

OBSOLETE

SITE COPY = CONTROL DATA ENGINEERS
UNIVERSITY OF ADELAIDE

6000 TRAINING SUPPLEMENT

PRELIMINARY EDITION

6000 TRAINING SUPPLEMENT

PRELIMINARY EDITION

FOR TRAINING PURPOSES ONLY

This book was compiled and
written by members of the
instructional staff of

CONTROL DATA INSTITUTE
CONTROL DATA CORPORATION

Publication No. 011568

January, 1968

Copyright 1968, Control Data Corporation
Printed in the United States of America

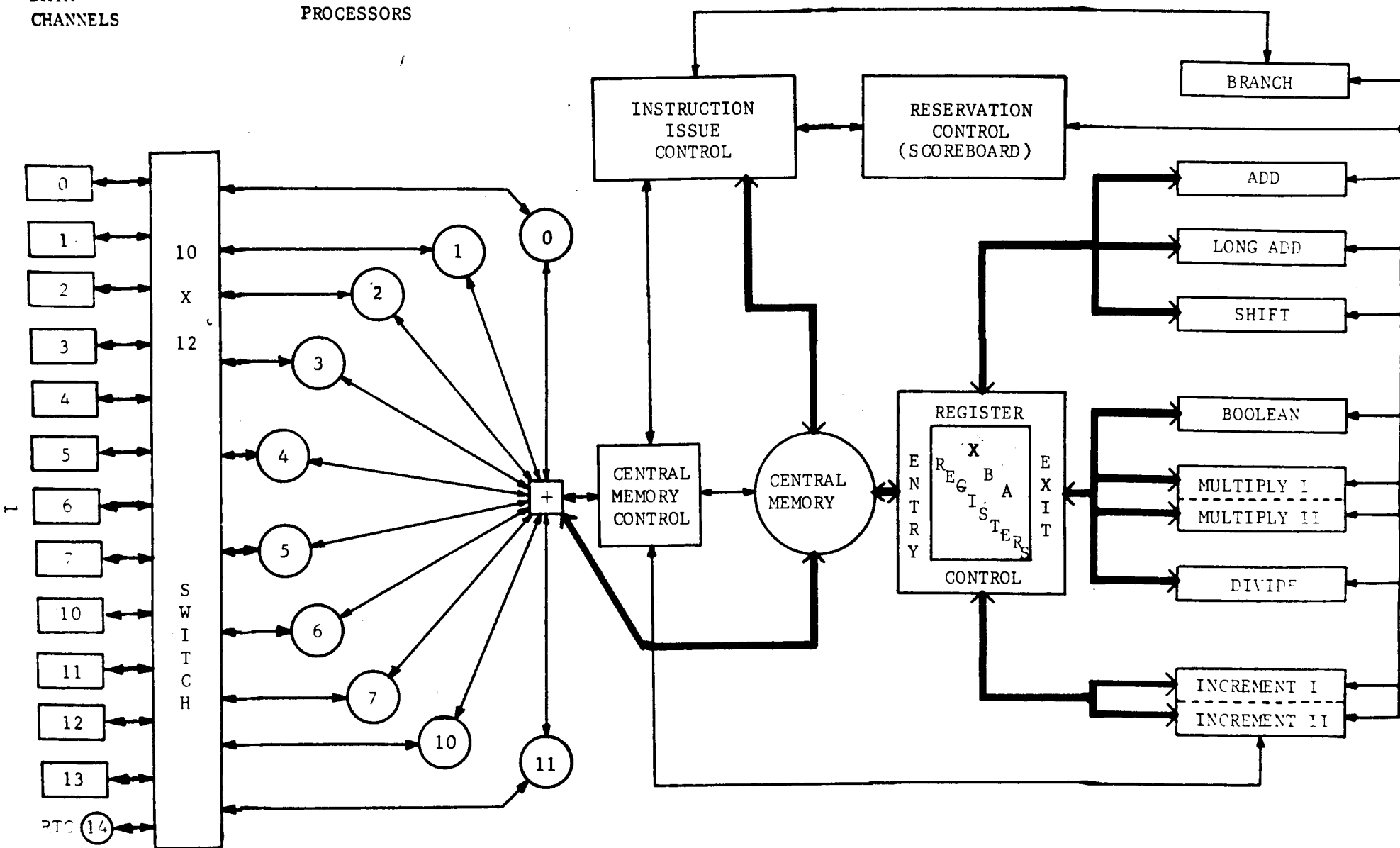
FOREWORD

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.

DATA CHANNELS

PERIPHERAL PROCESSORS

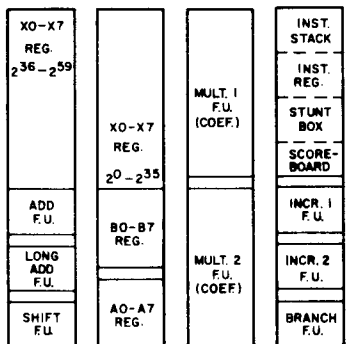
CENTRAL PROCESSOR



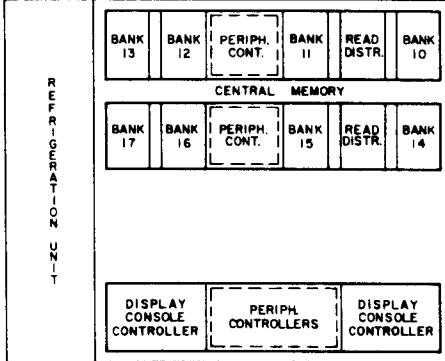
A GENERAL AND OVERALL 6600 COMPUTER BLOCK DIAGRAM

WING 2

REFRIGERATION UNIT

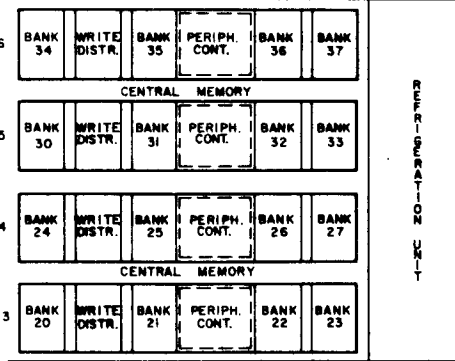


CH. 8 CH. 7 CH. 6 CH. 5

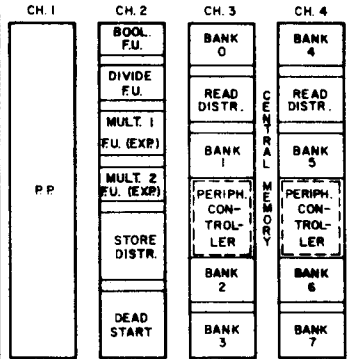


CH. 9 CH. 10 CH. 12

INTERCHASSIS CABLES
(37 LOGIC & 1 POWER/CH. MAX.)



CH. 16 CH. 15 CH. 14 CH. 13




REFRIGERATION UNIT

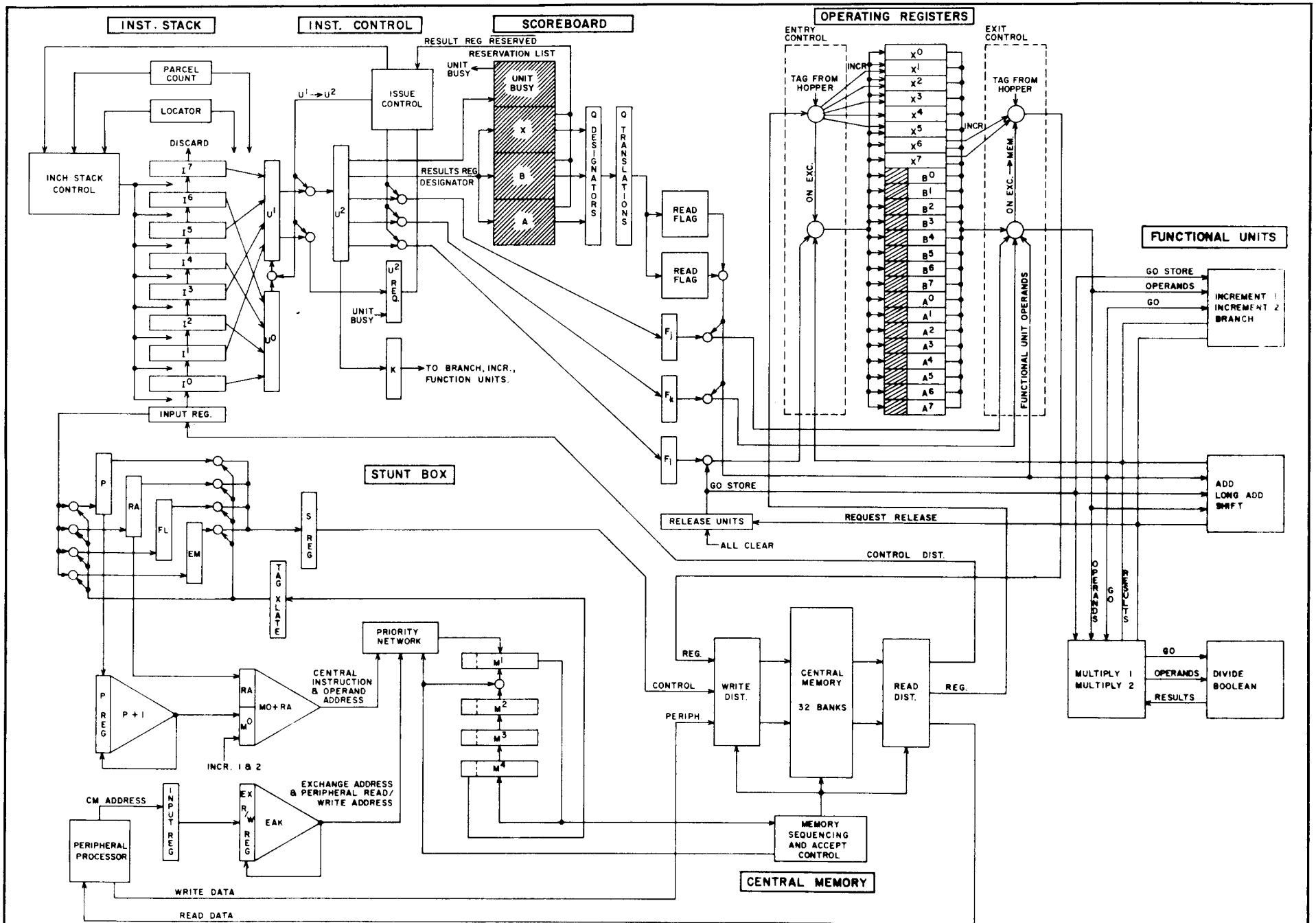
WING 1

WING 3

WING 4

 CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE CENTRAL PROCESSOR CHASSIS LAYOUT	PRODUCT 6601
	SIZE DRAWING NO. C 60119300	REV k
	SHEET 2	

2



3

CHAPTER I

CONCEPT OF THE 6600 CENTRAL PROCESSOR

INTRODUCTION

The CONTROL DATA® 6600 Computer System, through use of high-speed transistor logic and a design philosophy based on concurrent (or parallel) processing, is today recognized as the world's fastest and most powerful computer. The rapid throughput achieved by the 6600 system can be attributed in part to the concurrency that exists in several areas of the Central Processor.

MEMORY BANK PHASING

The Central Memory is divided into memory banks, each of which contains $4096_{(10)}$ 60-bit central processor words. A 131K central memory is composed of 32 such banks; a 65K memory has 16 banks. Since each bank has its own circuitry for the X & Y drive lines, inhibit lines, sense lines and memory cycle timing, each is capable of operating independently. This, in turn, permits memory cycles to be phased (overlapped) by 100 nanoseconds, to effectively reduce minimum access time to 100 nsec (e.g., a memory cycle is one microsecond in duration, but ten may be initiated each usec as long as they are to different banks). The bank phasing scheme, in addition to a memory cycle which is in itself extremely fast, eliminates a great portion of the memory waiting time that is inherent in the majority of computers.

INSTRUCTION STACK

A group of flip-flop registers referred to as the Instruction Stack is provided in the 6600 for the purpose of holding an iterative sequence of instructions (a program loop). The Stack can hold a loop containing up to 27 instructions (up to 4 instructions per word) which may then be executed without the need for instruction word memory references (RNIs).* Initially, the eight stack registers (I registers) are filled by reading instruction words from central memory. As each word is read into and executed from IO (See Figure 1-1), the preceding words move up in the stack and a new word is entered into the first I register. When the stack is filled, the movement of instruction words causes the top word (in

* Although 27 instructions may at first appear to limit the programmer's capability, it should be considered that the 6600 is designed primarily as a scientific machine. Consequently, a good many programs will be of a mathematical nature (i.e., matrix analysis). Also, each instruction can designate two source operands and one result destination. When viewed in this light, 27 instructions are, in most cases, more than adequate.

®Registered trademark of Control Data Corporation.

17) to be discarded. When instructions are being executed in the stack (looping), no movement occurs and the stack information remains static. In this manner, the necessity of fetching each instruction from memory is eliminated during short loops. The memory access time savings should be obvious.

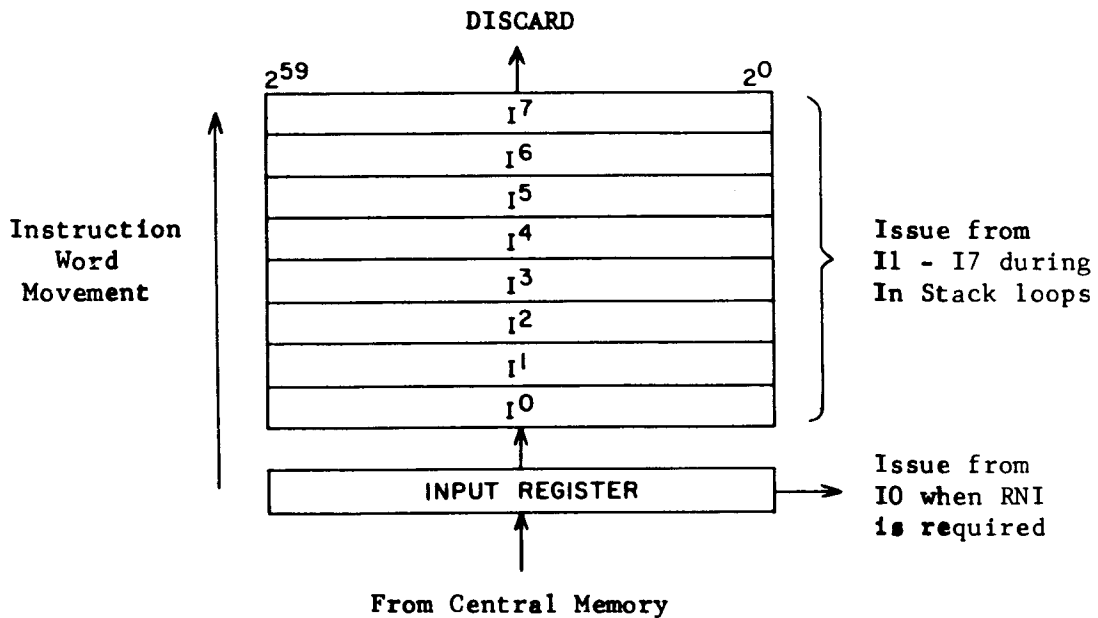


Figure 1-1

Operating Registers

Another property of the Central Processor decreases the number of memory references required to read and store operands. Twenty-four operating registers provide a flip-flop storage facility for 60-bit operands and 18-bit addresses and indexing values. Eight 60-bit registers (designated X0 - X7) provide for the storage of integer and floating point values in a 60-bit format. Eight 18-bit registers (designated A0 - A7) provide storage for central memory addresses of operands which are read or stored in memory. Eight 18-bit registers (designated B0 - B7) provide for storage of indexing values, used for modification of addresses and operands. Since most central processor instructions can designate two source operands (taken from X, B or A registers) and one result destination (X, B or A register), considerable operand manipulation can take place by use of the operating registers, thereby further decreasing the number of memory accesses needed. Proper use of the instruction stack and operating registers makes possible, execution of program loops which require no memory references -- for instructions, operands or storage of results.

Functional Units

Another area of concurrency in the 6600 Central Processor is that of parallel arithmetic (functional) units. Ten logically independent functional units are provided to allow several instructions to be in various stages of execution at the same time. The following list describes the functional units and their corresponding cycle times:

<u>UNIT</u>	<u>TIME</u> (nanoseconds)
1) ADD (floating)	400
2) MULTIPLY 1 (floating)	1000
3) MULTIPLY 2 (floating)	1000
4) DIVIDE (floating)	2900
5) BOOLEAN (logical)	300
6) LONG ADD (integer)	300
7) SHIFT	300 - 400
8) INCREMENT 1 (indexing)	300
9) INCREMENT 2 (indexing)	300
10) BRANCH (branch instructions)	800 - 1400

Each unit is assigned a group of instructions which it, and only it, processes. For example, the ADD unit processes all single precision, double precision, rounded and unrounded floating point add opcodes. The SHIFT unit handles opcodes that require shifting: left and right shifts, normalize operations, packing, unpacking, etc.

Separate functional units eliminate the necessity for sequential execution of program steps, a property which is inherent in most present-day computers. Instead, unrelated instructions may be processed out of sequence, causing a considerable decrease in the over-all execution time of a program. Of course, if a source operand for one unit is the result operand of another, the first unit must wait until the second completes its calculation and returns the result. Also, if two division steps are needed in sequence, the second must wait until the first completes, since only one divide unit exists. On the other hand, two multiply operations may take place at the same time because two multiply units are provided. The point to be stressed is that in most operational programs the instructions need not be executed in sequence. Instead, the majority of problems are composed of a series of smaller steps which are only indirectly related. The following programming comparison should illustrate this point.

The problem that follows is solved first by using a sequential computer and secondly, by using the 6600 with its functional units. Individual instruction execution times are assumed to be the same in both machines. Also, both have the capability of reading two.

source operands and returning one result by use of operating registers (X, B and A).

THE PROBLEM:

$$\left(\frac{A+B}{C}\right) \cdot (A^2 + B^2 + C)$$

THE OPERATING REGISTER CONTENTS: (where () means "the contents of")

(X1) = the value, A
 (X2) = the value, B
 (X3) = the value, C

THE PROBLEM THUS BECOMES:

$$\left[\frac{(X1) + (X2)}{(X3)}\right] \cdot [(X1)^2 + (X2)^2 + (X3)]$$

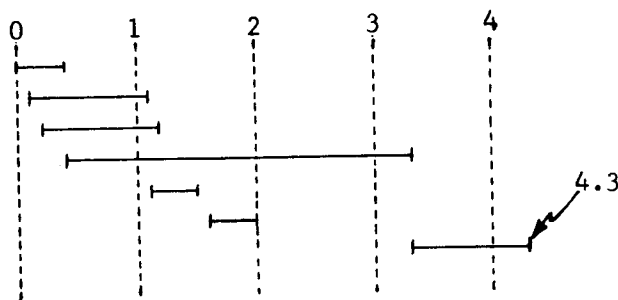
USING THE SEQUENTIAL COMPUTER: (where → means "replaces")

<u>Instructions</u>	<u>Time (nanoseconds)</u>
1. (X1) + (X2) → (X4)	400
2. (X4) / (X3) → (X5)	2900
3. (X1) * (X1) → (X6)	1000
4. (X2) * (X2) → (X7)	1000
5. (X6) + (X3) → (X0)	400
6. (X0) + (X7) → (X7)	400
7. (X5) * (X7) → (X6)	<u>1000</u>
TOTAL TIME =	7100

Since the instructions must be executed in sequence, the total execution time is the sum of the individual execution times, or 7.1 microseconds.

USING THE 6600:

1. (X1) + (X2) → (X4)
2. (X1) * (X1) → (X6)
3. (X2) * (X2) → (X7)
4. (X4) / (X3) → (X5)
5. (X6) + (X3) → (X0)
6. (X0) + (X7) → (X7)
7. (X5) * (X7) → (X6)



(NOTE: Time is shown in microseconds)

Using parallel functional units, the program execution time is only 4.3 microseconds, a reduction of approximately 40%.

Although the same saving will not occur in all programs, the example illustrates that, through efficient programming, a considerable decrease in execution time occurs. Even when a program is not optimized, a time saving will be realized. Details of time implications from the preceding chart are considered in later topics.

Summary

Several unique features are incorporated in the design of the 6600 central processor, including: 1) thirty-two (or sixteen) 4K, phased memory banks, 2) an instruction stack containing eight 60-bit registers, 3) twenty-four operating registers and 4) ten independent functional units. These provisions work in conjunction with each other to provide extremely rapid program execution times. Whenever parallel processing capabilities are provided in a computer, control circuitry is required to ensure that all features work together (without calamity) to produce a high-speed processing system,

BLOCK DIAGRAM ANALYSIS

CENTRAL MEMORY ADDRESS CONTROL

References to Central Memory can be initiated from various sources in the 6600. Peripheral Processors make central memory references during the central read, write and exchange jump instructions. The Central Processor uses central memory to fetch instruction words or to read and store operands. An orderly means for handling these memory requests and distributing the associated data must be utilized. This is complicated by the fact that the 6600 memory banks are phased to allow several memory cycles to be in progress at any one time. Therefore, it is very possible that a memory reference request be made to a bank that is already busy processing a memory cycle, so that the address must be saved and then re-issued. It is also conceivable that two requests occur simultaneously, requiring that a decision be made regarding which address will be issued first. Not only must the address be manipulated methodically, but the source or destination of the data associated with each address must be "remembered" by the control logic. These functions are accomplished for the most part, by the Central Memory Control logic, more often referred to as the Stunt Box.

Analysis of the Stunt Box takes place in the following sequence:

- 1) Hopper
- 2) Priority Network
- 3) Tag Generation and Distribution

HOPPER

The Hopper is a mechanism used to save conflicting addresses so they may be re-issued to the memory banks repeatedly, if necessary, until accepted and processed. Along with addresses, the Hopper saves gating information used to enable the data corresponding to each address through the memory Data Distribution logic to or from memory.

Physically, the Hopper is four flip-flop registers (designated M1, M2, M3 and M4) each of which stores an 18-bit address, 6-bit tag and a Full bit (except M2, which has no Full bit). Refer to Figure 1-2. The registers are connected to each other in such a manner as to allow information to circulate through each of the registers (the concept is similar to the Peripheral Processor barrel). A 75 nanosecond time interval exists between each register and produces a total re-circulation time of 300 nsec. For example, an address entered into M1 at time 00 enters M4 at t75, M3 at t150, M2 at t225 and (if it must be re-issued) re-enters M1 at t300.

6600 STUNT BOX

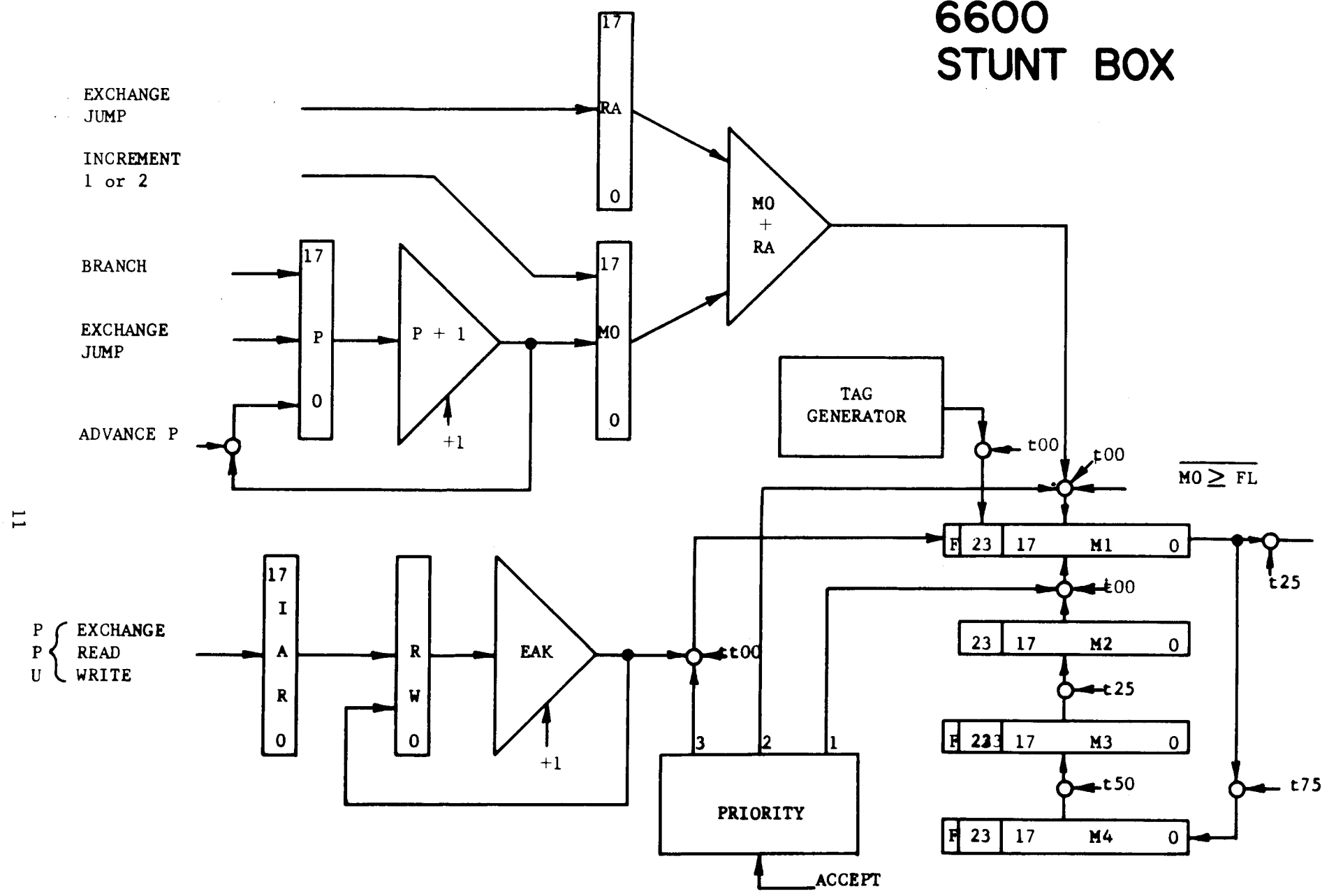


Figure 1-2

In actuality, only 17 of the 18 available address bits are used in a 131K central memory. The 18th bit is found throughout the memory circuitry, but is never utilized. The 6-bit tag is generated when an address is first entered into the hopper (specifically, M1) and it contains all the information necessary to properly distribute the associated data. Tag generation and distribution is almost a subject in itself and is treated separately later in this section. The Full bits found in M1, M3 and M4 indicate that a meaningful address and tag are contained in the respective register. It is set when an address and tag are entered into M1. There is no full bit in M2, since the Accept signal (explained below) serves a similar purpose.

Approximately 50 nsec after entering an address into M1, the address is automatically sent to the memory banks, where the lower 5 bits are examined to select one of the 32 banks. If the desired bank is not in use ($\overline{\text{BUSY}}$), an Accept signal is returned to the Stunt Box to indicate that no conflict exists and the memory cycle has been initiated. The address saved in the Hopper is then discarded. If the desired bank is busy, an Accept will not be returned. Its absence causes the associated address to be re-entered into M1 (from M2) and subsequently, reissued to the memory banks. The cycle will recur every 300 nsec until the address is accepted.

Figure 1-3, a timing diagram, verifies that the Accept is returned to the Stunt Box in time to disable (or if not returned, enable) the transfer of M2 to M1. If an Accept is generated, it will be received on Chassis 5 about 175 nsec after entering the associated address into M1 (t_{175}). This allows 125 nsec of logic delay time before the Accept is used to disable the M2 \rightarrow M1 transfer (t_{300}). Since M2 was not transferred to M1, the following transfer of M3 to M2 will destroy (write over) the contents of M2.

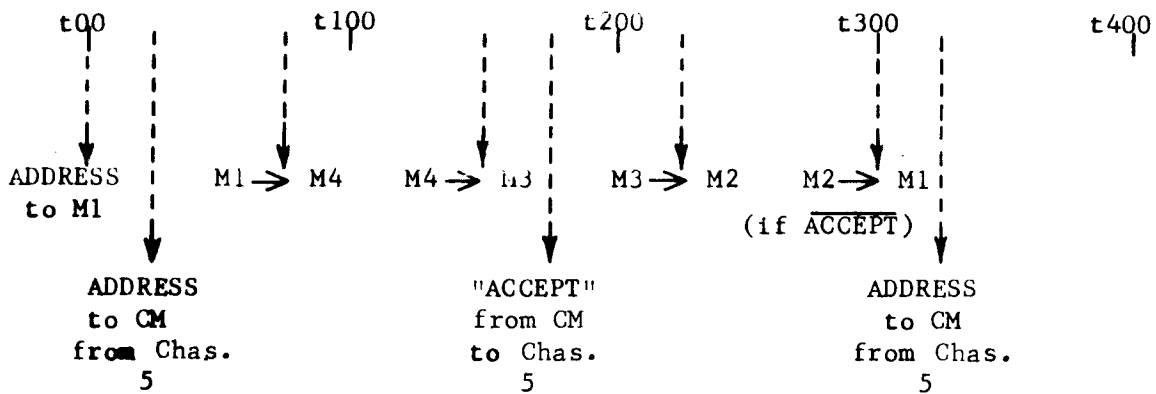


Figure 1-3.

PRIORITY NETWORK

Because more than one source exists for addresses entered into M1 (See Figure 1-2) a Stunt Box Priority Network is necessary to ensure organized handling of simultaneous memory requests. Each address source has a fixed priority, as follows:

First	Hopper (M2 → M1)
Second	Central Processor (M0 → M1)
Third	Peripheral Processor (ERW → M1)

HOPPER PRIORITY

In only one circumstance is re-entry of an address to M1 required: when an address has been sent to the memory banks and was not accepted due to a bank conflict. Non-acceptance of an address is indicated by not receiving an Accept from the memory banks 175 nsec. after issuing an address. Before enabling an M2 → M1 transfer it should also be determined that M2 contains a meaningful address. This is indicated by the presence of a Full bit. Since M2 does not contain a Full bit, the M3 Full bit is checked. (It is time delayed to ensure that M3 has been transferred to M2 before the check is made.) Thus, two conditions must be met to grant first priority:

(M3 Full) (Accept)

CENTRAL PRIORITY

Two sub-priorities exist under Central Priority because central processor memory references may be originated in two independent operations

- 1) Instruction word fetching (RNI's)
- 2) Reading and storing operands

In the first case, the address is obtained from the P register and in the second, from one of the two Increment Functional Units. If requests from both sources occur simultaneously, the operand address is entered first, then the instruction address. In either case, the address is entered into an 18-bit register, M0 (Figure 1-2). At the same time a control flip-flop called "Enter Central" is set and indicates that an address is in M0 waiting for entry into M1. (In a sense, the Enter Central flip-flop requests priority #2). Thus, one condition required for priority #2 is that Enter Central is set to indicate that an address is in M0 waiting for entry into M1.

