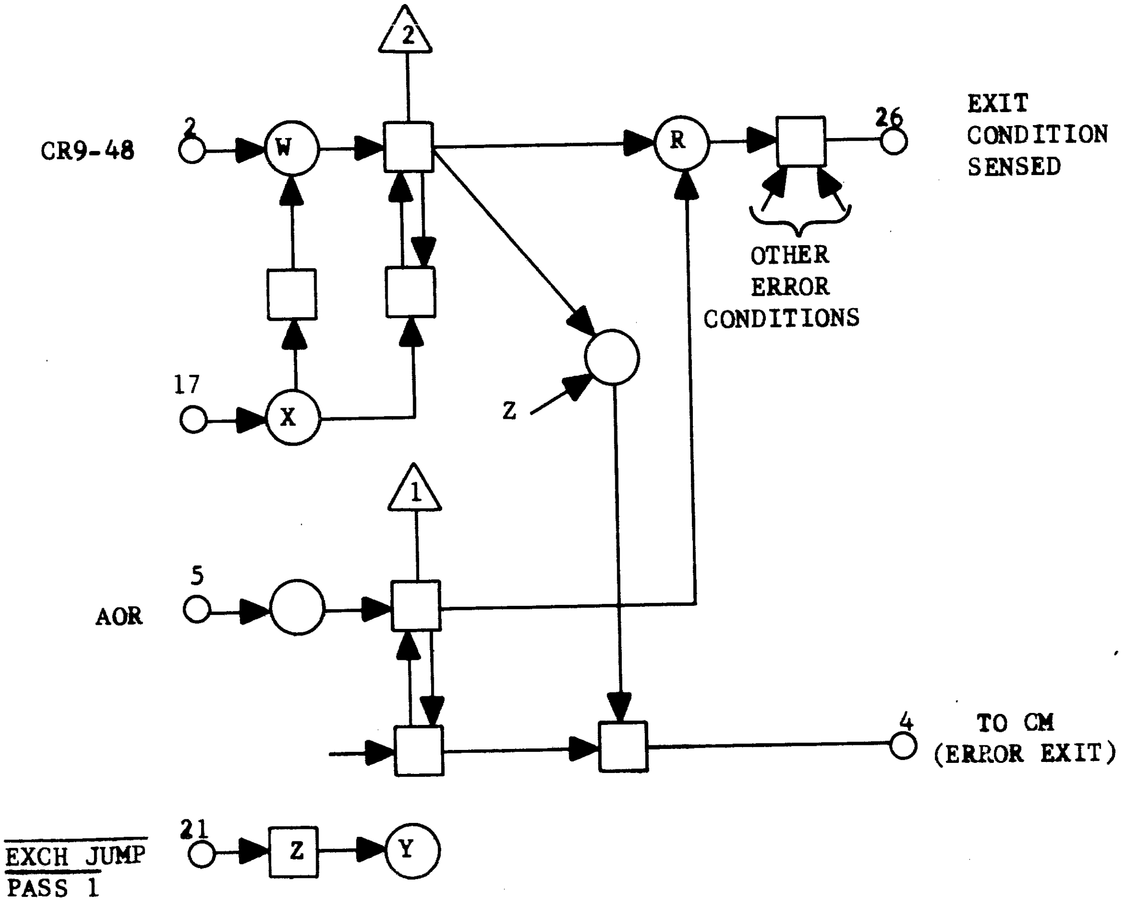


Figure 4-15. Typical EM Stage

The Error Exit flip-flop, shown below the EM state, sets if that type of an error occurs. If both EM-48 and EE-48 go set, the Error Exit flip-flop sets and an Error Exit operation occurs. If the EE-48 flip-flop sets and EM-48 does not set, a slightly different operation occurs (to be explained in detail later). The 17 inverter group output is used to record EM/EE during this operation.

RAecs, FLecs (24 bits each)

The RAecs and FLecs regsiters serve the same purpose for ECS as does RA and FL for CM. Since ECS can be much larger than CM, 24 bits are necessary. These two registers are used only during an ECS operation. Since the small arithmetic section has only an 18-bit adder, the additions and checks are made in the D adder of the large arithmetic section (Figure 4-16 shows a typical RAecs/FLecs stage).

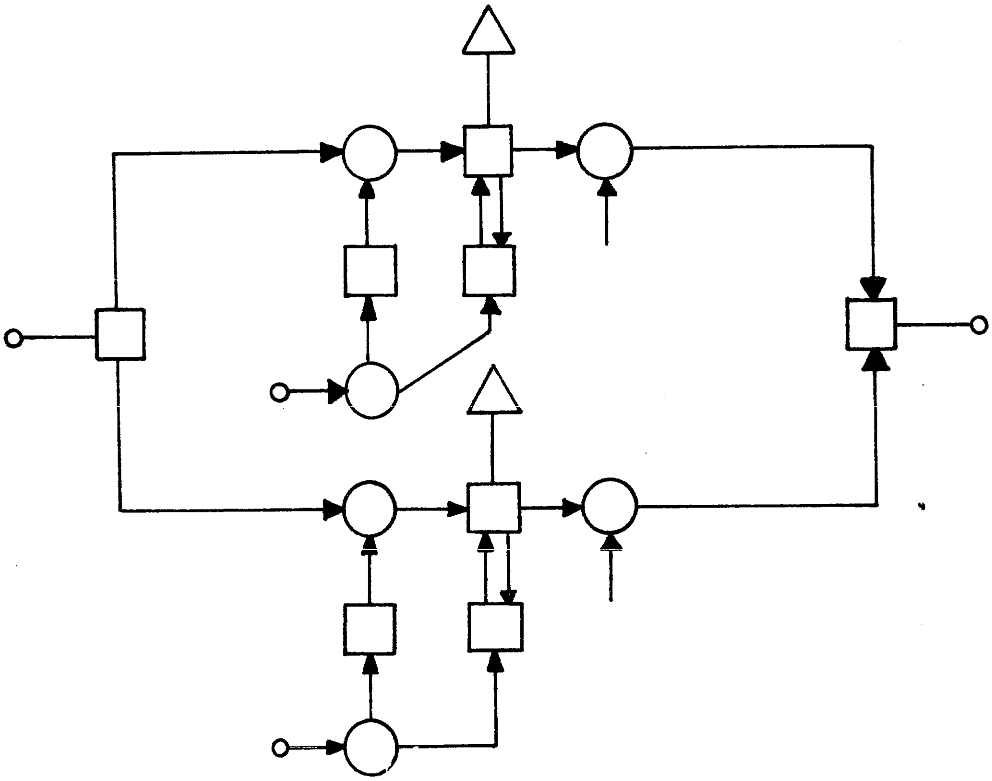


Figure 4-16. Typical RAecs/FLecs Stage

MA Register (18 bits)

The Monitor Address register is loaded only during an exchange jump operation, and is changed only by another exchange jump. It is used mainly with the Operating System program on a 6400 installation using ECS. Figure 4-17 illustrates a typical MA stage.

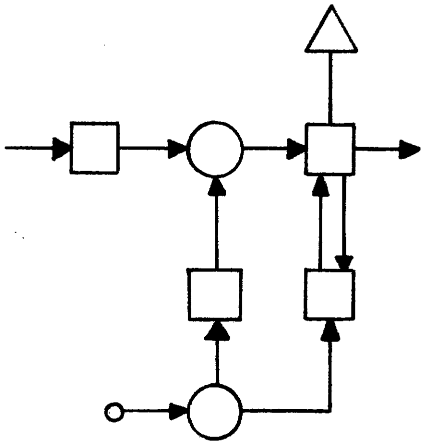


Figure 4-17. Typical MA Stage

OPERATING REGISTERS

The operating registers serve as the source and the destination of operands during execution of instructi ons. There are 24 operating registers divided into three groups:

8 Address Registers (A) - 18 bits each

8 Index Registers (B) - 18 bits each

8 Operand Registers (X) - 60 bits each

All of the operating registers have two inputs. One input is used to load the register during an exchange jump from memory via CR9. The other input is used to store the result of an instruction. Since there are eight registers in each group, a manner of feeding these registers by one path through a select circuit and feeder network is used (see Figure 4-17). There is a separate feeder and select circuit for each operating register group.

Each of the feeder circuits is identical except for size. The A and B feeder are 18 bits; the X feeders are 60 bits. For the X registers, the input to the feeders are from CR9 and from the C register (C-00 to C-58 and C107). A word intended for an X register is sent to the feeders where it is fanned out to the select circuits for X0 - X3. At the same time, the word is passed on to a second select circuit for X4 - X7. To take advantage of all the circuits on the RC modules being used in the feeder network, each pair of bits in the word are alternated in the select process. For instance, bits 00 and 01 are first fanned out for the X0 - X3 selection and then fanned out for the X4 - X7 selection. The next pair of bits, 02 and 03, are first fanned out for the X4 - X7 selection and then fanned out for the X0 - X3 selection. The gating terms "Data to X-" are the result of a clear/set pulse developed when a Store C in Xi command is issued by the controls during instruction execution.

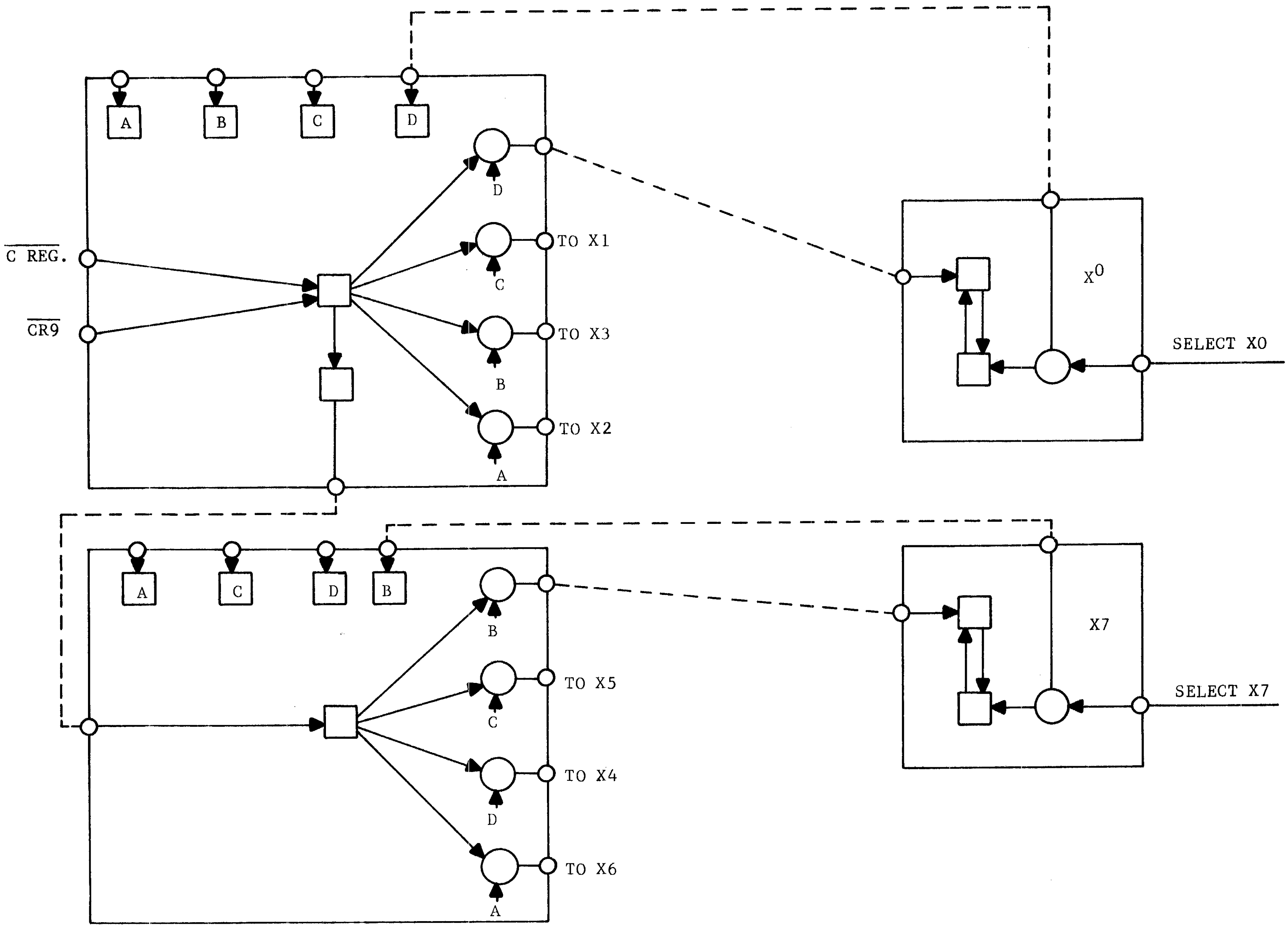


Figure 4-18. X Feeder Network

The A and B registers use the same type of select scheme. For the A and B registers, data come from the E register during instruction execution, from CR9 during an exchange jump.

Selection of data from the X and A operating registers uses a double-select scheme. For example, when the controls issue the Select Xj command, two X registers are initially selected. A second select then eliminates one of the registers, resulting in only one of the two registers being selected (see Figure 4-18). Table 4-2 illustrates this scheme.

PRIMARY SELECT	SECONDARY SELECT	RESULTING SELECT
X0 · X4	X0 - X3	X0
	X4 - X7	X4
X1 · X5	X0 - X3	X1
	X4 - X7	X5
X2 · X6	X0 - X3	X2
	X4 - X7	X6
X3 · X7	X0 - X3	X3
	X4 - X7	X7

Table 4-2

The A registers use the same scheme. The important thing to remember is that one control command, such as Select Xj, produces both the primary and secondary selection. The B registers use a Select 1-of-7 scheme, instead of the 2-of-8, then 1-of-2 scheme used in the X and A registers.

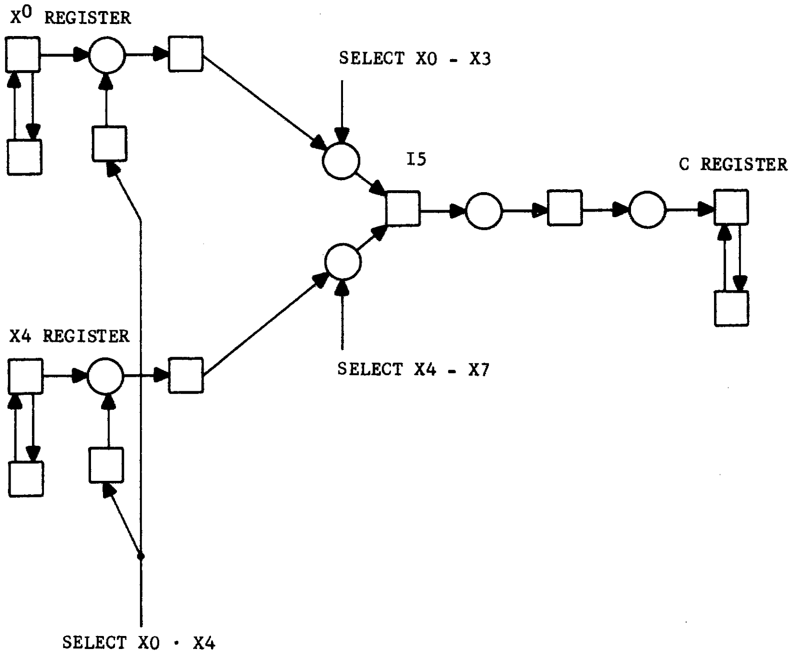
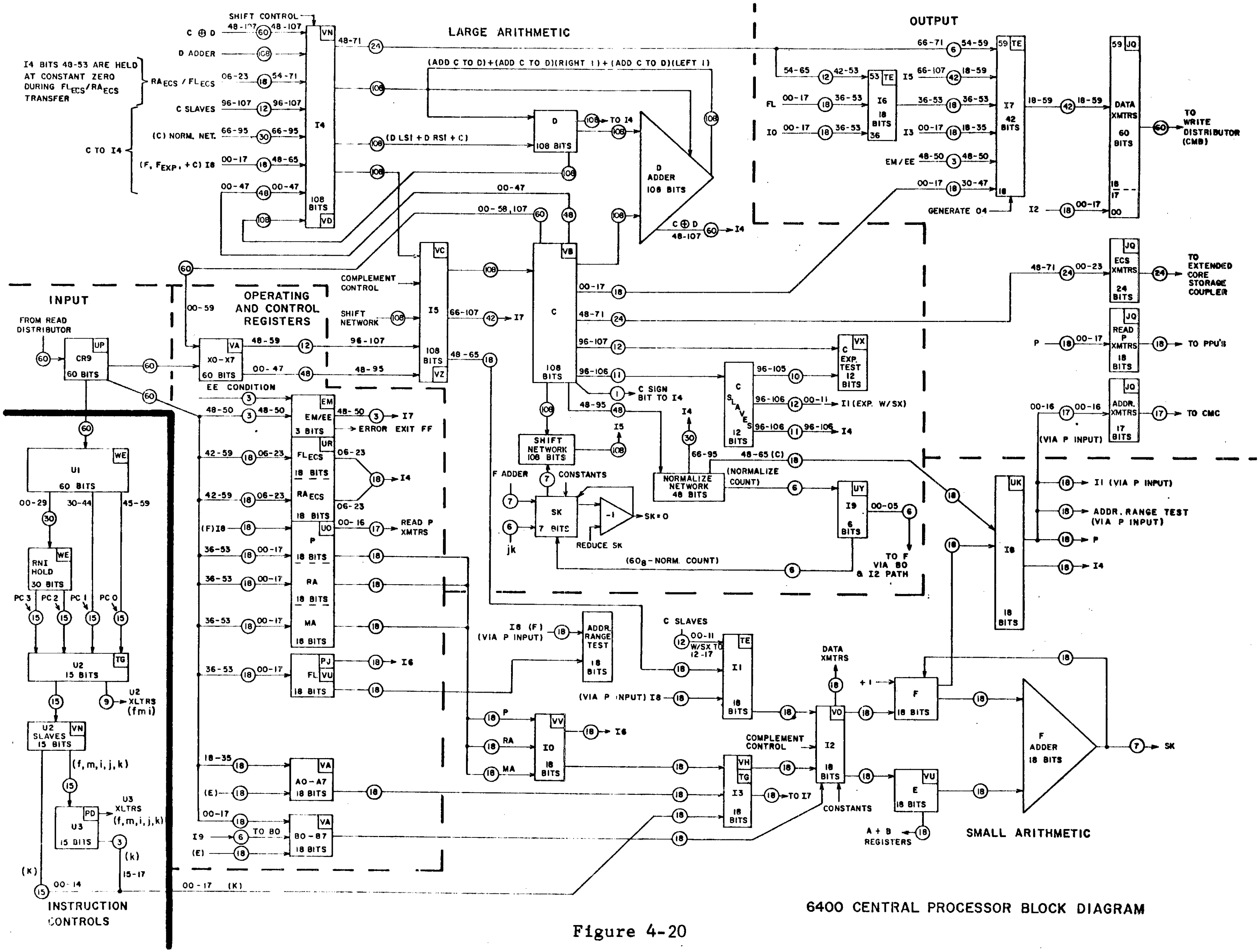


Figure 4-19. Selecting X Registers



6400 CENTRAL PROCESSOR BLOCK DIAGRAM

Figure 4-20

INSTRUCTION CONTROLS

The instruction controls section of the CPU receives, temporarily holds, decodes, and then executes instructions. The section contains the receiving, holding, opcode translating, and sequencing logic. Figure 4-20 shows the instruction controls section minus sequencing.

All instructions flow from CM through the Read Distributor to CR9. From CR9 the instructions continue to the U1 register, from where they are disassembled. A 60-bit instruction word can be made up of four 15-bit instructions, two 15-bit and one 30-bit instructions, or two 30-bit instructions (Figure 4-21).

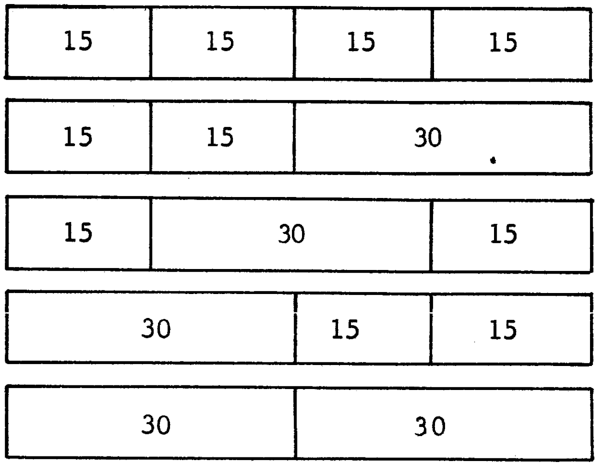


Figure 4-21. Instruction Position Variations

The disassembling of a 60-bit instruction word is called "parceling". A "parcel" in the 6400 is a group of 15 bits. A 60-bit instruction word is grouped as shown in Figure 4-22. Parcel 0 is always copied from U1 first, then Parcel 1, Parcel 2, and finally Parcel 3.

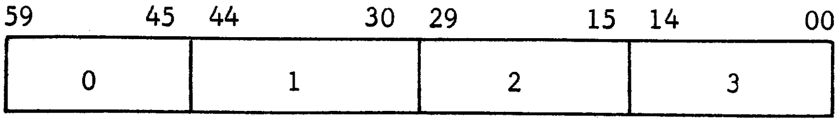


Figure 4-22. A 60-bit Instruction Word Grouped in Parcels

A Parcel Counter circuit enables each parcel from U1 through U2 to the U2 translators and slaves. Normally, the enable is up for most of an instruction execution time; however, the gate is not made into U3 until the next parcel is needed.

One parcel is one 15-bit instruction; however, a 30-bit instruction needs two parcels. Because only one parcel is moved at a time, the upper 15 bits of a 30-bit instruction moves to U3 first, then the parcel counter is advanced. This enables the next parcel, (the lower 15 bits of the 30-bit instruction) through U2. Notice that the U2 slaves also feed the I3 inverter group and that the lower 3 bits (k) of U3 also have a path to I3--"k" and the lower 15 bits make up the "K" of a 30-bit instruction. "K" is gated into I3 early during the execution of an instruction and the parcel counter is advanced again. Thus, the next 15-bit parcel is enabled through U2 and would be the next parcel gated into U3.

Figure 4-23 illustrates how the Instruction Controls section of the 6400 CPU reads and disassembles 60-bit instruction words. Notice that after parcel 0 and parcel 1 have been taken out of U1, the remaining 30 bits are sent to the RNI Hold register. This register holds the last two parcels while the next 60-bit instruction word is read from CM. When the parcel counter equals 2 or 3, the bits are enabled from the RNI Hold register through U2. When the parcel counter is advanced from 3, it will go to 0, enabling parcel 0 from U1. This will be the 1st parcel of the next 60-bit instruction word.

An RNI occurs after execution of the uppermost instruction in each 60-bit instruction word. Consequently, if the uppermost instruction is 15 bits in size, RNI occurs between parcels 0 and 1; if the uppermost instruction is 30 bits in size, RNI occurs between parcels 1 and 2.

Types of RNI Operation

Normally, obtaining an instruction for the 6400 CPU to execute requires a parcel from U1 or the RNI Hold register contents to U3. This type of reading next instruction (RNI) operation is called Parcel RNI.

After the 1st parcel RNI and the execution of the 1st instruction, the contents of the P register are advanced and added to the RA register contents to obtain the actual CM address of the next 60-bit instruction word. This addition needs the Small Arithmetic Section circuits for only 200 ns, after which time the CPU waits for CM. Reading up of the next 60-bit instruction word while the 2nd (and possible 3rd and 4th) instructions are being executed is called a Full RNI.

After jump instructions, (Exchange Jumps, conditional, unconditional, etc.) the next instruction is in the following 60-bit instruction word, still in CM. Therefore, a parcel cannot be taken from U1 or RNI Hold registers. The CPU must stop and wait until the next instruction is available in U1. This process of sending an address to CM and then waiting until it is in U1 is called an Initial Start RNI, or simply an Initial Start. Usually the address sent to CM would be (P) + (RA); however, the Return Jump instruction advances (P) during its Initial Start.

Unlike the Parcel RNI, the Initial Start operation does not place an instruction into U3 and start executing it. Rather than duplicate a

Parcel RNI, the 6400 CPU stops an Initial Start after the word is in U1, then starts a standard Parcel RNI operation.

LARGE ARITHMETIC SECTION (LAS)

The LAS is used during execution of most Central Processor instructions. Most instructions using 18-bit operands will not use this area of the CPU. Figure 4-24 places the LAS in its proper position of the CPU block diagram. The focal point of the LAS is the 108-bit D Adder.

The C and D registers, each 108 bits in size, serve as feeder registers for the D Adder as well as transfer paths used during instruction execution. The other important parts of the LAS are the I4 and I5 inverter groups, which serve as fan-in, fan-out, and control inverters during instruction execution. The following descriptions of the LAS will aid in understanding the over-all operations handled by the LAS.

I4 INVERTER GROUP

The I4 inverter group is 108 bits in size. It can be considered in two parts, the upper 60 bits and the lower 48 bits. In discussing control, the entire 108 bits will usually be considered. In some circumstances, however, the upper and lower parts of I4 do differ in control.

The lower 48 bits of I4 have inputs from two sources, the C register (00 - 47) and the D register (00 - 47). The transfers possible with the lower 48 bits of I4 are:

C → D
C → I5
D → I5
D (Right or Left One) → D

Since I4 is an inverter group and, therefore, only a path for a transfer, a second select term is usually necessary. For instance, when C to I4 is selected, (ENTER D + SELECT I4 → I5)(ENTER C) is necessary to complete the transfer. When a control such as C → I4 → I5 is noted in the command timing, it means the C to I4 and the I4 to I5 selections are being used.

The right and left shift capabilities are internal to I4. For instance, if D enters I4, then the output from I4 could be shifted right or left one position. The right shift is end off, no sign extension; the left shift is not end around. Figure 4-25 illustrates the lower 48-bit positions of I4.

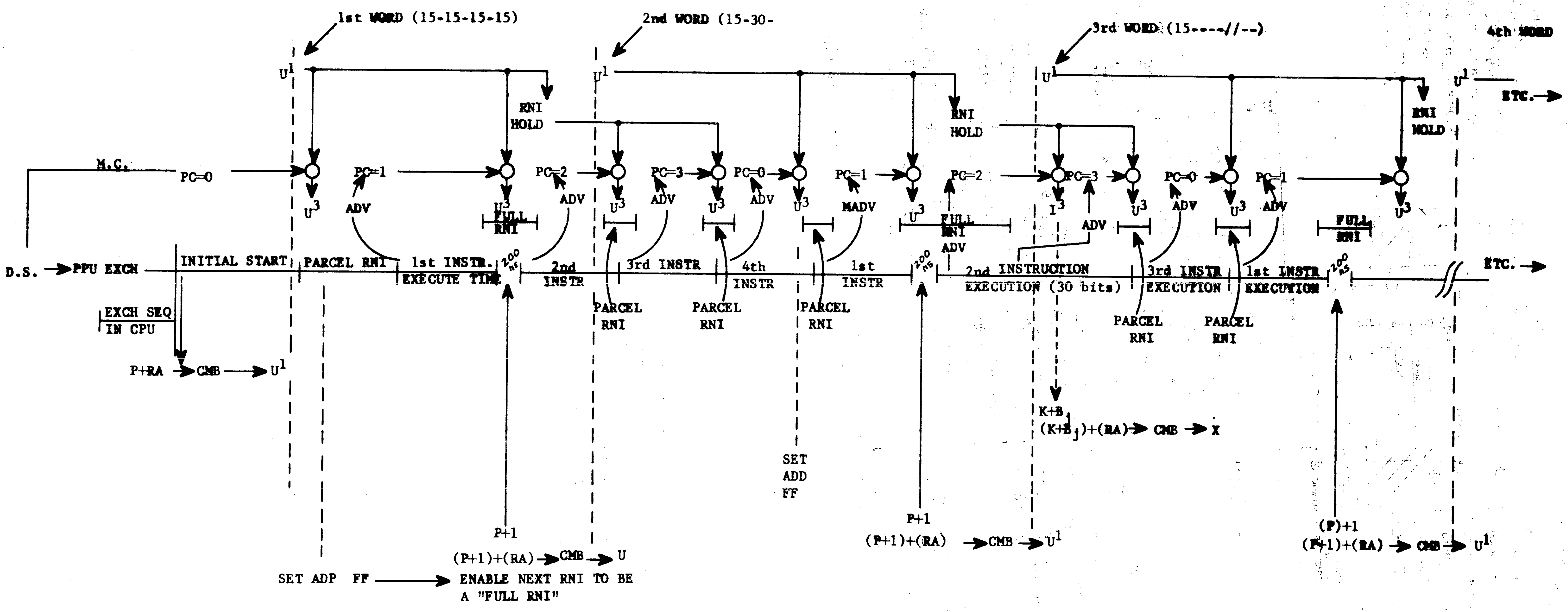
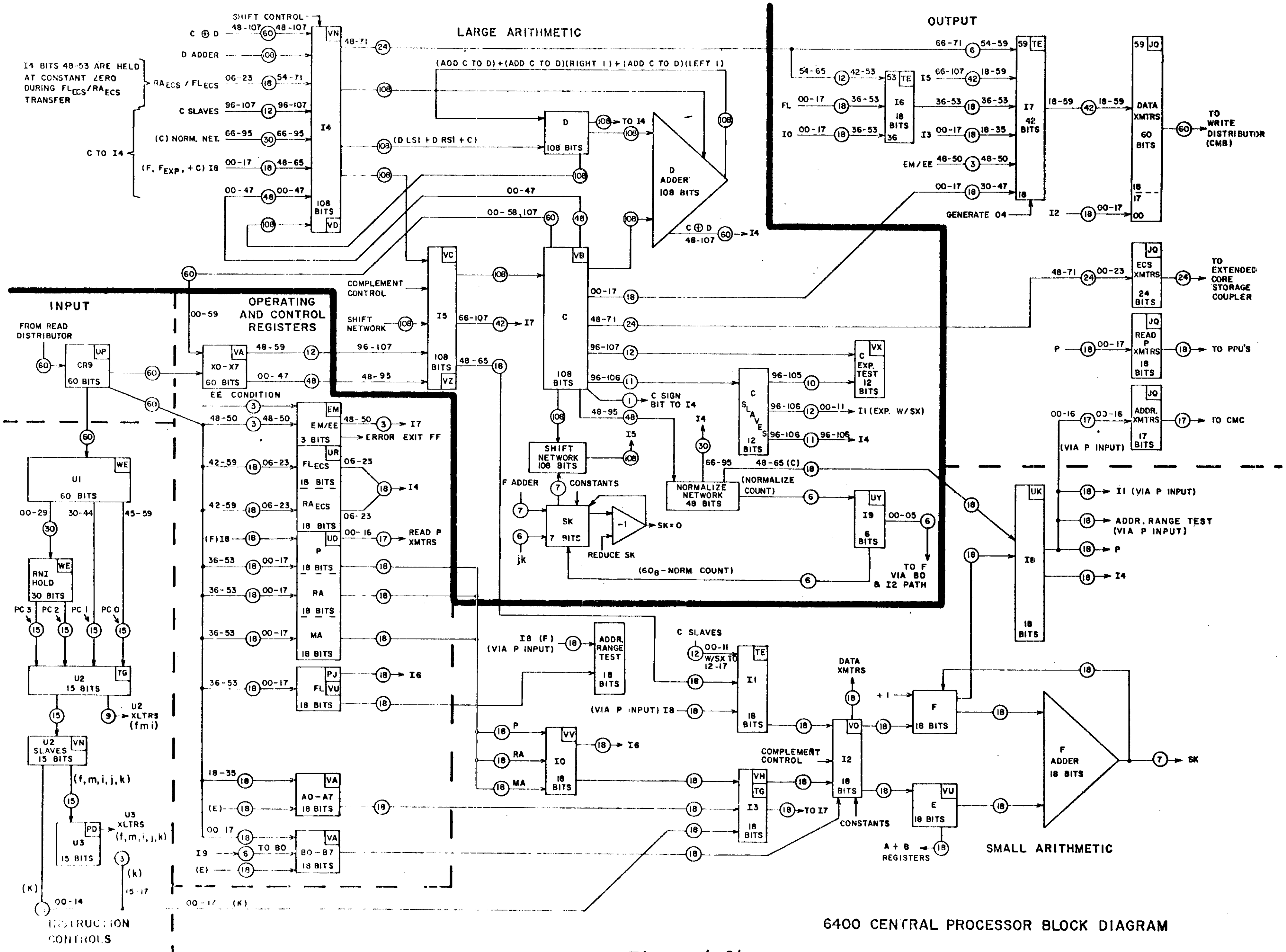


Figure 4-23 6400 CPU Instruction Flow Chart



6400 CENTRAL PROCESSOR BLOCK DIAGRAM

Figure 4-24

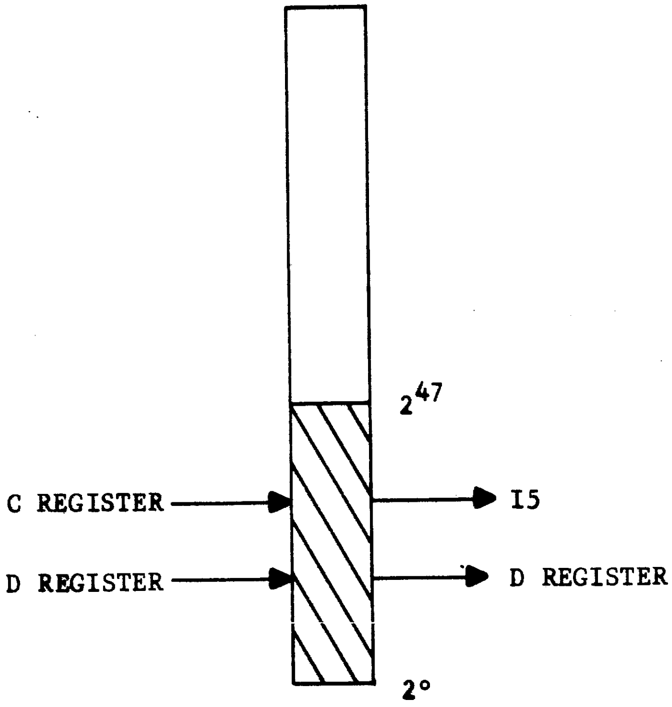


Figure 4-25. I4 Lower

The upper 60 bits of I4 tend to be more complex, because of the complication that evolved in making the transfer, C ⟶ I4. There are six transfers possible through I4 upper. They are:

1. C ⟶ D
2. C ⟶ I5
3. D ⟶ I5
4. D (Right or Left 1) ⟶ D
5. Exclusive OR of C · D ⟶ I5
6. RAecs/FLecs ⟶ I5

Complications arise on a C to I4 transfer since many of the bits pass through other inverter groups during the transfer. If a 108-bit transfer of C to I4 is needed, the lower 48 bits are transferred directly from C to I4. The upper 60 bits, however, travel a devious route before reaching I4. Figure 4-26 illustrates the upper part of a 108-bit transfer of C to I4. The important point to keep in mind is that many of the various inputs to I4 are part of a single transfer of C to I4.

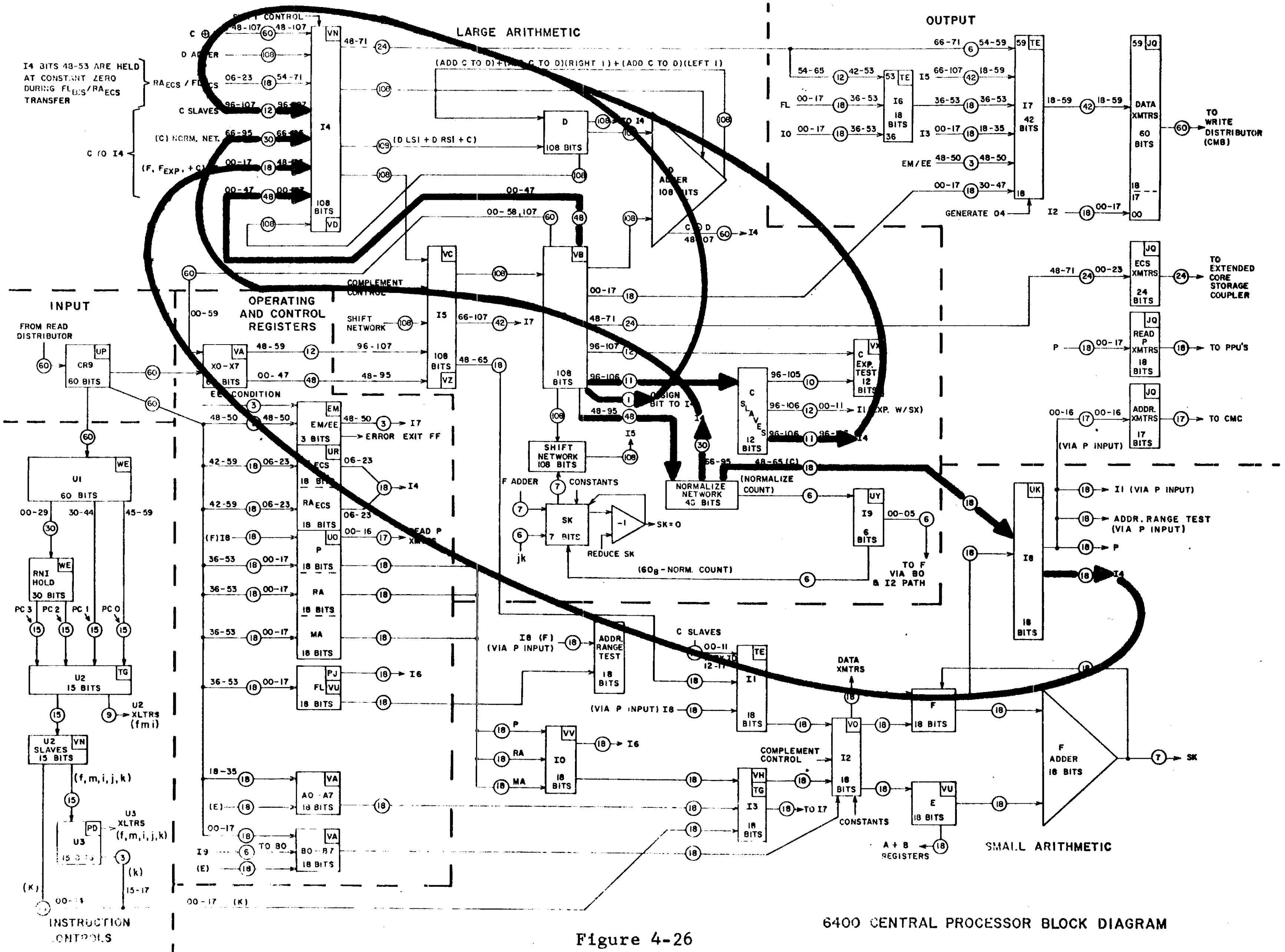


Figure 4-26

6400 CENTRAL PROCESSOR BLOCK DIAGRAM

RAecs/FLecs transfer into I4 is a path used during ECS operations and during an Exchange Jump. The Exclusive OR of C and D input to I4 comes directly from the first rank of the adder. Actually, the Exclusive OR is present in all positions of I4; but, in the lower 48 bits, it merely passes through and continues back to the adder. On the other hand, the upper transfer picks up the signal to be used when the Boolean instruction is executed.

The important outputs of I4 are to the D register and to I5. Both of these transfers are 108 bits and require the controls for the upper and lower parts of I4 to work simultaneously.

I5 INVERTER GROUP

The I5 inverter group is 108 bits in size. As for I4, so too the I5 inverter group can be considered in two parts: I5 upper and I5 lower. The upper section (60 bits) is made up of VC modules; the lower half (48 bits) is made up of VZ modules. I5 is one of the two places in the machine where an entire operand can be complemented. All 108 positions have this capability.

I5 is the secondary select position for the X operating registers. I5 can input any of the X registers completely or without the upper twelve bits (when floating point operands are selected from the X registers without the exponent). I4 can transfer 108 bits to I5. Likewise, the output of the shift network sends 108 bits to I5.

NORMALIZE NETWORK

The process of normalizing a floating point coefficient is one of left shifting the number until the upper-most bit of the coefficient area is a "1". If the coefficient is negative, it is complemented before sending to the Normalize Network. The Normalize Network always assumes a positive coefficient.

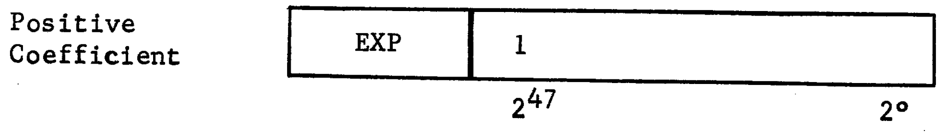


Figure 4-27. Normalized Coefficient

Later, when the coefficient has been used, it will be recomplemented before returning to memory. The process of normalizing a coefficient in the Central Processor is accomplished by inspecting the coefficient to develop a shift count used to left shift the coefficient the correct number of places. The normalize network, therefore, is the area where the necessary shift count is generated.

By dividing the 48-bit coefficient into six groups of eight bits each, the first inspection for the most significant "1" is made easier. Each of these groups is called an octum and is numbered 0 through 5, starting with the upper-most group.