A second condition needed for central priority is that priority #1 does not exist (i.e., the address in M2 was accepted or M2 does not contain a meaningful address).

A special circumstance arises which also must be considered in granting central priority. This is the case when read and store requests are made to the same memory address. This might occur when an instruction word modification is made followed by an RNI request for the modified word. If the two addresses enter the hopper in sequence (store location X, then read location X) storing before reading cannot be guaranteed because a bank conflict may exist with the store address. The operation (read or store) that is performed first depends strictly on when the bank goes BUSY. Whichever address is sent to the banks first (after BUSY) will be accepted and will cause a conflict for the second reference to the same location. Thus, it would be possible, in the above instance, to read the unmodified instruction word when actually, the modified word was desired. The reverse situation might also occur, wherein a location was to be read before modification.

To resolve the above cases, additional logic is required in the priority #2 circuitry which prevents a Central Read address from being entered into the Hopper if any (Peripheral or Central) Write address is in the Hopper. Also, if a Central Write is attempted, no Central Read address may be in the Hopper. (Prevention of a Central Write and Peripheral Read out of sequence is a software responsibility.)

The fourth, and final condition needed for Central Priority, is that the address being referenced must not be out of the bounds for this particular program. Memory bounds for a program are defined upon initialization of the routine (EXCHANGE JUMP) by the RA (Reference Address) and FL (Field Length) values. RA specifies the lower bound and $RA + FL - 1$, the upper bound. Each central memory reference adds to the value RA, the content of P (for RNI's) or the Increment I or II address (for operand references). Thus, the address being referenced (P, Incr. I or Incr. II) is said to be the "relative" address. The absolute CM address is the sum of the relative address and the content of RA. The relative address is always entered into MO. A special Adder adds MO to RA and yields the absolute address. Another circuit compares the content of MO with the content of FL. If $MO \geq FL$, the desired reference is "Out of Bounds", and the memory reference will not take place because Central Priority will not be granted. Thus, the condition $MO < FL$ is also a condition required for granting Priority 2.

The following Boolean formula summarizes the conditions required for granting Central Priority:

$$\begin{aligned}
 &(\text{Enter Central}) \quad \overline{(\text{Priority 1})} \quad (MO < FL) \quad (\text{Attempt Read}) \\
 &\overline{(\text{Write in Hopper})} + (\text{Attempt Write}) \quad \overline{(\text{Central Read in Hopper})}
 \end{aligned}$$

PERIPHERAL PRIORITY

Peripheral priority for CM references is granted only if neither Hopper nor Central priority exists and there is a peripheral processor request for a CM access. Since only one PPU request can occur at a time, no sub-priorities are required.

The PPU's request CM references in three situations:

- 1) Read central memory
- 2) Write central memory
- 3) Exchange jump.

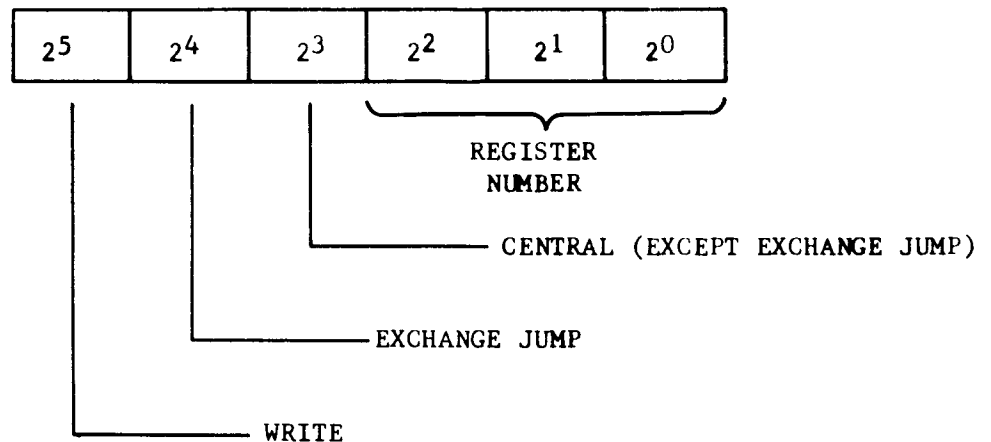
In all three cases, a PPU will send an 18-bit address to the Input Address Register (IAR) of the Stunt Box (See Figure 1-2). To specify the type of reference being requested, a Read, Write or Exchange pulse accompanies the address. These are used as a control function, to properly gate information to and from CM (See Hopper Tag discussion which follows). The presence of one of the three control pulses results in the Request for Priority #3. Thus, the Boolean expression for Peripheral Priority is as follows:

$$(PPU \text{ Read} + \text{Write} + \text{Exchange}) (\overline{\text{Priority 1}}) (\overline{\text{Priority 2}})$$

During peripheral processor read and write operations in central memory, a new address is sent to the IAR for every memory reference desired. For exchange jumps, only the starting address of the exchange jump package (in CM) is sent. It is the responsibility of the central processor to advance this address automatically in order to exchange the required information. This is accomplished by the Exchange Address Counter (EAK) which is utilized only during exchange jumps. It increments the exchange address for each of the 16 locations referenced.

HOPPER TAG GENERATION AND DISTRIBUTION

As previously mentioned, when an address is entered into M1, a 6-bit tag is also entered. It is used to properly gate data into and out of Central Memory. The tag bit positions are named as follows:



The bit (2⁵) will be set any time the associated address is that of information to be stored (written) into Central Memory.

The bit (2⁴) is set only during exchange jumps to indicate that the associated data is to be exchanged with registers in the CPU.

The bit (2³) is set any time a memory reference is initiated by the Central Processor (Priority 2) and allows information to be gated to or from the CPU, as opposed to a PPU. During exchange jumps, the bit is set to indicate that an X register is to be exchanged, or cleared to indicate that A, B and Control Registers (P, RA, FL, etc.) are to be exchanged.

The bits (2⁰ - 2²) indicate (when applicable) which X, B, or A register number is to be stored, read into, or exchanged. Table 1-1 lists all legal tag numbers (in octal) and their meaning. Decoding circuitry exists only for those tags listed. Any other bit combination will either not be decoded, or will be decoded as one of the legal tags.

TABLE 1-1. HOPPER TAGS

00	Peripheral Read	63	Exchange EM, A3, B3
10	CP RNI	64	Exchange RA(ecs) A4, B4
11	CP Read → X1	65	Exchange FL(ecs) A5, B5
12	CP Read → X2	66	Exchange A6, B6
13	CP Read → X3	67	Exchange A7, B7
14	CP Read → X4	70	Exchange X0
15	CP Read → X5	71	Exchange X1
40	Peripheral Write	72	Exchange X2
50	Return Jump + Error Stop	73	Exchange X3
56	CP Write X6	74	Exchange X4
57	CP Write X7	75	Exchange X5
60	Exchange P, A0	76	Exchange X6
61	Exchange RA(cm) A1, B1	77	Exchange X7
62	Exchange FL(cm) A2, B2		

A tag = 00 indicates a Peripheral Read address since all bits equal zero. This is interpreted as meaning:

$\overline{\text{WRITE}}$ $\overline{\text{EXCHANGE}}$ $\overline{\text{CENTRAL}}$ or Peripheral Read.

In this case the register bits ($2^0 - 2^2$) have no meaning and are not translated.

A tag = 10 indicates a Central Read Next Instruction (RNI) since the Central bit (2^3) is set and all other bits are cleared. Since a Central Read of Memory to X0 is not possible, the clear state of bits $2^0 - 2^2$ in this case indicate that an instruction word is to be read from Memory.

A tag = 11 indicates that a Central Read to X1 is to be performed. Bits $2^0 - 2^2$ in this case indicate the X register number. Tags 12-15 are also Central Reads to X registers, but to X2 - X5, respectively.

A tag = 40 indicates a Peripheral Write operation, since the write bit (2^5) is set and the Central and Exchange bits (2^3 & 2^4) are both cleared.

A tag = 50 indicates a Central Return Jump or Error Mode Stop memory reference. Bits $2^0 - 2^2$ are meaningless in this case since a Central Write (tag = 5X) of X0 is not possible. Since storage of information in central memory is required in the above cases, the 50 tag is reserved for this purpose.

Tags = 56 & 57 are generated when storage of X6 or X7, respectively, is desired. Bits $2^0 - 2^2$ again indicate the register number.

Tags 60 - 77 are all generated during an Exchange Jump operation. Bit $2^3 = 0$ indicates that A, B or Control registers are to be exchanged. Bit $2^3 = 1$ indicates that an X register is to be exchanged. Bits $2^0 - 2^2$ specify the operating register number (i.e., X, B or A) or the control register (i.e. P, RA, FL, or EM) to be exchanged. Note that these are the only cases when bit 2^4 (the Exchange bit) is set.

After a memory reference is initiated, the associated tag is decoded and will enable the gating of the desired information into and/or out of Central Memory, to or from the desired location (XBA registers, control registers, read or write pyramids, etc.).

CENTRAL MEMORY

The 6600 Central Memory is composed of 60-bit words located in 16 or 32 memory banks each of which contains 4K words. This results in 65K or 131K memory sizes, respectively. In either case, 4 banks are contained on a chassis.

Selections of bank and chassis are made by decoding the lower 4 (for 65K memories) or 5 bits (for 131K memories) of the address. For example, in 131K system,

bits 2^0 and 2^1 select one of 4 banks on a chassis, while bits 2^2 , 2^3 & 2^4 select one of 8 chassis. The address is sent from Chassis 5 to all memory chassis of a system and all chassis decode the lower bits of the address. Only one chassis will recognize its bit configuration ($2^2 - 2^4$). By decoding bits 2^0 and 2^1 , the bank selection is made. If the selected bank is free (i.e., a memory cycle is not already in progress) the Accept signal is returned to the stunt box and a Go signal is sent to the selected Storage Sequence Control circuit (SSC). The SSC is a simple flip-flop timing chain which generates the read/write memory cycle.

The selected address within the selected bank is determined by decoding the remaining 12-bits of the 17-bit address (16 bits for a 65K system). While a memory cycle is in progress, the bank busy signal (bank not free) disables initiation of other memory cycles within that bank. It also disables the return of the Accept signal to Chassis 5, which causes the address to be retried at the 300 nsec stunt box rate.

The information being read from or stored into central memory is gated by a circuit called the Read/Write distributor. It, in essence, distributes information to and from the 4 Chassis connected to central memory as shown in Figure 1-4.

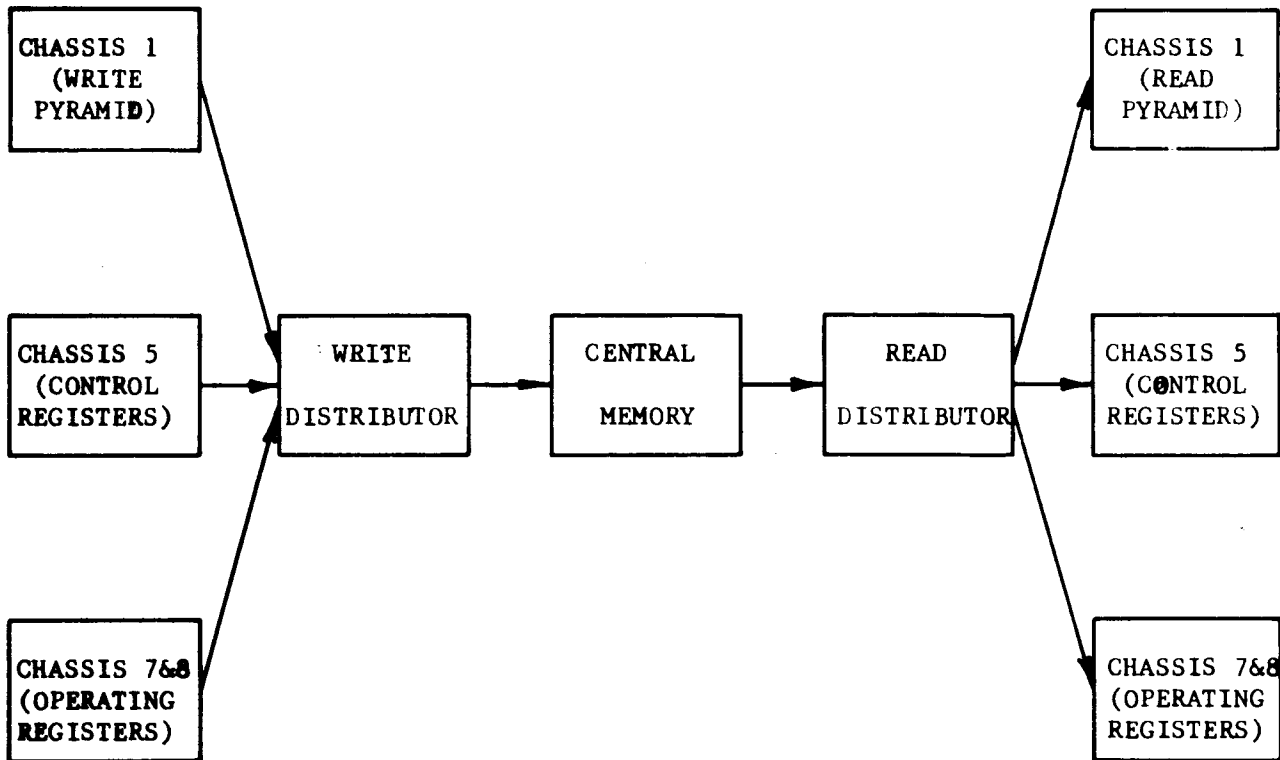


Figure 1-4.

CENTRAL MEMORY

ADDRESSING

The CP programs are stored in CM, and all PPs may use CM for supplementary storage or inter-communication control. Thus CM addresses are generated by the CP and all PPs.

Each processor sends a CM address to a common address clearing house, or stunt box, from where they are sent on to CM. The stunt box can accept addresses from the several sources at 100-nsec intervals (maximum rate) on a priority basis and in turn issue one address every 100 nsec to CM.

An address goes to all banks of CM for decoding, and the referenced bank returns an accept signal to the stunt box if the bank is not busy (free) with a previous reference. The stunt box saves each address that it sends to CM in a hopper mechanism, and, if the address is not accepted, it is recovered from the hopper and re-issued to CM and again saved. The issue-save cycle repeats until an accept is received to void the hopper address. Up to three addresses can be saved in the hopper. However, an address is always accepted within 2000 nsec (worst case because of bank conflict) of the first time it is issued.

DATA DISTRIBUTION

Data to and from CM is distributed from a data distributor. The word from a read reference goes from CM to the data distributor and then to the requesting processor. A word to be stored during a write reference goes from the processor to the data distributor to CM. The distributor can transfer a word to or from CM every 100 nsec. A store word goes to all banks of CM, but separate storage control mechanisms for each bank insure that the word is stored in the proper bank.

The distributor routes data to and from proper origins and destinations as directed by control information or tags received from the stunt box. The tags are entered in the stunt box along with each address and serve to identify the address sender, origin or destination of data, and nature of the address, e.g., read, write, or PP exchange jump. The stunt box sends the tags to the data distributor (and to destinations in the processors for read references) when an address is accepted, and the distributor accomplishes the data transmission. For write references, the data source sends the word to the distributor, where it is held temporarily before it is stored.

STORAGE

The many banks of storage in CM are evenly distributed on 8 chassis in the computer. There are four banks per chassis.

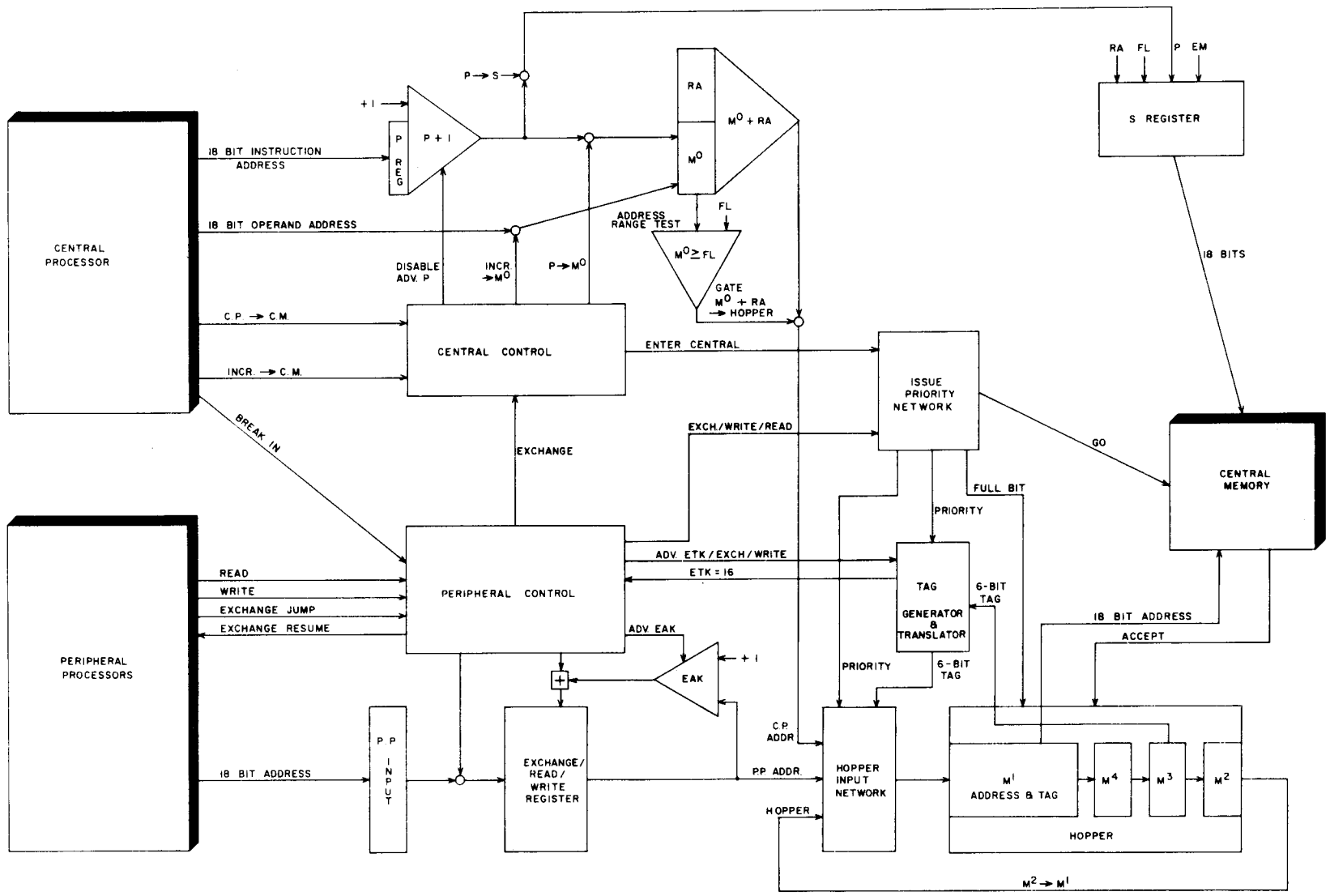
The circuit organization allows the four banks to operate independently and be phased into operation at 100-nsec intervals, which corresponds to the maximum rate at which the stunt box issues addresses. A chassis input register receives the 17-bit address from the stunt box and distributes the 12-bit address to 1 of 4 storage address registers associated with the four banks. Hence 32 consecutive addresses referencing 32 separate banks may be accepted at 32 consecutive minor cycle intervals and result in a data word flowing to or from CM in 32 consecutive minor cycle intervals. The independent controls for each bank and treatment of the address and data word insure that only one bank is in a given time segment of its 1000 nsec storage cycle at any one time. At least one minor cycle separates the storage cycle of all banks.

A word read from any bank is sent to a common temporary storage register and to the data distributor by a common path. A word to be restored is then sent to a write register by way of a buffer register. The write register sends the word to 1 or 4 restoration registers for restoring in the proper bank.

A word from the data distributor during a write reference goes to the temporary storage register on all chassis and then follows the restore path for writing in memory. Only one of the many banks is in the proper time spot in its storage cycle to store the word received, and this bank is the one associated with the write address.

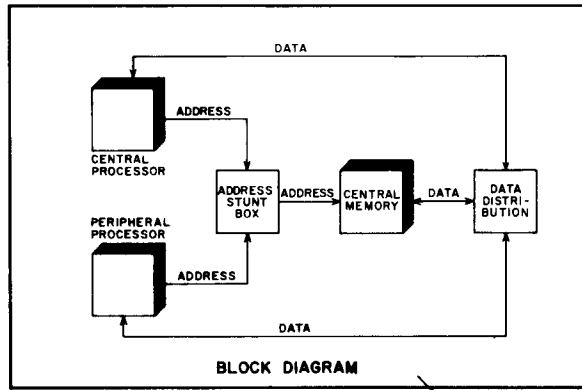
A go signal with each address from the stunt box allows a group of four banks (one chassis) to recognize and translate the bank bits. The referenced bank, if not busy, sends an accept to the stunt box and starts 1 of 4 storage sequence control circuits, which in turn direct the 1000 nsec storage cycle for the selected address.

A write signal may also accompany each address from the stunt box. It distinguishes read and write references and controls the path to the restoration registers. The CM uses the same 12-bit storage module as used in the PPs, but five are driven in parallel to hold the 60-bit word.

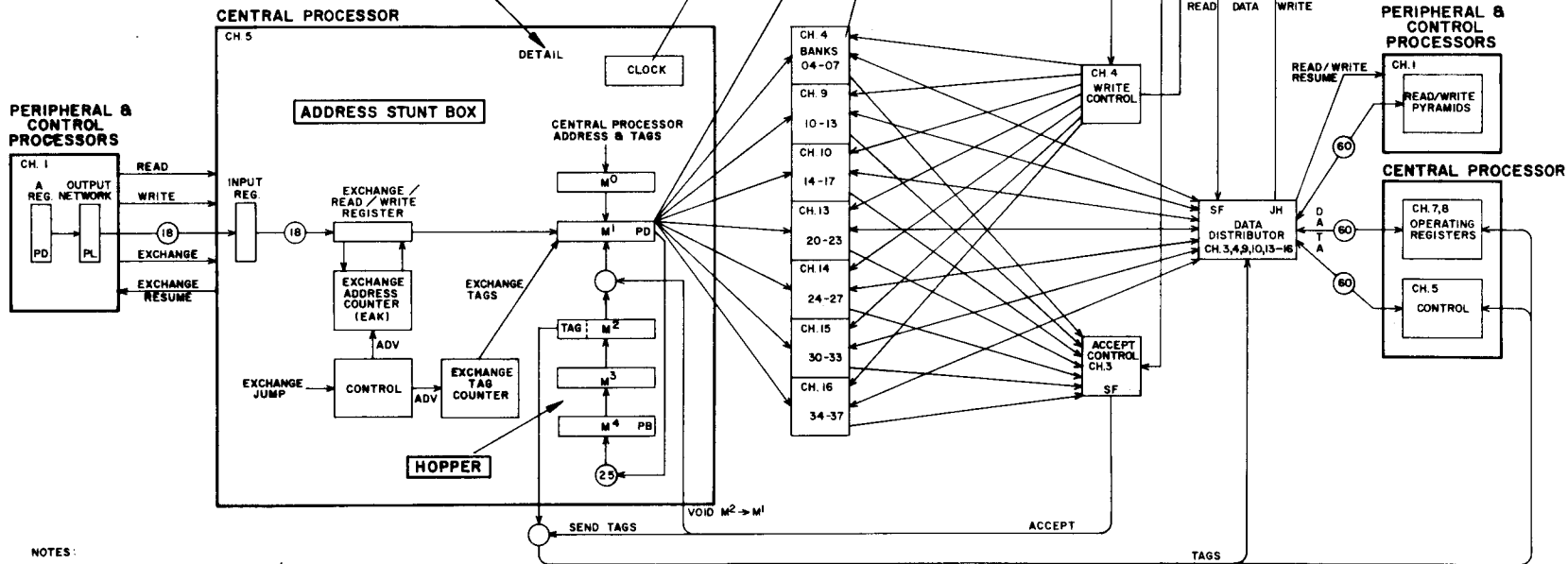
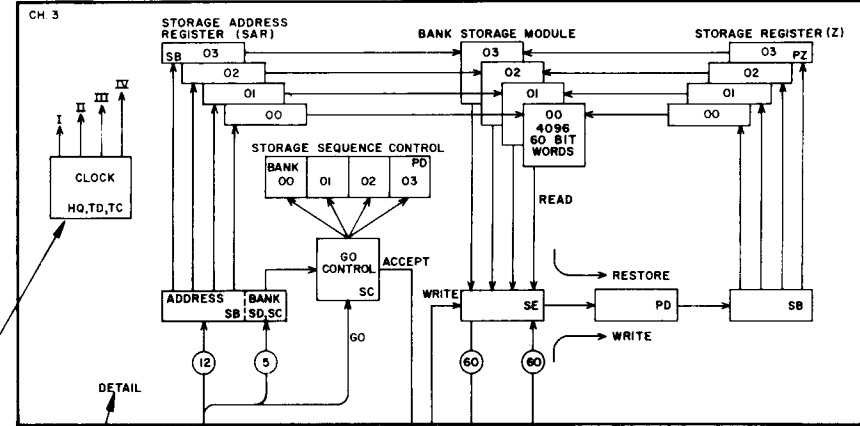


21

 CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE	6601/04	PRODUCT
		CENTRAL PROCESSOR STUNT BOX BLOCK DIAGRAM	6601/04
	SIZE	DRAWING NO.	REV.
	C 60119300	k	
	SHEET	PAGE	
	201	75	



BANKS 00-03



NOTES:

1. ADDRESSES SENT TO CM FROM M¹ AT MINOR CYCLE RATE.
2. DATA MOVES TO/FROM CM AT MINOR CYCLE RATE.
3. ADDRESS TAGS DEFINE ORIGIN / DESTINATION OF DATA.
4. TIME FROM M¹ → CM TO RESPONSE TO CM ACCEPT IS 200 nSEC.
 - a. M¹ STORED IN M⁴ AT ISSUE TIME AND MOVES TO M² IN TIME SEQUENCE.
 - b. ACCEPT VOIDS RE-ISSUE OF ADDRESS FROM HOPPER.

CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE CENTRAL MEMORY ADDRESS - DATA FLOW	PRODUCT 6601
		SIZE DRAWING NO. REV C 60119300 K
		SHEET 38 I

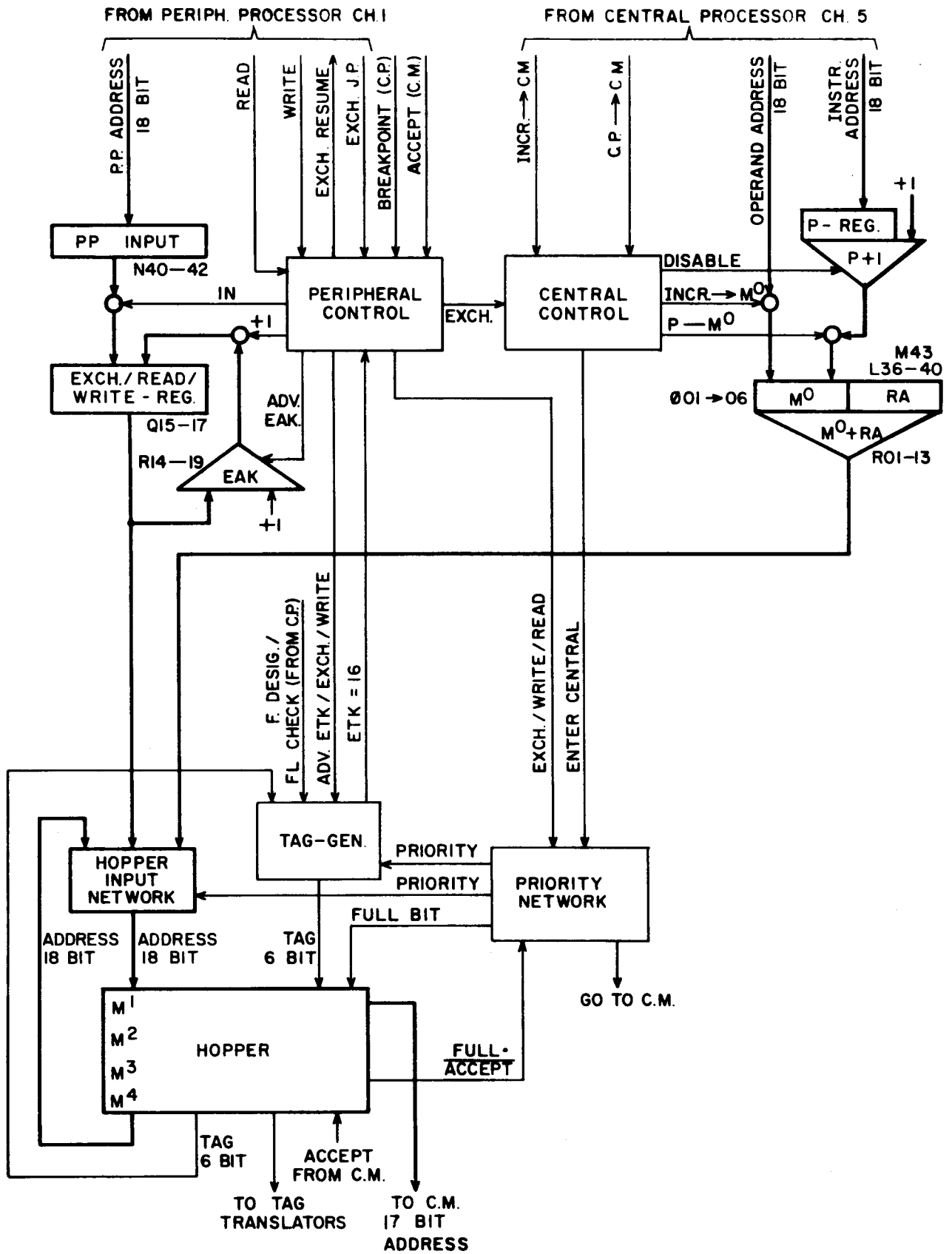


Figure 2-1. Stunt Box Block Diagram

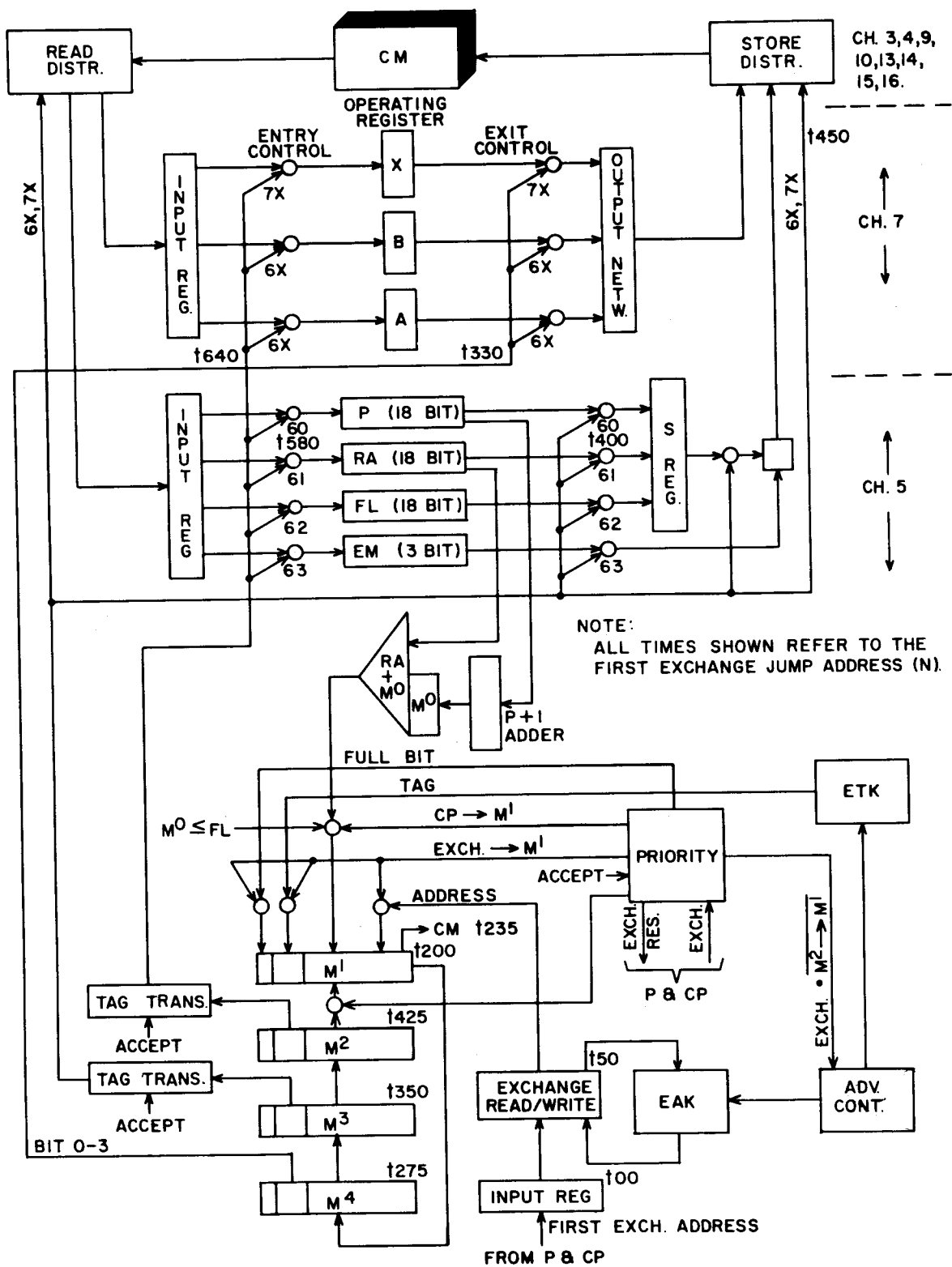


Figure 2-10A. Exchange Jump

READ DISTRIBUTOR

The read distributor accepts read words from the 8 CM chassis and routes them to the several destinations.