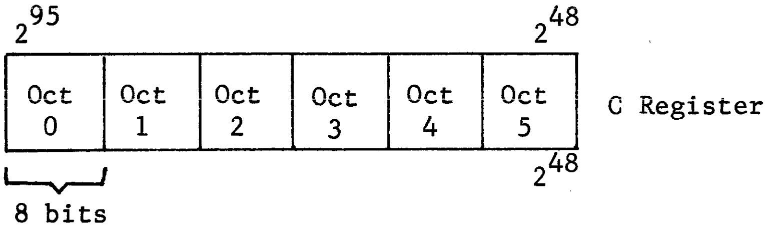


Figure 4-28. Octums

A point to remember here is that a 6-bit shift count is being formed for use by the shift network. The upper three bits of the shift count can be developed by knowing in which octum there is a "1" bit. For instance, if the upper-most octum contains a "1", the left shift will be less than 10_8. Therefore, none of the upper three bits in the shift count will be set. The following table illustrates the development of the upper three bits of the shift count depending on the octum where the first "1" bit was found.

"1" in Octum	Upper Bits =
0	000
1	001
2	010
3	011
4	100
5	101

Table 4-2

Development of the three lower bits is accomplished by examination and translation of the bits within the octum that had the most significant "1" bit (Figure 4-29 illustrates the procedure).

Actually, all of the octums are being translated to determine the lower three bits of the shift count, but only the translation from highest octum (0 = highest) holding a "1" bit is used. For example, assume that the first "1" bit was in position 41 (C register position 89). First,

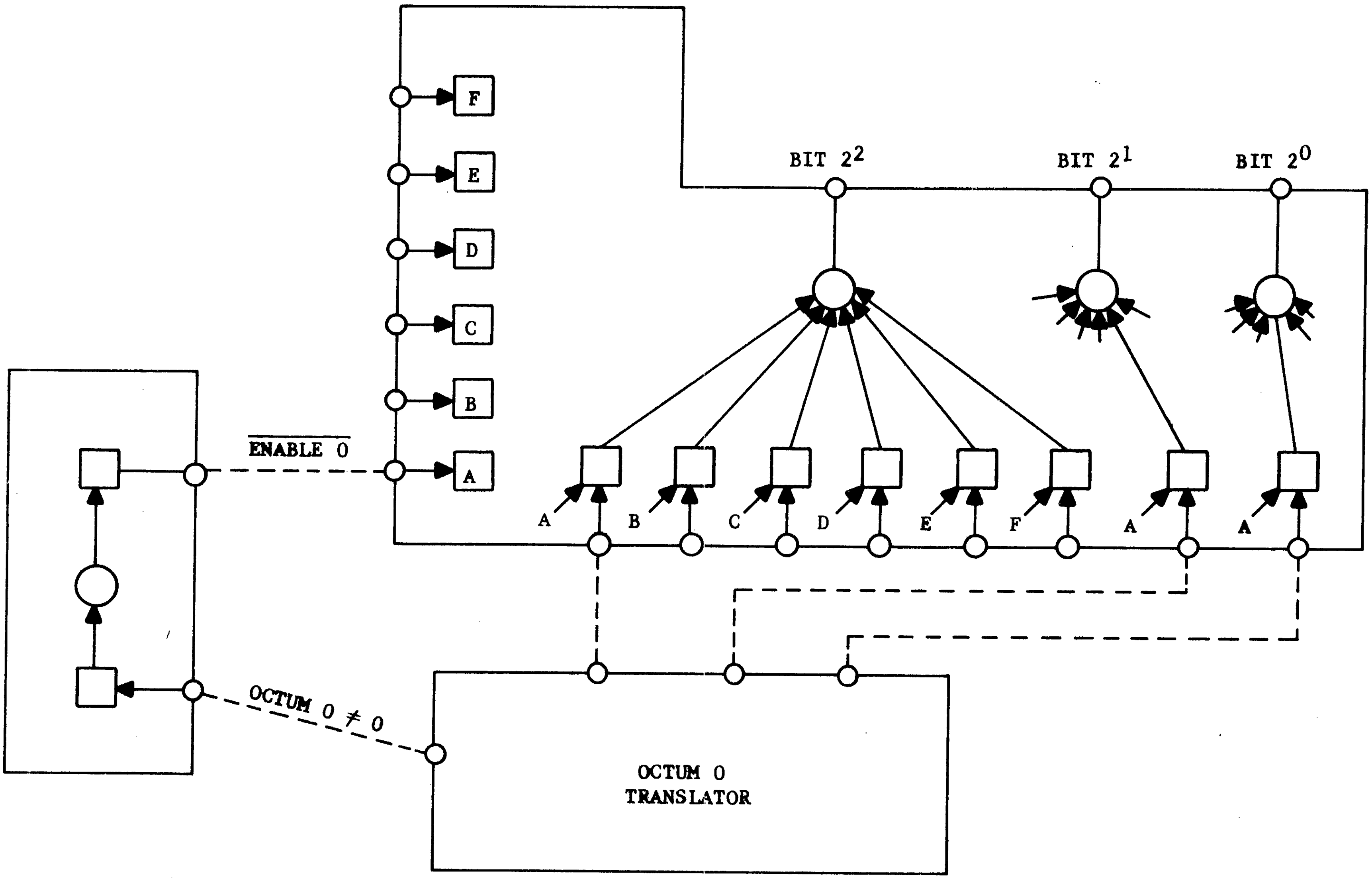


Figure 4-29. Lower Bits from Normalize Net

Octum 0 would be determined as the octum with the needed "1" bit. This translation would enable the output of the Octum 0 translators, which would be six.

The shift count is sent to the I9 inverter group where modifications will be made before setting the SK register. (I9 operation will be discussed later.) Should the coefficient be equal to zero, a special translation denoting this condition causes a signal to be sent to I9, creating the necessary results in the SK register.

SK REGISTER AND DECREMENTER

The SK (shift count) register is seven bits in size although only the lower six can be considered in a programming sense. The seventh bit is determined only by the operational sequences. The SK register is used to determine the number of places that a quantity is to be shifted in the shift network. This may be for execution of a shift or normalize instruction, or it may be for shifting a number in order to alter its position in a data path.

Shifts are determined by the binary value at each position of the SK register, or by the binary sum of the register.

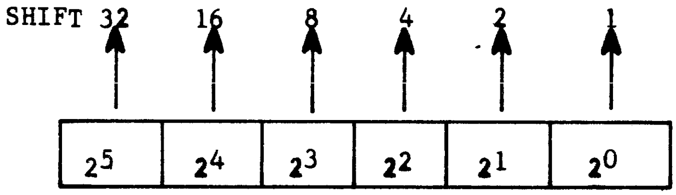


Figure 4-30. SK Register

The determination of right or left is a responsibility of another control circuit.

The shift register must be able to reduce the count during the execution of iterative instructions. During these instructions, the SK register is as a counter to maintain a record of iterations that have yet to be performed.

Error Mode Register (EM), (3 bits)

The Error Mode register is loaded during the Exchange Jump operation. It is 3 bits in size, but is effectively 18 bits during operations and is illustrated as such in the Exchange Jump package.

The Exit Mode register contains a code that will decide whether the CPU will stop on certain error conditions.

If 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).

= 030000 Address or operand out of range.

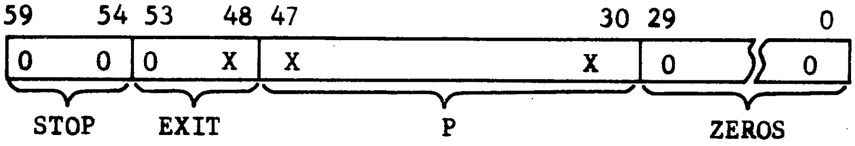
= 040000 Indefinite operand - floating point arithmetic unit (floating Add, Multiply, or Divide) attempted to use an indefinite operand (see Range Definitions).

= 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, the Exit condition (bits 48, 49 or 50 only) and the Program Address at exit time in the following format, and jumps to P = 0 (RA), thereby stopping.



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

Address Out of Range

A. Condition not selected.

1. 5X, i = 6 or 7 instruction.
The CMB address is sent to CMC, but an Abort signal is also sent which clears the Address Catching Register (CRD). The Write bit is in CRO-17 at that time, so CMC assumes a Read of Address 000000. Address 000000 is cycled, but contents not used. CPU continues executing instructions.

2. 5X, i = 1-5.
Same operation as in #1 above, but no X register contents sent to the Write Distributors. The Accept from CMB (via CMC) gates in the contents of Address 000000 from the Read Distributor. This address normally has 0   0. Thus, the X register used, equals 0   0. CPU continues executing instructions.

3. Jump or RNI/IS operations.
Same operation as in #1 above, but the contents of address 000000 is used as an instruction. This normally means a Stop instruction. P would be one address larger than the address of the instruction in error.

B. Condition selected.

1. 5X, i = 6 + 7.
The CMB address is sent to CMC, but an Abort is also sent which clears the Catching register (CRO). The Write bit (bit 17) is in CRO at that time, so CMC assumes a Read of address 000000. The contents of X6 or X7 is sent to the Write Distributor, but are not selected into the Y1 register. Thus, the CMB address from the CPU is not affected, address 000000 is cycled but not used.

2. 5X, i = 1-5.
Same conditions occur as in #1 above with the exception that data is sent to CMB (i = 1-5 implies a read operation). Also, the contents of address 000000 is read into Xi. The contents of the P register, the error condition, and a 00 instruction are stored at address RA; then the P register is cleared. When the next RNI occurs, the contents of RA is read up and executed. This would be a Stop instruction.

3. Jump or RNI/IS operations.
Same operation as in #1 above. The contents of address 000000 is sent to the U1 register.

The following block diagram illustrates the basic paths associated with the SK register.

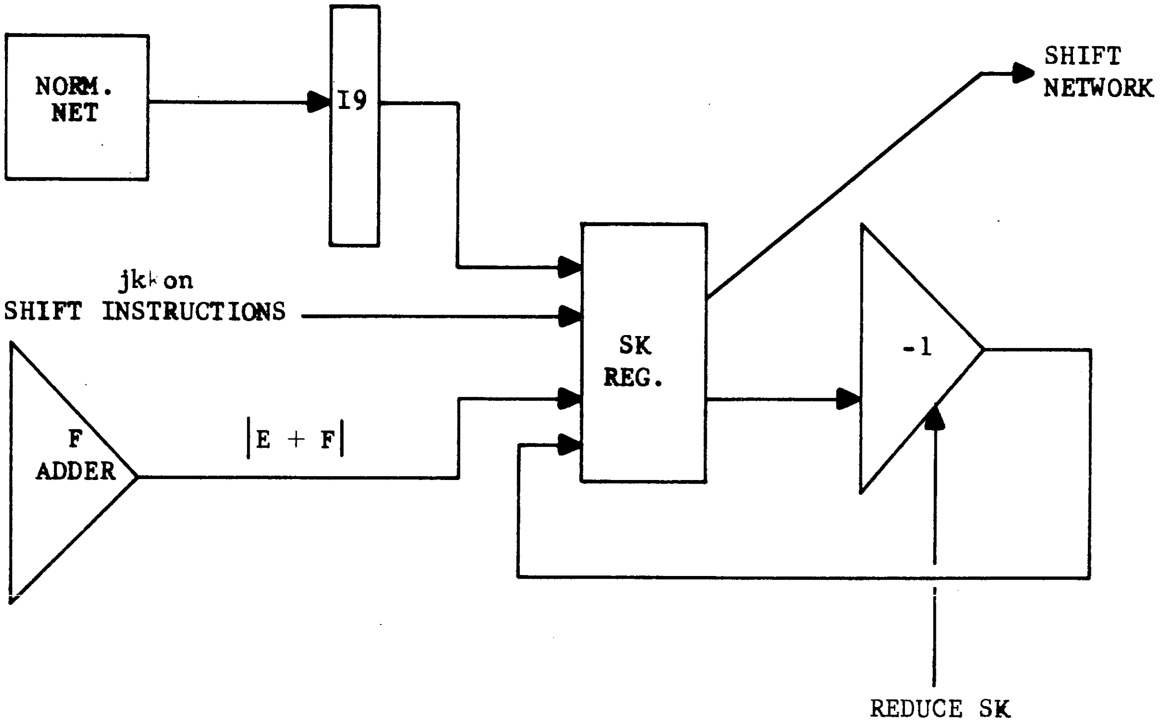


Figure 4-30

Inputs

The SK register has five inputs, each one has a definite purpose. First, there is the input that comes from the jk portion of an instruction via U2 during the execution of a Shift instruction. Another input is from the SK Feeder circuit. This input is the difference between two floating point exponents--in absolute (positive) form--as derived by the F adder. The absolute difference is necessary to equalize the exponents of two floating point numbers that are to be added or subtracted. The process is to subtract the smaller exponent from the larger, which results in a difference that can be used as a shift count. The coefficient of the smaller exponent is right-shifted the necessary number of places, resulting in both floating point numbers having the same exponent. The computer does not care which exponent is larger or smaller. The Xk exponent is always subtracted from the Xj exponent. After the subtract, the sign is checked to determine the absolute value. If the sign is positive, the shift count is correct. If the sign is negative, the shift count is complemented. The entire process is handled in the Feeder network.

Bits 2^{12} - 2^{17} contain the sign of the result. These bits, along with the lower six bits, are sent to the SK feeder as complement enables. A typical path is shown in Figure 4-31.

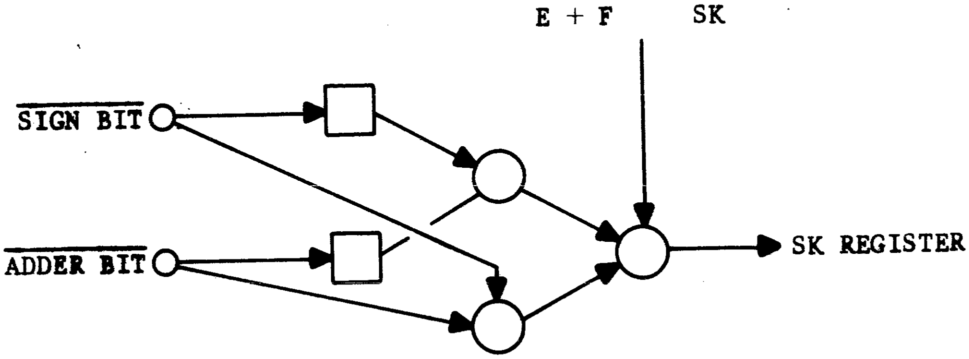


Figure 4-31

If the sign bit is negative, the complementing path is taken. If the sign is positive, the uncomplementing path is taken. The decision as to which coefficient should be shifted is also decided by the sign of the result. Since Xj is always the minuend, if difference sign is negative, the minuend must have been larger than the subtrahend (Xk).

A third input to the SK register is Reduce SK. This input is coming from the decrementer that is fed by the SK register.

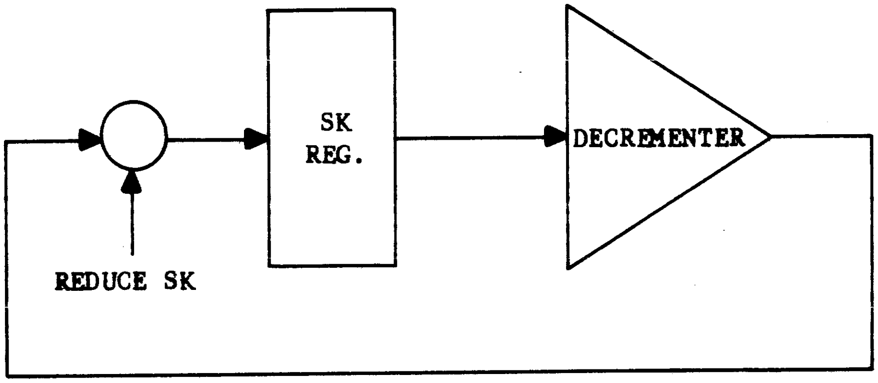


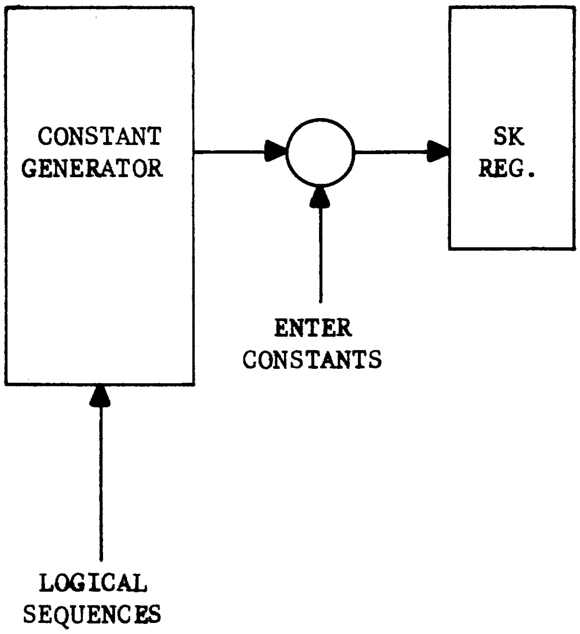
Figure 4-32

The decrementer is, effectively, a two's complement binary subtracter. The count in SK will be reduced one count for each iteration of a multiply or divide operation. A translator off the SK register determines when the SK register equals zero, to end the operation.

A fourth input to the SK register comes from the I9 inverter group. This input is the shift count generated by the Normalize network. Before I9 sends the shift count, however, a modification is made: the shift count is subtracted from 60_8. The reason for this is that during a normalize operation, the coefficient is in the C register in bit positions 95 - 47. After normalization has completed by the Shift network, the coefficient must be stored in an X register. This means we must right-shift the number in the C register 60 places to position it in bit positions 00 - 47.

Since noramalizatin involves a left shift and relocation of the numbere involves a right shift, the complete operation can be handled in one pass through the shift network. This is done by subtracting the normalization count from 60 and sending the result to the SK register. For instance, if the normalization count was 7, a right shift of 51 places would accomplish the same result as a left shift of 7, followed by a right shift of 60_8.

The last input to the SK register consists of the constants needed during final normalization or during iteration operations. These constants will come from the Constant Generator and will be gated in at the proper time by the control sequences.



SHIFT NETWORK

The Shift Network is 108 bits in size and is made up of 5 inverter ranks interlaced with data paths and the necessary controls. Each rank of the network shifts the number right, left, or straight ahead, depending on the controls. The ranks that shift are determined by the count in SK.

The Shift Network must be considered in parts--the upper 60 and the lower 48 bits. The upper 60 bits can shift right, left, or no shift, while the lower 48 bits can only shift right or no shift. Frequently, the upper 60 bits must be considered as a Shift register in itself, since a left shift will end-around to position 48 of the network. All right shifts will have the sign bit extended. Usually a right shift of 77_8 is considered maximum, but during the equalization of exponents when executing a floating add or subtract, a right shift of 177_8 is possible. Since the Shift network can only handle shifts of 77_8, this extra right shift of 64 is wired in between the C register and the first rank of the Shift Network.

The controls for shifting are from the SK register. The binary value of each position represents the number of places to shift. If we consider a number entering Rank I of the network (usually in the upper 60 positions), the output of Rank I could be right-shifted 32, unchanged in position, or left-shifted 32. (Left shifting is in the upper 60 bit positions only.) Rank I feeds Rank II where a shift of 16 positions to the right or left could occur. This process continues until the number has filtered through the network. Sign extension is controlled by the C Sign Record control, which previously monitored the 108th bit of the C register. This control extends ones or zeros into the Shift Network. (See Figure 4-34 for an illustration of the Shift Network.) The final shift position of one place makes use of the I5 inverter rank. If a shift of one is needed, either right or left, the last rank is fed to I5.