The distributor is organized on chassis 3, 4, 9, and 10, each of which handles 15 bits of the 60-bit word. Chassis cable limitations dictate the organization. The listing below shows the bits handled by each chassis.

CHASSIS	BITS
3	0-14
4	15-29
9	30-44
10	45-59

Chassis 13-16 each send the same 15-bit group to chassis 3, 4, 9, and 10. A read word from chassis 3 retains bits 0-14 but sends remaining bits in three groups to chassis 4, 9, and 10. Read words from chassis 4, 9, and 10 are handled similarly. Intra-chassis coaxial cables are

used on chassis 3, 4, 9, and 10 for their 15-bit portions so that timing is consistent with the chassis receiving the data.

Each read word is sent unconditionally from chassis 3, 4, 9, and 10 to chassis 5 (CP control) and chassis 7 and 8 (CP registers). A read peripheral tag from the stunt box is sent to chassis 4 and then on to chassis 3, 9, and 10. The tag gates the read word to the C^5 register in the read pyramid on PP chassis 1.

The read peripheral tag also enters a time delay chain and is returned to the PP as a resume signal. The resume sets the C^5 full FF in the PP (after data word is in C^5) to signal the presence of the read word. The same resume also clears the central busy FF to indicate to PP control that the address has been accepted by the stunt box and CM has delivered the word. This allows the PPs to proceed and send another address to the stunt box.

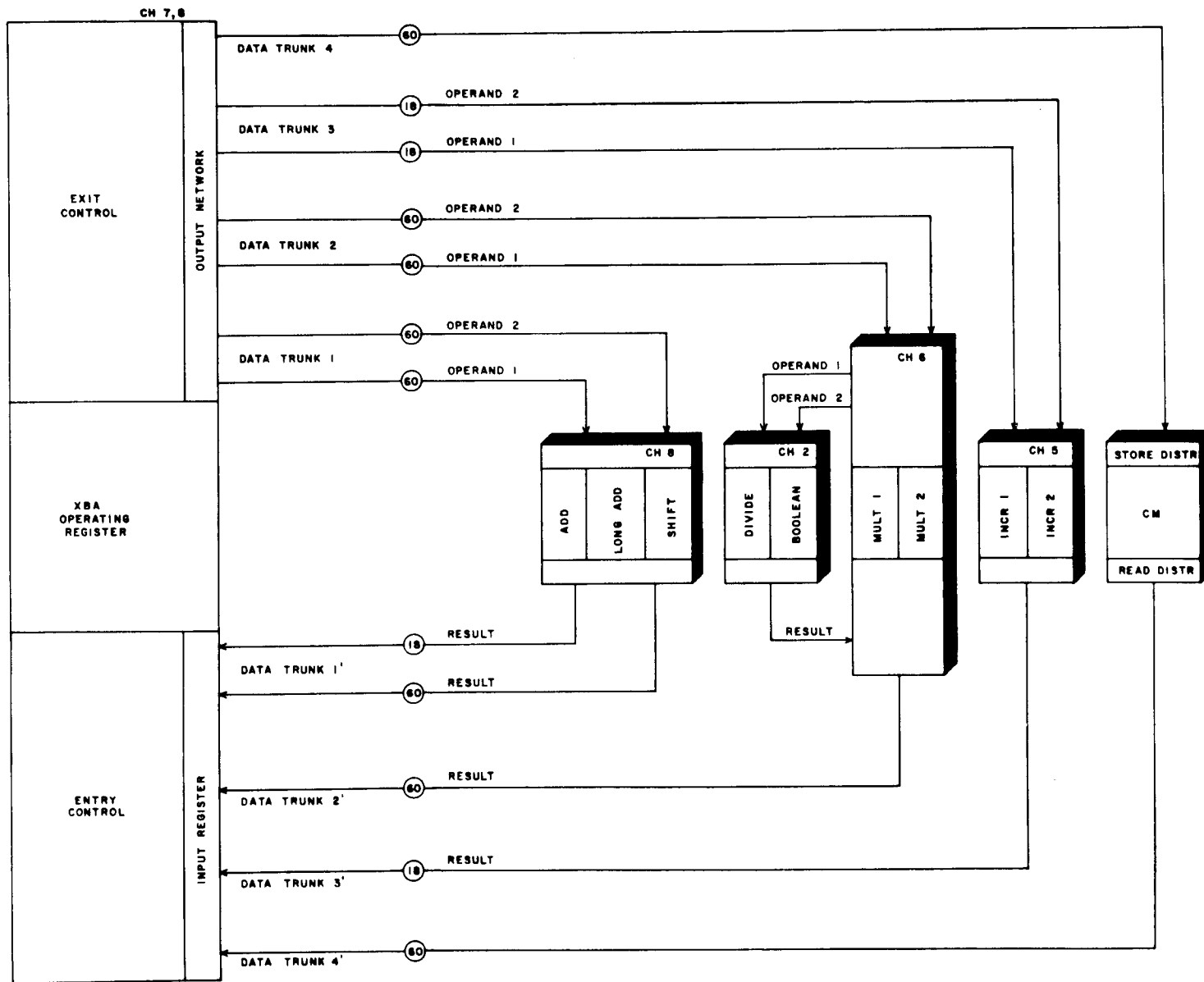


Figure 6-5. Data Trunks

Data Trunk 2 connects the output network of the XBA-Registers to the Divide, Boolean, Multiply 1 and Multiply 2 functional units. The lower 48-bits of the operands go via chassis 6 (Multiply 1 and 2) to chassis 2 (Divide and Boolean). The upper 12-bits go directly to chassis 2 from EXIT Control and are not sent to chassis 6 (except bit 2^{59}) which contains only the Multiply coefficient logic.

The results coming from the functional units and the data coming from Central Memory will be transmitted by Data Trunks 1', 2' 3', and 4' to the Input Register of the Entry Control network.

Determination of which information is to be gated is made by decoding of the Hopper tag associated with each address and ANDing the decoded signal with the Accept for that address.

Information is gated from the Write Pyramid on Chassis 1 to CM via the write distributor during PPU central memory write operations (tag = 40).

Information is gated from Chassis 5 during return jumps and Error Mode Stops (tag = 50).

Information is gated from Chassis 7 and 8 during Exchange Jumps (tags = 60 - 77) and central processor store operand operations (tags = 56 or 57).

Information is sent from C.M. via the Read distributor to Chassis 1 during PPU central memory read operations (tag = 00).

Information is sent to Chassis 5 during Exchange Jumps (tags = 60 - 65) and RNI (tag = 10) operations.

Information is sent to Chassis 7 and 8 during Exchange Jumps (tags = 60 - 77) and central processor operand read (tags = 11 - 15) operations.

It is re-emphasized that in all cases of gating the Read/Write distributor the Accept signal is necessary. This ensures that the information desired is properly timed for entry to or exit from memory.

INSTRUCTION CONTROL

Instructions in the Central processor are executed from the Instruction Stack (shown in Figure 1-1). Each 60 bit Stack register can contain up to four instructions, since the Central processor employs both a 15 bit and a 30 bit instruction format, and as few as two. The responsibility of Instruction Control is to determine that a 60 bit Instruction word has become available to the stack, sort out the 15 or 30 bit instructions within that word, and then deliver the instructions to Reservation Control so they can be executed.

Initially all instruction words (a 60 bit memory word fetched by an RNI request) move into the bottom rank of the stack (I0) when Instruction Control receives the RNI tag (TAG = 10g and accept) from the Stunt Box. The RNI tag also signals Instruction Control to begin the process of sorting instructions within that word and transferring them to Reservation Control. The total process is called Instruction Issue, and the sorting of instructions is referred to as Parcelling.

INSTRUCTION ISSUE

Instructions can be issued from any rank of the stack, however if we were to assume an initial condition such as at the end of an Exchange Jump sequence we would see issue beginning with the upper instruction in I0. Once a program is in execution, program control can be transferred to some higher rank of the stack by a Branch instruction. This situation forces Instruction control to keep track of which rank of the stack the Program address is currently indicating. The "Locator" (L) register and counter perform this function. Control of the L count would be very similar to control of the Program Address. However, L refers to a particular rank of the stack so it would only vary between 0g through 7g. Example: L count = 0g indicates program control is currently in I0. L count = 7g indicates program control in I7. An initial Master Clear would set the L count to 0g, so we can see that Instruction Issue would start from the bottom rank of the stack.

NOTE: The L register contains the complement of the L count.

Now that we have selected a particular rank of the stack, we must concern ourselves with sorting out the instructions within that rank. In other words we must parcel the instructions from the selected rank of the stack. Each rank is considered to have four overlapping 30-bit Parcels.

These are: parcel 0 - bits 30 through 59
 parcel 1 - bits 15 through 44
 parcel 2 - bits 00 through 29
 parcel 3 - bits 44 through 14 (end around)

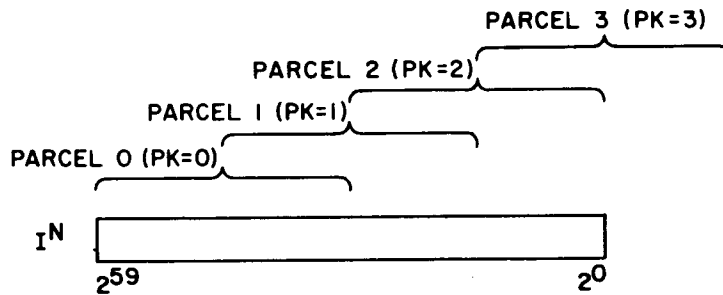
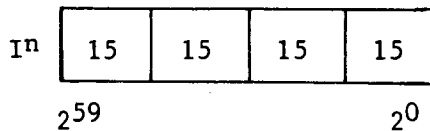


Figure 1-5 Instruction Word Parcels

By examining each word with the 30 bit parcels we guarantee the detection of any possible combination of instructions within that word. If the instruction contained within the parcel happens to be a 15 bit instruction the lower 15 bits of the parcel are discarded and the next sequential parcel is extracted. However, if a 30 bit instruction is encountered, the entire parcel would be used and the next sequential parcel would be skipped.

Example:



Here each parcel would be extracted in sequence with the lower 15 bits of each parcel being discarded.

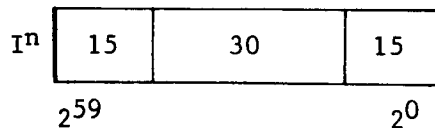
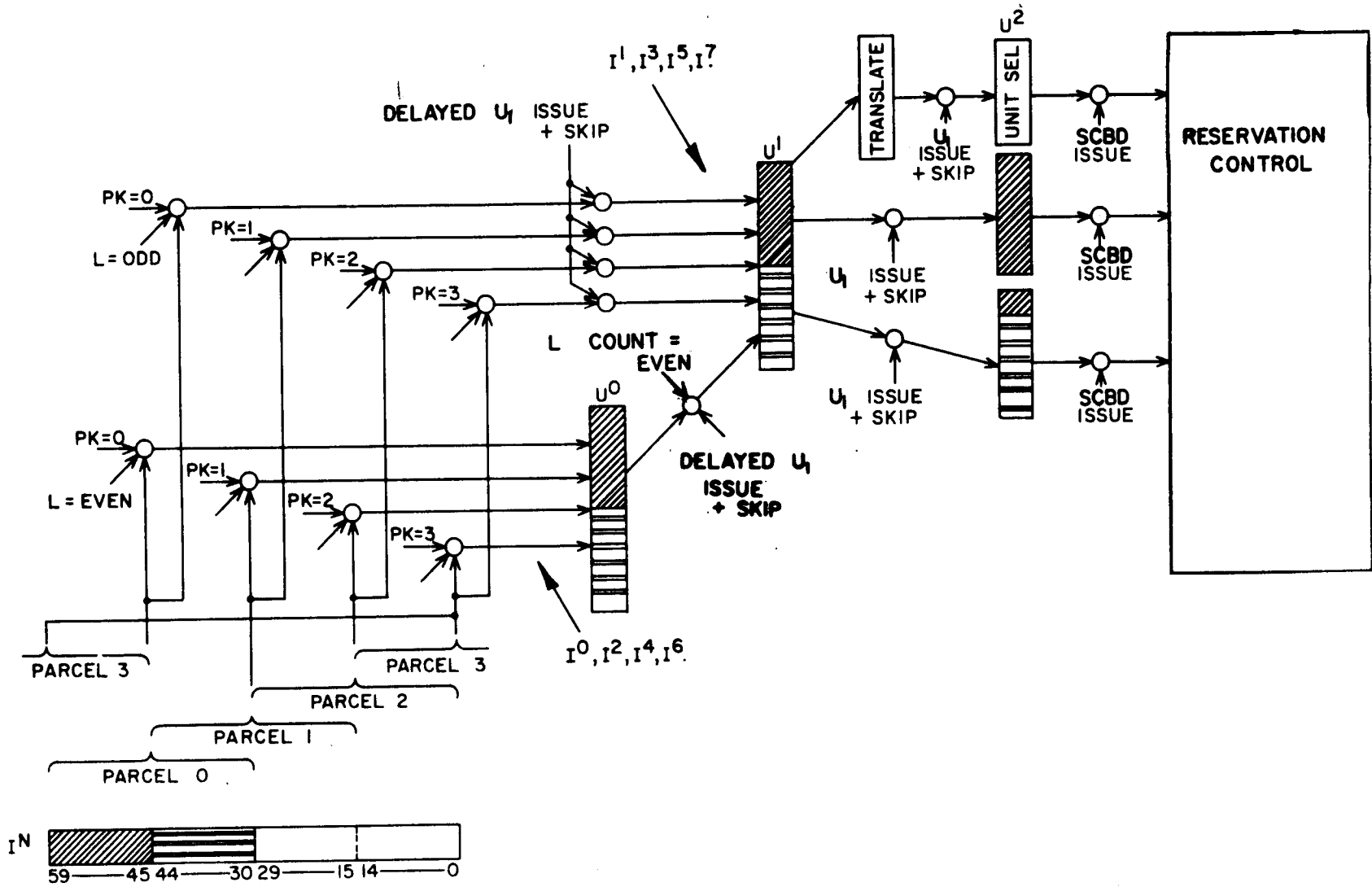


Figure 1-6 Parcel Counter



In this example parcel 0 and 1 would be extracted in sequence all 30 bits of parcel 1 would be used, so parcel 2 would be skipped.

Instruction Control uses a two bit Parcel Counter (PK) to keep track of the current parcel and will generally advance the parcel counter after each parcel is extracted. An initial PK = 0 condition would be set by Master Clear.

Initial Instruction issue begins with the L count = 0 (I0), PK = 0 (bits 30 through 59), and, as previously mentioned, the first Instruction word after an Exchange Jump comes from Central Memory to I0 as a result of the RNI tag. The tag also starts the Issue operation, so the rest of our analysis can now be concerned with moving the instructions to Reservation Control. Figure 1-B shows the path each parcel will take.

Issue Control, generates two types of issue pulses. These are:

U1 issue - A pulse that gates the selected parcel to the U1 and U2 instruction registers and advances PK. This pulse occurs at a minor cycle rate during the issue sequence.

Scoreboard

Issue - A pulse that gates the parcel from the U2 instruction register to Reservation Control. This issue can also occur at a minor cycle rate.

With eight different L counts and four different Parcel counts, it is easy to see that 32_{10} different parcels must move through the U1 and U2 registers. Sixteen of these parcels (PK = 0, 1, 2, 3 L count = 0, 2, 4, 6) move from the even-numbered ranks of the stack to U0 before U1 issue would move then to U1. It is not necessary to have an issue pulse to move parcels to U0, so we would see the selected parcel from the selected even rank of the stack move into U0 automatically. In our initial case Parcel 0 of I0 would be the first parcel extracted to U0 and the first U1 issue pulse would move the parcel to U1.

Notice that at this time there have not been meaningful parcels in U1 or U2, so as far as the U2 register is concerned it receives "Trash" on the first U1 issue. Also, no scoreboard issue should be generated until after the first meaningful parcel has moved into U2. The PK being advanced to 1 by U1 issue would cause parcel 1 of I0 to be extracted to U0, so on the next U1 issue parcel 1 would move to U1, parcel 0 would enter U2, and PK would advance to 2.

One more U1 issue would move parcel 2 to U1, parcel 1 from U1 to U2, so at this time the first Scoreboard issue must occur to issue parcel 0 to Reservation Control. From this point both U1 and Scoreboard issue can continue at a minor cycle rate until parcel 3 is issued to Reservation Control (three more issue pulses).

All of the possible instructions in I0 have now been put into

execution, and issue must stop until the next 60-bit Instruction word becomes available from Central Memory. This is called a "Pause".

The Pause could be quite lengthy if Instruction Control had not had the foresight to request another RNI from Central Memory. This request is made any time L count = 0, PK = 0, and U1 issue. It is easy to see why the request is made under those conditions, once it is realized that I0 is the bottom rank of the stack and after issuing from I0 there wouldn't be any place to go for the next instruction. There is one other operation that comes into play at this time, and it is the process of moving the current instruction words of the stack up to make room for the next instruction word from Central Memory (Inching).

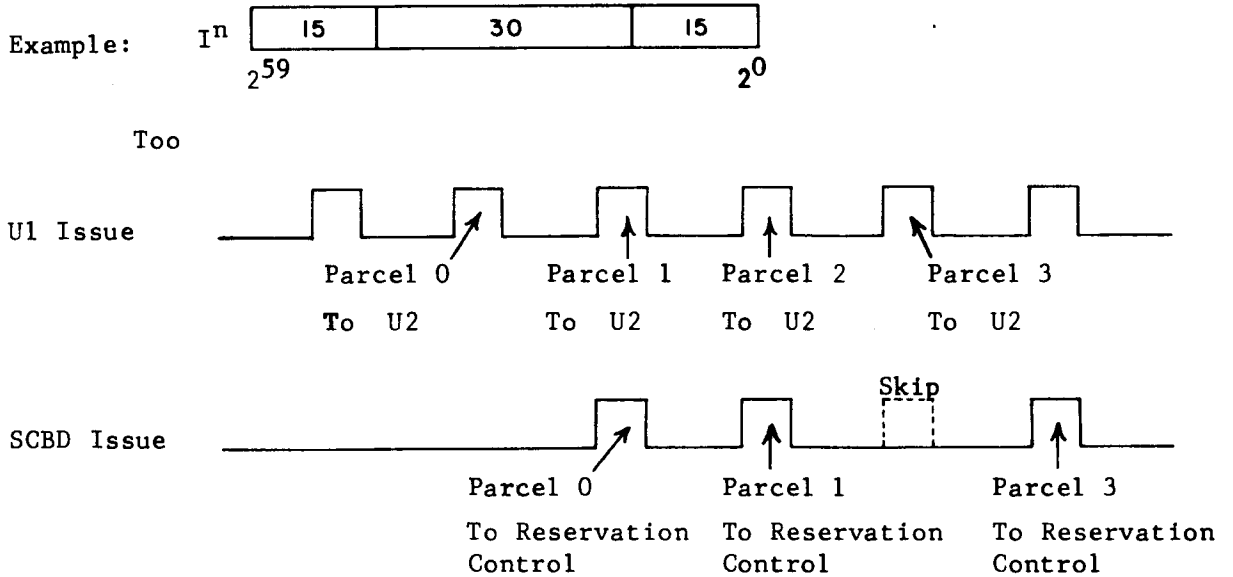
Inch is also started when L count = 0, PK = 0, and U1 issue. During the Inch process each rank of the stack is shifted up starting with an I6 to I7 transfer and continuing with I5 to I6, I4 to I5, I3 to I4, I2 to I3, I1 to I2, and I0 to I1 in that sequence.

Four minor cycles (400 nsecs) are necessary to complete the Inch, and it becomes necessary to advance the L count to 1, since the last Inch transfer moves the current Instruction word from I0 (L count = 0) to I1 (L count = 1). An important point to realize here is the I6 to I7 transfer destroys the Instruction word that was in I7. Consequently, a program loop that is to be executed within the Stack must fit in the stack between I1 and I7. A quick examination of the stack reveals a maximum in Stack Program length of 27₁₀ instructions.

I7	15	15	15	15
I6	15	15	15	15
I5	15	15	15	15
I4	15	15	15	15
I3	15	15	15	15
I2	15	15	15	15
I1	15	15	30 Bit Branch	
I0				

Figure 1-7 Maximum In Stack Loop

The analysis of Instruction Issue to this point has assumed straight line program execution with no complications. There are, however, many special situations which may be encountered. Whenever a 30-bit Instruction is encountered in a parcel, Instruction Control must cause the next sequential parcel to be skipped. The skipping is accomplished merely by not generating a Scoreboard issue when the unwanted parcel is in U2.



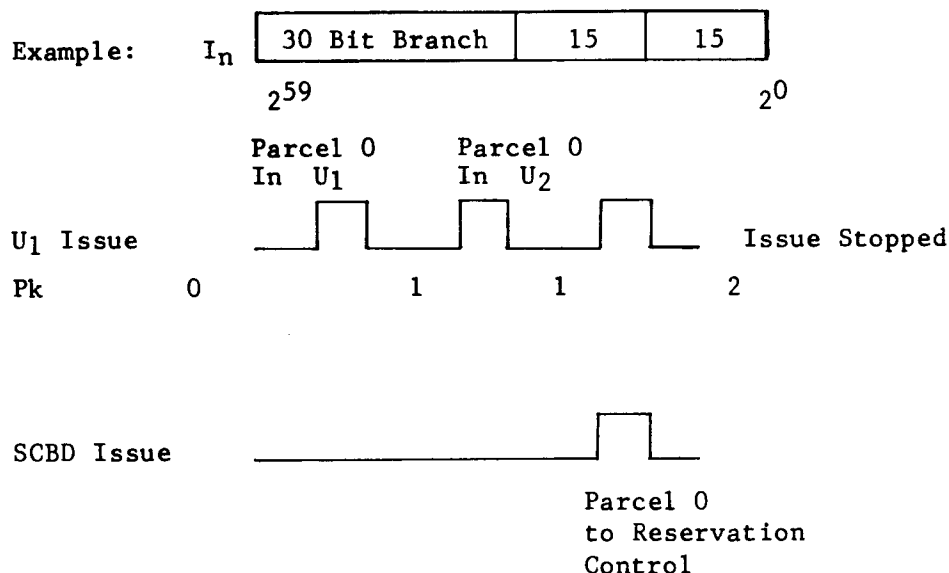
There is also the possibility that Issue may have to be stopped if either the Functional Unit or the Result register (required by the parcel) are Busy. This is accomplished by Instruction control translating the parcel when it is in U1 and setting a Unit Request FF and Result Register Select FF when the parcel enters U2. Each Unit Request FF interrogates the corresponding Unit Busy FF in Reservation Control, and, if the Unit is busy, a signal is generated which blocks all Issue pulses until the Unit becomes not busy.

A similar operation occurs with the Result register except the Result Register Select FF must be ANDed with a translation for the "i" portion of the Parcel to determine which portion of the Reservation List should be examined for a Reservation. The Reservation List (XBA) is where all Result register reservations are held by Reservation Control.

Probably the most involved operation in Instruction Control occurs when a Branch Instruction is encountered, and, even though the Branch Unit will execute the instruction, Issue Control must set itself up for proper operation. The reason for the complication of course, is that a Branch Instruction can do one of the following three:

- Loop - a conditional Branch, condition met, and In Stack.
- Jump - an unconditional Branch or a conditional branch, condition met, and not In Stack.
- No Branch - a conditional Branch, condition not met.

Instruction Control will set itself up for the No Branch Condition by setting the Parcel Count equal to the parcel count of the next parcel to be issued, and then stopping Issue after issuing the Branch.



By controlling the Parcel count in this manner it is possible to restart issue the same as if it were starting after an RNI. Once the Branch has been issued, the Branch Unit makes a series of tests to determine whether the branch is to an instruction word already in the Stack.

The test results are only enabled on the Conditional Branch instructions 03₈ through 07₈ and enable these instructions to Loop. The first test is made by subtracting the current Program Address (P) from the Jump Address (R), and if the difference (T) is +7 or less the Branch may be In Stack.

However, a further test must be made to see if a jump of T places can be made relative to the current position in the Stack which is reflected by L. This is the L-T test and, if there is not an end-around borrow from the test, the branch still may be In Stack.

If R-P gave a positive result, the jump was forward and the L-T test being made successfully would say In Stack, but if R-P was negative, the branch was backward and a further test must be made to see if there is a usable instruction in the rank of the stack to which the jump is being made. Conveniently, the result of the L-T test would be the new L setting if the Branch is to be made, and this is subtracted from the Stack Depth Counter register (D). The D-(L-T) test is only necessary on the backward jumps (R-P negative) and, if it is successfully made, the branch would be In Stack. The Branch unit uses the Long Add unit to make the Branch Condition test for the 03_g instructions, and the Increment units to test the 04_g through 07_g instructions. If the condition is met, a Go Branch signal is generated. If the R-P, L-T, D-(L-T) have all been successful, a Loop Proceed is generated.

If Go Branch occurs and R-P, L-T, and D-(L-7) were not successful, a Jump is generated. If Go Branch does not occur, a No Branch Proceed is generated. On a Loop Proceed, the Jump Address (R) is transferred to P, the result of the L-T test is gated to L, the Parcel Count is set to 0_g, and Issue is restarted. On the Jump an R to P transfer is also accomplished, but L is set to 7_g (L count = 0_g), the Parcel count is cleared, and an RNI request is made to the Stunt Box. (Issue would restart as a result of the RNI.) The No Branch Proceed merely restarts Issue, since this is why Instruction Control has been set up. There are many special cases that affect Instruction Control during branch instructions, but these will be covered in Section 4 along with a more detailed explanation of the other Instruction Control operations.

RESERVATION CONTROL
(SCOREBOARD)

The need for reservation control logic in the 6600 Central Processor arises due to the parallel processing concept of the CPU. This capability necessitates an orderly means of utilizing the functional units, operating registers and memory circuitry, since it is possible that several instructions require the same functional unit, operating register, etc. The scoreboard, then, makes the required reservations of each instruction and provides a means for the orderly handling of conflicts which may occur between instructions.

Conflicts are categorized into three groups - first, second and third order. The types of conflicts are defined as follows:

- 1) FIRST ORDER: A conflict between two instructions that require the same functional unit or the same result registers.

EXAMPLE 1: Functional Unit Conflict

$$\begin{aligned} \text{F X 6} &= \text{X1} \oplus \text{X2} \\ \text{F X 5} &= \text{X3} \oplus \text{X4} \end{aligned}$$

Both instructions need the Floating Add functional unit for their calculation. Since only one such unit exists, the second instruction must wait until the first is finished, before it can be executed. Note that if two Multiply instructions are coded in sequence, no functional unit conflict occurs since two multiply units are provided.

EXAMPLE 2: Result Register Conflict

$$\begin{aligned} \text{F X6} &= \text{X1} + \text{X2} \\ \text{F X6} &= \text{X4} * \text{X5} \end{aligned}$$

Both instructions require X6 for their result. In this case, the Floating Add result would be returned to X6 before the Multiply result was desired.

There are then, two types of First Order Conflicts - functional unit and result register. In all cases of first order conflicts, issuance of instructions stops until the conflict is resolved. In other words, no further instructions are initiated (including the one which "sees" the conflict) until the first of the conflicting instructions has completed. In conclusion, first order conflicts temporarily stop issuance of instructions at the point of conflict.

- 2) SECOND ORDER: A conflict that occurs when an instruction requires the result register of a previously initiated instruction as a source operand.

EXAMPLE:

$$\begin{aligned} F \text{ X6} &= X1 + X2 \\ F \text{ X7} &= X5 / \text{X6} \end{aligned}$$

In this case, the Divide unit needs X6, which is the result of the Add instruction, as one of its source operands. The Divide Unit must obviously wait for the Add unit to time out, but instruction issue will not stop. Instead, the Scoreboard will delay the start of the Divide instruction until the Add unit has stored its result. Subsequent instructions may be issued as long as no First Order conflicts exist.

The result of a Second Order Conflict is to delay the execution of the conflicting instruction only.

- 3) THIRD ORDER: A conflict that occurs when one instruction must store its result in a register which is to be used as a source operand for a previously issued instruction.

EXAMPLE:

$$\begin{aligned} F \text{ X3} &= X1 / X2 \\ F \text{ X5} &= \text{X4} * X3 \\ F \text{ X4} &= X0 + X6 \end{aligned}$$

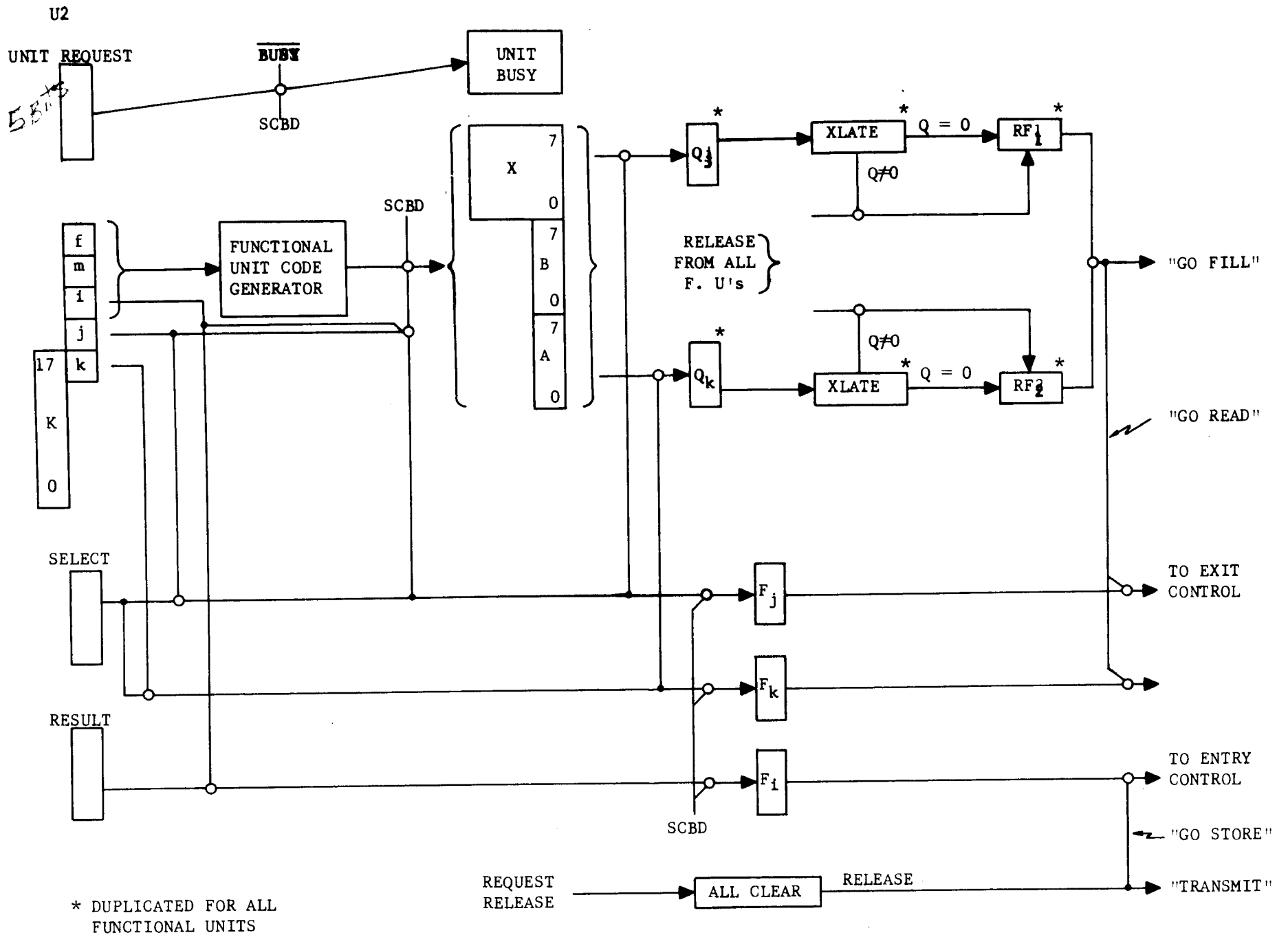
In this example, due to the relatively long execution times of the Divide and Multiply op. codes and the second order conflict (X3) of these units, the Add instruction will complete its calculation before the Multiply unit has read its operands (both operands are always read at the same time; therefore, all second order conflicts must be resolved). Since the Multiply instruction is intended to read X4 before it is changed by the Add instruction, storage of the Add result must be delayed until the Multiply Unit begins its calculation. Thus, third order conflicts do not delay issue or calculation, but rather the storage of a result operand.

The following discussion explains, at the block diagram level, how these conflicts are handled by the scoreboard. Figure 1-8 should be used in following the explanation.

FIRST-ORDER CONFLICTS

First-order conflicts are defined as either functional unit or result register conflicts. If either type exists, no issues can be generated.

Functional unit conflicts are determined by checking the Unit Request flip-flops against the Unit Busy flip-flops. Recall that Unit Requests are set at the



SCOREBOARD BLOCK DIAGRAM
Figure 1

time an instruction is transferred from U1 to U2. The Unit Request flip-flop sets depending upon the op.code that is translated (from U1 fmi portion). When an instruction is issued to the scoreboard, among other things, a Unit Busy flip-flop will be set. This in effect, reserves a functional unit for a particular op.code - it will remain reserved until the given instruction has completed execution, at which time the Unit Busy flip-flop is cleared to allow a subsequent instruction to use that functional unit. Thus, when a given unit is needed (as determined by a set Unit Request flip-flop) it can be used only if the associated Unit Busy flip-flop is cleared. If the Unit Busy is set, a functional unit conflict exists and generation of issues is disabled until the unit is freed.

In determining the existance of result register conflicts, a comparison of the request and reservation logic is also made. Upon issuing an instruction to U2, Result flip-flops are set according to the op.code translation. Four such flip-flops exist and specify a result register group (i.e. Xi, Bi, Ai or Bj). The specific register within a group is determined by translating the i or j octals (as specified by the Result flip-flops). For example, if the Xi flip-flop is set and the U2 i digit = 3, X3 is the result register desired.

The result register reservations are placed in the "XBA reservation list" when an instruction is issued to the scoreboard. This list is composed of 24 "slots", where-in codes are placed to specify which functional unit has reserved each of the 24 operating registers. For example, a code of 16₍₈₎ in the X4 slot of the reservation list indicates that X4 is reserved for the result of the LONG ADD functional Unit. The complete list of possible codes follows:

UNIT	CODE
Increment 1	01
Increment 2	02
Shift	03
Boolean	04
Divide	05
Multiply 1	06
Multiply 2	07
Read Memory, Channel 1	11
Read Memory, Channel 2	12
Read Memory, Channel 3	13
Read Memory, Channel 4	14
Read Memory, Channel 5	15
Long Add	16
Add	17

Any slot that contains an all-zero code indicates that the associated operating register is not reserved. Any non-zero code indicates that the associated register is reserved for a result. Thus, if translation of U2 indicates that X4 is specified as a result register and the X4 slot of the reservation list is zero, no conflict occurs. If the X4 slot is not equal to zero, the register is reserved. Thus, a conflict exists and issues are disabled until the conflict is resolved.

Notice from the list of codes that 5 are named Read Memory, Channel X. These are necessary for the 5 X 1 - 5 X 5 instructions. They return results from Memory to X 1 - X 5 and must make a result register reservation. In this sense, Memory acts like a functional unit.

In summary, both cases of first-order conflicts are handled similarly in that requests for units or result registers (made at U2 time) are checked against reservations existing in the Scoreboard. If a conflict exists, issues are disabled until the conflict is resolved.

SECOND-ORDER CONFLICTS

Second-order conflicts occur when a functional unit requires as a source operand, the result of another functional unit. The source operands are defined by the j and k octals of U2 in conjunction with the select flip-flops which are set with a U2 transfer. The select flip-flops define the source register group as well as the octal digit specifying the register within that group (i.e. Xj, Bj, Aj, Bk or Xk). By ANDing select flip-flops with the j and k octal digit translations, specific registers are selected.

Determination of whether or not the desired registers are reserved is made by looking at the content of the XBA reservation list, but not directly. Each functional unit has 4-bit Q designators which, when an instruction is issued to the scoreboard, receive the contents of the XBA slot associated with the desired source operand registers. For example, the following instruction sequence causes a second order conflict:

$$\begin{aligned} \text{F X 5} &= \text{X3} * \text{X2} \\ \text{F X 6} &= \text{X2} + \text{X5} \end{aligned}$$

The Multiply I unit reserves X5 by placing a code of 06₍₈₎ in the X5 slot of the reservation list. Assuming that no other instructions have been issued, no other reservations exist when the Add instruction is issued. Since the Add unit wished to read X2 and X5, it transfers to its Qj and Qk designators the content of the X2 and X5 slots, respectively. At this point, the Add Q designator equals 00₍₈₎ and Qk equals 06₍₈₎. In essence, this tells the Add unit that its j operand (X2) is not reserved by the Multiply I unit. Since a functional unit does not begin calculation until after it can read both operands, the Add unit must wait until Multiply I returns its result to X5.

Associated with each functional unit are flip-flops, called Read Flags, which when set, indicate that the desired operand (s) can be read. The Add unit has two Read Flags, one for the Xj operand and one for Xk. Read Flags can be set in two ways, both of which result from translating the Q designators.

- 1) If $Q = 00(8)$, a Read Flag can be set since the desired operand is not reserved.
- 2) If $Q \neq 00(8)$, a Read Flag cannot be set until the functional unit, whose code is in Q , has completed its calculation and returned its result to the result register. Completion of a functional unit's operation is indicated by a signal called Release (discussed in detail under third order conflicts).

In the above example, the Add unit's X_j Read Flag is set immediately, since $Q_j = 0$. The X_k Read Flag is set when the Release for Multiply 1 occurs, since Q_k translates as $06(8)$. Each possible non-zero Q translation is tied to the "Release" signal for the associated functional unit, so it is possible to set a Read Flag by any translation of Q , ANDed with the associated "Release" signal or, by $Q = 00(8)$.

Once both Read Flags are set, it is necessary to send the functional unit its operands and to send a Go signal to the unit, allowing it to begin its calculation. The Go F.U. signal is sent as soon as both Read Flags are set. This signal starts the functional unit timing chain. At the same time, the source register selection codes are sent (by a Go Read signal) to Register Exit Control to gate the proper operands to the unit. These codes are obtained from the F designators associated with each unit. These are 3-bit designators which are used to remember the source and result operand register numbers. They also are set when the scoreboard is issuing an instruction. In the above example, once both Read Flags are set, the content of the F_j and F_k designators of the Add unit are sent to Register Exit Control and will allow X_2 and X_5 to be gated to the Add unit. The Read Flags are cleared during the minor cycle after both are set. Set Read Flags then, indicate that an operand is waiting to be read.

Thus, the general second-order conflict case delays the start of a functional unit until both source operands can be read. Some special cases exist, which are discussed in detail in the logic analysis sections of this manual. At this point, it is appropriate to understand the general case.

THIRD-ORDER CONFLICTS

The possibility of third-order conflicts occurs when a functional unit has generated a result and wishes to store in an operating register. If the desired result register is waiting to be read, the unit must wait to store until after the read has occurred.

Whether or not a register is waiting to read is determined by checking the F_j and F_k designators against the associated Read Flags in all the functional units. The result register of a unit is given by the F_i (and, in some cases, F_j) designator of that unit. When a unit requests to store a result, its result register number is checked against the Read Flags and F designators of all other units. If any Read Flag is set AND the associated F_j or F_k designator translation is the same as the F_i designator of the storing unit, a third-order conflict exists. The unit will therefore be prevented from storing until the conflicting unit's Read Flag is cleared. This, of course, occurs once a unit has set both of its Read Flags.

The general sequence in handling third-order conflicts is as follows. First, a unit desiring to store a result sends a Request Release signal to the scoreboard near the end of its calculate time. This signal is then ANDed with an All Clear signal to generate the Release gate, which allows storage of the result. The All Clear is the result of checking all Read Flags with the associated Fj and Fk designators and comparing with the Fi (or Fj) designator (for the result) of the unit requesting release. The Release signal accomplishes several necessary tasks in the scoreboard. It sends a transmit signal to the functional unit to gate the result to the data trunk. It also generates a Go store signal which gates the Fi (or Fj) designator to Register Entry Control to select the desired result register. Release also clears reservations in the scoreboard (i.e. XBA designators, Unit busy flip-flops, etc.) and checks all Q designator translations in the event that a unit is waiting to use this result as a source operand. The Release then, indicates final termination of an instruction, and in essence, removes that instruction from the scoreboard.

REGISTER EXIT/ENTRY CONTROL

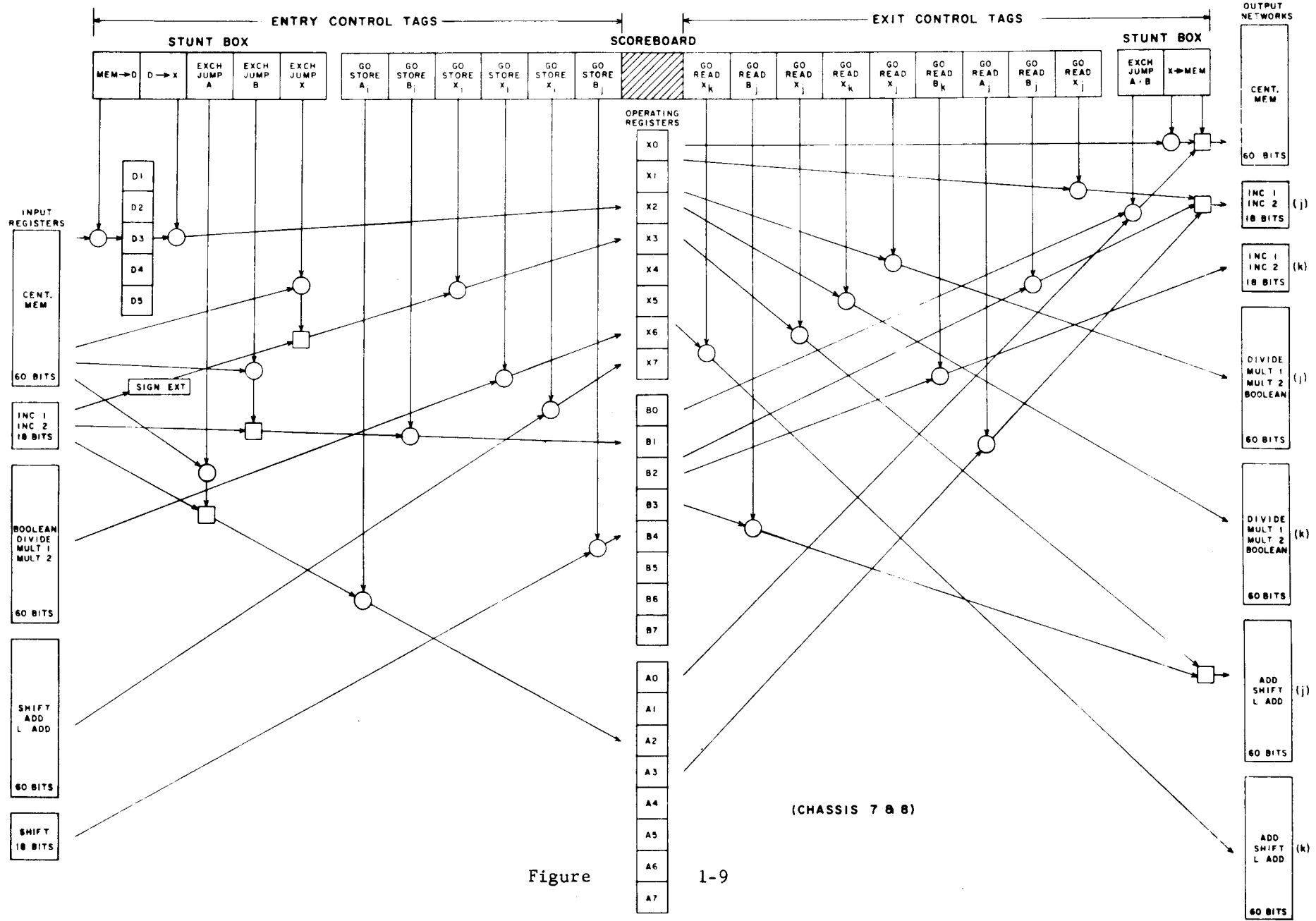
As the name implies, Register Exit/Entry Control is the control logic used for gating data into and out of the 24 operating registers. It is in essence, a large translating network which decodes tags sent from the Stunt Box or Scoreboard to enable the transfer of data to and from central memory or the functional units. Figure 1-9 is a block diagram which should be used during the following discussion.

ENTRY CONTROL

Entry Control is shown on the left half of Figure 1-9. To the extreme left are the four general sources of information for the X, B and A registers:

- 1) Central Memory
- 2) Data Trunk #1 (Shift, Add and L. Add)
- 3) Data Trunk #2 (Boolean, Divide, Multiply 1 and Multiply 2)
- 4) Data Trunk #3 (Increment 1 and 2)

Data is entered into the operating registers from Central Memory during Exchange Jumps and during the central read operand instructions (5 X 1 - 5 X 5). Since memory references are involved, all of the gating tags are sent from the Stunt Box and are composed of the lower four bits ($2^0 - 2^3$) of the Hopper Tag ANDed with the Accept signal for the associated address. In other words, when an Exchange Jump or a Read Operand address is accepted, the four-bit tag is sent from the tag timing chain to Entry Control where it enables the information from Central Memory to the proper X, B, or A register.



45

Figure 1-9

Notice that during Read Operand references, the 60-bit operand is first sent into the D register (1 - 5) associated with the X register (1 - 5) which will ultimately receive the information. This is enabled by the Mem → D signal which results from the simple translation: (tag 11 - 15) (Accept). The operand will be temporarily stored in D, until any third order conflict which may exist is resolved. (A Read Flag may be set for the X register which is to receive the operand from memory). Thus, when the All Clear signal occurs, the D → X signal is generated and completes the transfer to X.

During Exchange Jumps, new information is entered into A, B and X registers by Hopper tags in the range, 60 - 77. Recall, that tags 60 - 67 enable the exchange of A and B registers, while tags 70 - 77 enable exchanging X registers. The Entry Control Tags, Exchange Jump A, Exchange Jump B and Exchange Jump X refer to the hopper tage 60 - 77 accepted.

A result generated by the Increment units may be entered into X, B or A registers, depending upon the instruction being processed (5X, 6X, or 7X). Thus, three Entry Control Tags are shown for the Increment Data trunk, namely, Go Store Ai, Go Store Bi and Go Store Xi. Recall that Go Store occurs after a functional unit has been released and enables the Fi designator content to Entry Control. Thus, the Go Store tags are generated by the Scoreboard at the completion of an instruction sequence. Note also, that Sign Extension occurs when storing an Increment result (18 bits) in an X register (60 bits).

Results generated by the Boolean, Divide, Multiply 1 or Multiply 2 units are always 60 bits in length and the result register of these functional units is always an X register. Therefore, one Entry Control Tag, namely Go Store Xi is shown for Data Trunk #2. The tag is also generated from the Fi designators of the units on this trunk when the unit is Released by the Scoreboard.

The units on Data Trunk #1, Shift, Add, and Long Add all generate a 60-bit result for X registers, but in addition, the Shift unit may generate an 18-bit result for a Bj register. (For example, during normalize or unpack operations.) Thus two Entry Control Tags are shown for this trunk: Go Store Xi and Go Store Bj. These are also generated by the Scoreboard when a unit releases from the Fi (or Fj in the special shift case) designators.

EXIT CONTROL

Similar to Entry Control, Exit Control has four general destinations for data from the operating registers:

- 1) Central Memory
- 2) Data Trunk #1
- 3) Data Trunk #2
- 4) Data Trunk #3

The Exit Control Tags are also generated similarly, that is, from the Stunt Box or the Scoreboard.

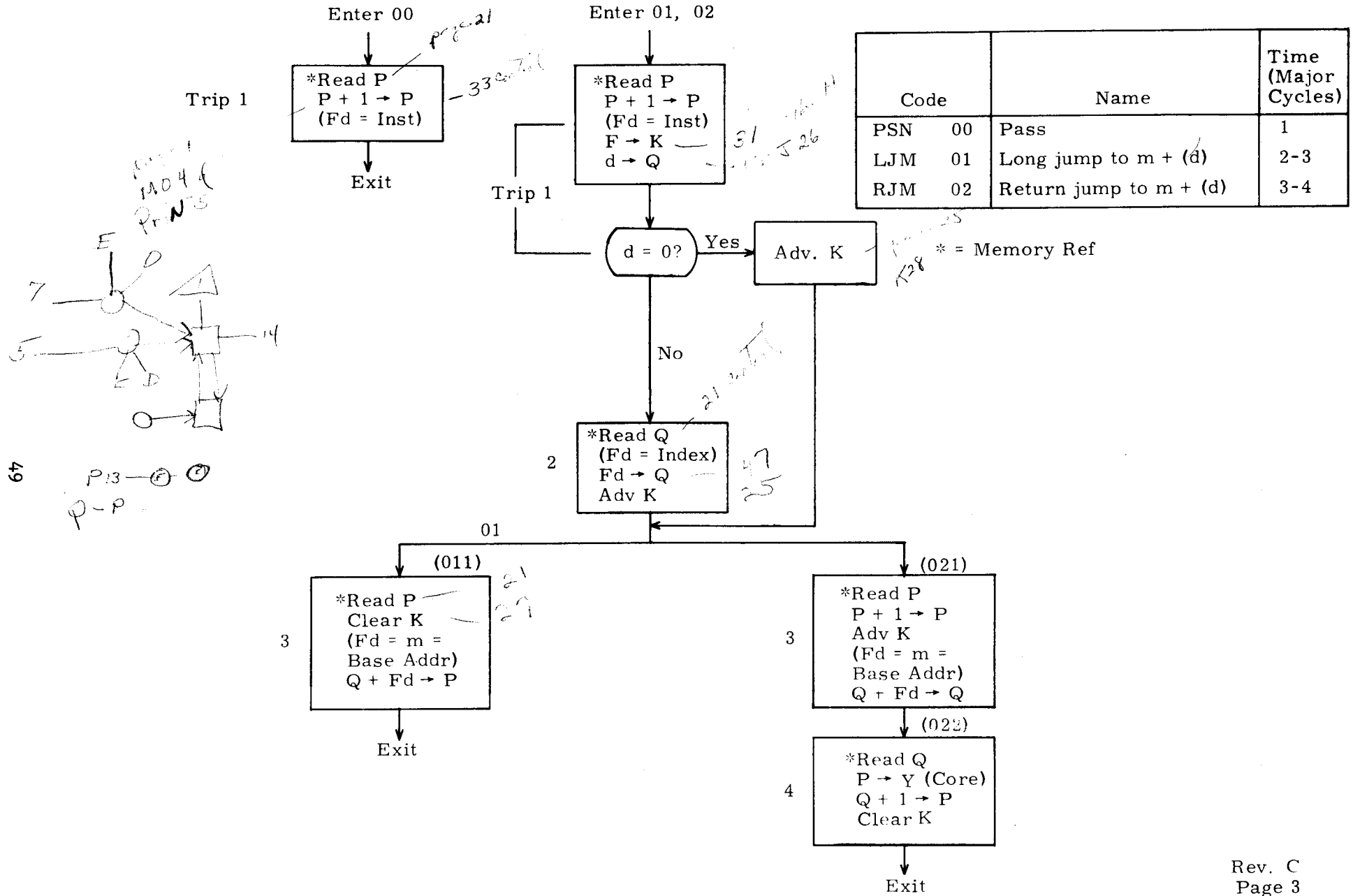
The Stunt Box generates tags for information to be sent to central memory, specifically, during Exchange Jumps or Central Store Operand instructions (5 X 6 or 5 X 7). During these operations, the lower four bits of the Hopper Tag are sent to Exit Control when the associated address has been Accepted (is not in conflict). The data, which may be A and B register or X register contents, are sent on the memory trunk and will be stored during the write portion of the memory cycle.

For the Increment Data Trunk, four Exit Control Tags are shown in Figure 1-9 since the Increment units may specify an A, B or X register with the j octal and only a B register with the k octal. Thus the four tags, Go Read Xj , Go Read Bj , Go Read Aj , and Go Read Bk are used to gate operands on this trunk.

For Data Trunk #2, all functional units specify only X registers as source operands. Therefore, only Go Read Xj and "Go Read Xk" tags are required.

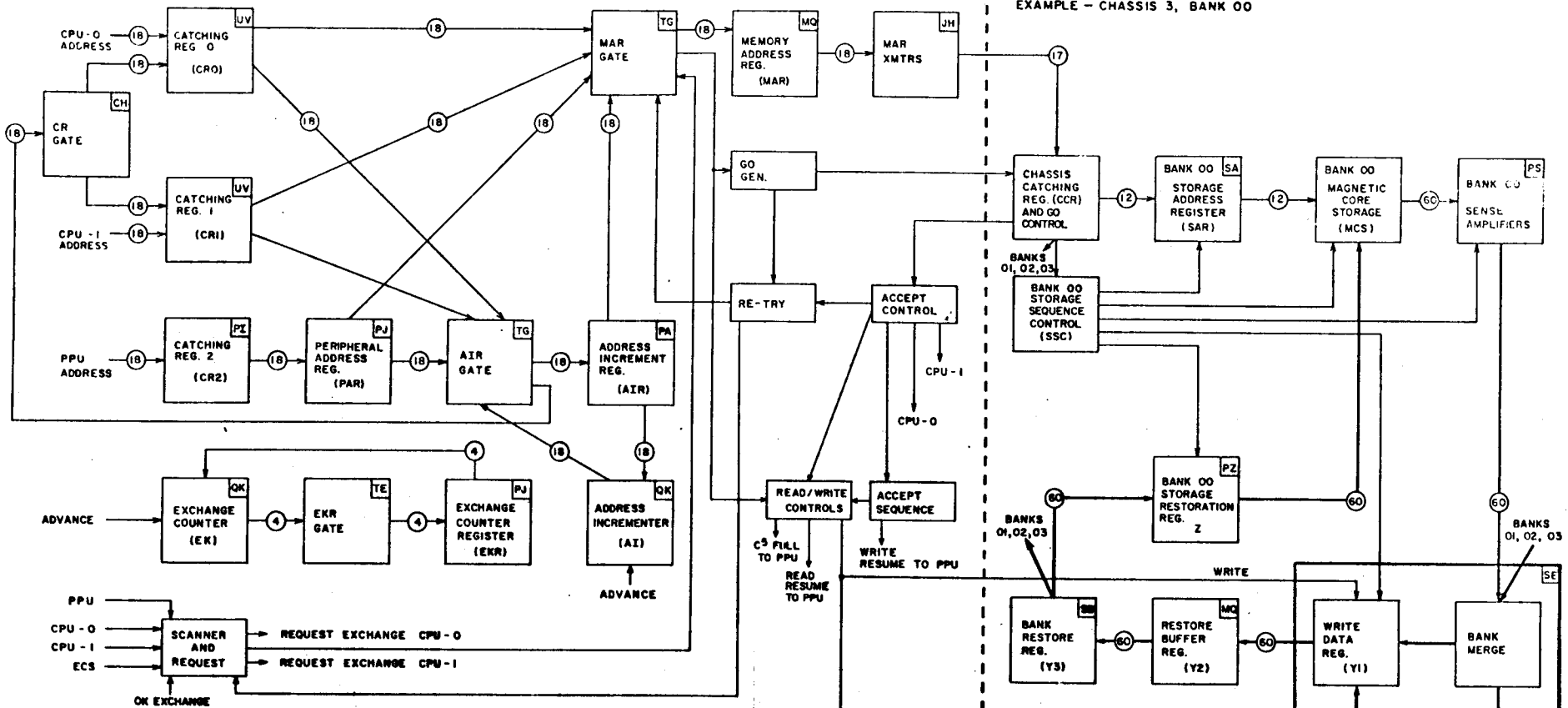
For Trunk #1, the Add, and L. Add units may specify an Xj or an Xk source register (or both) while the shift unit may specify an Xj or Bk register (or both). Thus the three tags, Go Read Xj , Go Read Bj and Go Read Xk are required for this data trunk.

All Go Read tags are generated by the Scoreboard when both Read Flags for a unit have been set. This enables the Fj or Fk designator to exit control and gates the proper register to the proper trunk.