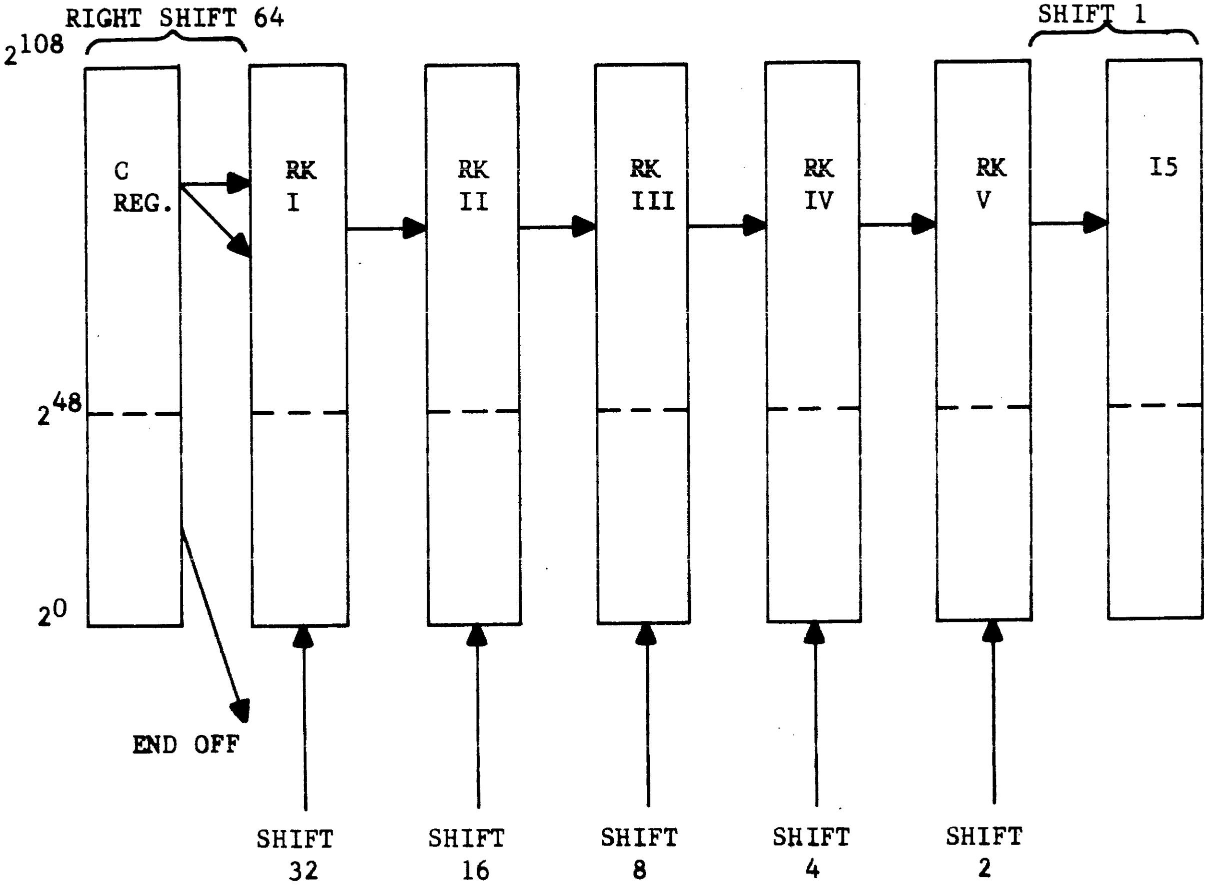


Figure 4-34. Shift Network

I9 INVERTER GROUP

Part of the I9 operation was explained in the SK register discussion. I9's only input is from the Normalize Network (6 bits). The outputs are to the SK register (60_8 minus normalize count) and to the F register. The transfer to the F register is a path used to store the shift count in Bj upon completion of the normalize operation. The shift count will follow the path F ⟶ I8 ⟶ I1 ⟶ I2 ⟶ E ⟶ Bj.

If the Normalize Network attempts to normalize a zero coefficient, I9 receives a signal that causes its output to generate a shift count of 60. The end result is a zero coefficient with an exponent of 60 less than the original.

C REGISTER

The C register consists of 108 flip-flops and serves as the feeder network to the D Adder and also as a data path used during many operations. There is only one input to C and that is from I5. C has outputs to the following circuits:

D adder
Shift network
I7 (Output inverter rank)
X operating registers
I4 inverter group
ECS transmitters
C slaves

Each of the outputs must be more specifically positioned since the complete C register is not usually used for each of the transfers.

Adder Output

All 108 bits of the C register are unconditionally fed to the D adder, where translation for the carry networks will be made. The D adder is always active; the output available upon control request.

Shift Network - Output

All 108 bits of C feed the Shift Network, but all positions do not serve the same purpose. The Shift Netwrok shifts a number right or left 77_8 except during a floating add or subtract when the shift network can shift right 177_8 places, since the equalization of exponents can be this great. This seventh bit in the shift count indicates a right shift of 64 (binary position value) and is wired between the C register and the Shift Network. Therefore, the C register feeds the Shift Network in all 108 positions, if the right shift 64 is not being used. If the right shift is needed, then the lower 64 positions of C will not go to the Shift Network, since right shifts are end off. Instead, bit 64 will go to position 00 of the Shift Network, bit 65 to position 01, bit 66 to position 02, etc. (see Figure 4-35).

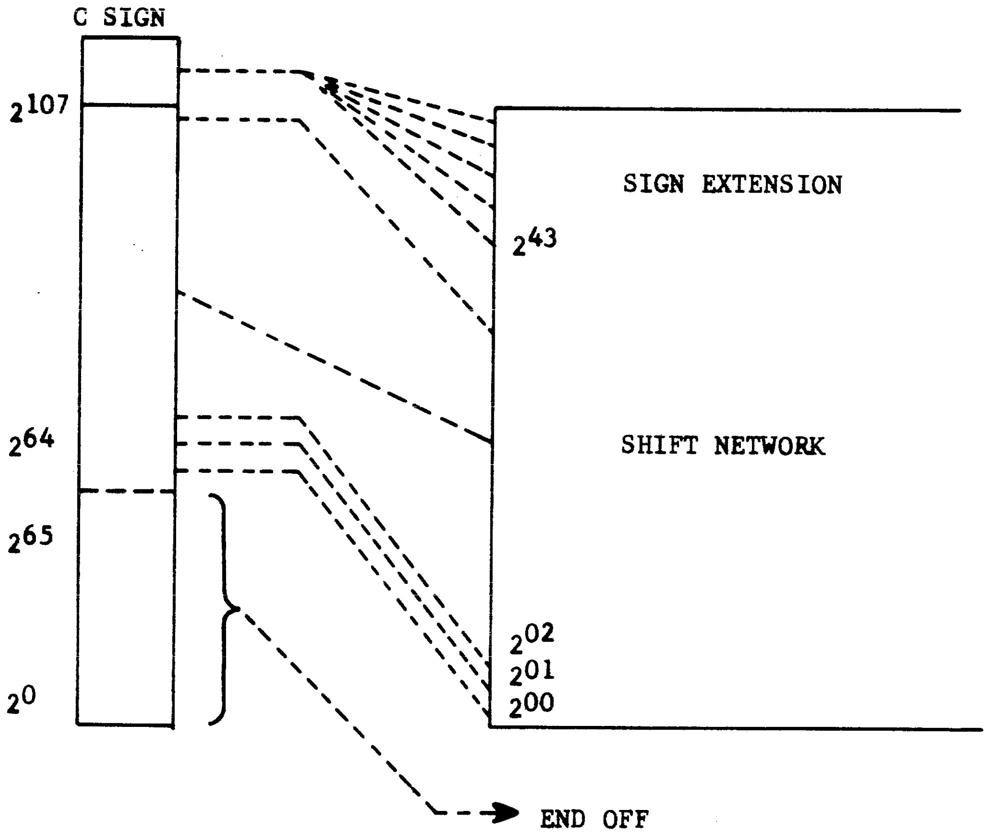


Figure 4-35. RS64

I6 and I7 Outputs

These outputs are used during an Exchange Jump to send the contents of certain control registers to memory.

X Operating Registers - Output

The output to the X operating register is used to store results at the completion of an operation. Only the lower 60 bits are used durint this operation. The C register data are sent to the X feeders and then to the proper X register. When this transfer is made, the lower 59 bits and the 108th bit (to indicate the sign of the number) are sent.

I4 Output

Whenever a C ⟶ I4 transfer is called for by the sequences, the lower 48 bits go directly to I4. However, the remaining 60 bits travel through various other circuits before reaching I4. See the text concerning I4 inputs to better understand this transfer.

ECS Transmitters

During ECS operations the Central Processor must send the ECS address to the ECS coupler. The ECS address was stored in X^0 and will travel through C via I5 to the transmitter networks.

C Slaves Output

The upper twelve bits of C are always sent to a fan-out circuit called the C slaves. The output of the slaves will feed the C Exponent Test circuit ant the sign extension controls for right shift.

D REGISTER

The D register consists of 108 flip-flops. D serves as one of the Feeder registers for the D adder, as well as the Result register for all operations using the adder.

The D register has two outputs, one is to I4 (108 bits). This output will be used on the D to I4 to C transfers. It will also be used during iterative operations for the D to I4 to D transfers. Bits from C and D are compared at the output of D in order to determine the carries, etc., needed during add operations.

Inputs to the D register come from I4 and from the last rank of the D adder. Data entering D from I4 can be the C register, or the D register ($D \longrightarrow I4 \longrightarrow D$), shifted right or left one (shifted in I4). The input to D from the adder can be the adder results or the results shifted right or left one. These shifted inputs are used during iterative operations.

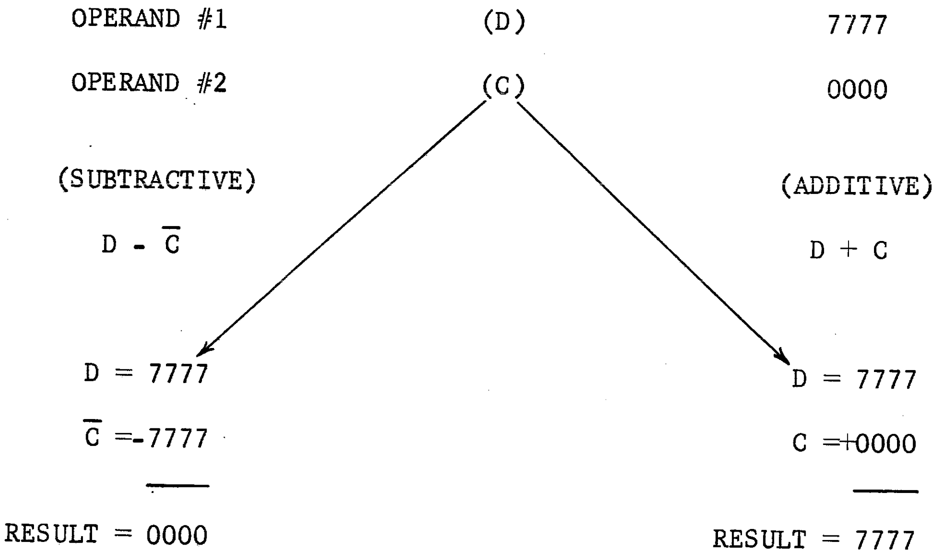
D ADDER

The D adder is the focal point of the LAS. All boolean, floating multiplies, divides, floating adds and subtracts, and integer add and subtract instructions use the adder. Various other operations, such as FL checking for ECS instructions, also use the adder.

Definition of the adder is arbitrary since all adders operate under the same basic theory. In general, adders are defined as being additive or subtractive. These definitions consider the development of positive or negative zero as a result when adding certain pairs of operands.

An adder that usually develops positive zero is more useful and, therefore, is the type desired in the CPU. Usually, this type of adder is defined as a subtractive adder since the only pair of operands that will produce a negative zero result is negative zero and negative zero. The equation for a subtractive adder would be $C - \overline{D}$, or $D - \overline{C}$, where C and D are the two original operands. The explanation of this type of adder would include the presence of borrows, and the consideration of inverting one of the operands.

Consider the following operands as they are added, using each set of rules.



OPERAND #1 (D) 7777

OPERAND #2 (C) 0000

(SUBTRACTIVE)

$D - \overline{C}$

$D = 7777$

$\overline{C} = -7777$

RESULT = 0000

(ADDITIVE)

$D + C$

$D = 7777$

$C = +0000$

RESULT = 7777

Figure 4-36

This adder is considered subtractive; therefore, the following set of rules are used:

	ACTUAL REGISTERS		ARITHMETICAL CONTENTS	
	D	C	D	$\overline{C}$
SATISFY	1	1	1	0
PASS	1	0	1	1
PASS	0	1	0	0
BORROW	0	0	0	1

Table 4-4

A borrow is when any stage generates a borrow from the next higher stage. A pass indicates that a stage will not generate a borrow, nor will it satisfy one, but will pass the borrow on to the next higher stage. A satisfy indicates that an incoming borrow will be satisfied. These terms can be applied to groups and sections of the adder as well as to stages. Before proceeding, a general review of an adder will be given. Any stage can be considered as the result of the logical addition of the two members. This is called the half add. This same stage must also be considered with the presence of an incoming borrow. This would be the full add. For the half add, there are four combinations to consider:

0	1	1	0	ARITHMETIC CONTENTS OF REGISTERS
0	1	0	1	
0	0	1	1	

It can be seen that there are two possible results for the four combinations. If both members are equal, the half add is a zero; if they are opposite, the result is a 1. Considering these same combinations with borrow inputs, it can be seen that the borrow will complement the results.

	0	1	1	0	ARITHMETIC CONTENTS OF REGISTERS
	0	1	0	1	
RESULTS WITH NO BORROW INPUT	0	0	1	1	
RESULTS WITH BORROW INPUT	1	1	0	0	

Therefore, if the members are equivalent and have a borrow input, the result is a 1. The following table defines all the combinations. (E = Equivalence, B = Borrow)

$$E \cdot B = 1$$

$$E \cdot \bar{B} = 0$$

$$\bar{E} \cdot B = 0$$

$$\bar{E} \cdot \bar{B} = 1$$

To make use of this equivalence check, the equivalence of a circuit must be represented with a 1. Therefore, a stage can be equivalent (1) or not (0).

The rules necessary to operate the adder can be applied to source operands and the results calculated using paper and pencil. For the sake of space, 12-bit operands are used.

$$\left.\begin{array}{rr} D = & 1\;2\;3\;4 \\ C = & \underline{+5\;6\;7\;0} \\ \text{RESULT} & 7\;1\;2\;4 \end{array}\right\} \longrightarrow \begin{array}{rr} D = & 1\;2\;3\;4 \\ \bar{C} = & \underline{-2\;1\;0\;7} \\ & 7\;1\;2\;4 \end{array}$$

Considering the above operands, first write them in binary notation. Then, above each stage, note the condition of that stage using the rules for borrows, passes, and satisfies.

```
                              , END AROUND
          ↓    ↓     ↓  ↓↓↓/  BORROW
         PBS  PSB  PSS  PBB
     D = 001  010  011  100 ⎫  ACTUAL
     C = 101  110  111  000 ⎭  CONTENTS
 EQUIV = 011  011  011  011
BORROW
INPUT  = 100  010  011  111
RESULT = 000  100  101  011

ANSWER = 111  011  010  100
```

```
                              , END AROUND
          ↓    ↓     ↓  ↓↓↓/  BORROW
         PBS  PSB  PSS  PBB
     D = 001  010  011  100 ⎫  ARITHMETIC
     C̄ = 010  001  000  111 ⎭  CONTENTS
 EQUIV = 100  100  100  100
BORROW
INPUT  = 100  010  001  111
RESULT = 111  001  010  100

ANSWER = 111  001  010  100
```

In the above examples, the equivalence of each stage (the equivalence is represented by a 1) is first formed. Next, the borrow <u>input</u> to each stage is determined. These inputs are represented by ones. Application of the rules produce the final answer. Note that when using the "actual contents" of C + D, the result must be complemented to obtain the correct answer. With this basic theory of operation a Subtractive adder, we can describe the physical and theoretical operation of the D adder.

The D adder is 108 bits in size, divided into 6 <u>sections</u> of 18 bits each. Each section is divided into 6 <u>groups</u> of 3 bits each (see Figure 4-37).

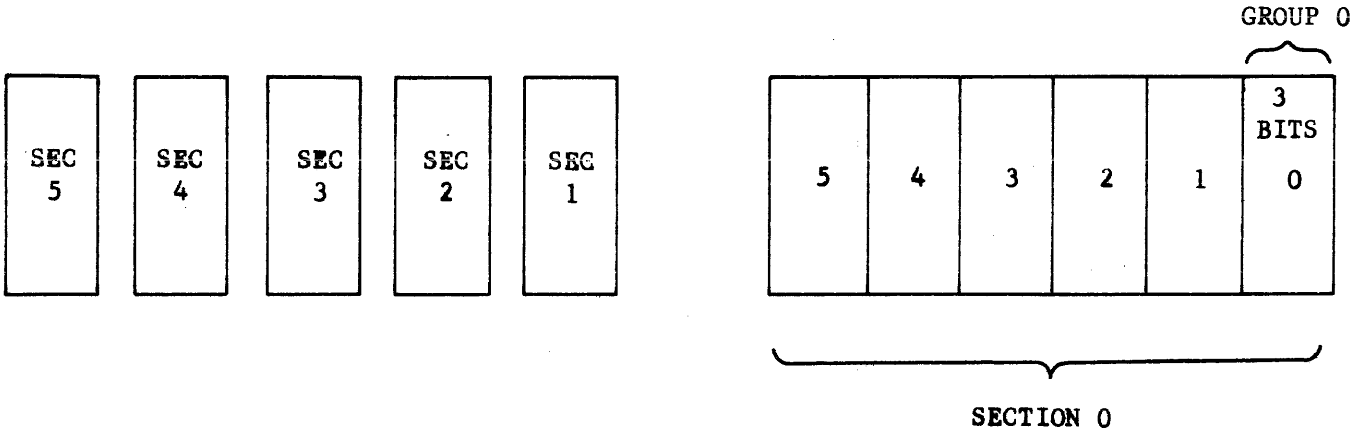


Figure 4-37

We can consider the adder to be made up of three main areas (see Figure 4-38). They are:

1. Half-adder
2. Borrow network
3. Summation network

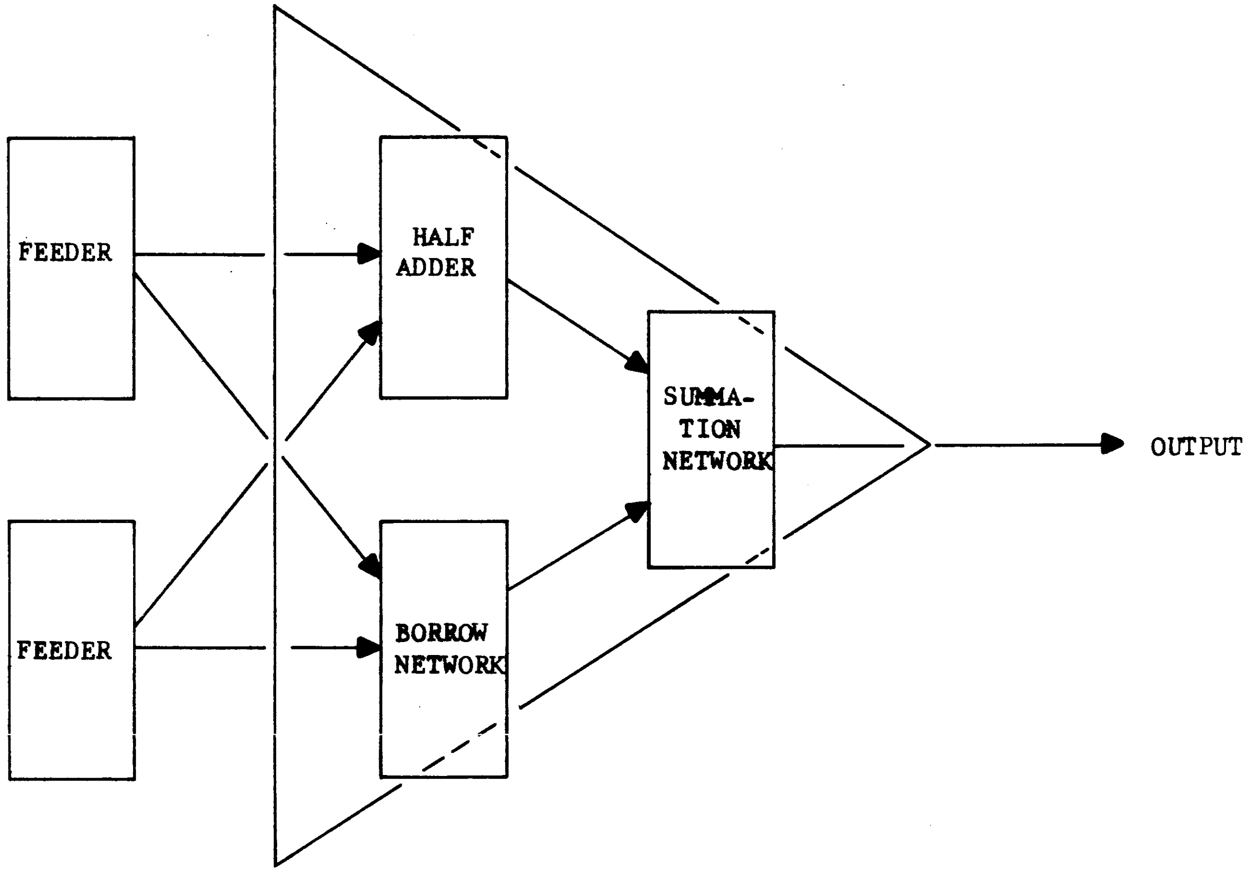


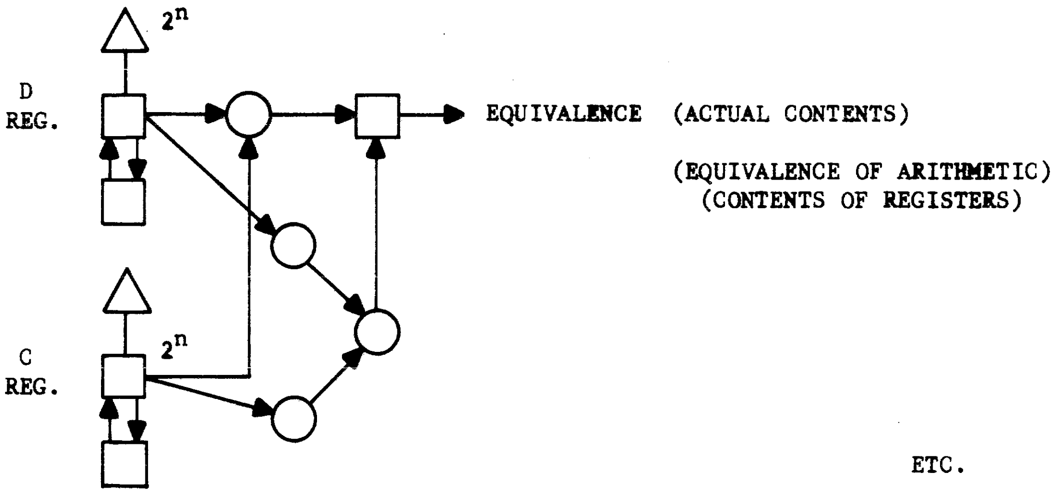
Figure 4-38

The C and D registers serve as the feeder networks for the two operands. Each stage of C and D are compared to determine their equivalence. The determination of borrows, satisfies, and passes are then made for each stage. The borrow network propagates this information through several ranks to determine the final state of each group and section. Finally, the borrow network and the half add are brought together to form the final answer (Figure 4-39).