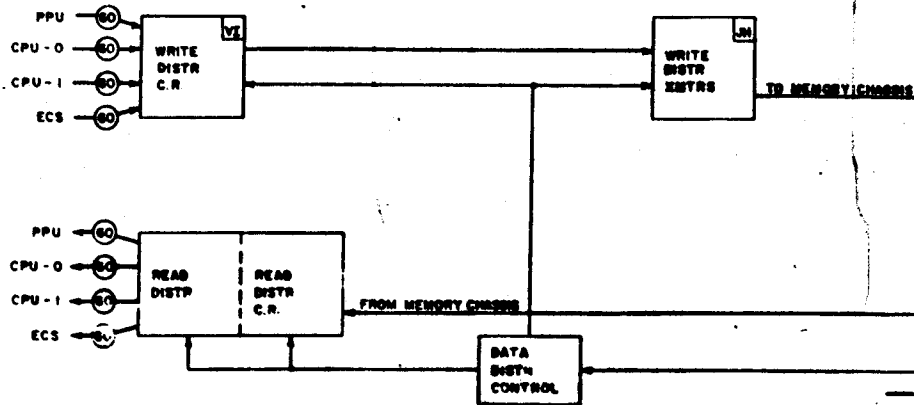


CENTRAL MEMORY CONTROL

CENTRAL MEMORY BANKS-TYPICAL MEMORY CHASSIS
EXAMPLE - CHASSIS 3, BANK 00



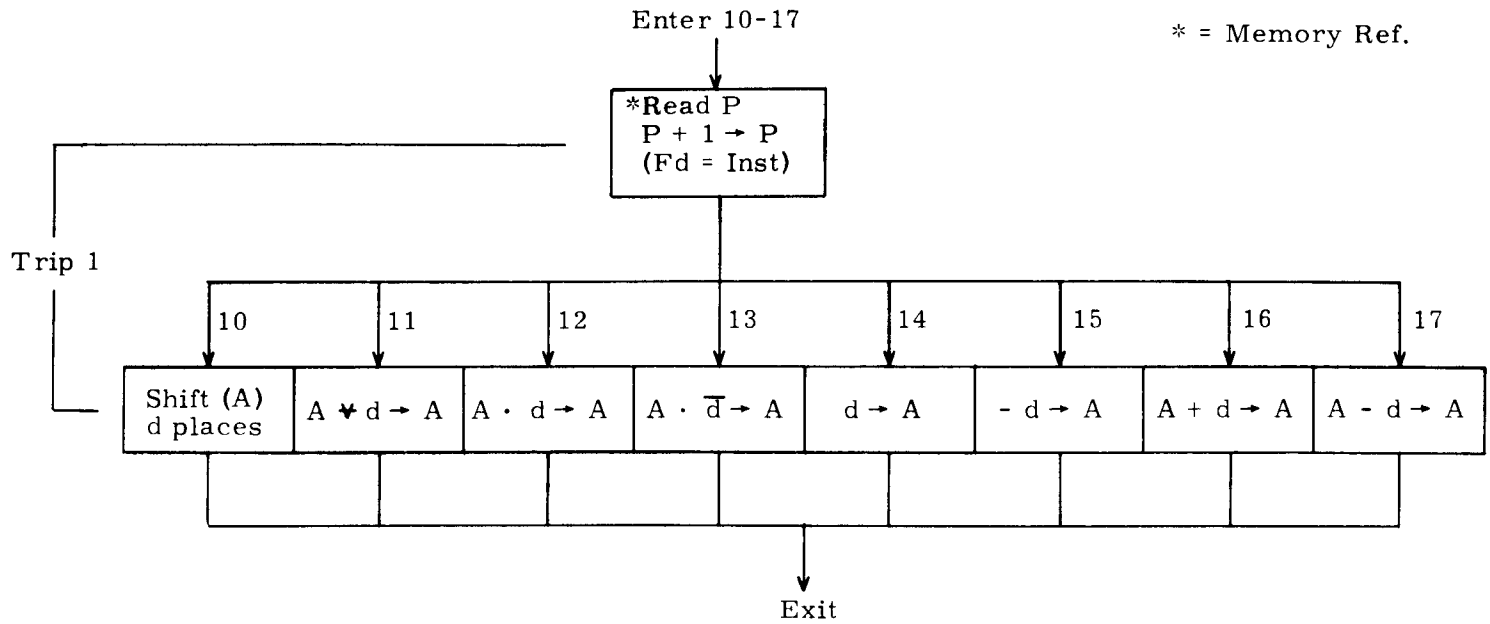
DATA DISTRIBUTOR



50

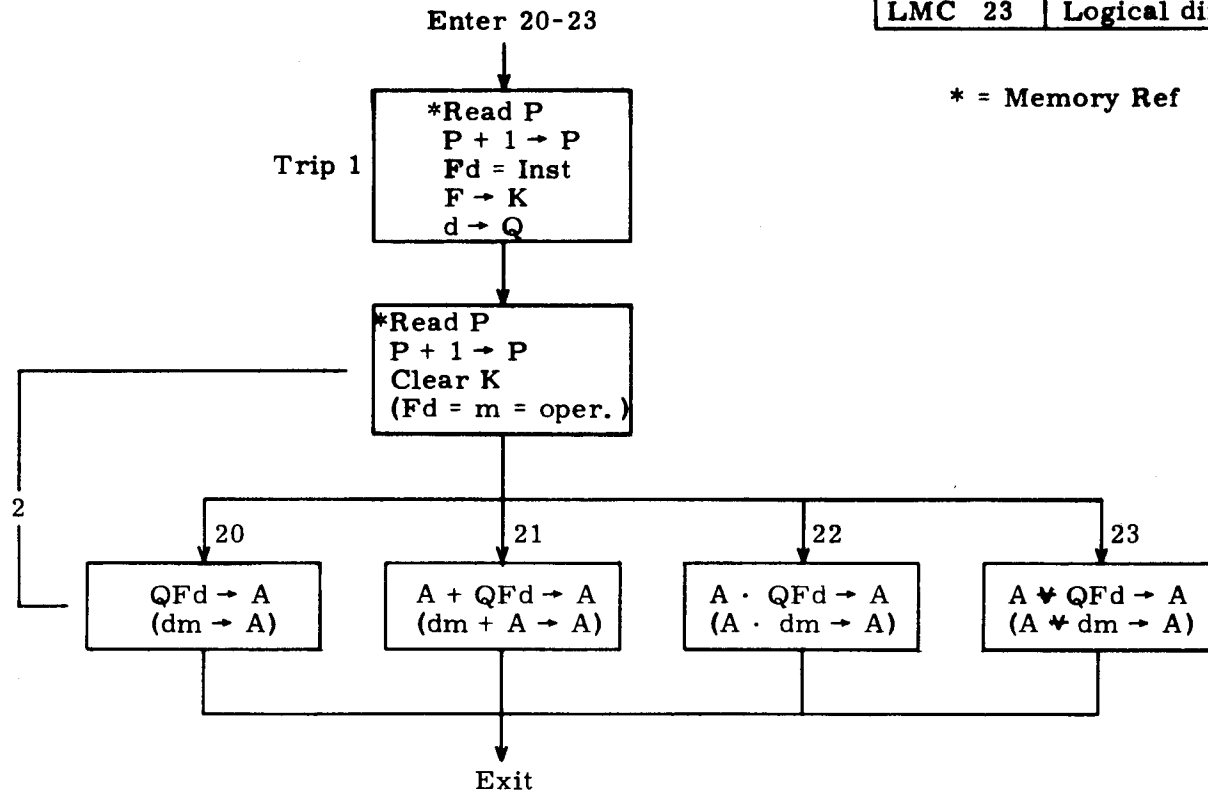
Code	Name	Time (Major Cycles)
SHN 10	Shift d	1
LMN 11	Logical difference d	1
LPN 12	Logical product d	1
SCN 13	Selective clear d	1
LDN 14	Load d	1
LCN 15	Load complement d	1
ADN 16	Add d	1
SBN 17	Subtract d	1

51



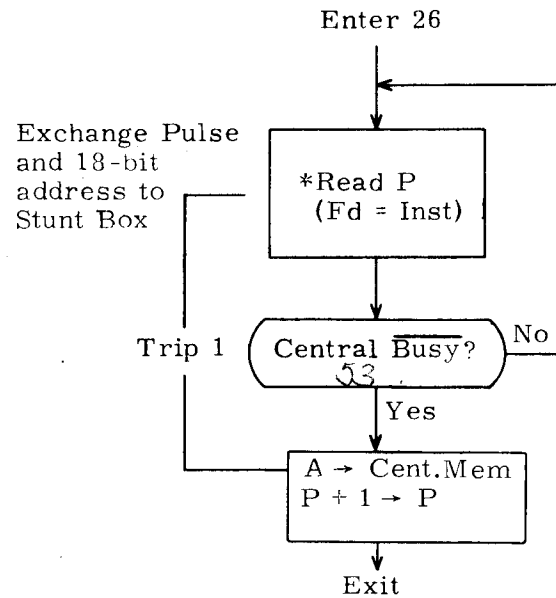
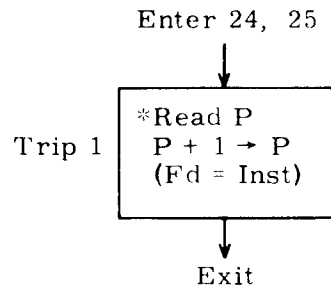
Code	Name	Time (Major Cycles)
LDC 20	Load dm	2
ADC 21	Add dm	2
LPC 22	Logical product dm	2
LMC 23	Logical difference dm	2

* = Memory Ref

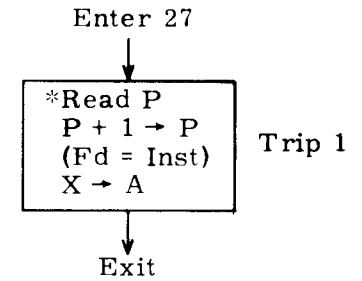


Code	Name	Time (Major Cycles)
PSN 24	Pass	1
PSN 25	Pass	1
EXN 26	Exchange jump	min. 2.0
RPN 27	Read program address	1

* = Memory Ref.

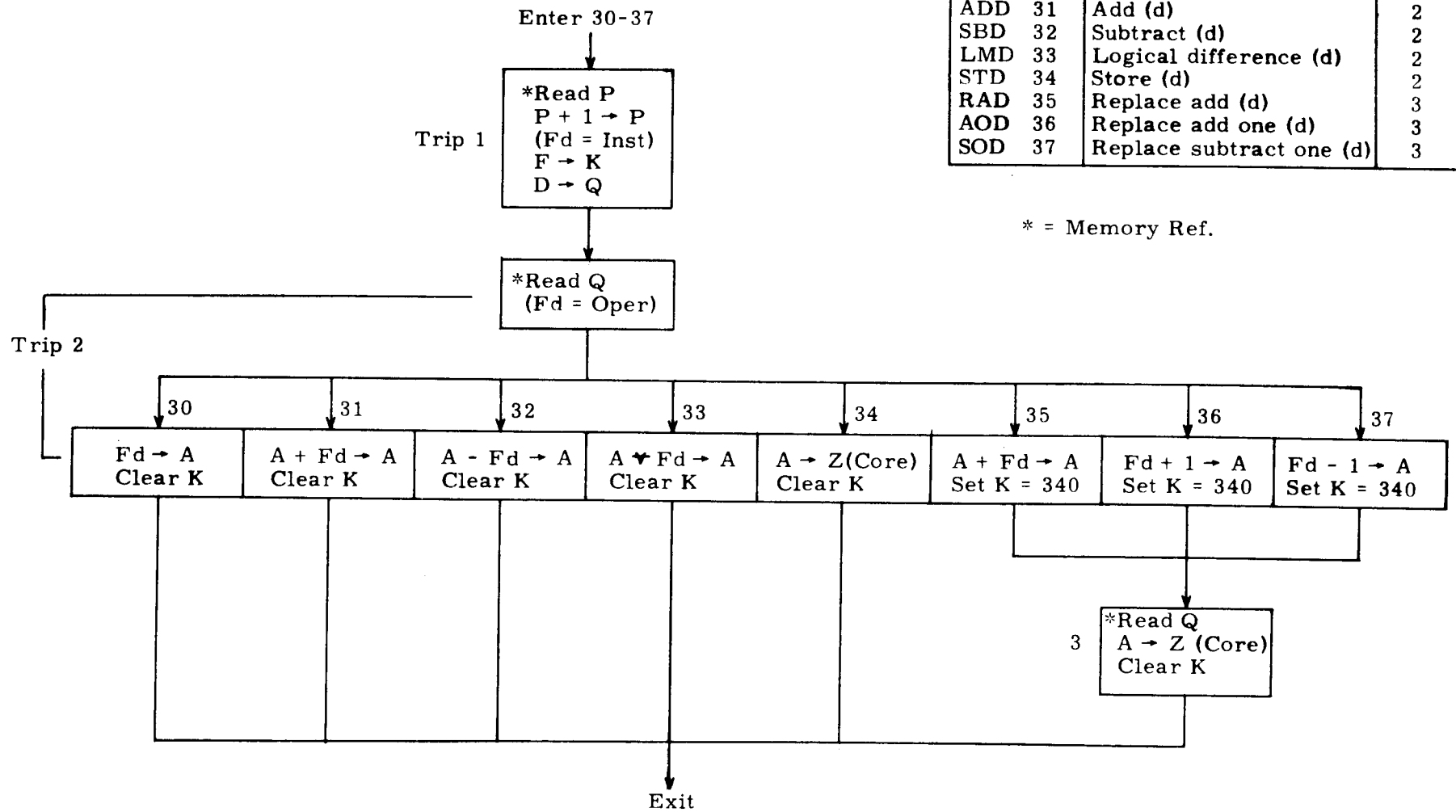


X = Central Proc (P) reg



Code	Name	Time (Major Cycles)
LDD 30	Load (d)	2
ADD 31	Add (d)	2
SBD 32	Subtract (d)	2
LMD 33	Logical difference (d)	2
STD 34	Store (d)	2
RAD 35	Replace add (d)	3
AOD 36	Replace add one (d)	3
SOD 37	Replace subtract one (d)	3

* = Memory Ref.



54

Enter 40-47

Trip 1

```

*Read P
P + 1 → P
(Fd = Inst)
F → K
d → Q

```

2

```

*Read Q
(Fd = Addr)
Fd → Q
Adv K

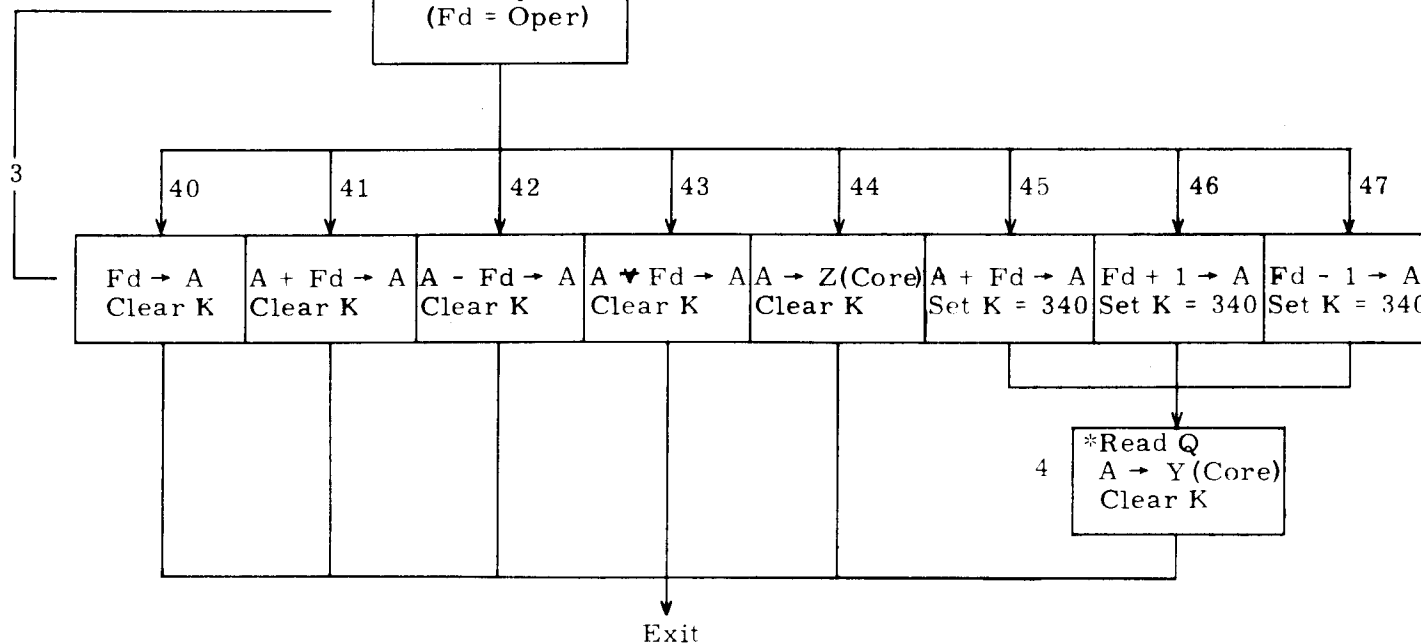
```

4X1

```

*Read Q
(Fd = Oper)

```

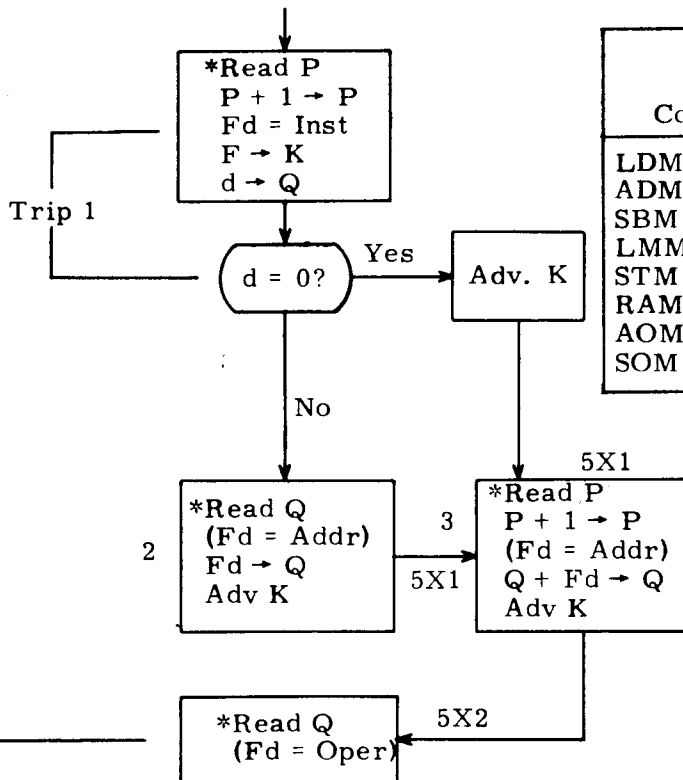


Code	Name	Time (Major Cycles)
LDI 40	Load ((d))	3
ADI 41	Add ((d))	3
SBI 42	Subtract ((d))	3
LMI 43	Logical difference ((d))	3
STI 44	Store ((d))	3
RAI 45	Replace add ((d))	4
AOI 46	Replace add one ((d))	4
SOI 47	Replace subtract one ((d))	4

* = Memory Ref

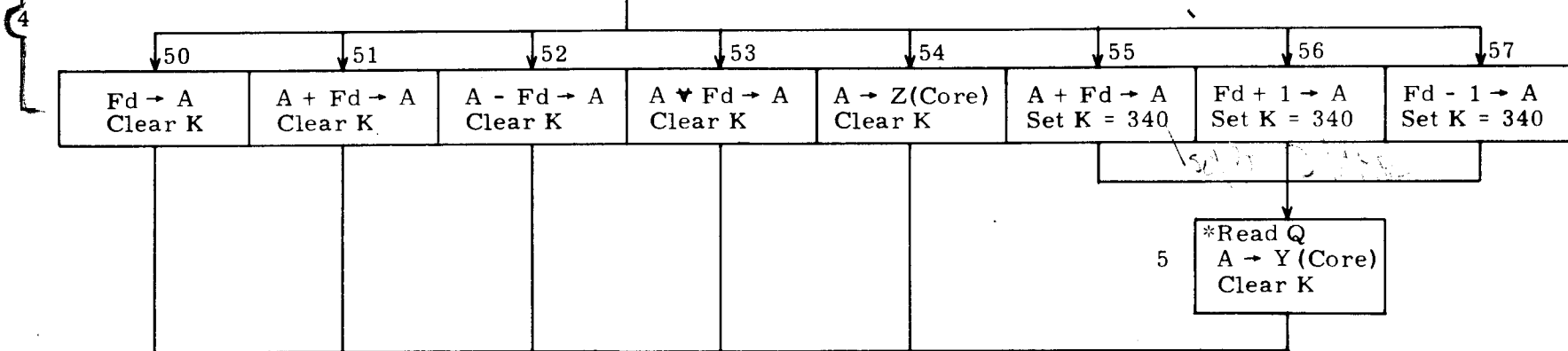
55

Enter 50-57

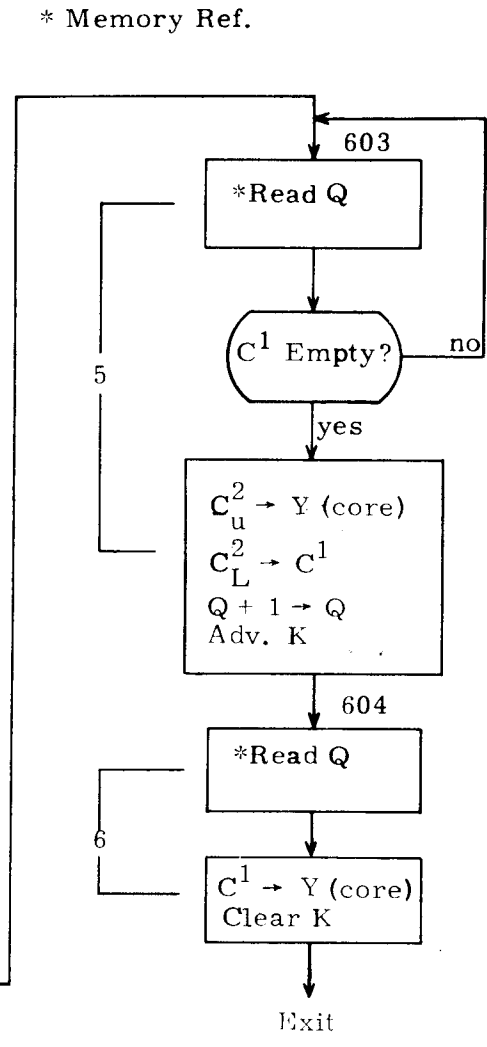
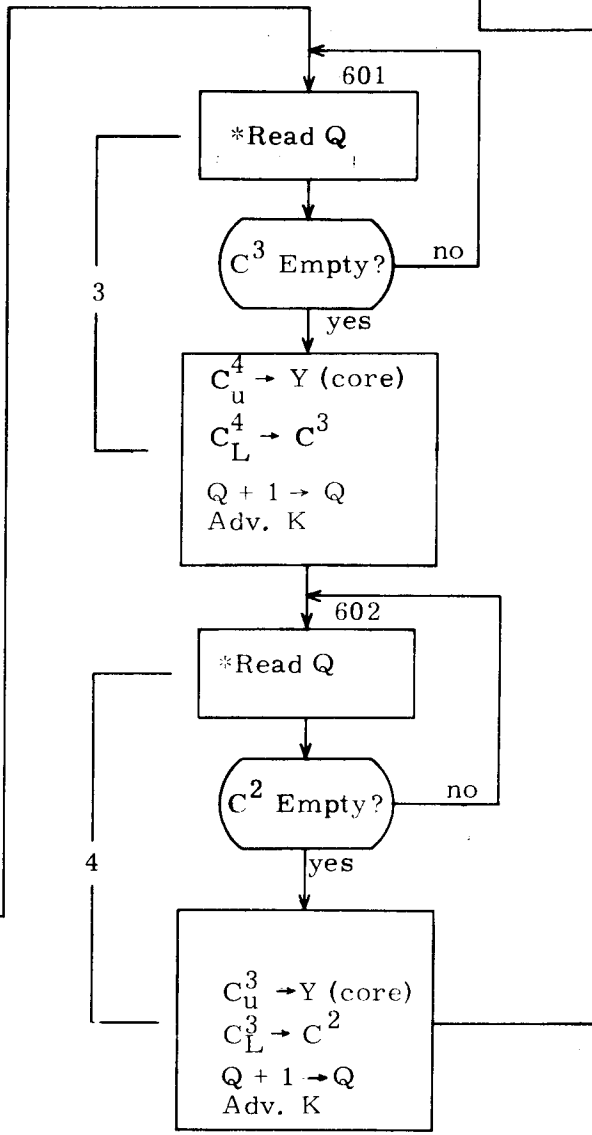
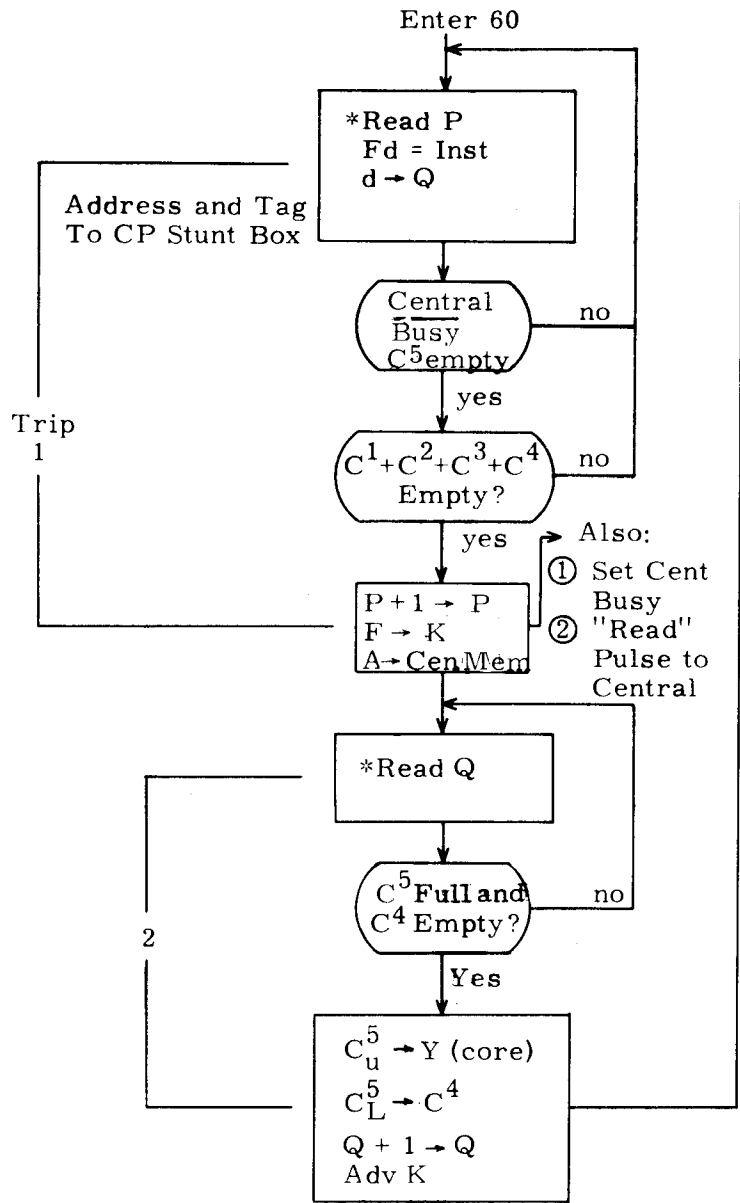


Code	Name	Time (Major Cycles)
LDM 50	Load (m + (d))	3-4
ADM 51	Add (m + (d))	3-4
SBM 52	Subtract (m + (d))	3-4
LMM 53	Logical Difference (m + (d))	3-4
STM 54	Store (m + (d))	3-4
RAM 55	Replace add (m + (d))	4-5
AOM 56	Replace add one (m + (d))	4-5
SOM 57	Replace subtract one (m + (d))	4-5

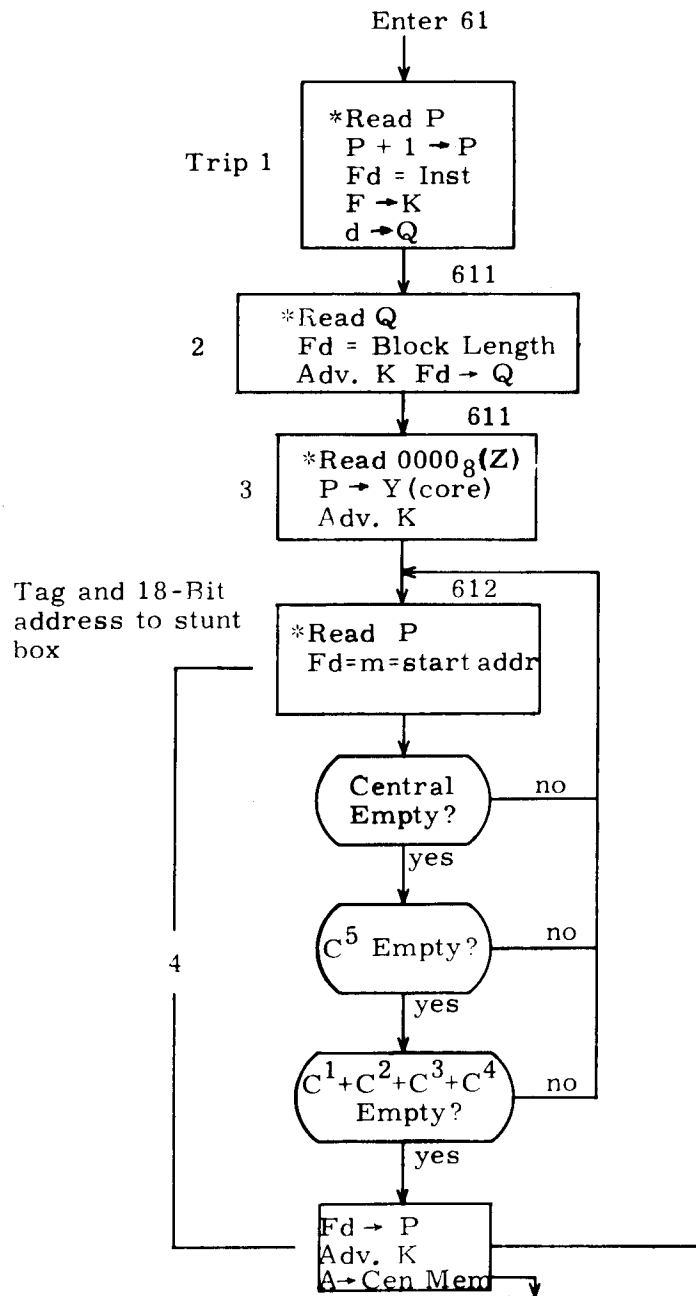
* = Memory Ref.



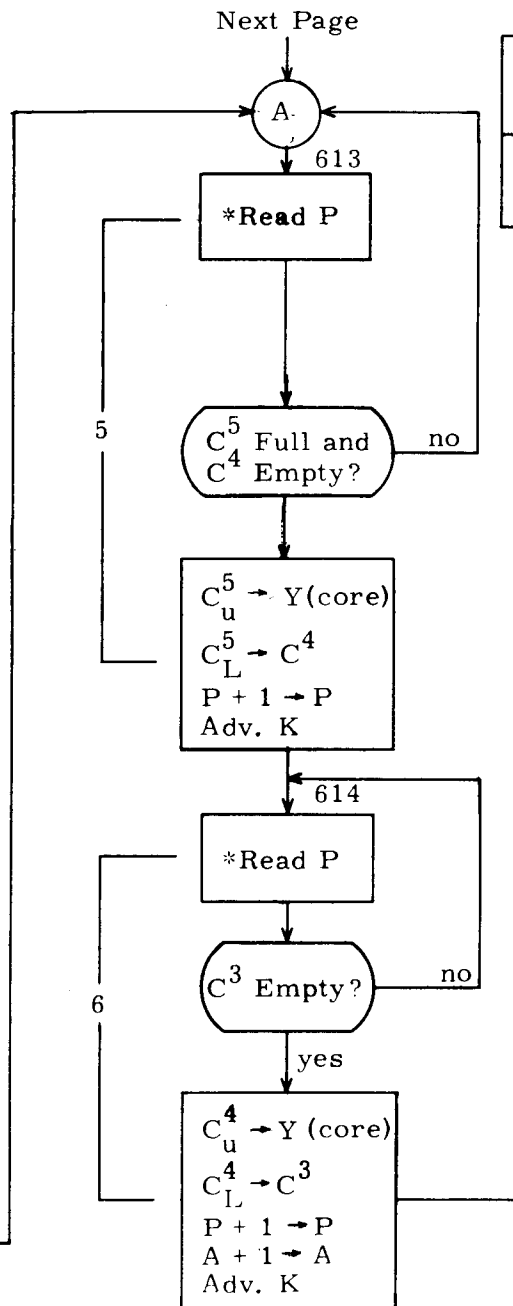
Exit



Code	Name	Time (Major Cycles)
CRD 60	Central read from (A) to d	min. 6

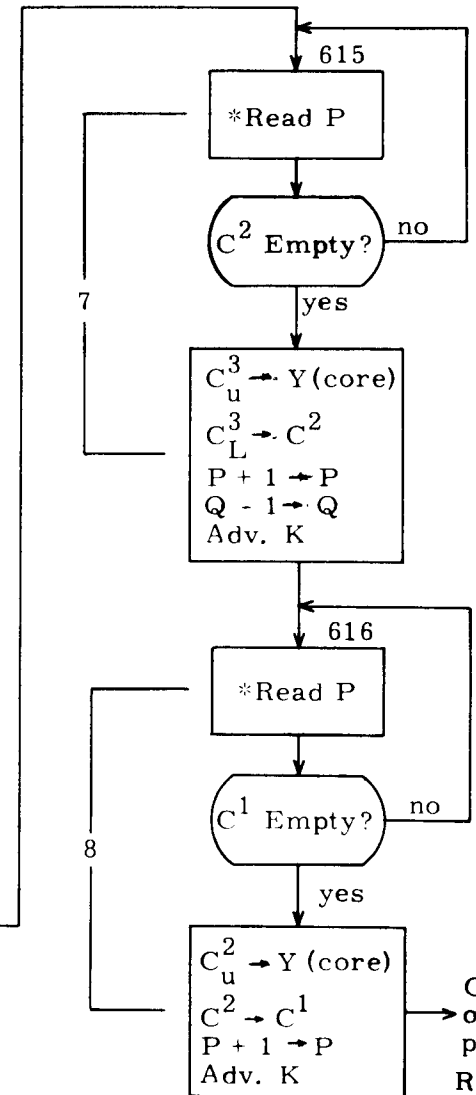


- Also:
- ① Set Cent Busy
 - ② Sent "Read" Pulse → Cent

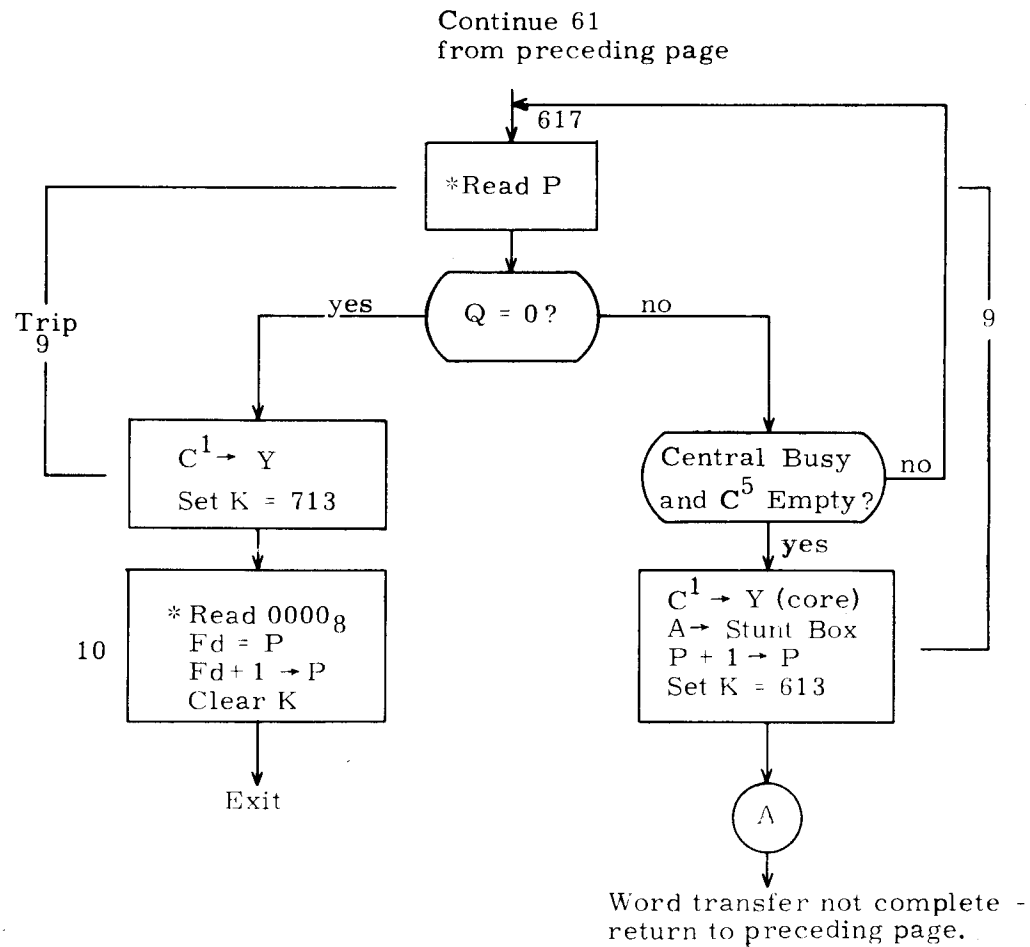


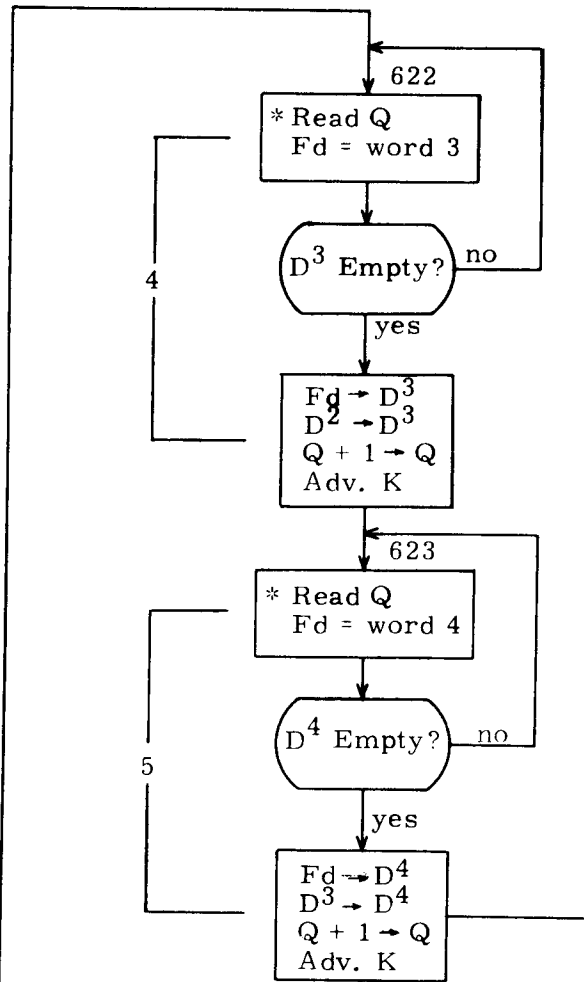
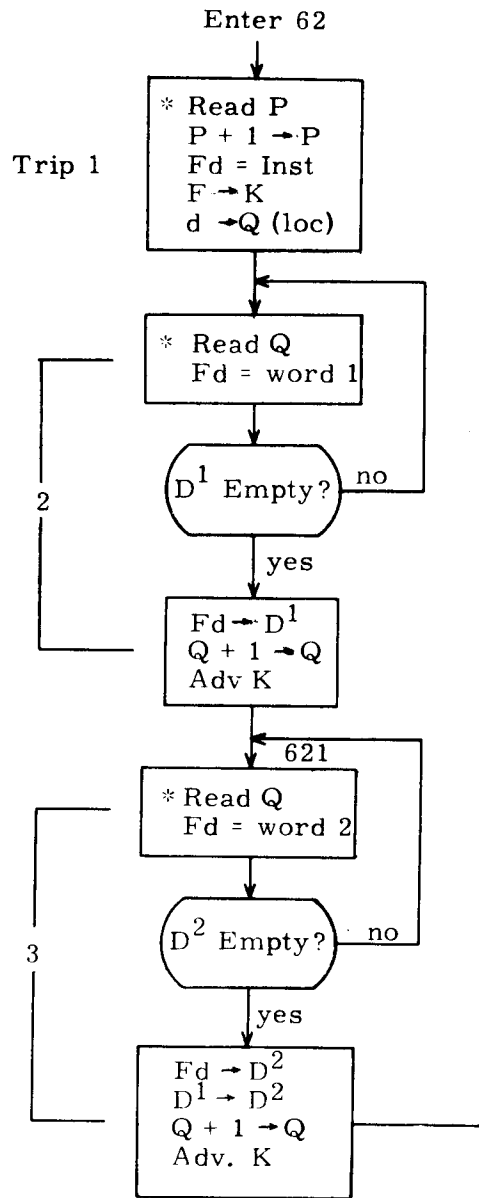
Code	Name	Time (Major cycles)
CRM 61	Central read (d) words from (A) to m	5 plus 5/word

* = Memory Ref.



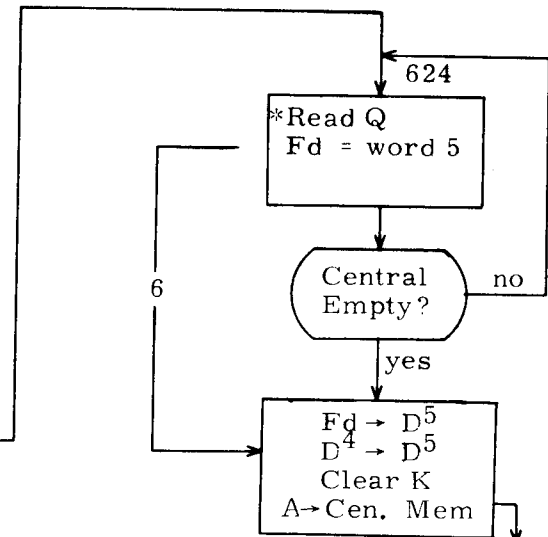
Continued on next page
Rev. C
Page 12





Code	Name	Time (Major Cycles)
CWD 62	Central write to (A) from d	min 6

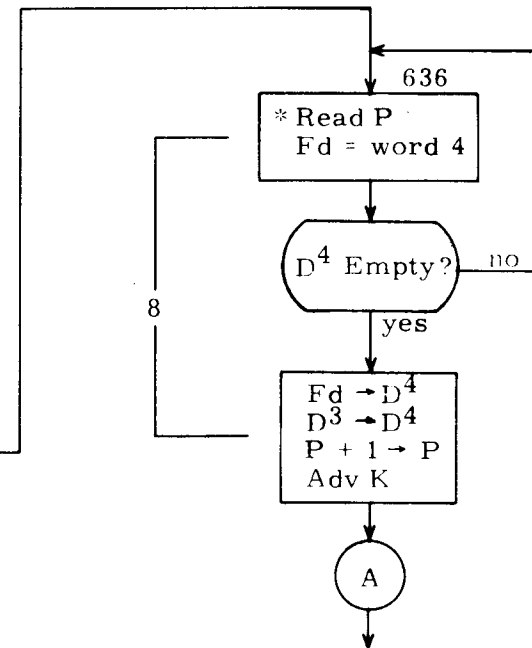
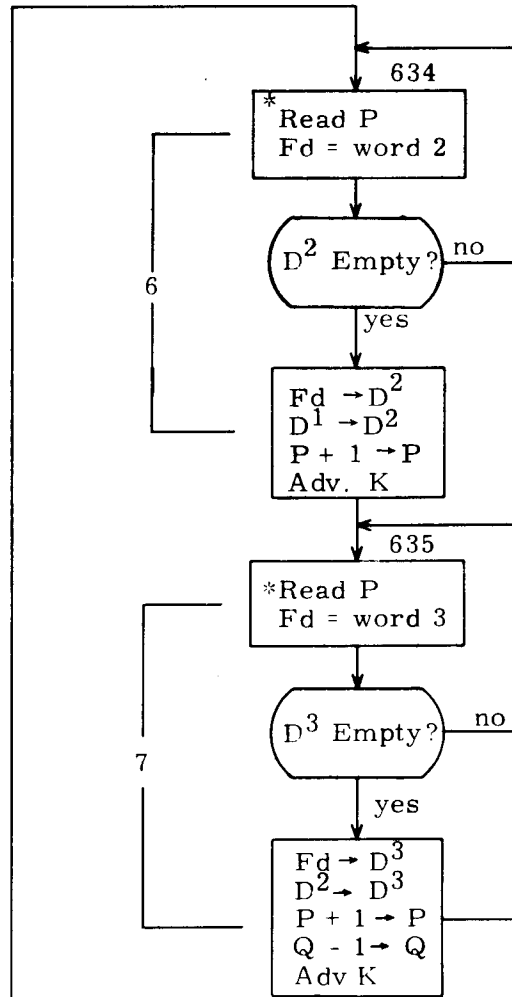
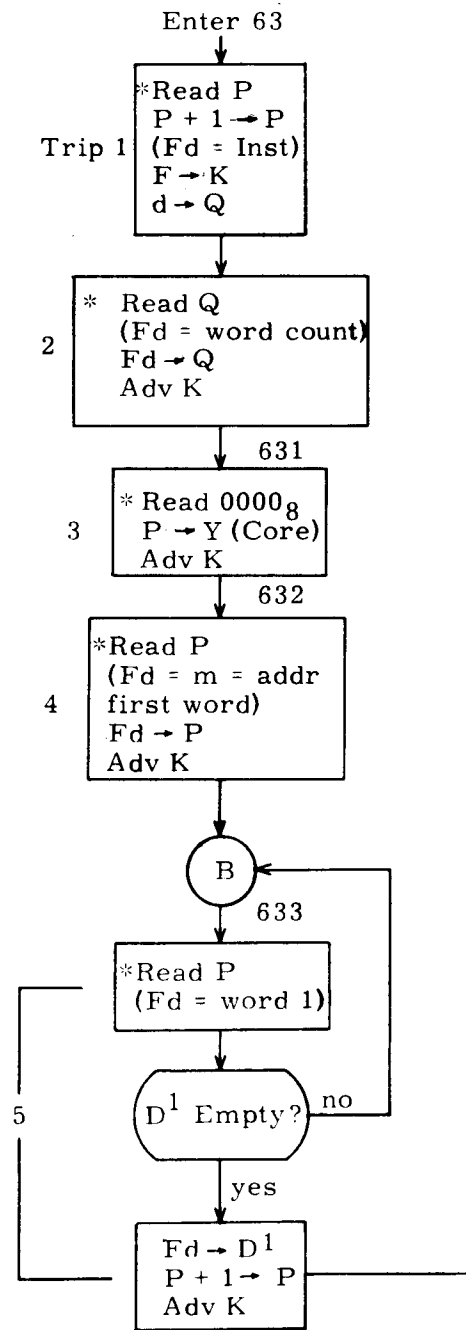
* = Memory Ref.



Transfer to D⁵ sends D⁵ to Central Memory

- Also:
- ① Send "Write" to Central
 - ② Set Cent Busy

This page left blank intentionally.

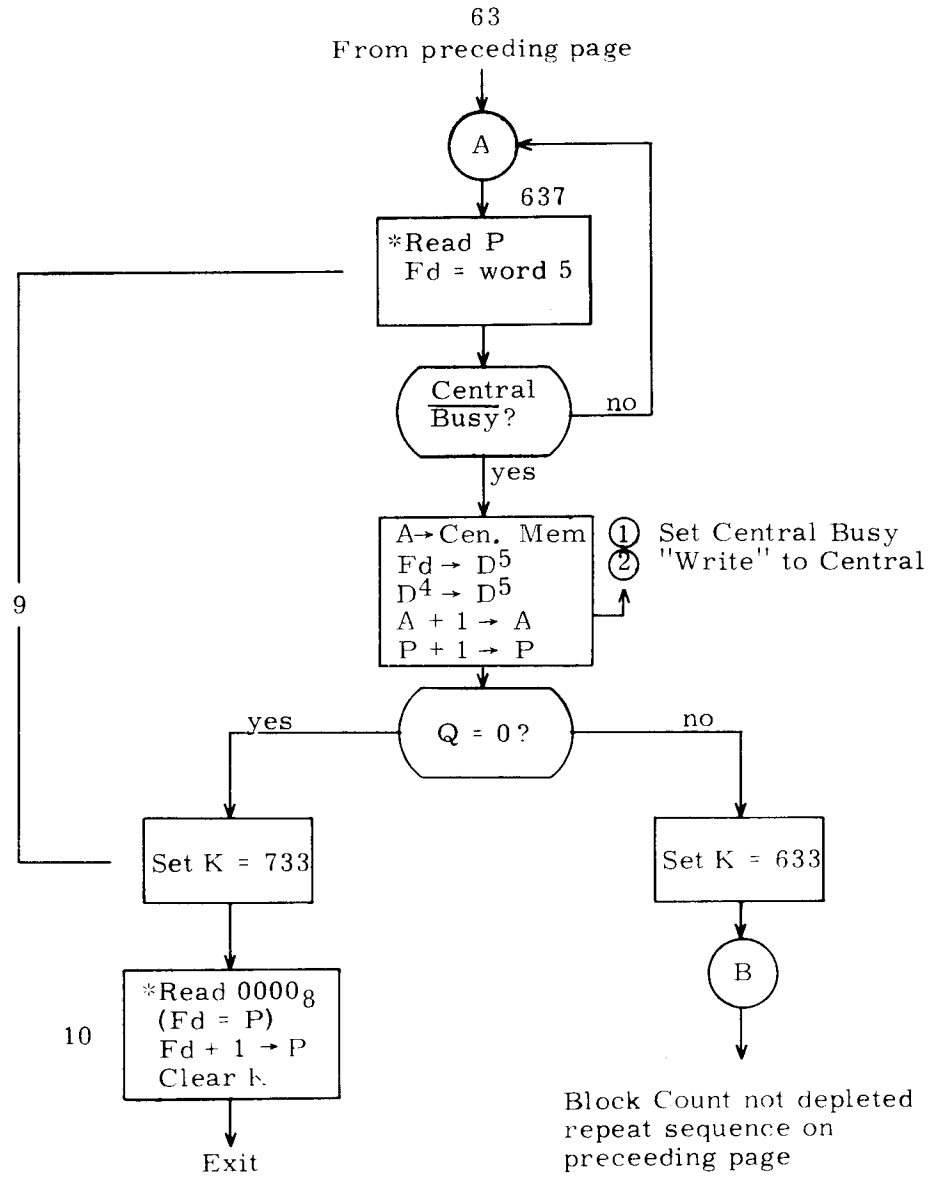


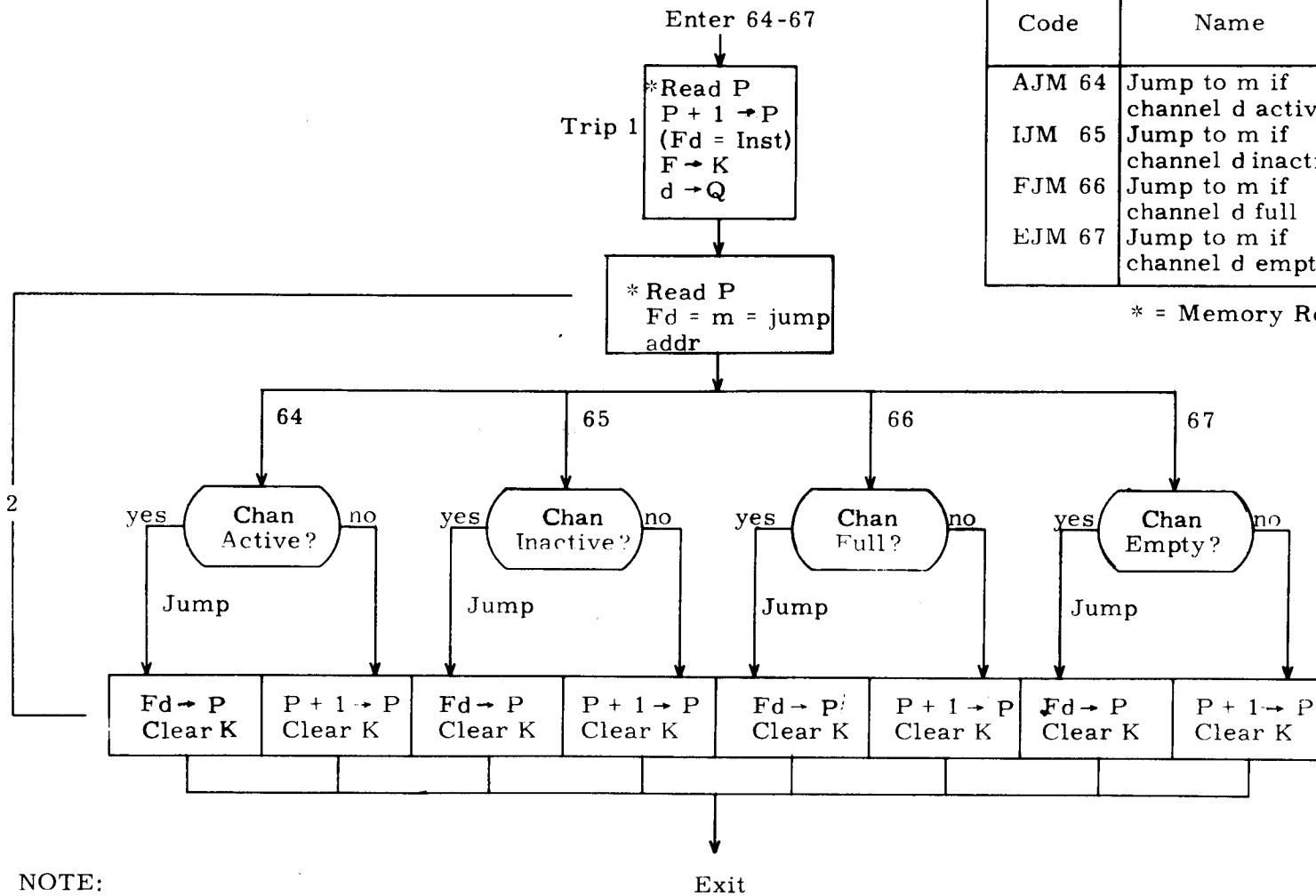
Code	Name	Time (Major cycles)
CWM 63	Central write (d) words to (A) from m	5 plus 5/word

* = Memory Ref.

Next Page

Rev. C
Page 16



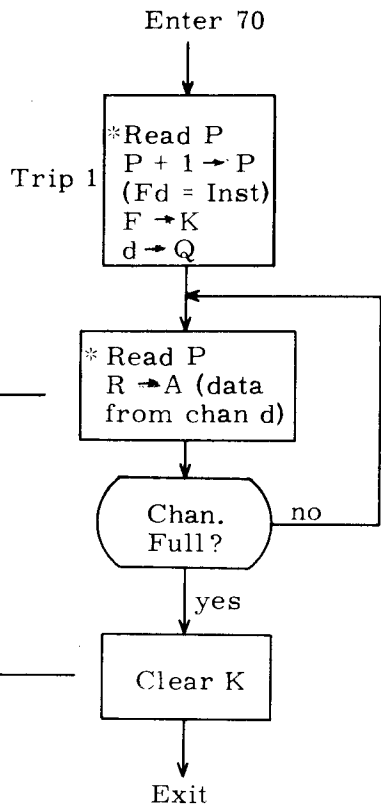


Code	Name	Time (Major Cycles)
AJM 64	Jump to m if channel d active	2
IJM 65	Jump to m if channel d inactive	2
FJM 66	Jump to m if channel d full	2
EJM 67	Jump to m if channel d empty	2

* = Memory Ref.

NOTE:

- FD → P → ① Fd → H
- ② Q → Q (Clears Q)
- ③ Q Adder → P

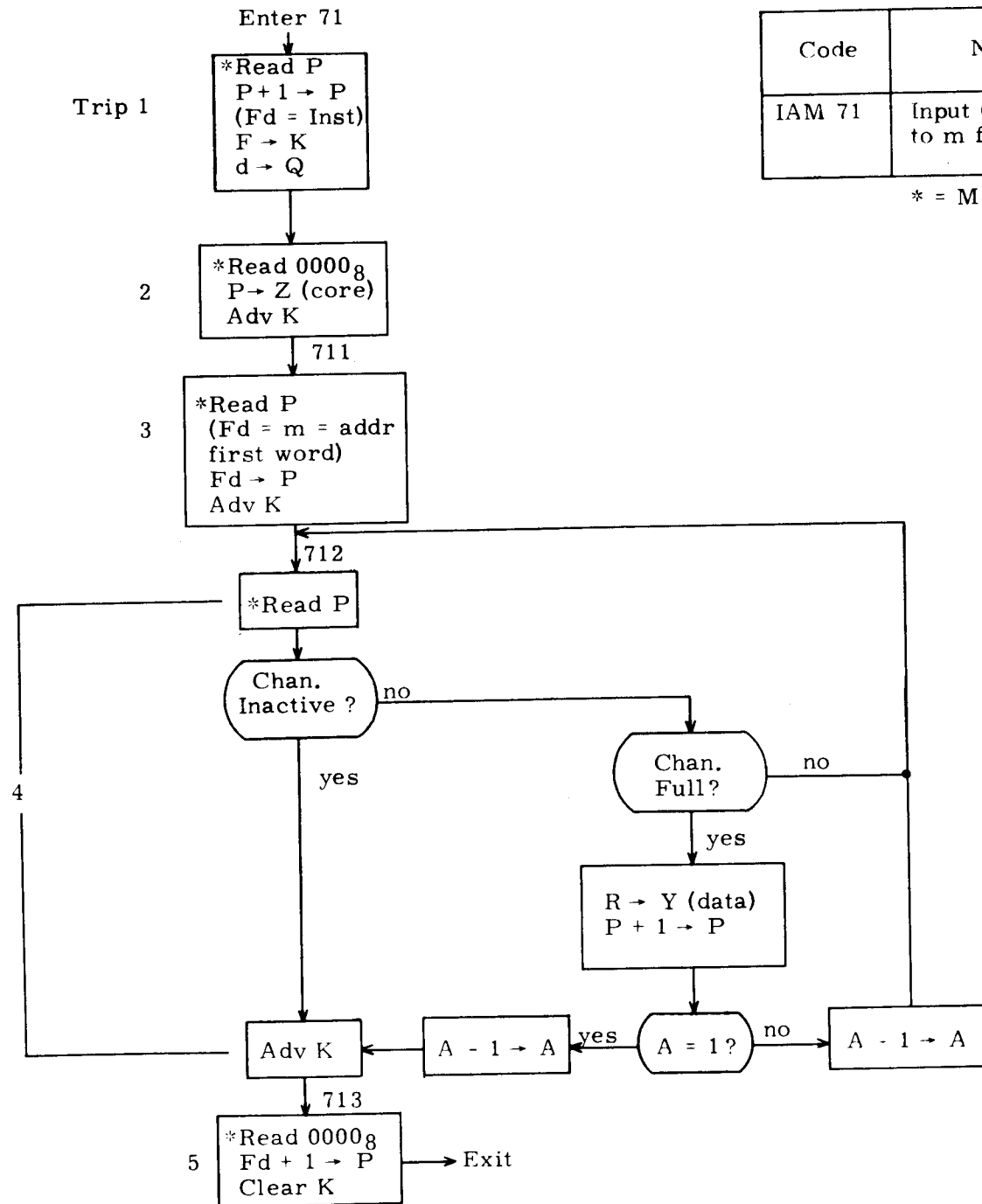


Code	Name	Time (Major Cycles)
IAN 70	Input to A from channel d	2

* = Memory Ref

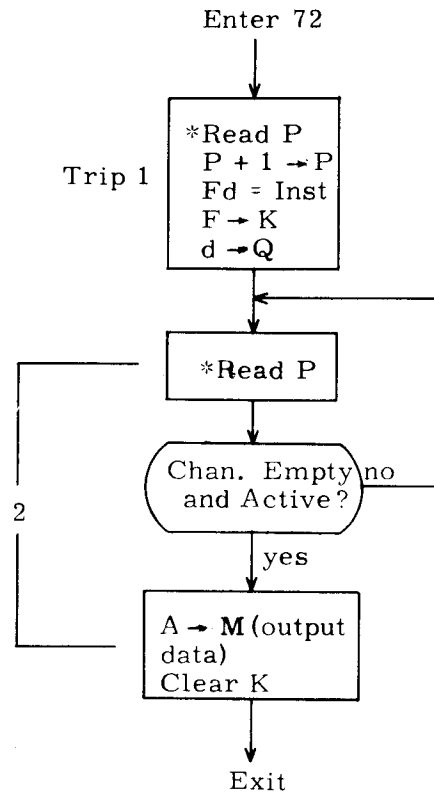
Code	Name	Time (Major cycles)
IAM 71	Input (A) words to m from chan d	4 plus 1/word

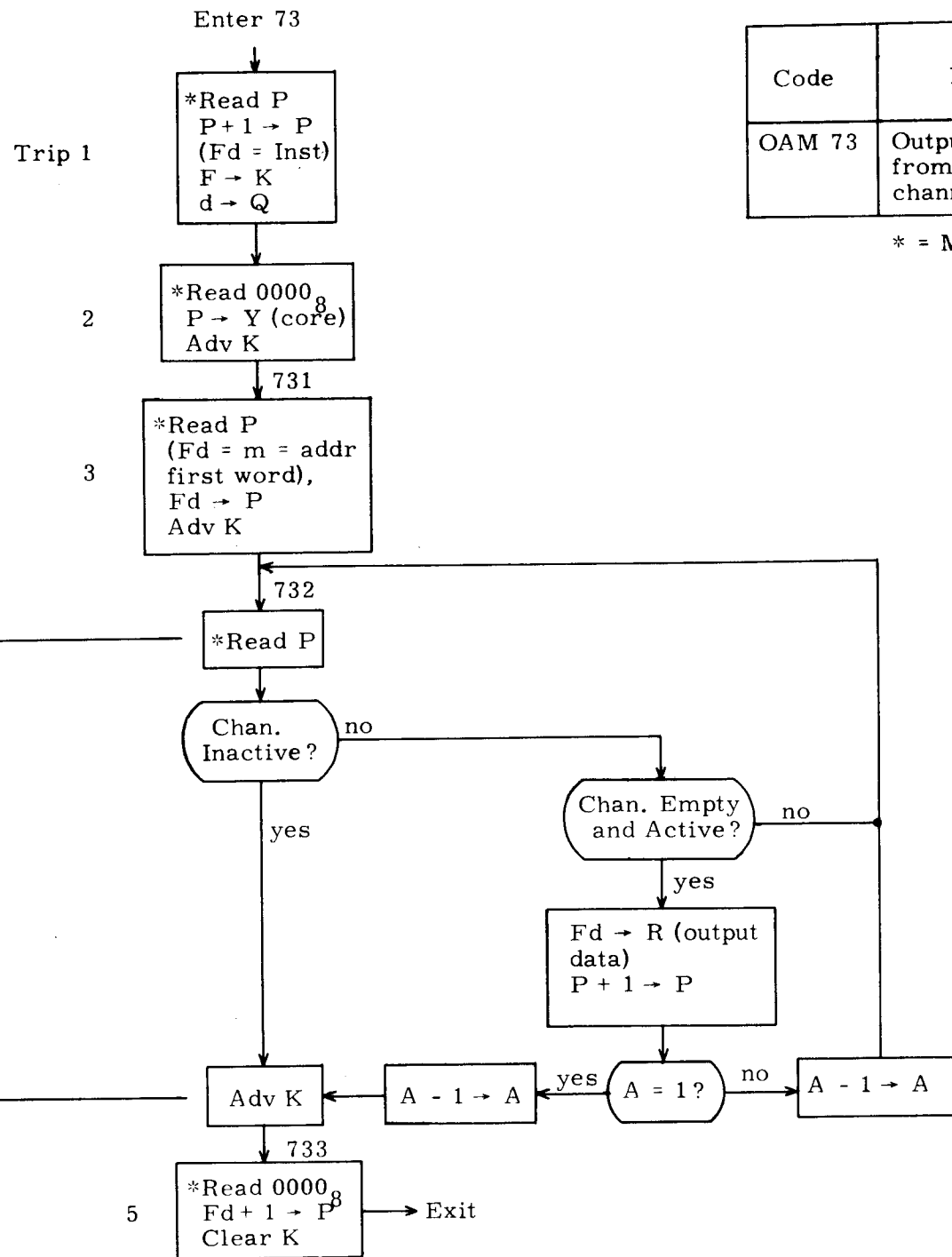
* = Memory Ref.