To analyze the adder, consider the 2^0 and 2^1 bits of the adder and the borrow network. Loking at the adder from this viewpoint allows a partial examination of all the circuits. Use Figure 4-40 as the operation is explained.

First, develop the equivalence from the lower bits of the C and D registers. Determination of an end around borrow must involve searching all stages of the adder for borrows and the possible satisfaction of such borrows before reaching the last stage of the adder.

Considering only the last stages of the adder (2^{107} - 2^{105}), determine if there will be an end around borrow. Using Figure 4-40 notice that if the



EQUIVALENCE CIRCUITS
Figure 4-39a

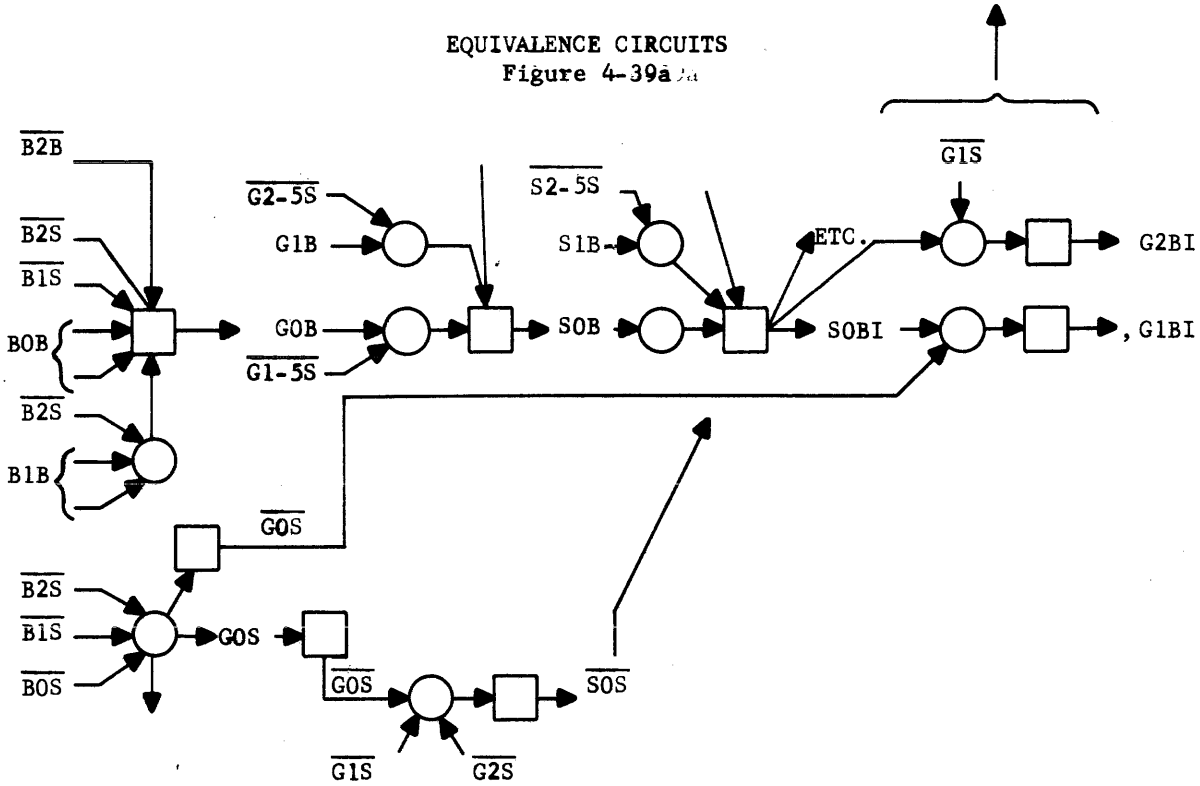


Figure 4-39b

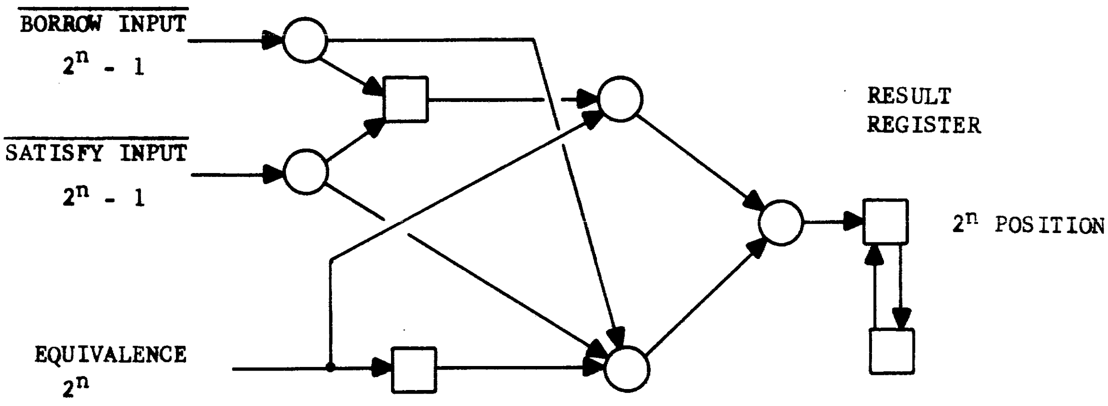

$\overline{\text{BORROW INPUT}}$
$2^n - 1$
$\overline{\text{SATISFY INPUT}}$
$2^n - 1$
EQUIVALENCE
2^n
RESULT
REGISTER
2^n POSITION

Figure 4-39c

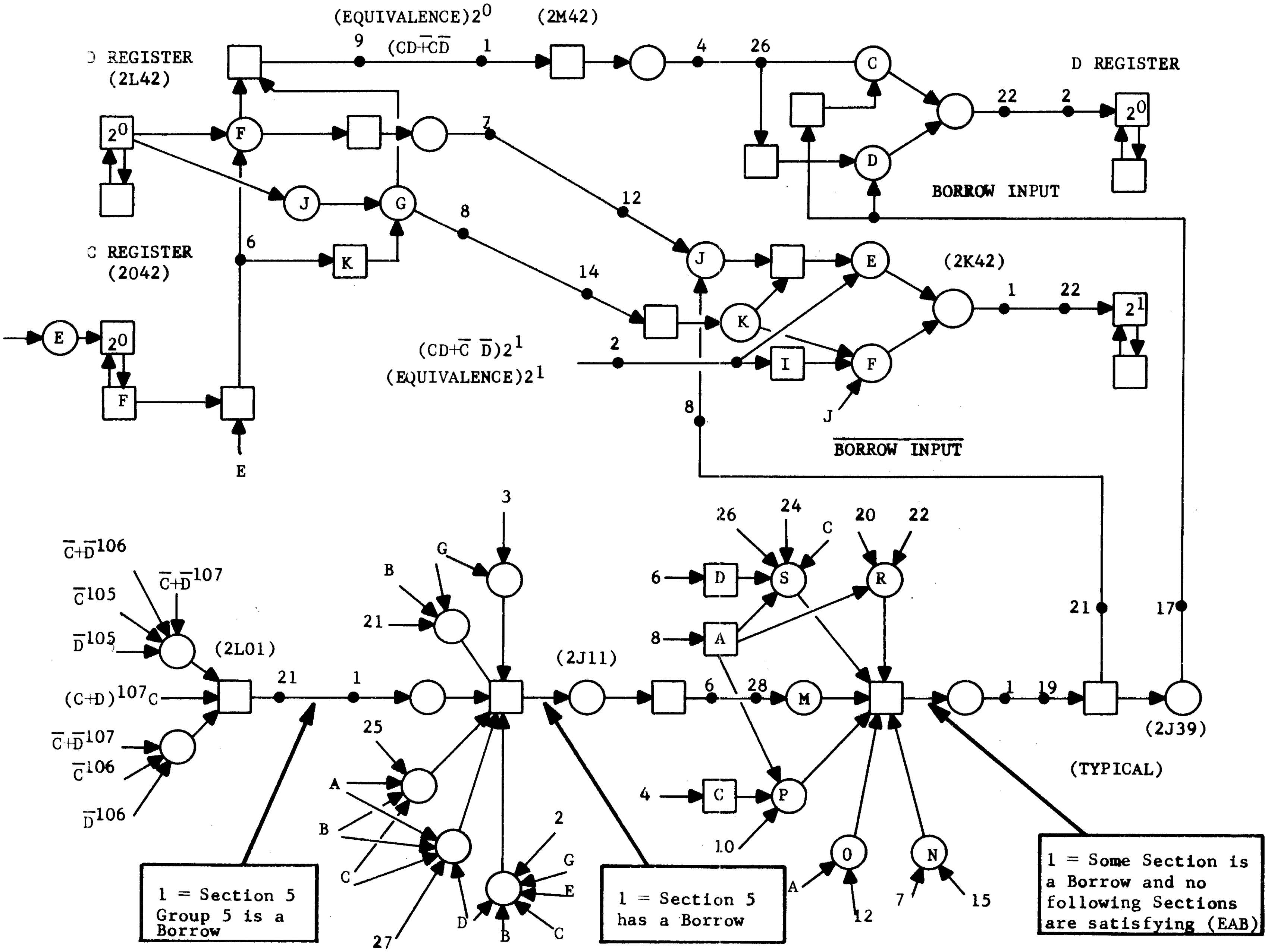


Figure 4-40. D Adder Analysis

conditions indicate an end around borrow, there is a "1" out of the OR. The input to the OR is: (Look for "0" inputs to the OR).

$$(1) \quad (\overline{C} + \overline{D})^{107} (\overline{C} + \overline{D})^{106} (\overline{C} \cdot \overline{D})^{105}$$

$$(2) \quad + (\overline{C} + \overline{D})^{107} (\overline{C} \cdot \overline{D})^{106}$$

$$(3) \quad + (\overline{C} \cdot \overline{D})^{107}$$

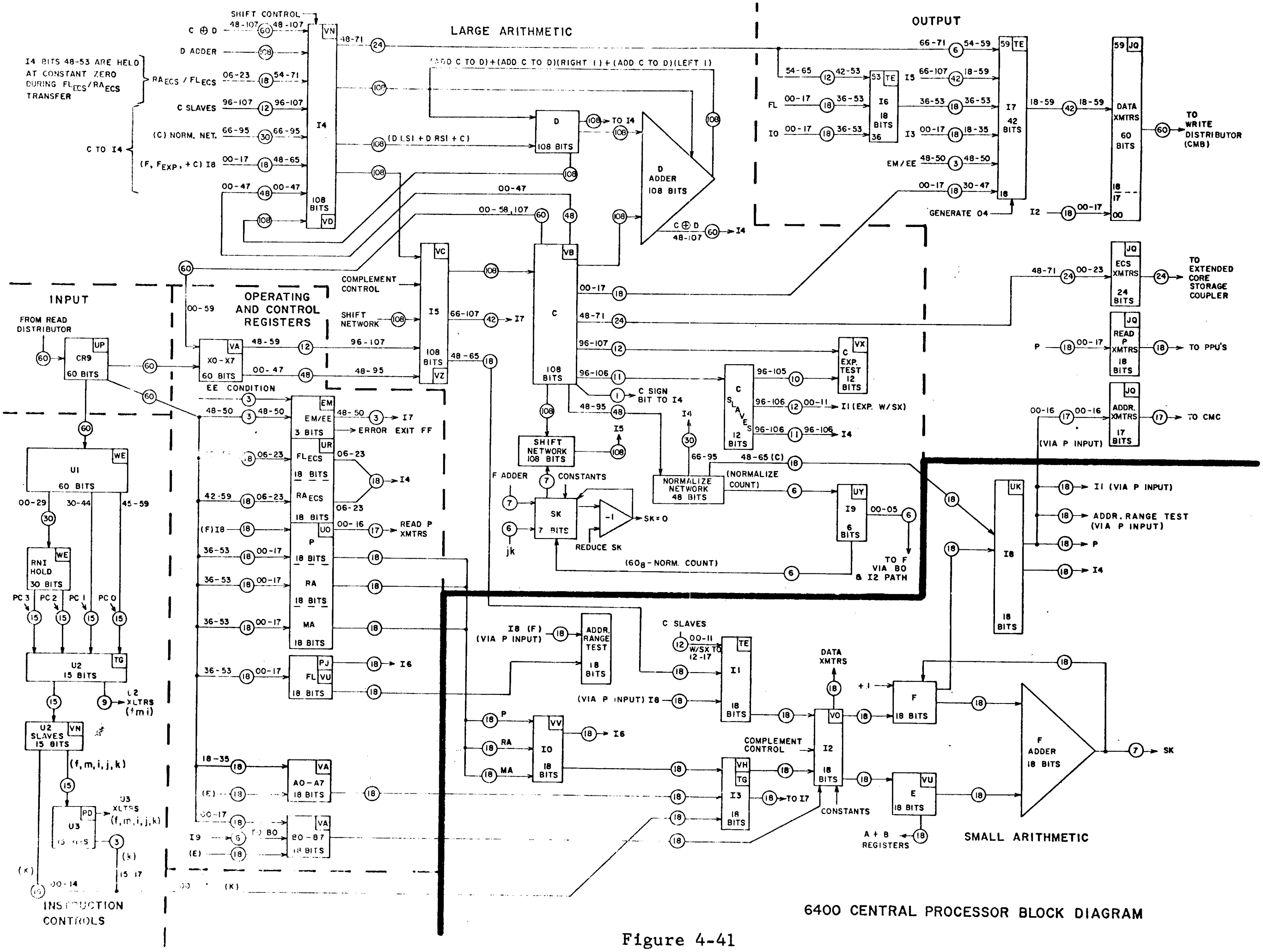
Term (1) indicates that there was a borrow in stage 105 and no satisfies after. Term (2) indicates there was a borrow in stage 106 and stage 107 was not a satisfy. Term (3) indicates that the last stage is a borrow $(C \cdot \overline{D})$. The output of the OR, therefore, says that the last stage of the last adder section is a borrow. If this is true, this output is all that is necessary to cause an end around borrow. However, assuming that the last stage is not a borrow, then the next circuit in line considers all groups in the last section of the adder. Specifically, this circuit examines all the groups in the section looking for a borrow and no following satisfies. The case could be that the last section is not a borrow; therefore, further examination of the adder is made in the next circuit. Here all the sections feed their results into an OR that decides if any section is a borrow and that no satisfies follow. With some study and a certain amount of spatial perspective, it can be seen that all of these circuits mentioned are duplicates of the circuits that are illustrated.

The final analysis of the adder involves looking at how an end around borrow affects the lower stage of the adder. It can be seen, using Figure 4-40 and applying the rules set forth before, that the end around borrow will inhibit the lowest stage of the result if the two original operands were equivalent in this position. By assuming the other possible conditions, all of the combinations can be tried and will be found to hold true.

The second stage of the adder is also illustrated in Figure 4-40 in order to show the effects of an end around borrow. It must be remembered that each higher stage may be influenced by an end around borrow, depending on what the lower stage did.

SMALL ARITHMETIC SECTION (SAS)

The SAS section of the Central Processor handles operations using 18-bit operands. This would be increment instructions, jumps, and exponent manipulation during floating point operations. The SAS includes several inverter groups, registers and an 18-bit adder. Figure 4-41 places the SAS in the Central Processor block diagram. A basic description of the integral parts of the SAS follows; this will allow a better understanding, later, of operations performed here.



6400 CENTRAL PROCESSOR BLOCK DIAGRAM

Figure 4-41

I1 Inverter Group

The I1 inverter group is 18 bits in size and consists of a three-way fan-in. (TE Module) I1 has only one output, to I2. One input is from the F register via I8. This would be used during the addition of RA to P before referencing memory.

The second input is bits 48 - 65 of I5. This transfer is necessary when executing an instruction which needs to reference an X register, or on an Exchange Jump (X ⟶ I5 ⟶ I2 ⟶ D.T.).

The third input is from the C register via the C slaves. This transfer is the exponent of a floating point number. The transfer is 11 bits with the sign of the exponent extended to fill I1. The bias is also removed.

The process of bias removed and sign extension is accomplished by examining the sign of the exponent (C^{106}) which would also be the bias, and extending this sign to the upper eight bits of I1. For instance, if the exponent was 2177, the sign of the exponent is positive. (If the coefficient was negative, a zero in C^{106} would represent a positive exponent.) Therefore, zeros are extended in the upper eight bits of I1. This produces an exponent of 000177, feeding the adder (bias removed and sign extended). For a negative coefficient having an exponent of 5600, C^{106} is equal to a zero (still a positive exponent) and ones are extended in the upper eight bits of I1. The result is 777600. However, the exponent is complemented in I2, resulting in 000177. Notice that the same final exponent is feeding the adder as was in the first example. This proves that 2177 and 5600 are the same exponent, but the coefficient is negative in one and positive in the other.

I2 Inverter Group

The I2 inverter group is 18 bits in size and has the capability to complement any operand passing through it. Complementing would be used is a negative coefficient during floating point operations, where the exponent must be complemented, and during other operations where constants must be subtracted in the F adder. I2 also has the B registers as two inputs. Any selection at the B register will come to I2. The output of I2 feeds the E and F registers that feed the F adder. There is also an output that feeds the data transmitters. This would be used during an exchange jump. One input to I2 comes from I3. This would be P, RA, the A operating registers or the K part of an instruction word when 30-bit instructions are being executed. The last input comes from I1.

I3 Inverter Group

The I3 inverter group feeds I2 and I7. I7 is used during exchange jumps. I3 is a four-way fan-in with inputs from the A operating registers, I0 (P and RA) and I10 (K on 30-bit instructions).