Code	Name	Time (Major Cycles)
OAN 72	Output from A on channel d	2

*Memory Ref.



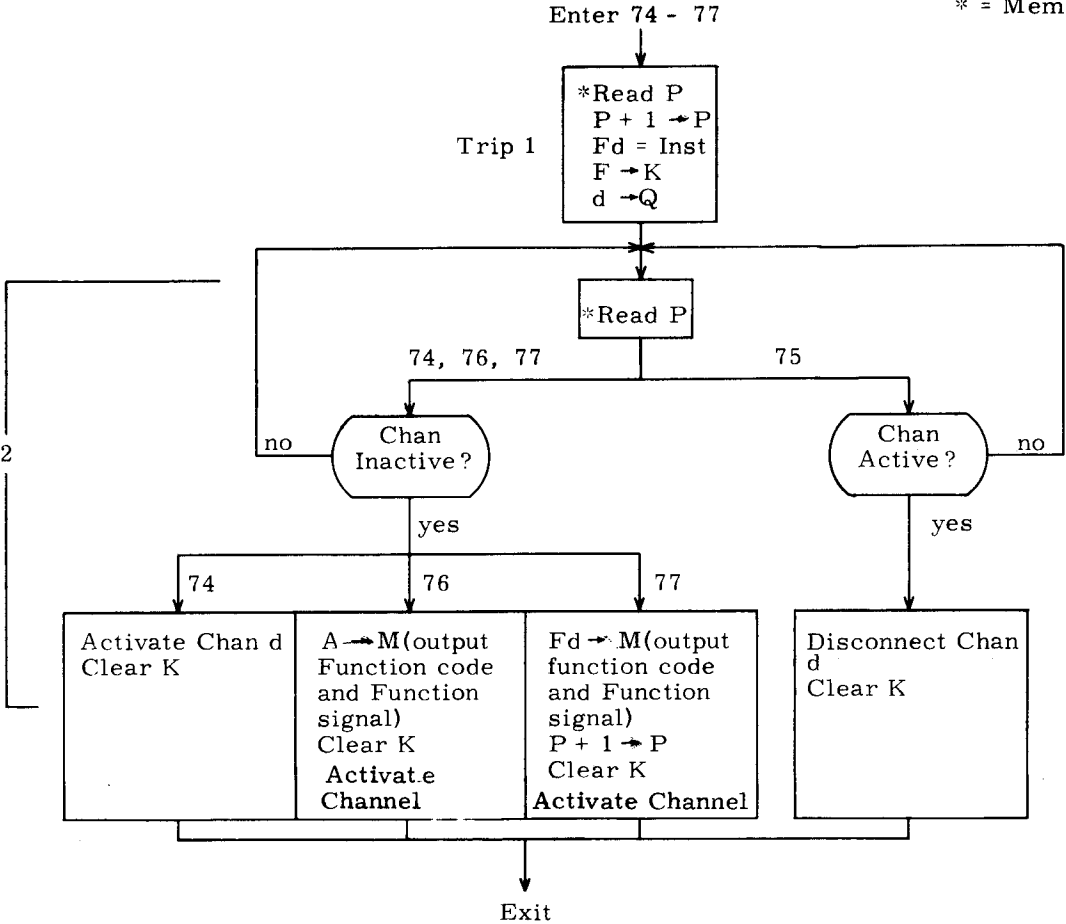


Code	Name	Time (Major cycles)
OAM 73	Output (A) words from m on channel d	4 plus 1/word

* = Memory Ref.

Code	Name	Time (Major cycles)
ACN 74	Activate channel d	2
DCN 75	Disconnect channel d	2
FAN 76	Function (A) on channel d	2
FNC 77	Function m on channel d	2

* = Memory Ref



70

PERIPHERAL AND CONTROL PROCESSORS

CONTENTS

Page	Title	Page	Title
	Peripheral and Control Processors, Introduction	37	A Adder
1	Overall Block Diagram	39	A Register Gates
2	Equation Lists	41	B Gates
3	Detail Block Diagram	43	Q Adder Block Diagram
4	Timing, Master Clock	45	Q Adder
5	Central Processor Master Clock (Chassis 1), Serials 1-7	47	Q and H Register Gates
6.1	Central Processor Master Clock (Chassis 1), Serials 8 and up	49	H Gates
6.2	Barrel, A Register, P Register, Q Register	50	Shift Network
7	A, P, Q Typical Barrel Paths	51	Shift Network, 18-Bit
8	K Register	52	Communication with Central Memory and Central Processor, Central Program Monitor, Exchange Jump
9	K _O Barrel and Slot Paths, and Typical Translations		
10	Slot	53	Central Read Control
11	Barrel Timing	54	Central Read
13	Barrel Map	55	Central Read Pyramid
14	Storage Sequence Control, Memory	56	Central Write
15	Storage Sequence Control	57	Central Write Pyramid
17	Storage Sequence Control Timing	58	Input/Output, Master Clear, Disconnect (75), Function (76 or 77), Activate (74)
18	Memory Cycle Path		
19	Typical Memory Cycle Path	59	Input/Output Paths
21	P → G, Q → G	60	Data Input Sequence, Status Request
22	K Register		Data Output Sequence.
23	K Translations, General	61	Dead Start
25	Adv. K	62	Dead Start, Load, Sweep, Dump
27	K → K Gate	63	712, 732, 505 → K (Dead Start)
29	Clr. K ₂ , Set K ₆	65	Set Q, Dead Start
31	Set K = 340, F → K Gates	67	Dead Start Controller
33	P Register Gates		
35	A Adder, Overall Block Diagram		
36	A Adder		

PERIPHERAL AND CONTROL PROCESSORS

INTRODUCTION

The CONTROL DATA 6601 Central Computer consists of ten peripheral and control processors, a central processor, central memory, and peripheral equipment controllers. Each peripheral and control processor is an independent computer with 4096 words of core storage and a repertoire of 64 instructions. The peripheral and control processors share access to central memory and to 12 bi-directional input-output channels.

The ten peripheral and control processors are combined in a multiplexing arrangement which allows them to share common hardware for arithmetic, logical, I/O, and other operations without sacrificing speed or independence. This multiplexing arrangement consists of the barrel, the slot, and common paths to storage and I/O channels.

The barrel is a matrix of FFs used to hold the quantities in the operating registers of the ten processors and to give each a turn to use the execution hardware in the slot (adders, shift network, etc.). The quantities in the barrel are shifted from slot output to slot input. Each time a processor's data enters the slot, a portion of the instruction is executed. A trip around the barrel requires 1000 nsec (one major cycle), of which each processor's data spends 900 nsec in the barrel and 100 nsec in the slot. Each processor has its own independent 4096 word memory which may be referenced once each major cycle (once each trip around the barrel).

The peripheral and control processors read data from input devices, perform preliminary arithmetic and logical operations, send data and programs to central memory, assign tasks to the central processor, read central processor results from central memory, and send results to external storage (magnetic tape, disc file, etc.) or to output devices (line printer, display console, etc.).

Characteristics of the peripheral and control processors are:

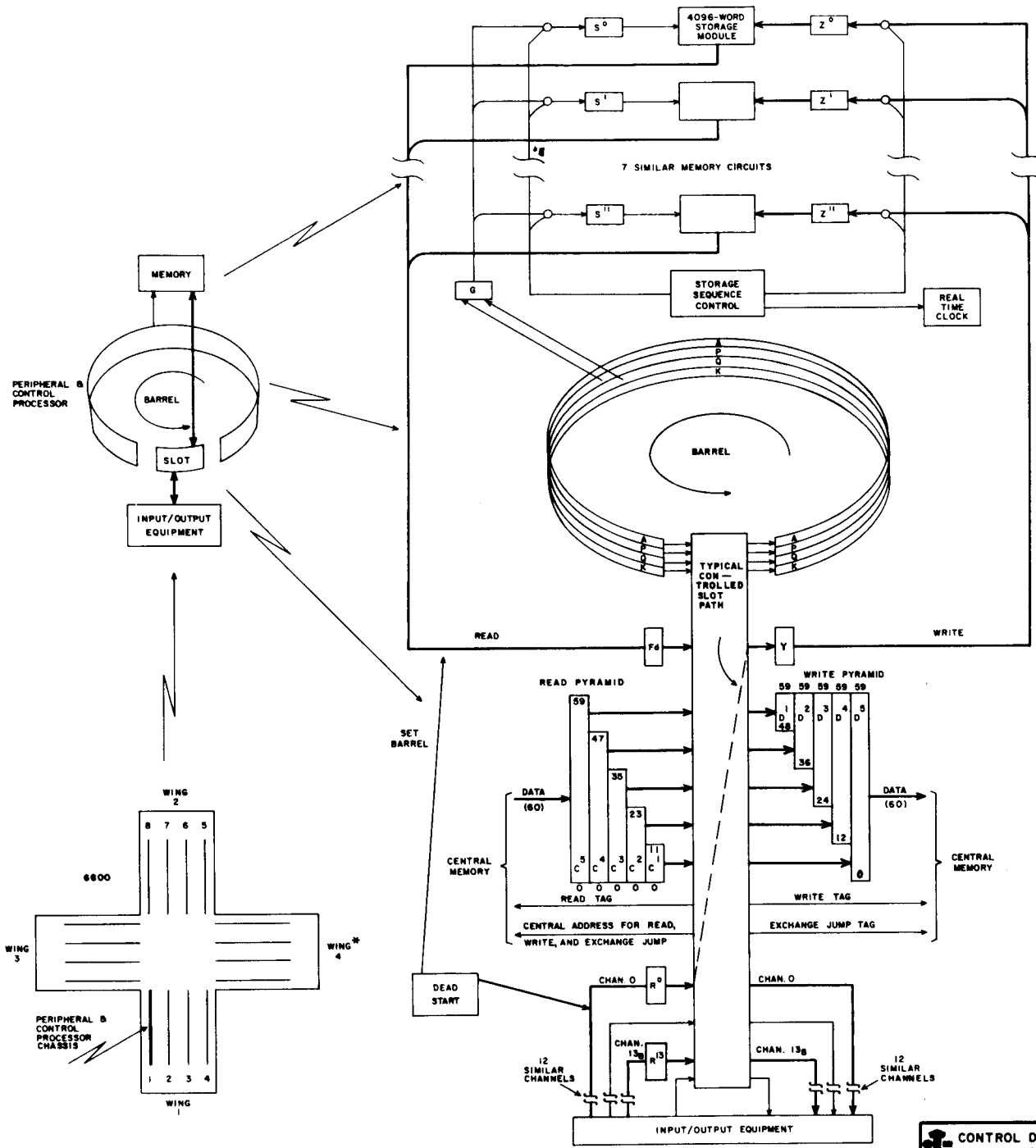
- 4096 word magnetic core storage (12-bits)
Random access, coincident current
Major cycle - 1000 ns
Minor cycle - 100 ns
- 12 bi-directional input-output channels
All channels available to all processors
Maximum transfer rate per channel - one word/major cycle
- Real-time clock (period 4096 major cycles)
- Instructions
 - Arithmetic
 - Logical
 - Input-output
 - Central memory read/write
 - Exchange jump
- Average instruction execution time - two major cycles
- Indirect addressing
- Indexed addressing

Pub. No. 60119300

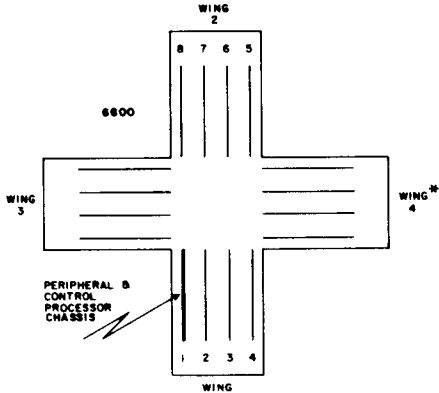
Rev. C

Peripheral and Control Processors

ii



— DATA (12-BIT, EXCEPT FOR CENTRAL PROCESSOR)
 — CONTROL
 A = 18-BIT, ADDRESS
 P = 12-BIT, PROGRAM ADDRESS
 O = 12-BIT, ADDRESS OR OPERAND STORAGE
 K = 9-BIT, INSTRUCTION CODE DESIGNATOR
 ALL SLOT INPUTS & OUTPUTS TIME-ORIENTED WITH BARREL.



* WING 4 ON 6601 ONLY.

CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR OVER-ALL BLOCK DIAGRAM	PRODUCT 6601/04
		SIZE DRAWING NO. C 60119300
		REV K
		SHEET 3

EQUATION LISTS

P Register and P Incrementer

P Incr. → $P = \overline{Q} \rightarrow \overline{P}$
Q Adder → $P = 011+022+(64X, \text{active})+(65X, \text{inactive})+(66X, \text{full})+(67X, \text{empty})+$
 $(03+(04, A=0)+(05, A=0)+(06, A \text{ pos})+(07, A \text{ neg})). (K=00X)$
Adv. P = $(K=00X). (26+60)+5X5+(26, \text{central busy}, K=00X)+021+5XX+2XX+$
 $64X+65X+66X+67X+713+733+(637, \text{central busy})+(617, Q \neq 0, \text{central}$
 $\text{busy}, \overline{C}^5)+613, \overline{C}^4, \overline{C}^3+614, \overline{C}^3+615, \overline{C}^2+616, \overline{C}^1+633, \overline{D}^1+634, \overline{D}^2+635, \overline{D}^3+$
 $636, \overline{D}^4+712, \text{full}+(732, \text{empty, active})+(77X, \text{inactive})+\text{central busy}, \overline{C}^5,$
 $(\overline{C}^1+\overline{C}^2+\overline{C}^3+\overline{C}^4), (K=00X), (P=60)$
Fd → $P = 612, \text{central busy}, \overline{C}^5, (\overline{C}^1+\overline{C}^2+\overline{C}^3+\overline{C}^4)+632+711+731+733+713$
Zero → $P = \text{dead start. (clock=7777)}$

A Register and A Adder

10000_B → $A = \text{dead start. (clock=7777)}$
 $X \rightarrow A = 27 (K=00X)$
Fd → $A = 37X+471+572+36X+461+562$
 $A \rightarrow A = 14X+15X+20X+27X+30X+36X+37X+401+461+471+502+562+572+70X$
 $R \rightarrow A = 70X$
 $+1 \rightarrow B_L = 36X+461+562+(637, \text{central busy})+614, \overline{C}^3$
 $-1 \rightarrow B_L = 712, \text{full}+37X+471+572+(732, \text{active, empty})$
 $+d \rightarrow B_L = 10+12+14+16+20X+21X+22X+31X+35X+411+451+552+30X+401+502$
 $-d \rightarrow B_L = 11+13+15+17+23X+33X+431+532+32X+421+522$
 $+F \rightarrow B_M = 20X+21X+22X+30X+31X+35X+401+411+451+502+512+552$
 $-F \rightarrow B_M = 23X+33X+431+532+32X+421+522$
 $00 \rightarrow B_M = 10+12+14+16+36X+461+562+(614, \overline{C}^3)+(637, \text{central busy})$
 $00 \rightarrow B_U = 30X+31X+35X+36X+401+411+451+461+502+512+552+562+10+12+14+$
 $16+(614, \overline{C}^3)+(637, \text{central busy})$
 $+Q_L \rightarrow B_L = 20X+21X+22X$
 $-Q_L \rightarrow B_L = 23X$

Q Adder Controls

Q_L → $Q \text{ Adder} = 010+020+4X0+5X0+630+64X+65X+66X+67X+610+K=00X$
P → $Q \text{ Adder} = (03+04+05+06+07) (K=00X)$
 $+1 \rightarrow H_L, H_U = 022+(60X, \overline{C}^5, \overline{C}^4, K=XX0)+(601, \overline{C}^3)+602, \overline{C}^2+603, \overline{C}^1+620, \overline{D}^1+$
 $621, \overline{D}^2+622, \overline{D}^3+623, \overline{D}^4$
 $d \rightarrow H_L = 00X+010+020+021+011+4X0+5X0+5X1+2XX+64X+65X+66X+67X+$
 $610+630$
 $F \rightarrow H_U = 011+021+5X1+2XX+010+020+610+630+64X+65X+66X+67X+4X0+$
 $5X0$
 $-1 \rightarrow H_L = 615, \overline{C}^2+635, \overline{D}^3$
 $+00 \rightarrow H_U = ((03+04+05+06+07), d^5)(K=00X)$

G Register

$P \rightarrow G = (Q \rightarrow G)+(713+733+710+730+611+631)$
 $Q \rightarrow G = 010+020+022+4X0+610+630+60X+62X+3X0+5X0+5X2$

K Register

$K \rightarrow K = \overline{2XX+(70X, \text{clock select})+(70X, \text{full})+(72X, \text{active, empty})+(74X,}$
 $\text{inactive})+(75X, \text{active})+(76X, \text{inactive})+3XX+4X1+5X2+011+022}$
 $+64X+65X+66X+67X+713+733+604+(624) (\text{central busy})+(77X,}$
 $\text{inactive})$
 $340 \rightarrow K = 35X+36X+37X+451+461+471+552+562+572$
 $F \rightarrow K = 01+02+20+21+22+23+3X+4X+5X+(F=5, 60)+(60, \text{central busy}, \overline{C}^5, (\overline{C}^1+$
 $\overline{C}^2+\overline{C}^3+\overline{C}^4))$
 $712 \rightarrow K = \text{load, dead start. (clock=7777)}$
 $732 \rightarrow K = \text{dump, dead start. (clock=7777)}$
 $505 \rightarrow K = \text{sweep, dead start. (clock=7777)}$
 $\text{Adv } K = 010+020+610+630+021+5X1+4X0+5X0+632+711+731+$
 $((712+732), \text{inactive})+611+631+710+730+((01+02+5X), d=0, K=00X)+$
 $(600, \overline{C}^5, \overline{C}^4)+(601, \overline{C}^3)+(602, \overline{C}^2)+(603, \overline{C}^1)+(613, \overline{C}^4, \overline{C}^5)+(614, \overline{C}^3)+$
 $(615, \overline{C}^2)+(616, \overline{C}^1)+(620, \overline{D}^1)+(621, \overline{D}^2)+(622, \overline{D}^3)+(623, \overline{D}^4)+$
 $(633, \overline{D}^1)+(634, \overline{D}^2)+(635, \overline{D}^3)+(636, \overline{D}^4)+(712, \text{full}, A=1)+(732,$
 $\text{empty}, A=1)+(612, \text{central busy}, \overline{C}^5, (\overline{C}^1+\overline{C}^2+\overline{C}^3+\overline{C}^4))$
 $\text{Clr } K^2 = 617, Q=0+617, \text{central busy}, \overline{C}^5,$
 $Q=0+637, \text{central busy}$
 $\text{Set } K^6 = 617, Q=0+637, \text{central busy}, Q=0$

A Adder Control

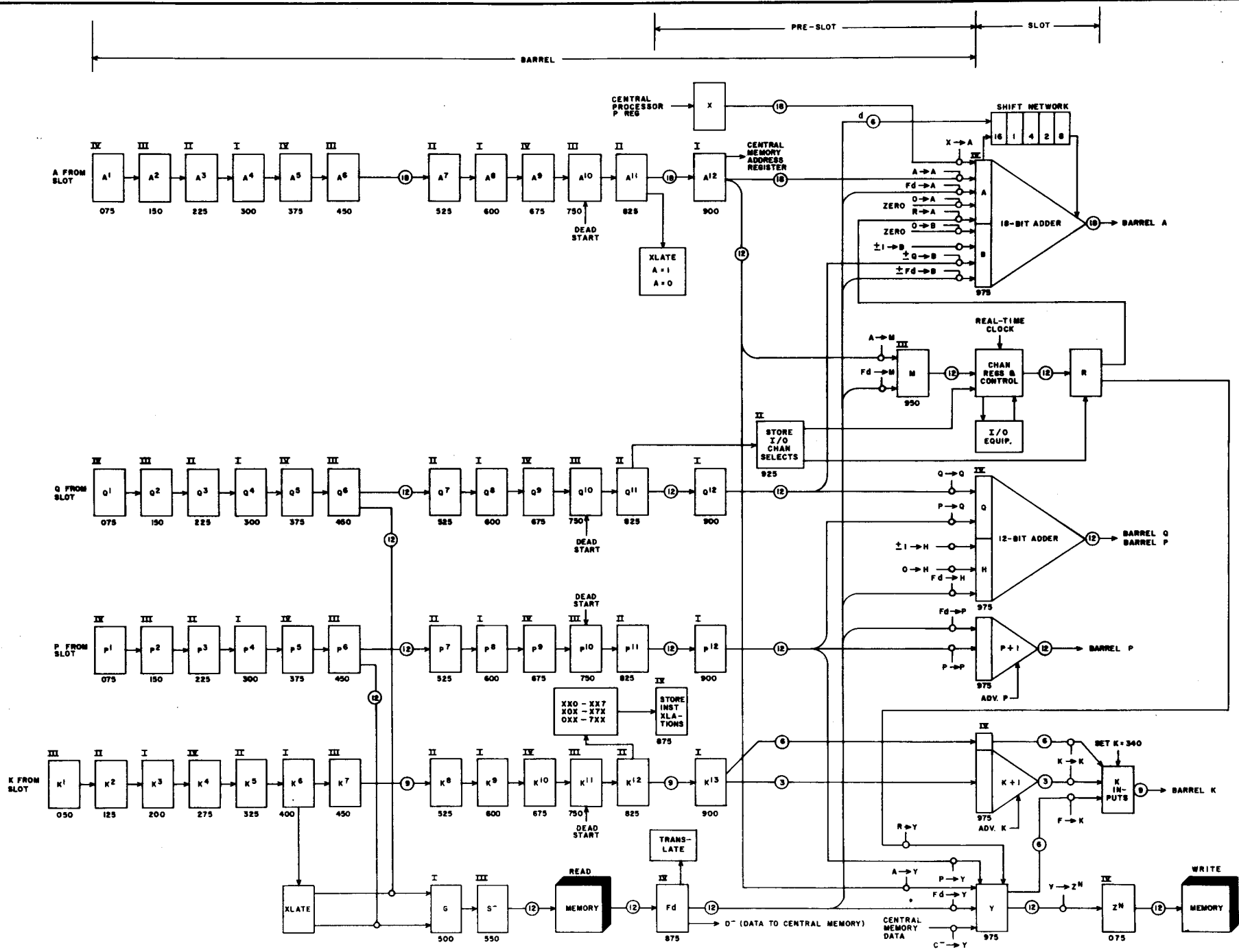
$\text{Add} = 10+11+12+13+22X+23X+33X+431+532$
 $\text{Selective} = 11+12+13+22X+23X+33X+431+532$
 $\text{Logical Prod.} = 12+13+22X$
 $\text{Shift} = 10$

Q Register and Q Adder

$000\ 000\ 00X\ XXX \rightarrow Q = \text{dead start (clock=7777)}$
 $1 \rightarrow Q^1 = \text{dead start (clock=7777)}$
 $\text{Minor cycle } 8+9+4+5$
 $1 \rightarrow Q^2 = \text{dead start (clock=7777)}$
 $\text{Minor cycle } 6+7+8+9$
 $1 \rightarrow Q^3 = \text{dead start (clock=7777)}$
 $\text{Minor cycle } 0+1$
 $1 \rightarrow Q^0 = \text{dead start (clock=7777)}$
 $\text{Minor cycle } 1+3+5+7+9$
 $Q \text{ Adder} \rightarrow Q = \text{Unconditional}$

Peripheral and Control Processors

Pub. No. 60119-60m
 Rev. K P. 2-2



NOTE:
 ALL PERIPHERAL AND CONTROL PROCESSOR MODULES ON CHASSIS 1
 THIS SHEET IS IDENTICAL TO APPENDIX A PAGE 2.

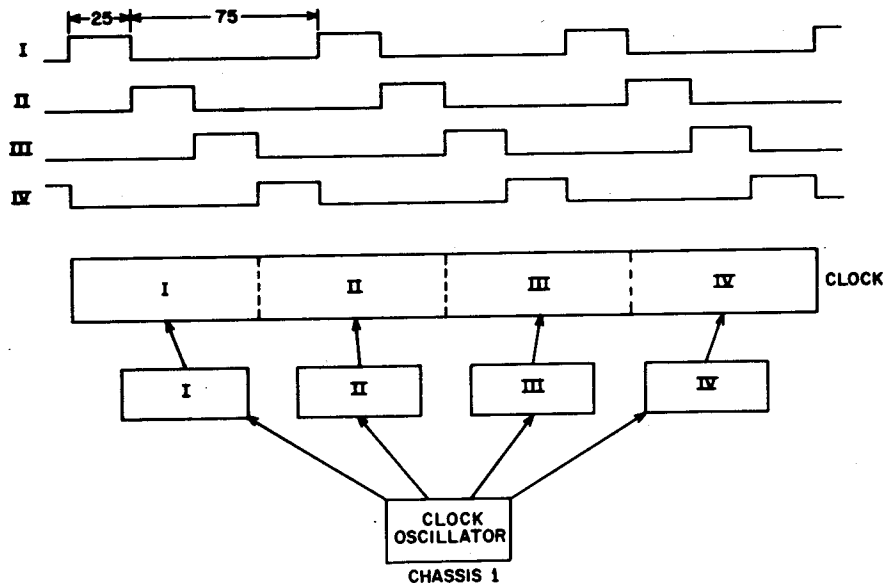
CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR DETAIL BLOCK DIAGRAM	PRODUCT 6601/04
	SIZE C 60119300	REV K
	SHEET 4	REV 3

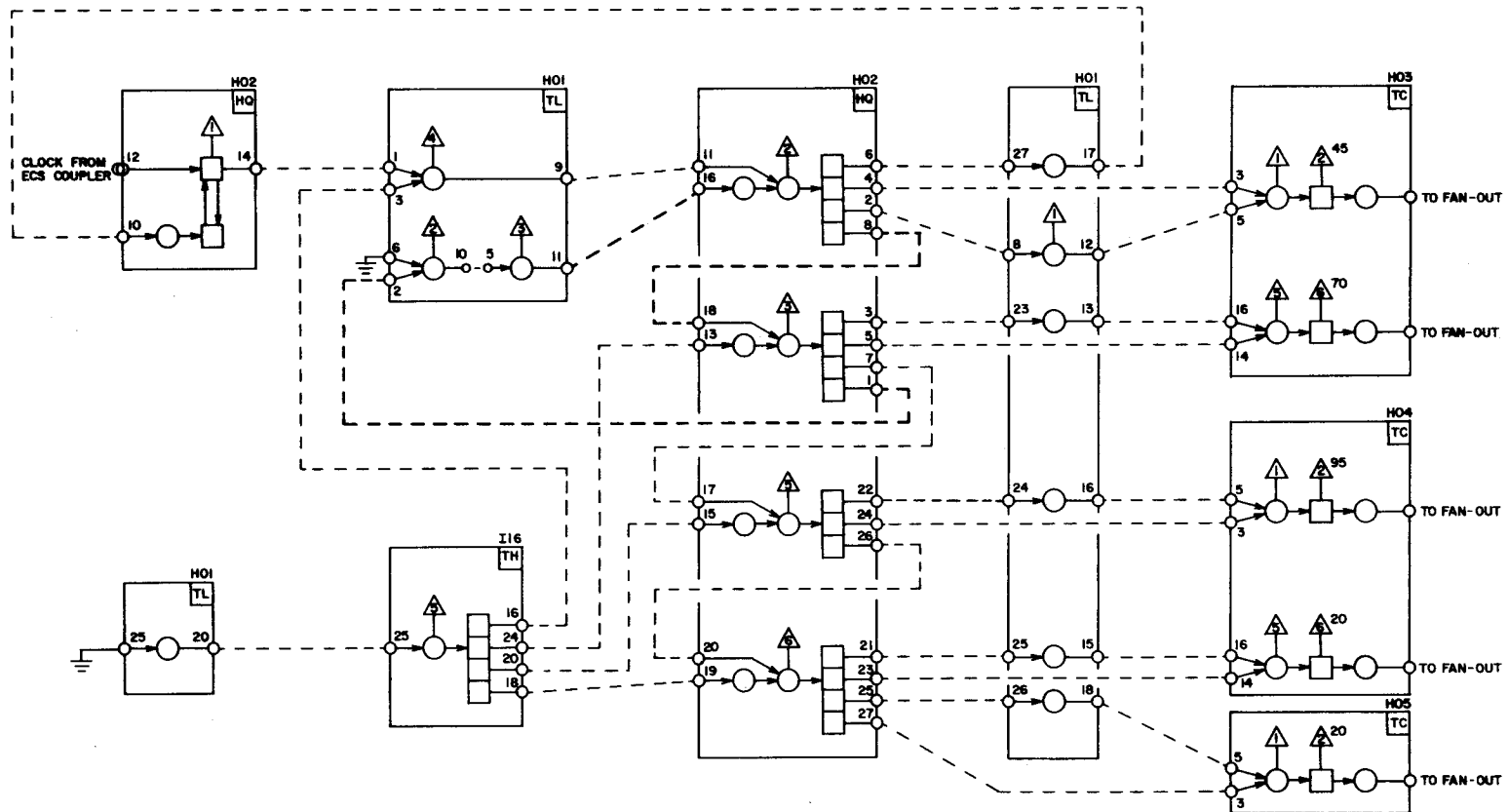
TIMING

Timing in the 6600 is controlled by a four-phase master clock located on the peripheral and control processor chassis (chassis 1). Four 25 nsec pulses are issued each minor cycle to control movement of data and instructions. A storage sequence control system, timed by the four-phase clock, controls storage references and defines the ten peripheral and control processors.