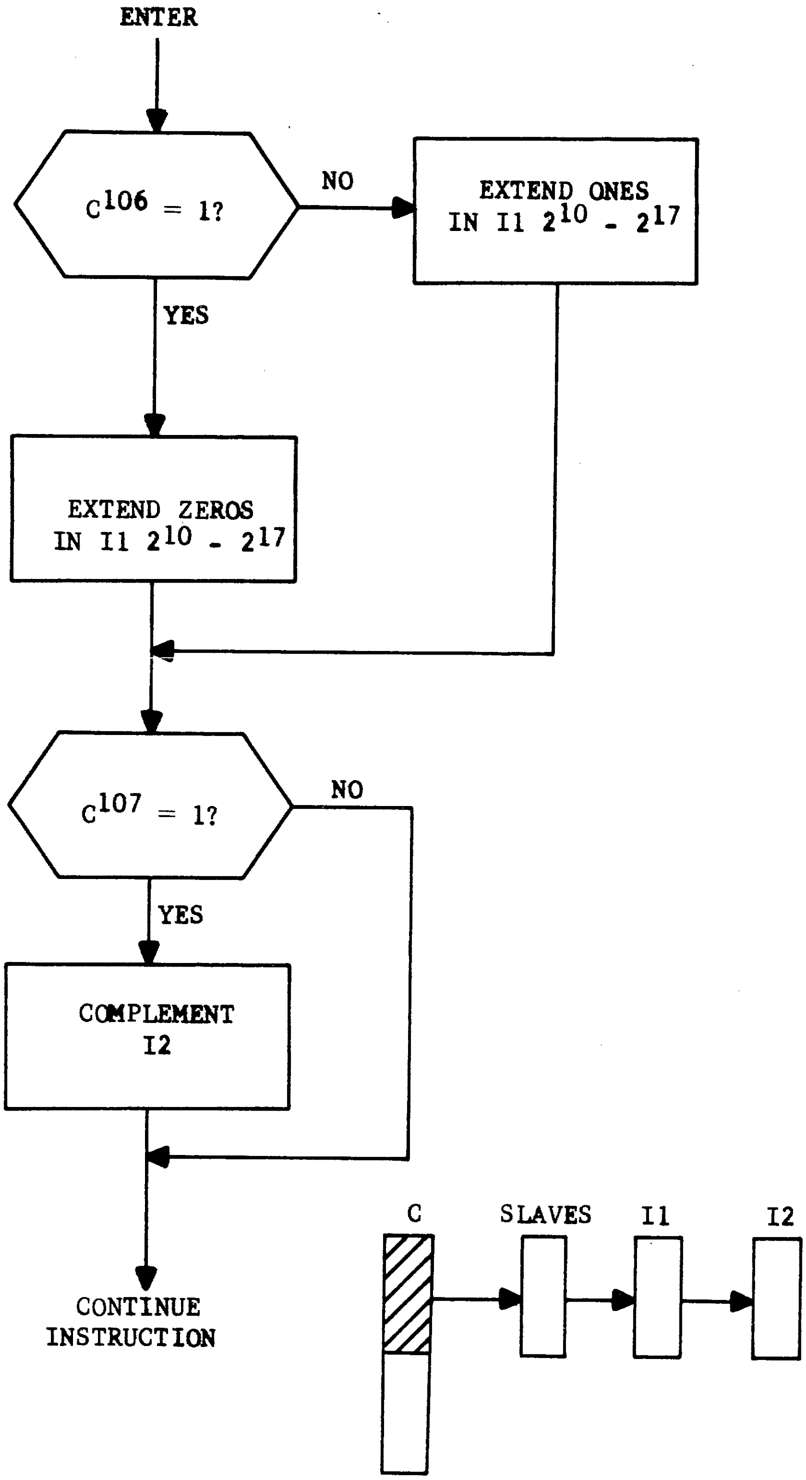

ENTER
$C^{106} = 1?$
NO
EXTEND ONES IN I1 $2^{10} - 2^{17}$
YES
EXTEND ZEROS IN I1 $2^{10} - 2^{17}$
$C^{107} = 1?$
NO
YES
COMPLEMENT I2
CONTINUE INSTRUCTION
C
SLAVES
I1
I2

Figure 4-42

E Register

The E register is 18 bits in size and serves as one of the feeder registers for the F adder. E register's only input is from I2. Outputs from E feed the first rank of the adder and the A and B operating registers, feeder networks.

F Register

The F register is the other feeder register for the F adder. F also serves as the output register for the results of the adder. Inputs to F are from Rank V of the adder, from I2 inverter group and from I9. The I9 input would be the shift count developed during normalization of a floating point coefficient and must be added or subtracted from the exponent. There is also the input of +1 as a constant necessary during advancing of P. The F register output also feeds I8. I8 is used for general distribution of data from the SAS.

F Adder

The F adder is the same as the D adder except for its size; the F adder, is one section of the D adder. Use the same explanation for the F adder as was used for the D adder, just remembering that there will be less complications.

I8 Inverter Group

I8 is a general distribution point for data in the SAS. Inputs are from the F adder and the Normalize Network. The input from the Normalize Network is used during a C ⟶ I4 transfer. The output of I8 distributes data to many points such as the FL checker, the exponent checker, and the P register.

Replacing the Bias in I8

I8 is the point where the bias will be replaced in the exponent before storing it. Remember that all manipulation on the exponents done in SAS is with the bias removed. Sometimes the bias is replaced inadvertently during manipulation of the exponents. The circuit merely toggles the bias bit (2^{10}), which may seem strange for some exponents, but it should be remembered that I5 can complement the entire bias and exponent.

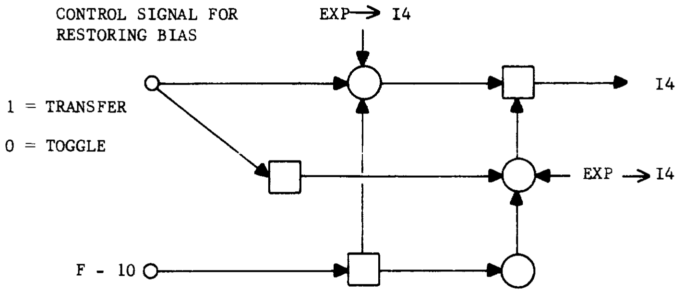


Figure 4-43

Output Area

All addresses and data pass through this area of the CPU. There are four basic sets of transmitters in this area. They are:

Data Transmitters (60)
ECS Address Transmitters (24)
Read P Transmitters (17)
Central Address Transmitters (17)

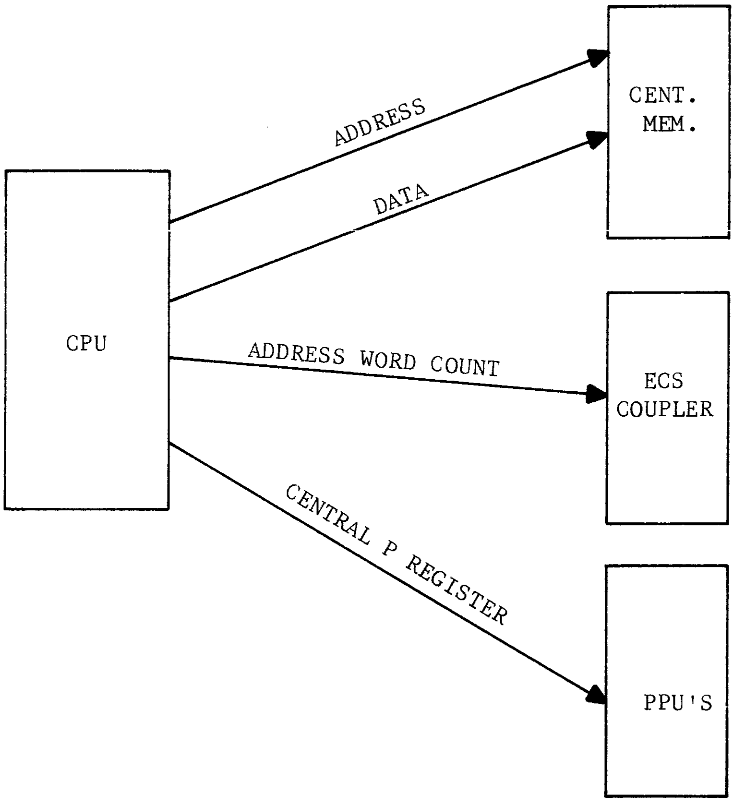


Figure 4-44

I6 and I7 inverter groups are the fan-in circuits that collect all the necessary data that will be sent by the data transmitters to Central Memory (usually on exchange jump). The Read P transmitters send the contents of the CPU P register to the Peripheral Processors unconditionally every 100 nanoseconds.

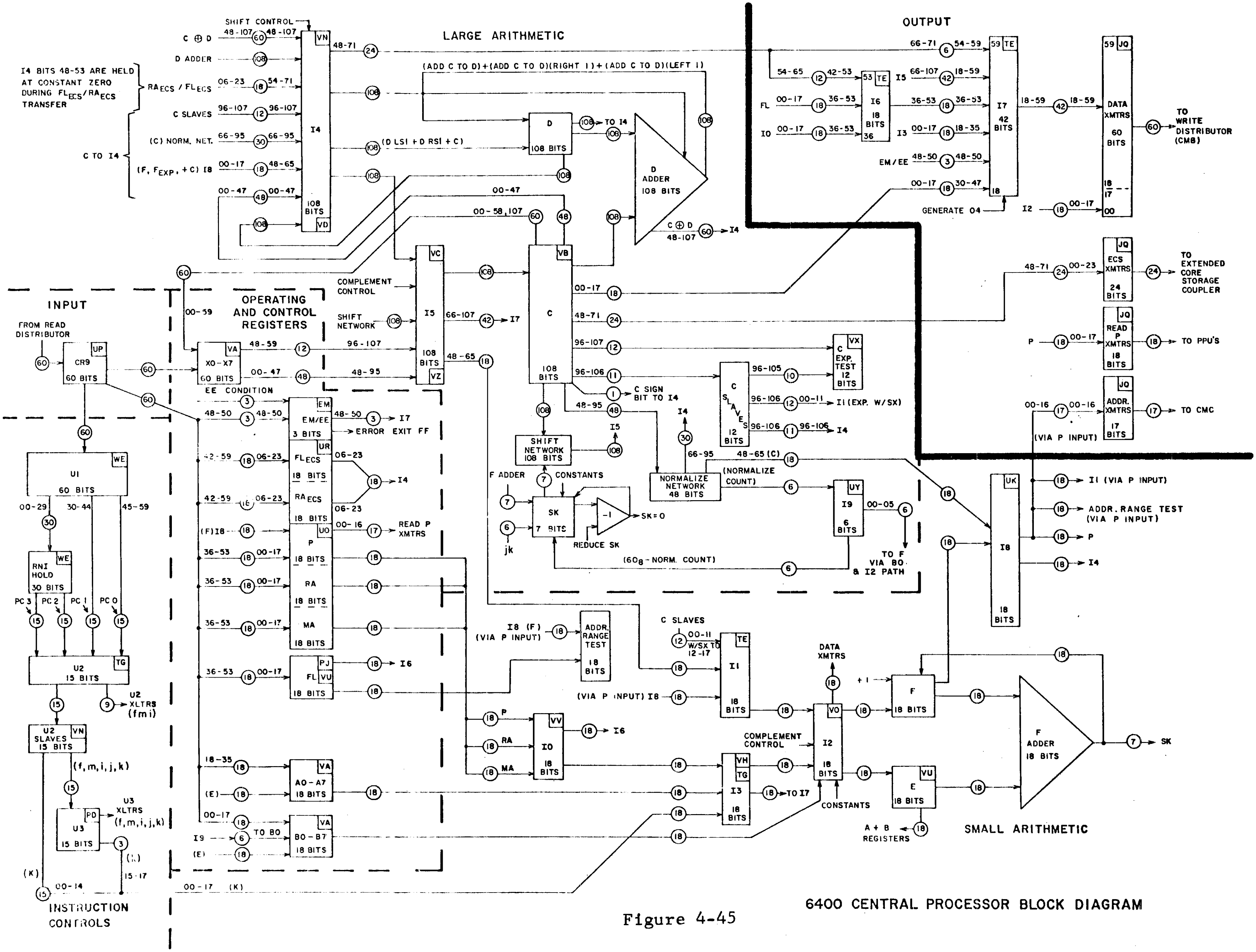


Figure 4-45

6400 CENTRAL PROCESSOR BLOCK DIAGRAM

CENTRAL PROCESSOR OPERATIONS

Execution of instructions in the Central Processor entails the operation on data through different areas of the circuitry under control of logical sequences. The logical sequences are initiated by decoding of the instructions as they come from memory.

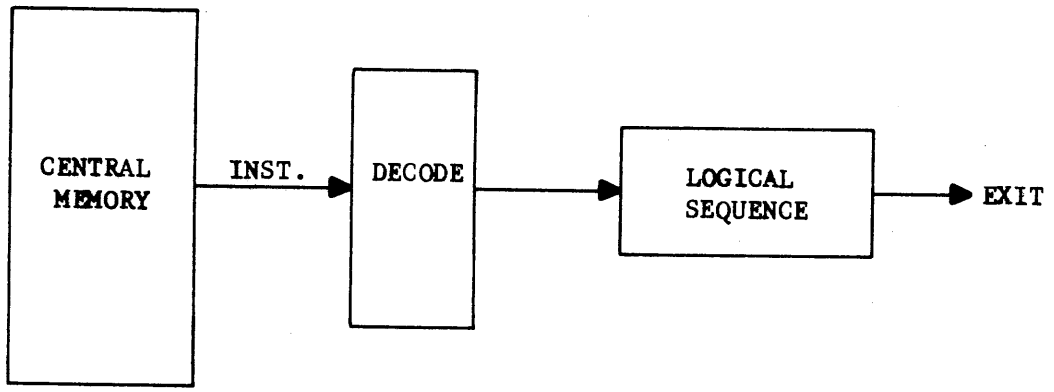


Figure 4-46. Sequence Initiation

The logical sequences are made up of flip-flops that will pass a signal along, according to timing signals and logical gates. Each flip-flop in the chain will cause a particular part of the needed controls to develop.

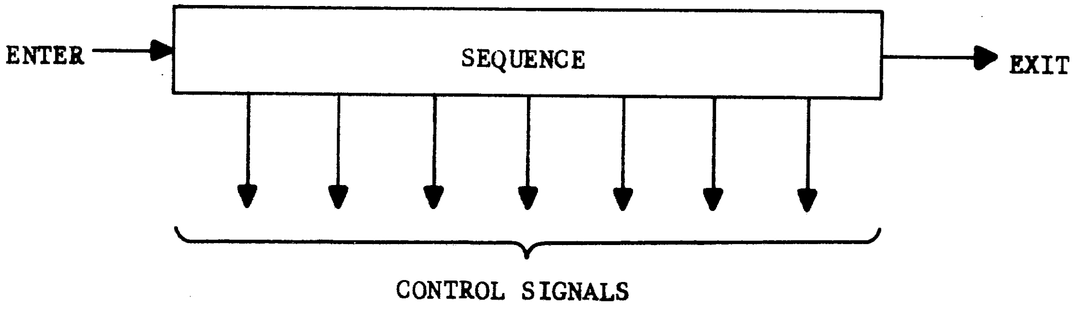


Figure 4-47

The command timing charts define the action of the controls for all the particular sequences.

In this section of the manual, discussions will look at the CPU theory of the operation but will not examine the logic to any detail. References will be made to the logic in order to aid the reader in following the operation through it. Following are some general comments concerning the Central Processor instruction set and a list of the instructions and their individual times.

GENERAL TIMING COMMENTS

The 6400 system Central Processor has a unified arithmetic unit. Instructions in the 6400 Central Processor, therefore, are executed in sequential fashion, with little concurrency. Factors influencing instruction execution time are summarized below.

1. The next instruction word is called up from Central Memory between the execution of the first and second instructions. This process takes two minor cycles to initiate (the remainder of the Read Next Instruction is parallel in time with the execution of the second instruction).

   Rule: Add two minor cycles for each instruction word in a program which does not have a jump, taken as the upper instruction.

2. All execution times listed are complete. The times include getting the next instruction ready to execute. The jump and return jump times include reading up and preparing to execute the new instruction word.

3. The return jump, jumps and load/store memory instructions pay a time penalty for being the second instruction of an instruction word. This penalty is caused by hardware limitations and is not due to memory bank conflicts.

INSTRUCTION	SECOND INSTRUCTION IN WORD TIME PENALTY
Jumps taken	1 minor cycle
Return jump	2 minor cycles
Load/store	2 minor cycles
Minimum execution of second instruction	8 minor cycles

4. If the second instruction references the same memory bank as (P + 1), there is an additional penalty of three minor cycles due to bank conflict.

5. A store (not load) as the first instruction of a word can cause a bank conflict with (P + 1). If this occurs, the penalty is three minor cycles.

The rules, then, for efficient coding in the 6400 include:

1. Put jumps in the upper parcel. This eliminates both the two minor cycle RNI penalty and the possibility of having a memory bank conflict with (P + 1).

2. Where possible, keep load/store instructions in the bottom two parcels.

3. Loads and stores in consecutive parcels will not cause memory conflicts with each other.

CENTRAL PROCESSOR INSTRUCTION EXECUTION TIMES

Central Processor instruction execution times for the 6400 system are given in Table 4-5.

Instruction execution times are listed in minor cycles, which is 100 nanoseconds in the 6400.

TABLE 4-5. INSTRUCTION EXECUTION TIMES: CENTRAL PROCESSOR

OCTAL CODE	BRANCH INSTRUCTIONS			6400
00	STOP			-
01	RETURN JUMP TO K			21
011	READ EXTENDED CORE STORAGE	[2]		
012	WRITE EXTENDED CORE STORAGE			
02	GO TO K + Bi[1]		[3]	13
030	GO TO K if Xj = zero	[4]		13
031	GO TO K if Xj $\neq$ zero			13
032	GO TO K if Xj = positive			13
033	GO TO K if Xj = negative			13
034	GO TO K if Xj is in range			13
035	GO TO K if Xj is out of range			13
036	GO TO K if Xj is definite			13
037	GO TO K if Xj is indefinite			13
04	GO TO K if Bi = Bj	[1]		13
05	GO TO K if Bi $\neq$ Bj			13
06	GO TO K if Bi $\geq$ Bj			13
07	GO TO K if Bi $<$ Bj			13

[1] GO TO K + Bi and GO TO K if Bi - - - tests made by Increment Instruction

[2] Execution times for Extended Core Storage operations are dependent upon several factors; refer to Extended Core Storage literature for timing information

[3] Jumps in which the jump condition is not met require 5 minor cycles

[4] GO TO I if Xj - - - tests made by Long Add Instruction

OCTAL		
CODE	BOOLEAN INSTRUCTIONS	6400
10	TRANSMIT Xj to Xi	5
11	LOGICAL PRODUCT of Xj and Xk to Xi	5
12	LOGICAL SUM of Xj and Xk to Xi	5
13	LOGICAL DIFFERENCE of Xj and Xk to Xi	5
14	TRANSMIT Xk COMP. to Xi[5]	5
15	LOGICAL PRODUCT of Xj and Xk COMP. to Xi	5
16	LOGICAL SUM of Xj and Xk COMP. to Xi	5
17	LOGICAL DIFFERENCE of Xj and Xk COMP. to Xi	5

OCTAL		
CODE	SHIFT INSTRUCTIONS	6400
20	SHIFT Xi LEFT jk places	6
21	SHIFT Xi RIGHT jk places	6
22	SHIFT Xk NOMINALLY LEFT Bj places to Xi	6
23	SHIFT Xk NOMINALLY RIGHT Bj places to Xi	6
24	NORMALIZE Xk in Xi and Bj	7
25	ROUND AND NORMALIZE Xk in Xi and Bj	7
26	UNPACK Xk to Xi and Bj	7
27	PACK Xi from Xk and Bj	7
43	FORM jk MASK in Xi	6

OCTAL		
CODE	ADD INSTRUCTIONS	6400
30	FLOATING SUM of Xj and Xk to Xi	11
31	FLOATING DIFFERENCE of Xj and Xk to Xi	11
32	FLOATING DP SUM of Xj and Xk to Xi[5]	11
33	FLOATING DP DIFFERENCE of Xj and Xk to Xi	11
34	ROUND FLOATING SUM of Xj and Xk to Xi	11
35	ROUND FLOATING DIFFERENCE of Xj and Xk to Xi	11

OCTAL		
CODE	LONG ADD INSTRUCTIONS	6400
36	INTEGER SUM of Xj and Xk to Xi	6
37	INTEGER DIFFERENCE of Xj and Xk to Xi	6

OCTAL		
CODE	MULTIPLY INSTRUCTIONS	6400
40	FLOATING PRODUCT of Xj and Xk to Xi	57
41	ROUND FLOATING PRODUCT of Xj and Xk to Xi	57
42	FLOATING DP PRODUCT of Xj and Xk to Xi	57

[5] Comp. = Complement; DP = Double Precision

OCTAL CODE	DIVIDE INSTRUCTIONS	6400
44	FLOATING DIVIDE Xj by Xk to Xi	56
45	ROUND FLOATING DIVIDE Xj by Xk to Xi	56
47	SUM of 1's in Xk to Xi	68
46	PASS	3

OCTAL CODE	INCREMENT INSTRUCTIONS		6400
50	SUM of Aj and K to Ai	[6]	6
51	SUM of Bj and K to Ai		6
52	SUM of Xj and K to Ai		6
53	SUM of Xj and Bk to Ai		6
54	SUM of Aj and Bk to Ai		6
55	DIFFERENCE of Aj and Bk to Ai		6
56	SUM of Bj and Bk to Ai		6
57	DIFFERENCE of Bj and Bk to Ai		6
60	SUM of Aj and K to Bi		5
61	SUM of Bj and K to Bi		5
62	SUM of Xj and K to Bi		5
63	SUM of Xj and Bk to Bi		5
64	SUM of Aj and Bk to Bi		5
65	DIFFERENCE of Aj and Bk to Bi		5
66	SUM of Bj and Bk to Bi		5
67	DIFFERENCE of Bj and Bk to Bi		5
70	SUM of Aj and K to Xi		6
71	SUM of Bj and K to Xi		6
72	SUM of Xj and K to Xi		6
73	SUM of Xj and Bk to Xi		6
74	SUM of Aj and Bk to Xi		6
75	DIFFERENCE of Aj and Bk to Xi		6
76	SUM of Bj and Bk to Xi		6
77	DIFFERENCE of Bj and Bk to Xi		6

[6] Times listed are when i = 0. When i = 1-5, the execution time is 12 minor cycles; with i = 6 or 7, 10 minor cycles is the execution time.

MULTIPLY OPERATION

There are three multiply instructions in the central processor instruction set. They are:

40 Floating Product of Xi and Xk to Xi (15 bits)

This instruction multiplies two floating point quantities located in operand registers j (multiplier) and K (multiplicand) and packs the upper product result in operand register i.

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.

41 Round Floating Product of Xj and Xk to Xi (15 bits)

This instruction attaches a round bit to the floating point number in operand register k (multiplicand), multiplies this number by the floating point number in operand register j, and packs the upper product result in operand register i. (No lower product available.)

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.

42 Floating DP Product of Xj and Xk to Xi (15 bits)

This instruction multiplies two floating point quantities located in operand registers j and k and packs the lower product in operand register i. The result is not necessarily a normalized quantity.

The multiplication of two floating point operands involves the multiplication of the coefficients and the addition of the exponents. The multiply process will be handled in the LAS and the exponent manipulation will be done in the SAS. In the 6400 CPU we can have a floating point coefficient of 48 bits. The remaining twelve bits of the 60-bit memory word are the signed exponent and the sign of the coefficient.

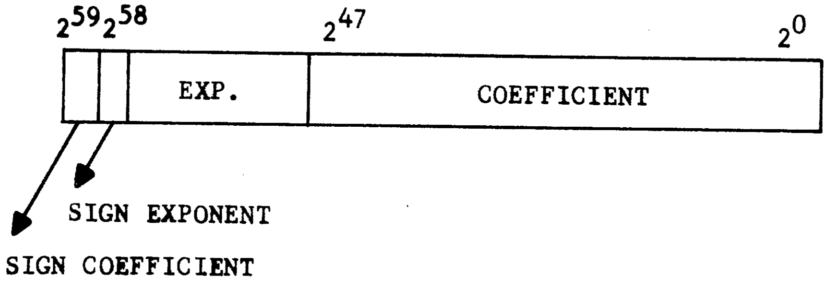


Figure 4-50

With a coefficient of this size, the development of a 96-bit product results. Since only 60 bits can be stored as an answer, a decision to use either the upper 48 or the lower 48 bits must be made. With the instructions available, a selection of either the upper or the lower half may be made except for rounded multiply.

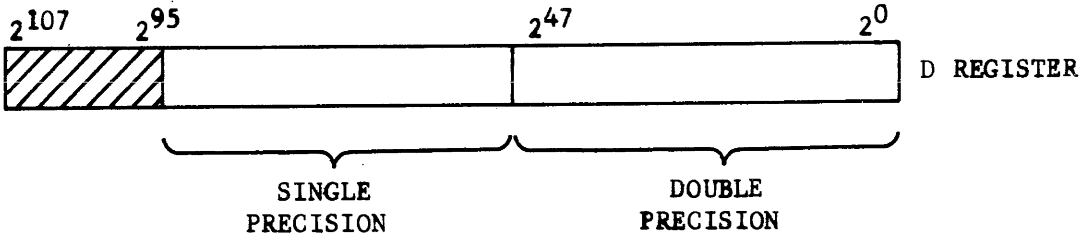


Figure 4-49

The location of the binary point is to the right of bit 2^0 in the result register. (The binary point for each of our original coefficient was also to the right of bit 2^0.) This means that all numbers are considered integers rather than fractions. The exponent developed in the SAS is relative to the complete 96-bit answer; therefore, double precision uses this exponent unchanged whereas single precision exponents must be plus 60 to reconcile the right shift necessary to move the binary point below the 2^{48} bit. (Effectively shift the number right 48. This puts the 2^{48} bit position in the 2^0 position, which now places the binary point just to the right of the answer.)

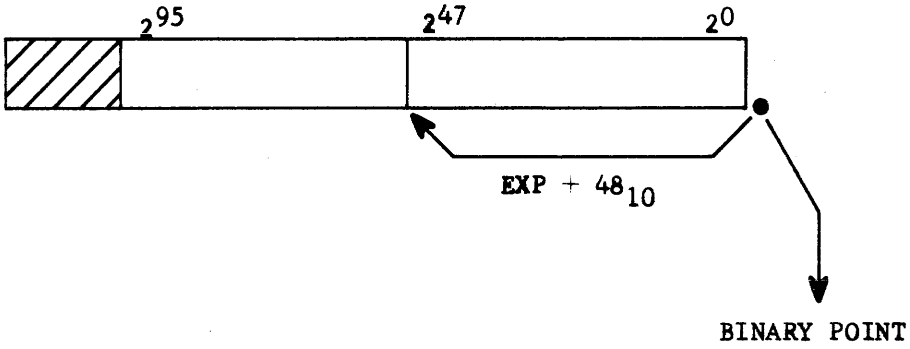


Figure 4-50

With the execution of a multiply instruction, the first floating operand will be selected from the specified X operating register. In the C register the 2^{107} bit will be checked. (Remember the transfer, Select Xj, places the contents of X into positions 48 - 107 of the C register.) If bit 107 is a "1", sign record (XjSR) will be set to indicate that the coefficient is negative. The setting of XjSR causes the entire coefficient and the exponent to become complemented before any operation is performed. XjSR also causes recomplementing of the final answer before it is stored (IF $\overline{\text{XkSR}}$).

The second operand (Xk) is now transferred to C where its sign will be checked in the same manner. (XkSR)

The Xj coefficient is selected from the X register for the second time, if the first transfer into C found the coefficient to be negative. This second transfer allows the coefficient to be complemented while it passes through I5. When the coefficient, Xj, is in the C register for the second time, it is checked for normalization by comparing position 2^{95} (upper bit of coefficient) with the XjSR. If the bits are unequal, the Both Operands Normalized (BON) flip-flop sets. If the two bits are unequal ($C^{95} \neq$ XjSR), it means that the sign bit is "0" and $C^{95} = 1$ (indicating a normalized positive coefficient) or that the sign bit (XjSR) is "1" and $C^{95} = 0$ (indicating a negative normalized exponent).

The same process is involved in the selecting Xk coefficient from the X registers for the second time. When Xk is in C for the second time, Sj will already have moved to the D register via the Shift Network, which will shift it right 60_8. The check for the Xk normalized situation is made during the second transfer and, according to the results, the BON flip-flop is cleared if Xk is not normalized and set if it is normalized.

At this time, the two coefficients are in place and ready for the multiply process to start.

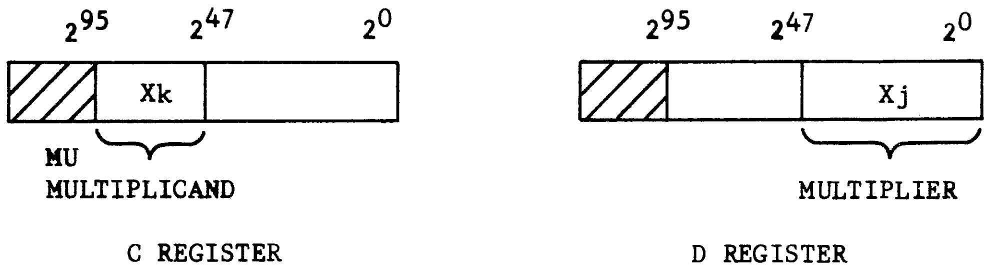


Figure 4-51

The process of multiplying is one of shifting and adding or just shifting according to the one bits in the multiplier. During the process, the C and D registers, the D adder, and the I4 inverter group are used. The SK register becomes set to 48_{10} to keep track of the number of iterations. Each pass reduced SK by one. A 1-bit register, called the D Flag, catches each bit of the multiplier as it is shifted, to determine whether an add is needed.

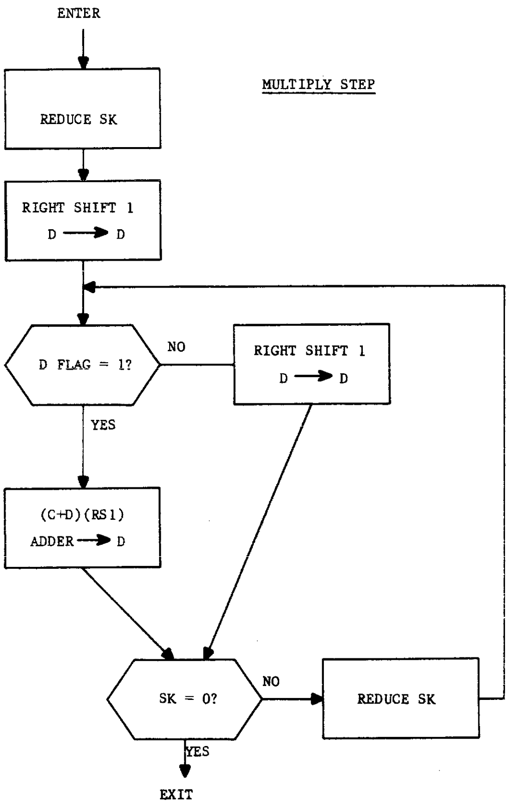


Figure 4-52

During the multiply operation the multiplier is shifted right into the D Flag. If the flag sets, the output of the adder is sent to D, shifted right one position. If the D Flag did not set, the output of the D register is sent through I4 (I4 will shift right one) and back to D. After the first iteration, the D register holds the partial product and the multiplier. The multiplicand remains in the C register unchanged throughout the operation. To visualize the operation of the multiply, the following illustration is used. For conservation of time and space, 3-bit coefficients will be used. Assume the C and D registers are 10 bits in size for this example.

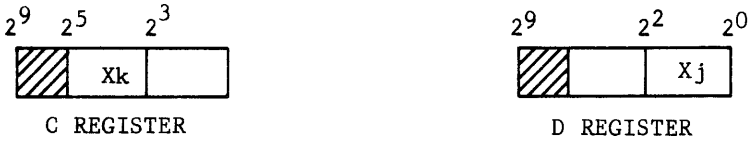
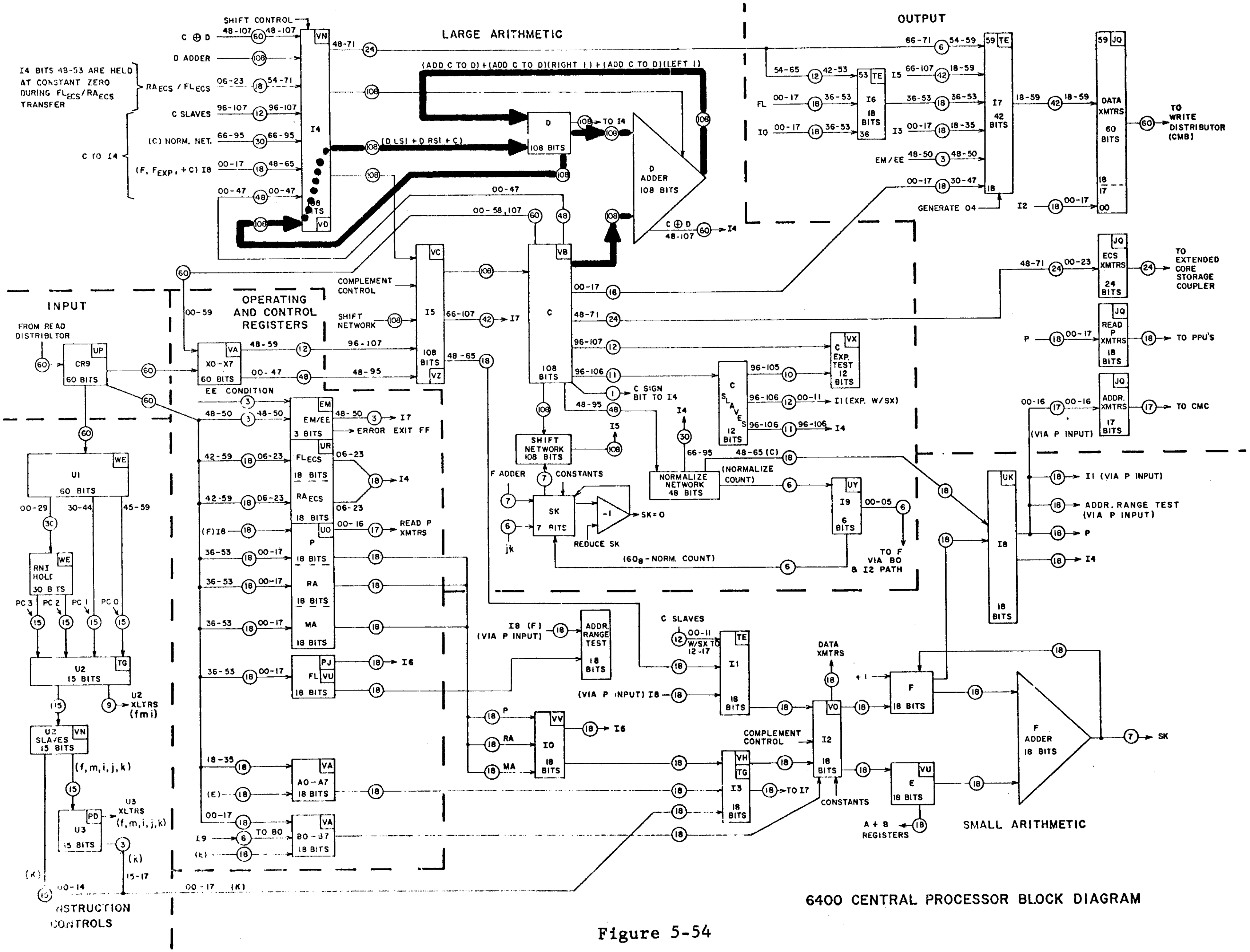


Figure 4-53

During the first iteration, the D register must shift right one position to load the D Flag, which determines the following operation. Assume that the computer is multiplying 5_8 x 4_8 = 24_8.



6400 CENTRAL PROCESSOR BLOCK DIAGRAM

Figure 5-54

MULTIPLY EXAMPLE

	C (MULTIPLICAND)	D (MULTIPLIER)	D FLAG
	0000100000	0000000101	
Right Shift 1D → D		0000000010	1
Add C + D and RS1 → D		0000010001	0
Right Shift 1D → D (D Flag = 0)		0000001000	1
Add C + D and RS1 → D		0000010100	

Figure 4-55

In this example, development of the final answer in the D register can be seen. After the first interation, the D register holds the partial product and the remaining bits of the multiplier. After the last iteration, only the final product remainsc since the multiplier has been shifted completely out of the register.

Some consideration must now be given to the form of the final answer. If either of the two original operands were not normalized, then the final answer will not necessarily be normalized. However, if they were both normalized, then a normalized answer is guaranteed (single precision only). This process of normalizing the result may include shifting of the final answer just one position different than the normal shift necessary before storing the results. Usually, most normalized operands automatically produce a normalized answer. However, where the smallest possible normalized operands are used, the answer may not be normalized. For example:

Xj Coefficient = 40————————0.

Xk Coefficient = 40————————0.

Answer = 20————————0.

In this example, the final answer is not normalized. When the right shift is performed which places the final answer (single precision) in the proper position for storing (remember that a single precision answer is in D^{95} - D^{48} but can only be stored in X from C^{48} - C^{00}), shifting will be one position less than usual. Normally, a right shift of 48 is performed; however, the shift now will be 47. This difference is made up in the exponent.

During double precision operations, normalized answers can never be guaranteed, even if the two original coefficients were normalized. However, when the computer executes a DP multiply and the coefficients are normalized, the whole 96-bit register is still considered and is normalized before the DP results are taken. This means a left shift of 1 is necessary to normalize the register. This difference, again, would be taken up in the exponent.

The manipulation of the exponent starts as soon as the first exponent (Xj) is sent to SAS. The bias is removed and the sign of the exponent is extended the full 18 bits of I1. In I2 the exponent may be complemented if XjSR is set. Later, the second exponent will arrive from LAS and the same process will occur. Once the two exponents have been placed in the E and F feeder registers, they are added to form the exponent of the 96-bit product. Let's assume the following exponents:

2001 Xj
2002 Xk

I1	000001
I1	000002
After Add	000003

Before being stored as part of the final answer, the exponent must have its bias replaced. This is done in I8:

I8 output 002003

There are certain possible adjustments which could be made to the exponent. For instance, if Single Precision Multiply was executed, 47_{10} or 48_{10} must be added to the exponent to adjust for the binary point.

Remember, whether 47_{10} or 48_{10} depends on the need or not the need for normalized answers. If double precision is executed, one can be subtracted from the exponent for original operands which were normalized but produced a 96-bit product which was not normalized. This subtraction of one compensates for the left shift of one. (Left shift increases the coefficient and, therefore, decreases the exponent.) Normally, the unadjusted exponent in SAS represents the 96-bit product and, therefore, the DP results.

Once the exponent has been adjusted, it is sent back to LAS. The path would be I8 ⟶ I4 ⟶ I5 ⟶ C. In the C register the exponent is re-united with the coefficient and the complete answer is stored in the X registers.

The Round Floating Multiply works in the same manner as was already described, except that a round bit is appended to the multiplicand before the first iteration.

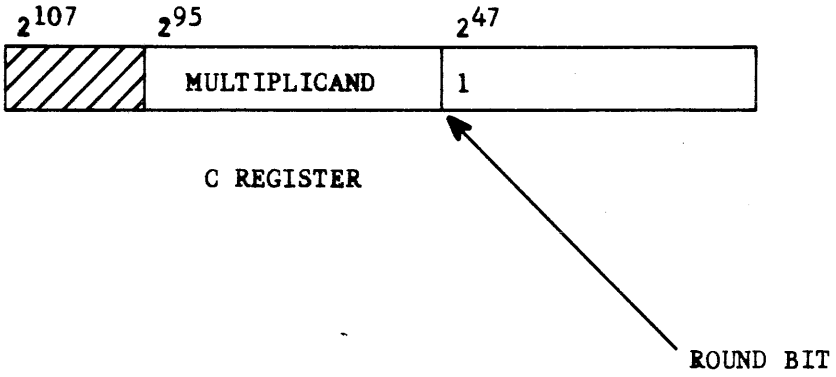


Figure 4-56

Effectively, this round bit is added in on each iteration where a "1" bit was present in the multiplier. The end result is to round the final product.

DIVIDE OPERATION

There are two divide instructions in the CPU set. They are discussed in the following topics.

44 Floating Divide Xj by Xk to Xi (15 bits)

This instruction divides two floating point quantities located in operand registers j (dividend) and K (divisor) and packs the quotient in operand register i.

The exponent of the result, if no overflow occurs, 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. The resultant exponent is the difference between the dividend and divisor exponents minus 47.

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

45 Round Floating Divide Xj by Xk to Xi (15 bits)

This instruction divides the floating quantity in operand register j (dividend) by the floating point quantity in operand register K (divisor) and packs the rounded quotient in operand register i. A 1/3 round bit is added to the least significant bit of the dividend before division starts.

The resultant exponent, if no overflow occurs, is the difference between 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. The resultant exponent is the difference between the dividend and divisor exponents minus 47. The result is a normalized quantity when both dividend and divisor are normalized.

The following general comments should aid in the understanding of the divide operation.

1. A normalized answer always results if both original operands are normalized.

2. A divide fault occurs if two operands are divided where the divisor   1/2 dividend (operands un-normalized).

3. If both operands are un-normalized, the result is not necessarily normalized.

The divide operation can be considered in two parts, just as for the multiply. The first phase considered is the preparation of the two operands; the second phase is the actual divide process.

The preparation of the operands includes the extraction of the exponents from each operand and sending them to the SAS where the final exponent will be computed. Also in preparation, the two coefficients will be checked for normalization and sign. This is done in the same manner as was done in the multiply. There is, however, one difference during preparation of the two divide coefficients that must be considered: a subtract is required during the divide process, rather than an add. This means that the divisor (Xk) must be in complement form ($D + \overline{C}$) and will, therefore, be complemented in I5, unless it was found to be negative ($C^{107} = 1$). If it is negative, complementing is omitted. After the preparation process, the dividend and divisor are located in the C and D registers, as illustrated in Figure 4-57.

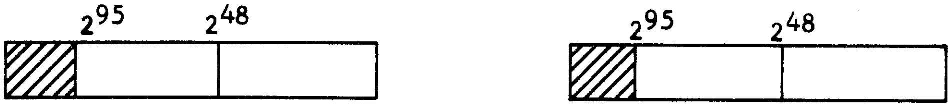
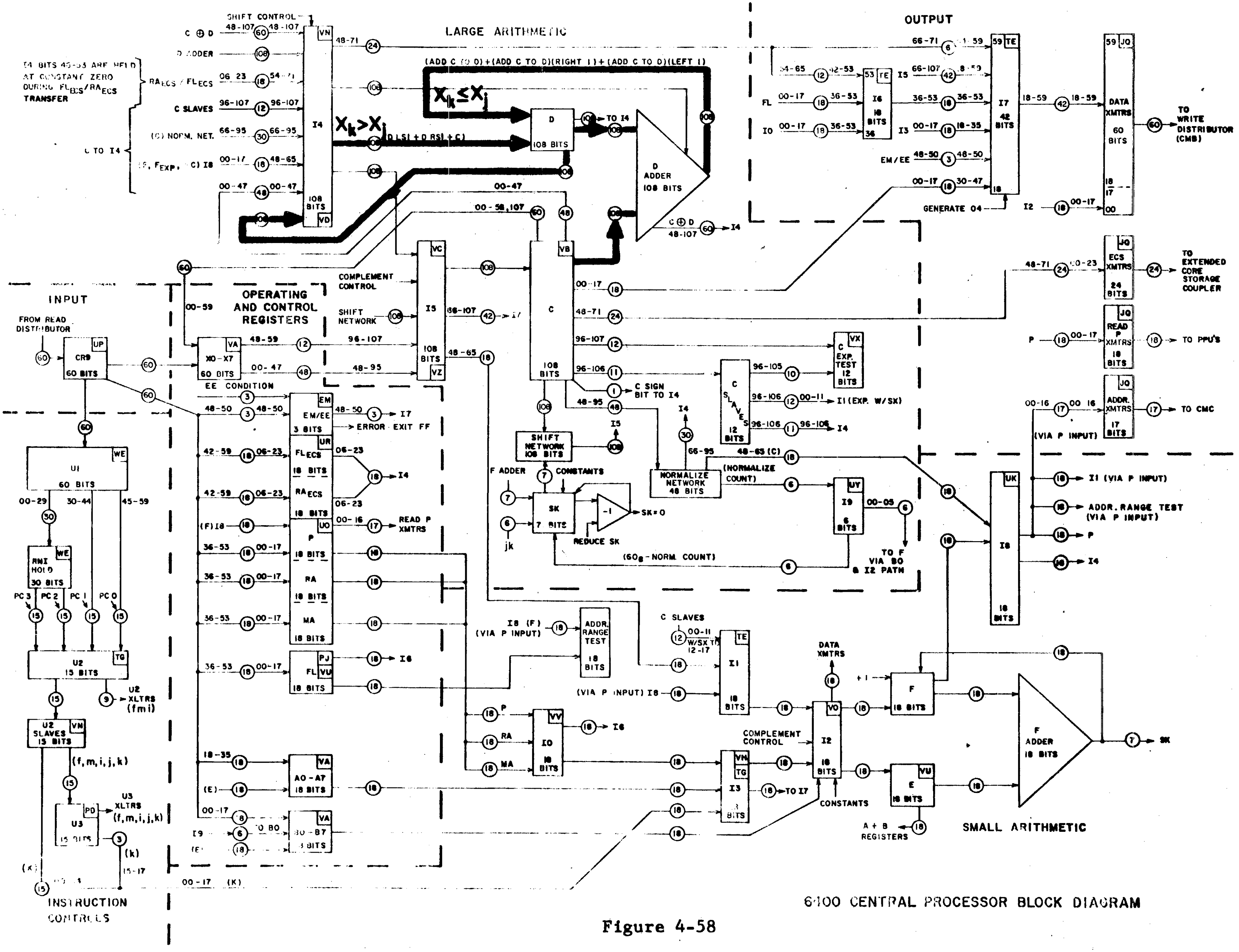


Figure 4-57



6400 CENTRAL PROCESSOR BLOCK DIAGRAM

Figure 4-58

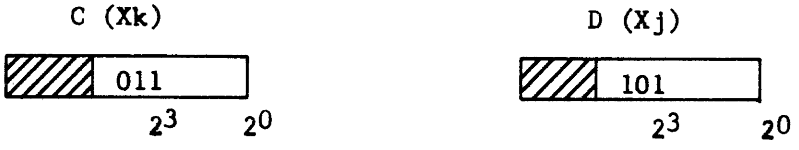
The elements of the LAs used during the divide are the C and D registers, the D adder, and the I4 inverter group. The SK register becomes set to 48_{10}, to establish the count of iterations.

The process of dividing is to subtract the divisor from the dividend to determine if it is smaller; if it is, a "1" bit is set into the partial quotient and the new dividend is shifted left one. (This has the effect of positioning the divisor so that the second subtract will be from a larger dividend.) If the subtraction of the divisor from the dividend does not find the divisor smaller, a zero is forced into the partial quotient and shift the dividend left shifted one. (Compare this method to paper and pencil division to better understand the theory.) This process of subtracting and shifting or just shifting continues until 48 iterations have been made.

The following illustration is offered to aid in the understanding of the theory. Assume 3-bit operands, to conserve space and time.

DIVIDE 5/4

The C and D registers hold the dividend and divisor. C and D feed the adder and determine the results of the first subtract.



ADDER OUTPUT (001)(CARRY)

Since there is an End Around Carry, the divisor must have been smaller; therefore, a "1" bit is forced into the partial quotient (lower part of D). This bit is developed by setting the lowest bit in D at the same time the shifted output of the adder feeds D.

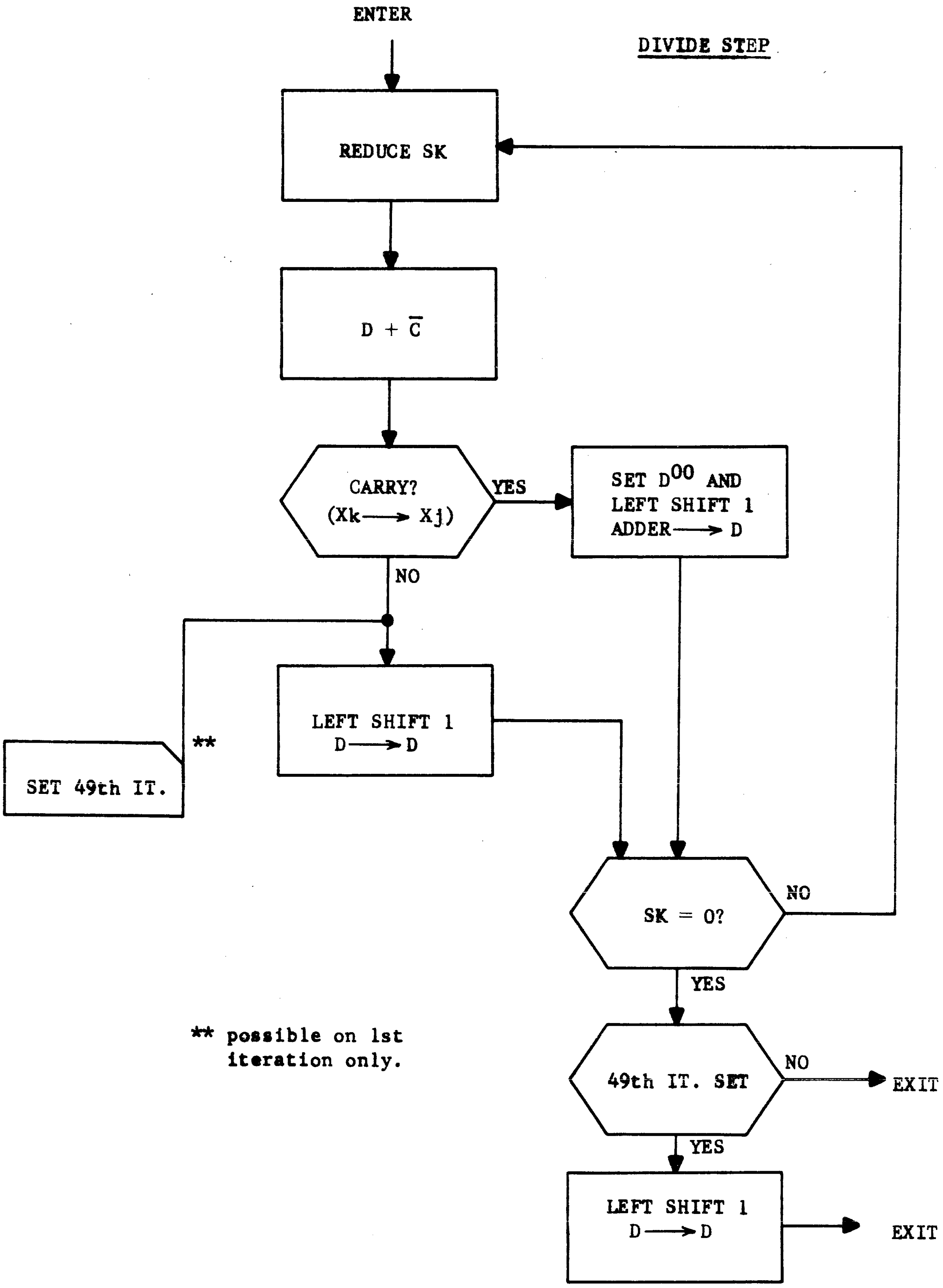

DIVIDE STEP

Figure 4-59

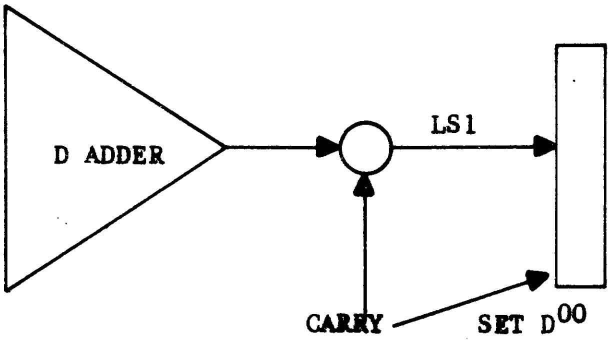
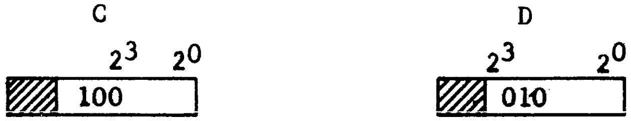


Figure 4-60

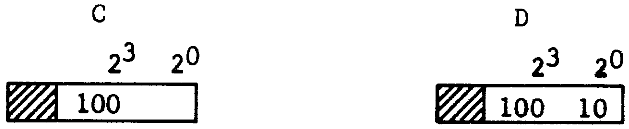
At the completion of this first iteration, the C and D registers look like:



C and D are again fed into the adder.

Adder Results (101)($\overline{\text{Carry}}$)

This time there is no End Around Carry (divisor > dividend) and, therefore, a zero is forced into the partial quotient. This is done by shifting the old dividend (the present contents of the D register) left one place (via I5). After the second iteration, the registers look like:



C and D feed the adder for the last iteration.

Output of Adder (111)(Carry)

This time an End Around Carry is generated (remember that "all pass" in the D adder is equivalent to a carry); therefore, another "1" bit is placed into the quotient. The final quotient becomes:

101_2

This answer does not mean much until the binary point is considered. Upon completion of the divide process, the binary point can be considered to be between bits 2^{46} and 2^{47}. This means the answer could be 1.X——X or 0.1X——X (consider normalized operands). With the preceding example, the answer would be: 1.2_8. However, this is not a floating point answer but an integer one instead. The binary point must be shifted to represent the number as an integer. This means that the final coefficient would be 5. Likewise, considering the full 48 bits of a possible answer, if the result was 1.X——X, the difference in the exponent must be made up by subtracting 47 from the exponent, rather than 48.

Another situation which must be considered is obtaining an un-normalized result after dividing two normalized coefficients. For instance, divide 4/5. The result would be 0.1X——X because the first subtract did not find the divisor smaller than the dividend and, therefore, formed a zero bit first in the partial quotient. To get a normalized answer in this instance, the 49th Iteration flip-flop is set. This 49th iteration is one extra left shift of the quotient, which effectively transfers the "1" bit from 2^{46} and 2^{47}. The binary point is now between position 2^{47} and 2^{48}. To represent the final answer as an integer, 48 must be subtracted from the exponent.

Considering this same example using the full 48-bit quantities; remember that each coefficient is placed in a 108-bit register C and D. Consider the following equation for the adder D + $\overline{C}$. 5/4

$\overline{C}$ divisor (Xk)	7 ——— 37 ——— 7
D dividend (Xj)	0 ——— 050 ——— 0
Adder output	0 ——— 010 ——— 0
Left shift into D	0 ——— 020 ——— 0
EAC = set D^{00}	0 ——— 020 ——— 1
$\overline{C}$ + D = adder output	7 ——— 060 ——— 0
$\overline{EAC}$ → left shift old dividend → D	0 ——— 040 ——— 2
$\overline{C}$ + D = adder output	0 ————————— 2
Left shift adder → D	0 ————————— 4
EAC → set D^{00}	0 ————————— 5

Since the remaining bits of the dividend are all zeros, we can discontinue representing the remaining iterations. Of course, these iterations will take place and the final result will be:

D final 50 ———— 0

Considering the binary point between positions 2^{46} and 2^{47}, the answer is 1.20———0_8. To represent this number as an integer, the binary point is placed to the right of the 2^0 bit. This difference is made up in the exponent by subtracting 47.

To summarize computation of the exponent, consider the following points.

When two coefficients are divided, the exponents (without bias and sign extended) are added and, depending on the 49th Iteration flip-flop, 47 or 48 are subtracted from the result. The 49th Iteration FF is set any time the divisor does not fit into the dividend on the first try; causing a leading zero. For two normalized coefficients, this leading zero would mean that the result is not normalized, unless shift left one more place. This extra left shift is made up in the exponent by subtracting 48_{10} from the exponent. If the 49th iteration is not made, 47 will always be subtracted from the exponent.

Rounded Divide

A rounded divide means that a round factor is inserted into the process which produces a more exact answer. The round bit is effectively 1/3 and is in the form of 25252525 added to the shifted dividend. Actually, the round bits are added one at a time. The original dividend was in the D register and will be shifted, either through the adder or through I4. As soon as the dividend has left shifted once, bit position 2^{48} is free. However, to add 2525..., an even count of the SK register (never the first count) is required. This means that when the SK register equals 46, the first round bit is added by setting bit 2^{48}. From time on, for every SK count, bit 2^{48} becomes set. The end result is that positions 2^{48} - 2^{95} of the D register (dividend) will have the round bits.

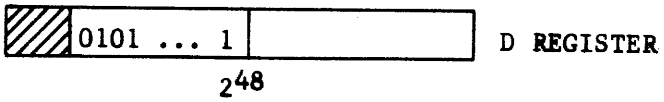


Figure 4-61

This round bit affects the final result by causing an End Around Carry into D^{00} (the partial quotient) if the digits after the binary point (after adjustment) were such as to allow it (7 2/3).

FLOATING ADD

30 Floating Sum of Xj and Xk to Xi (15 Bits)

This instruction forms the sum of the floating point quantities in operand registers j and k and packs the result in operand register i. 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 96-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 i.

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

32 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.

34 Round Floating Sum of Xj and Xk to Xi (15 Bits)

This instruction forms the round sum of the floating point quantities in operand registers j and k and packs the upper sum of the double precision result in operand register i. 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 situations.

A floating add operation involves the addition of two floating point coefficients which have equal exponents. Since equal exponents is unusual, the CPU equalizes the exponents before the add. This is done by right shifting the coefficient with the smaller exponent. A right shift decreases the size of the coefficient (moves the binary point left) and, therefore, the exponent is made larger. Once the exponents are equalized, the add is accomplished in the 108-bit D adder.

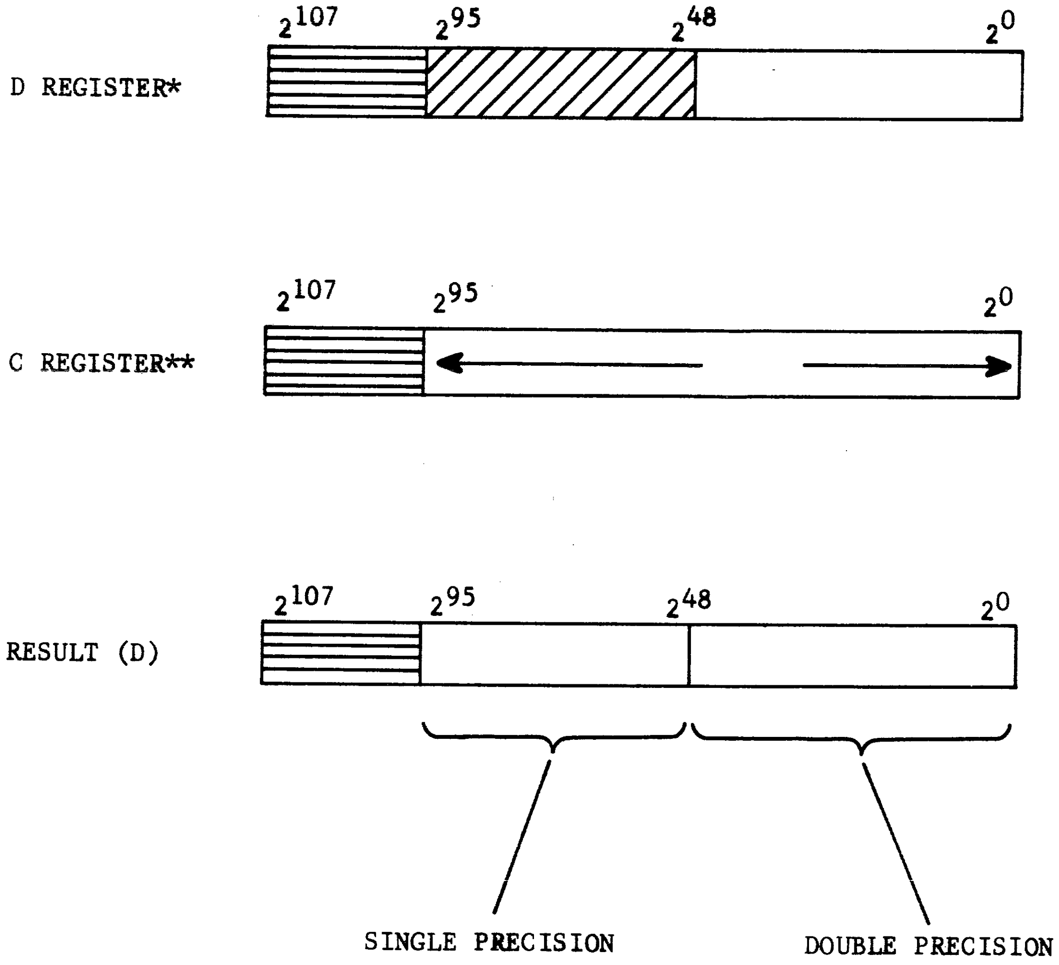
The binary point is considered to be between bit positions 2^{47} and 2^{48} of the 96-bit result register. Therefore, single precision add uses the results from position 2^{48} - 2^{95} and takes the computed exponent. Double precision takes the lower 48 bits of the D register and subtracts 48_{10} from the computed exponent to make up the difference in shifting of the binary point (result must be represented as an integer).

The first phase of a floating add operation involves selecting of the floating point operands from the X register and checking the sign of the coefficients. If the coefficients are negative, they will be complemented. This would be done in I5 when the coefficient is re-selected from the X registers. The exponent of each argument is sent to the SAS where the resultant exponent is computed.

The two exponents are equalized by subtracting one from the other. In I1, the bias of each exponent is removed and the sign extended. One of the exponents is complemented in order to perform a subtract. The two exponents are then sent to the F adder where their absolute difference is determined. The sign of the difference determines which exponent was the larger. Accordingly, an absolute value is sent to SK to be used to shift the coefficient of the smaller exponent. This coefficient is then sent to the Shift Network where it is right shifted the indicated number of places. Upon conclusion of this process, the two coefficients are in position in the C and D registers, ready for the add. The coefficient of the larger exponent will be in the D register, positions 2^{95} - 2^{48}. The shifted coefficient will be located, depending on the shift, somewhere in the C register.

The add is performed and, depending on whether single or double precision was selected, the results will be stored in the X register along with the proper exponent.

<u>FLOATING ADD</u>



* Coefficient of the larger exponent.

** Coefficient of the smaller exponent, can be found anyplace in the register depending on the number of right shifts necessary to equalize the exponents.

COMMENT SHEET

6400 CENTRAL PROCESSOR TRAINING MANUAL

Publication Number 60257200

FROM: Name: ______________________________

Address: ______________________________

COMMENTS: (Describe errors, suggested additions or deletions, and include page numbers, etc.)

CONTROL DATA INSTITUTES

Arlington, Virginia 22204
3717 Columbia Pike

Atlanta, Georgia 30327
500 Interstate North

Dallas, Texas 75235
Suite 224 Garden Mall
Frito-Lay Building

Frankfurt, Germany
Bockenheimer Landstrasse 10
6 Frankfurt/Main

Los Angeles, California 90045
5630 Arbor Vitae Street

Miami, Florida 33131
174 East Flaglar

Minneapolis, Minnesota 55408
3255 Hennepin Avenue South

Rockville, Maryland 20952
11428 Rockville Pike

St. Louis, Missouri 63108
Des Peres Hall
3694 W Pine Street

Southfield, Michigan 48075
23775 Northwestern Highway

Waltham, Massachusetts 02154
60 Hickory Drive

New York, New York 10011
Control Data Computer Training School
66 West 12 Street

San Francisco, California 94102
760 Market Street, Suite 600

Oakland, California 94612
508 16th Street

Chicago, Illinois 60603
37 South Wabash

CONTROL DATA
CORPORATION