MASTER CLOCK

The master clock oscillator consists of a TD module and a TI module. To form the 25 usec clock pulses, a pulse from the TD is ANDed with a similar pulse which has been delayed and inverted by the TI. The result is a series of pulses (primary clock) which are fanned out through TC modules to be used as timing control. In addition to forming the clock pulses on chassis 1, the master clock sends pulses to chassis 5 and from there to all the other chassis. On each chassis, the incoming clock pulses are used to form a clock system similar to chassis 1. The clocks on all chassis are synchronized so that time 00 on any chassis is the same as time 00 on any other chassis.





NOTES:

1. WIRE AS SHOWN FOR SYSTEMS WITHOUT ECS. WITH ECS COUPLER DRIVING THE CLOCK, RUN I16-16 TO HO1-6 AND GROUND HO1-3.
2. TURN TO PAGE 3 FOR COMPLETE CLOCK FAN-OUT ON CHASSIS 1.

79

THIS SHEET IS IDENTICAL TO 6601/04/13/14 CENTRAL PROCESSOR CLOCK, P. 4.1

 CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR MASTER CLOCK CHASSIS 1, SERIALS 8-UP	PRODUCT 6601/04/13/14	
	SIZE C	DRAWING NO. 60119300	REV. M
	SHEET 266	6.1	

BARREL

The barrel contains the A, P, Q, and K registers for each of the ten processors. The functions of these four registers in the barrel are:

- A (18 bits) A holds one operand for add, shift, logical and selective operations. The 18-bit quantity in A may be an arithmetic operand, central memory address, or an I/O function or data word.
- P (12 bits) P is the program address register. (P) is also used as a data address in certain I/O and central instructions.
- Q (12 bits) Q holds the d portion of instructions or may hold a data word when d is an address.
- K (9 bits) K holds the F portion of an instruction word and the trip count (the number of times an instruction has been around the barrel).

A REGISTER

The A register in the barrel receives the result of add, shift, logical or selective operations in the slot. This quantity may be stored, returned to the slot unaltered or used to condition other operations. A is always tested to determine its sign and whether it is zero, non-zero, or one. The result of these tests may be used to condition jump or other instructions. The quantity in A may be a full 18-bit central address or a 12-bit peripheral word (in which case the upper 6 bits will be zero).

The connections to A in the barrel are:

Outputs

- A → M - (A) may be sent as a data or function word on one of the I/O channels.
- A → Central Address Register - (A) is the central memory address in central read and write and exchange jump instructions.

Peripheral and Control Processors

A → Y - For a store instruction, (A) is sent to Y and then to storage.

A → Translation Networks

Inputs

- X → A - The content of the central program address register is sent to the peripheral X register every minor cycle. A 27 instruction sends (X) to A and enables a peripheral and control processor to monitor the progress of the central program.
- R → A - An input to A instruction gates a word from an I/O channel into A.
- Fd → A - A data word from storage is entered into A by the Fd → A path.
- A → A - When the quantity in A is to be returned to the slot unaltered, the A → A gate is enabled.

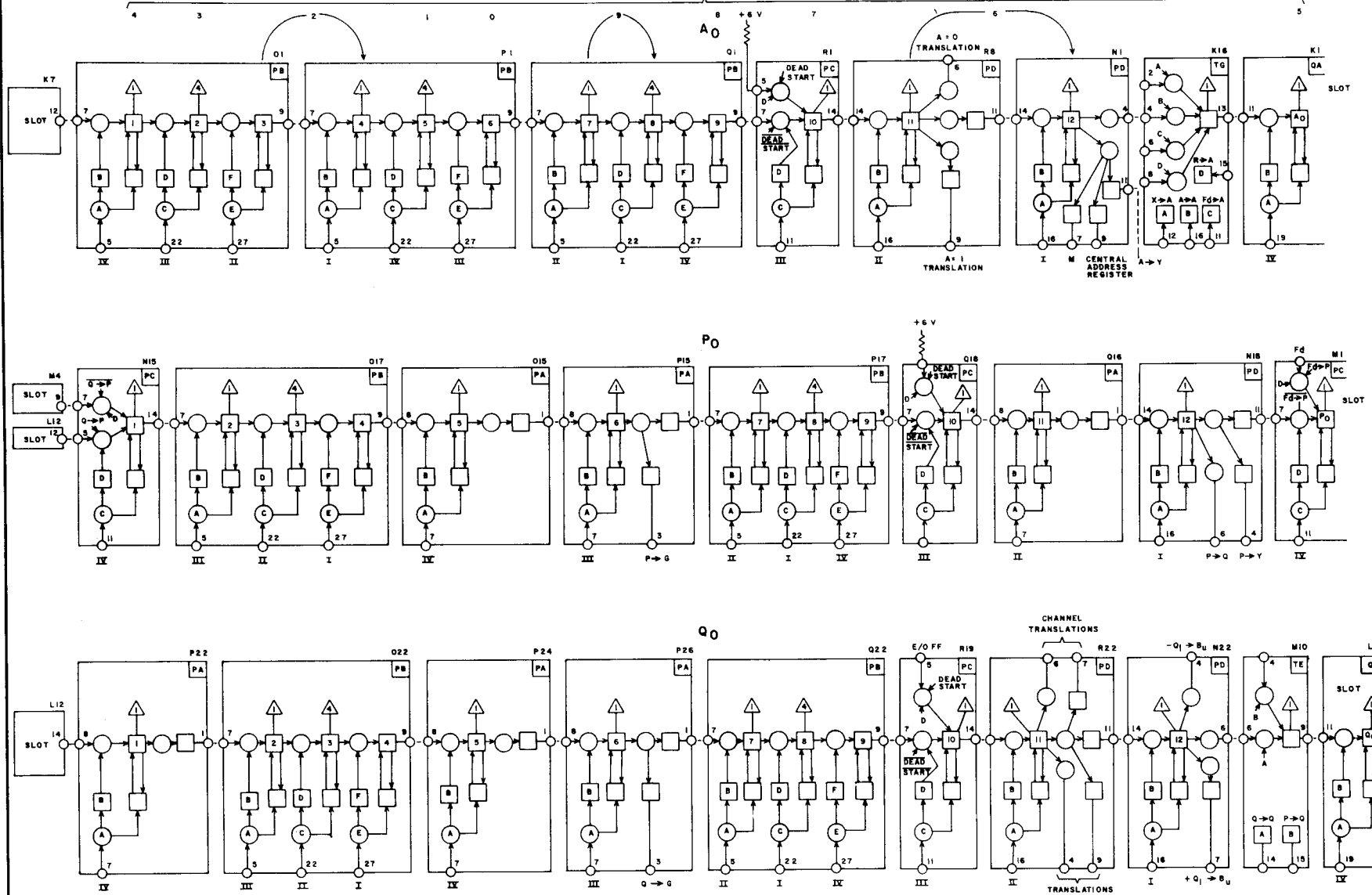
P REGISTER

P holds the program address and is not changed in the barrel (except by Dead Start). (P) is sent to a storage unit from stage 6 in the barrel. This allows time to read a word from storage and make it available at slot time. (P) is sent to the G register which feeds all ten storage address or S registers. When a jump is called for, P is sent to Q from barrel stage 12. Q is then altered by the Q adder in the slot and the new address returned to P at the first stage of the barrel.

Q REGISTER

Q holds the d portion of an instruction and has several outputs to translation networks which make channel selections for I/O instructions. When d is an address, (Q) is sent from the slot to P in the barrel and the word obtained from that address is entered into Q in the slot. When a jump is called for, the quantity in Q is added to or subtracted from (P) in the Q adder and the result sent to P. When an instruction calls for an 18-bit operand, the lower six bits of Q are sent to the upper six bits of A to form the 18-bit quantity dm.

LOCATIONS OF PROCESSORS WHEN STORAGE SEQUENCE CONTROL ENABLES G → 5. STORAGE 0
PROCESSOR 5 IS IN THE SLOT

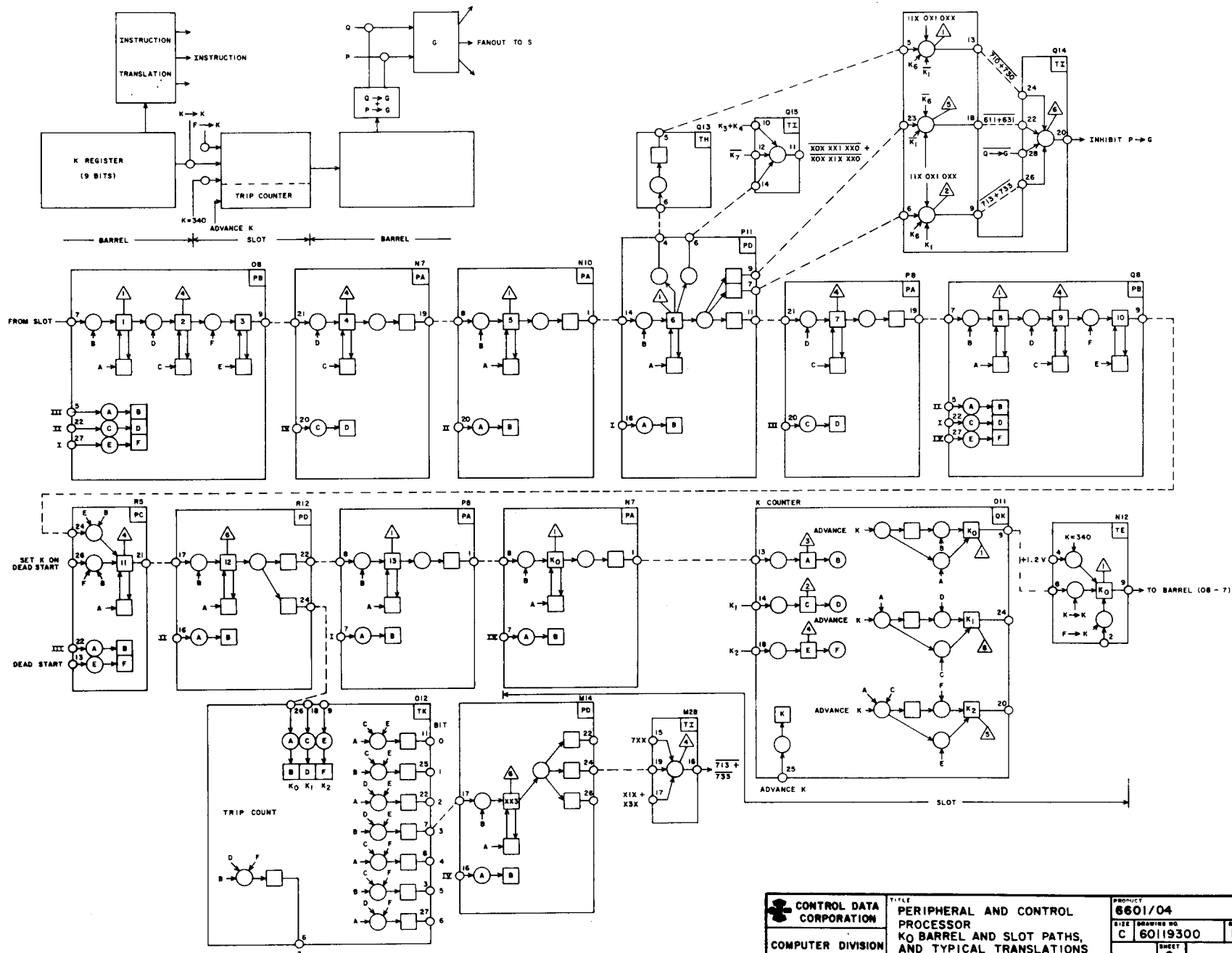


COMPUTER DIVISION	CONTROL DATA CORPORATION	TITLE PERIPHERAL AND CONTROL PROCESSOR A, P, Q TYPICAL BARREL PATHS	PRODUCT 6601
			SIZE DRAWING NO. C 60119300
			REV D
		SHEET 7	7

K REGISTER

K holds the F portion of an instruction word and a 3-bit trip count which sequences the execution of an instruction. K is translated at two different times during a trip around the barrel; first, to determine if a storage reference is needed, and second, to provide the proper commands at the slot. During the barrel trip in which a new instruction is being read from storage, a translation of K = 00X enables

translations from Fd in the storage cycle path to be used in place of K translations. This eliminates the need for a separate "Read Next Instruction" trip through the barrel and allows certain instructions to be read from storage and executed all in one trip. The K = 00X translation arises from the fact that K is cleared at the end of each instruction.



 CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR K ₀ BARREL AND SLOT PATHS, AND TYPICAL TRANSLATIONS	PRODUCT 6601/04	
	SIZE C	DRAWING NO. 60119300	REV K
	SHEET 8		9

SLOT

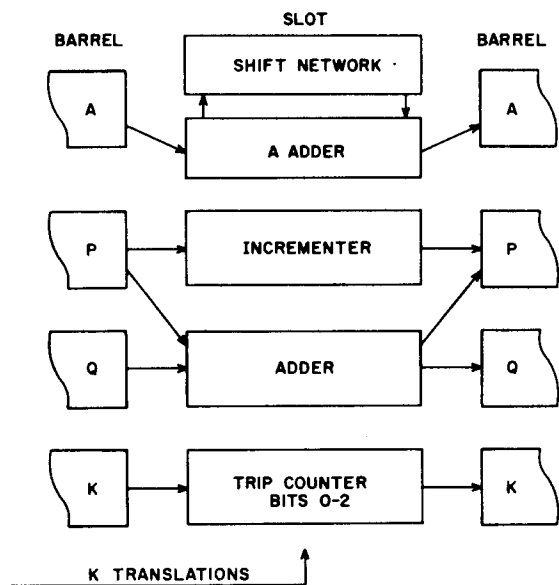
The slot contains the execution hardware for A, P, Q, and K. Each processor is allowed one minor cycle in the slot every major cycle.

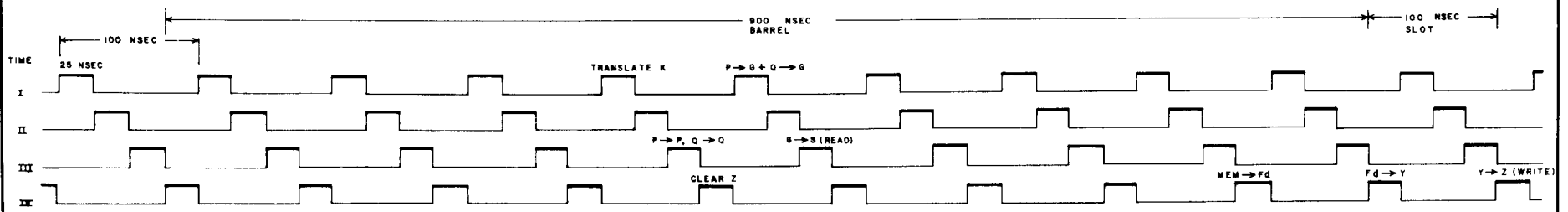
Included in the slot are:

- A Adder
Shift Network
Logical Circuits
Selective Circuits
- P Incrementor
Inputs from P or Q in the barrel

- Q Adder
Input Path from Fd
- K 3-bit Trip Counter
Input from F
K = 340 Gate

As A, P, Q, and K enter the slot, K translations (started earlier in the barrel) become available and a portion (or all) of an instruction is executed. The results are gated back into the barrel to be stored, used again, or sent to I/O equipment.





SLOT TO BARREL
 (BARREL SHIFTS
 RIGHT AT LEADING
 EDGE OF TIME
 INCREMENTS SHOWN)

A 18 BITS																				
P 12 BITS																				
Q 12 BITS																				
K 9 BITS																				

RE-ENTER
 BARREL
 AT LEFT

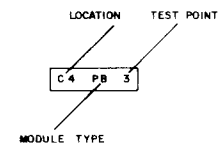
CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR BARREL TIMING	PRODUCT 6601/04	SIZE C	DRAWING NO. 60119300	REV 1
		SHEET 9	OF 11		

A

	IV	III	II	I	IV	III	II	I	IV	III	II	I	GATES		IV		
A0	Q1 PB 1		4		P1 PB 1		4		Q1 PB 1		4		R1 PC 1	R8 PD 1	N1 PD 1	K16 TG 1	K1 QA 1
A1	2		5		2		5		2		5		2		2		2
A2	3		6		3		6		3		6		5		5		5
A3	Q2 PB 1		4		P2 PB 1		4		Q2 PB 1		4		6		6		6
A4	2		5		2		5		2		5		R2 PC 1	R9 PD 1	N2 PD 1	K17 TG 1	K2 QA 1
A5	3		6		3		6		3		6		2		2		2
A6	Q3 PB 1		4		P3 PB 1		4		Q3 PB 1		4		5		5		5
A7	2		5		2		5		2		5		6		6		6
A8	3		6		3		6		3		6		R3 PC 1	R10 PD 1	N3 PD 1	K18 TG 1	
A9	Q4 PB 1		4		P4 PB 1		4		Q4 PB 1		4		2		2		2
A10	2		5		2		5		2		5		5		5		5
A11	3		6		3		6		3		6		6		6		6
A12	Q5 PB 1		4		P5 PB 1		4		Q5 PB 1		4		R4 PC 1	R11 PD 1	N4 PD 1	K19 TE 1	K5 QA 1
A13	2		5		2		5		2		5		2		2		2
A14	3		6		3		6		3		6		5		5		5
A15	Q6 PB 1		4		P6 PB 1		4		Q6 PB 1		4		6		6		6
A16	2		5		2		5		2		5		R5 PC 1	R12 PD 1	N5 PD 1		5
A17	3		6		3		6		3		6		2		2		2

P

	IV	III	II	I	IV	III	II	I	IV	III	II	I	IV		
P0	N15 PC 1	O17 PB 1		4		Q15 PA 1	P15 PA 1	P17 PB 1		4		Q18 PC 1	Q16 PA 1	N18 PD 1	M1 PC 1
P1	2		2	5		2		2		5		2		2	2
P2	5		3	6		3		3		6		5		3	5
P3	6	O18 PB 1		4		4		4	P18 PB 1		4		6		6
P4	N16 PC 1		2	5		5		5		2		5	Q19 PC 1	N19 PD 1	M2 PC 1
P5	2		3	6		6		6		3		6	2		2
P6	5	O19 PB 1		4		Q16 PA 1	P16 PA 1	P19 PB 1		4		5	Q17 PA 1		5
P7	6		2	5		2		2		5		6		2	6
P8	N17 PC 1		3	6		3		3		6		Q20 PC 1	N20 PD 1	M3 PC 1	
P9	2	O20 PB 1		4		4		4	P20 PB 1		4		2		2
P10	5		2	5		5		5		2		5		5	5
P11	6		3	6		6		6		3		6		6	6



Q

	IV	III	II	I	IV	III	II	I	IV	III	II	I	GATES		IV	
Q0	P22 PA 1	O22 PB 1		4		P24 PA 1	P26 PA 1	Q22 PB 1		4		R19 PC 1	R22 PD 1	N22 PD 1	M10 TE 1	L1 QA 1
Q1	2		2	5		2		2		5		2		2		2
Q2	3		3	6		3		3		6		5		5		5
Q3	4	O23 PB 1		4		4		4	Q23 PB 1		4		6		6	4
Q4	5		2	5		5		5		2		5	R20 PC 1	R23 PD 1	N23 PD 1	5
Q5	6		3	6		6		6		3		6	2		2	6
Q6	P23 PA 1	O24 PB 1		4		P25 PA 1	P27 PA 1	Q24 PB 1		4		5		5		M11 TE 1
Q7	2		2	5		2		2		5		6		6		2
Q8	3		3	6		3		3		6		6		6		3
Q9	4	O25 PB 1		4		4		4	Q25 PB 1		4		2		2	4
Q10	5		2	5		5		5		2		5		5		5
Q11	6		3	6		6		6		3		6		6		6

K

	III	II	I	IV	II	I	III	II	I	IV	III	II	I	IV		
K0	O8 PB 1		4		N7 PA 4	N10 PA 1	P11 PD 1	P8 PA 4	Q8 PB 1		4		R5 PC 6	R12 PD 6	P8 PA 1	N7 PA 1
K1	2		5		5		2		2		5		6		6	2
K2	3		6		6		3		3		6		2		2	3
K3	O9 PB 1		4		N8 PA 4		4		6	P9 PA 4	Q9 PB 1		4		5	P9 PA 1
K4	2		5		5		5		5		5		2		2	2
K5	3		6		6		6		6		6		3		3	3
K6	O10 PB 1		4		N9 PA 4	N11 PA 1		5	P10 PA 4	Q10 PB 1		4		2	2	P10 PA 1
K7	2		5		5		2		6		5		5		5	2
K8	3		6		6		3		6		6		6		6	3

CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR BARREL MAP	PRODUCT 6601
		SIZE DRAWING NO. C 60119300
		REV C
		SHEET 10
		13

STORAGE SEQUENCE CONTROL

Timing for memory references is controlled by the Storage Sequence Control, which is a timing chain of FFs gated by clock pulses. As a "1" passes down the chain, each FF is set for one minor cycle during which it issues commands to the storage logic. This chain reinitiates itself after each cycle and runs continuously. One memory reference is initiated each minor cycle. The Storage Sequence Control overlaps the references as shown in the typical stage "a".

The stages of storage sequence control are numbered according to the processor for which they initiate a memory reference. The commands issued by the first half of a typical stage are:

G → S, Storage a
Clear Z, Storage a + 1
Set Z, Storage a + 5
Enable Sense, Storage a + 7

The second half of state "a" issues the commands:

Read, a
Write, a + 5
Stop Read, a + 6
Stop Write, a + 1

These commands and other signals from storage sequence control

define and separate the peripheral and control processors.

The reset circuit which reinitiates storage sequence control senses whether stages 0-8 are set; if not, stage 0 is reinitiated just after stage 9 has issued its commands.

A memory reference is initiated from stage 6 in the barrel, so that information from memory is available at slot time. Thus, a memory reference for processor 0 (storage 0) is initiated while processor 5 is in the slot.

MEMORY

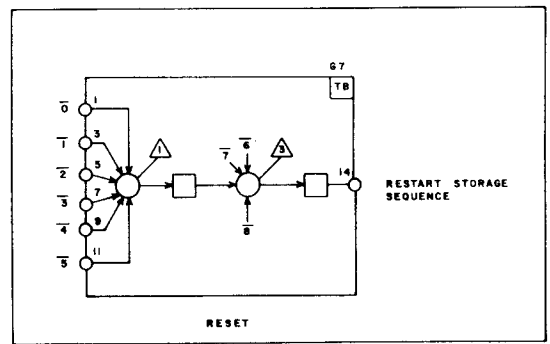
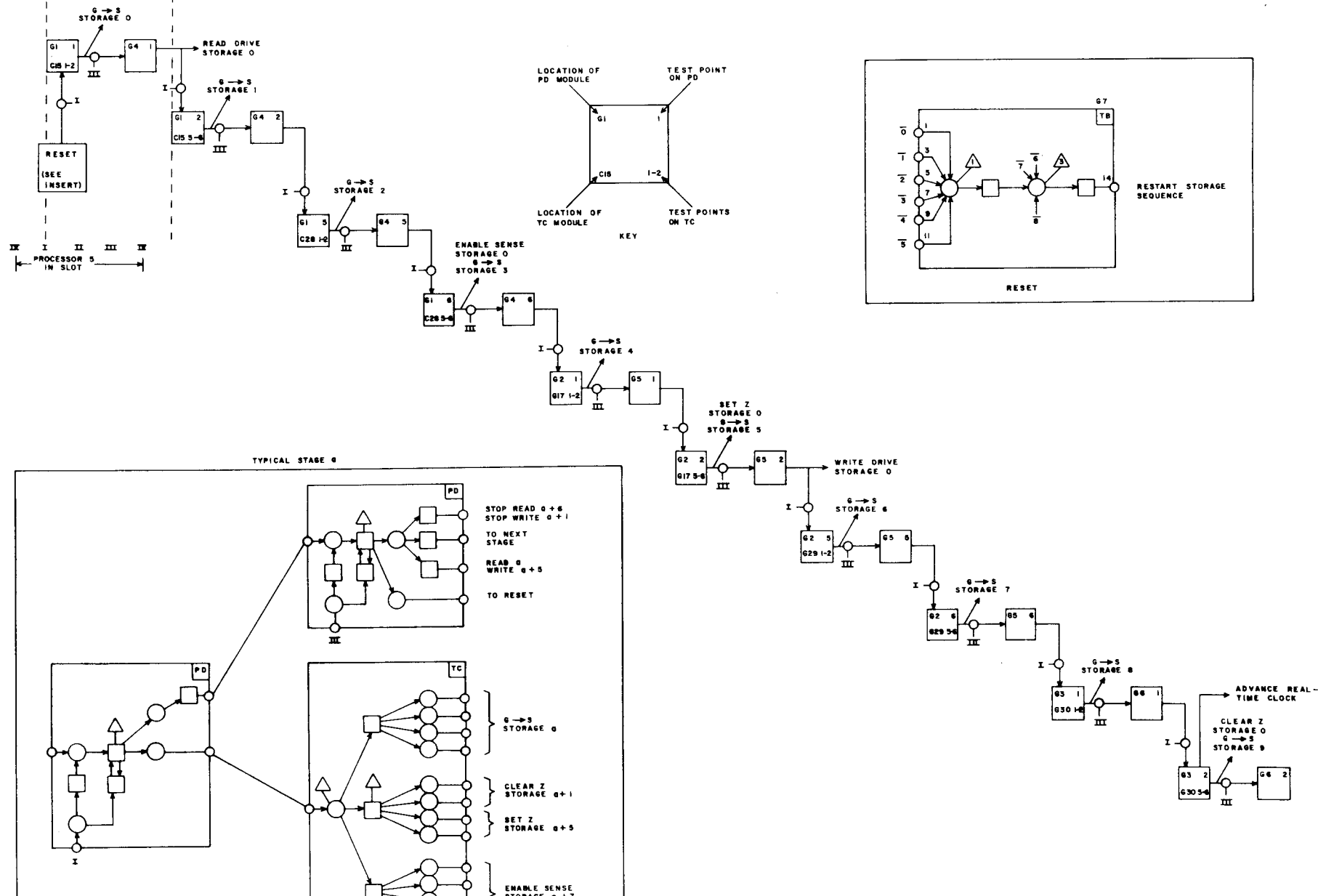
Each of the ten peripheral and control processors has its own independent core-storage unit with a capacity of 4096 12-bit words. Each has its own address register (S), sense amplifiers, and restoration register (Z). However, the ten storage units share a common memory cycle path and common paths to and from the barrel.

Each peripheral and control processor makes one memory reference each major cycle. When no memory reference is called for by the current instruction, address 0000 is read and restored.

Peripheral and Control
Processors

Pub. No. 60119300
Rev. C Page 14

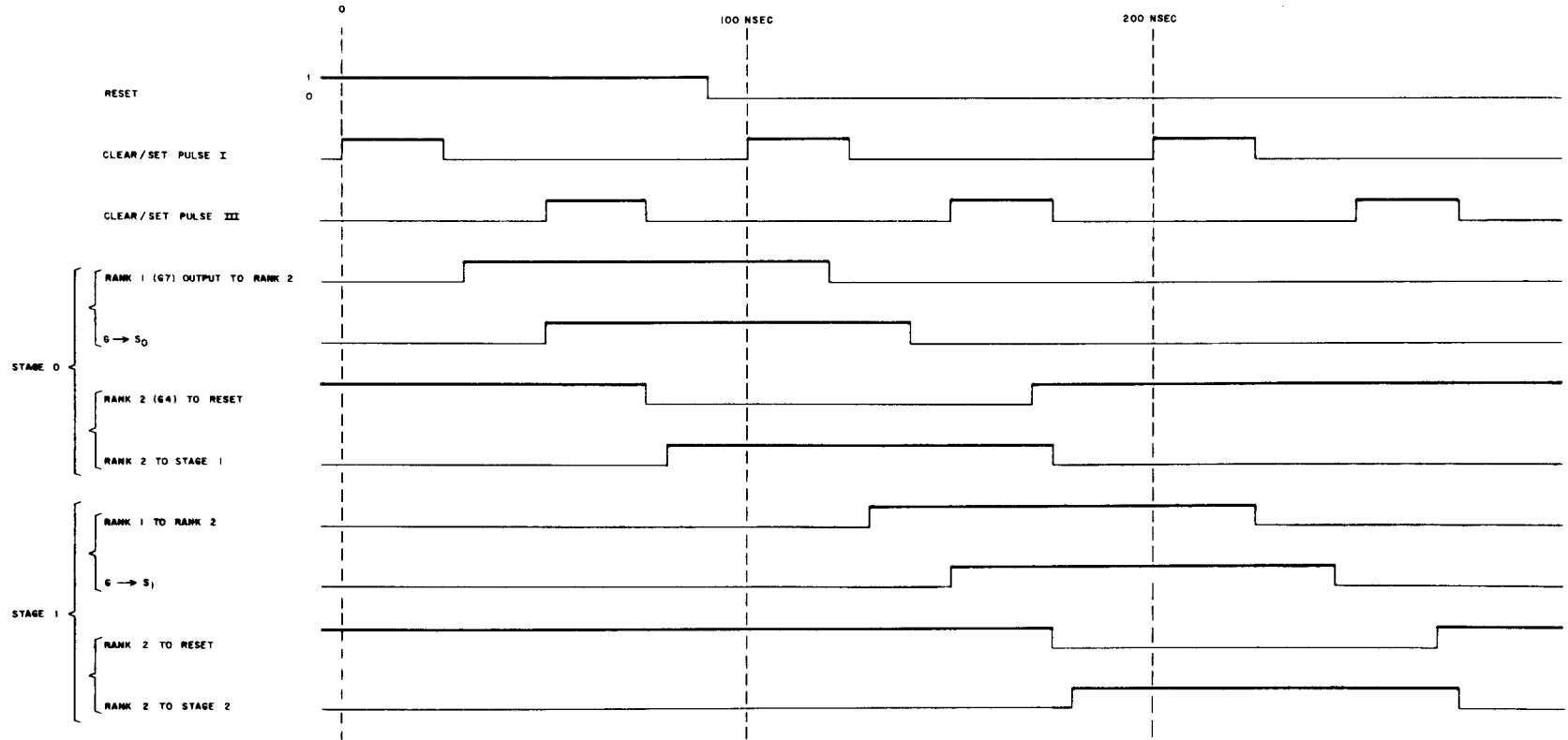
NSEC 00 100 200 300 400 500 600 700 800 900 00



89

CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR STORAGE SEQUENCE CONTROL	PRODUCT 6601
	SIZE DRAWING NO. C 6019300	REV C
SHEET 11		REV 15

90



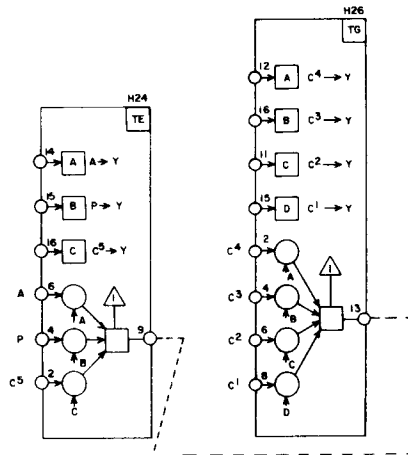
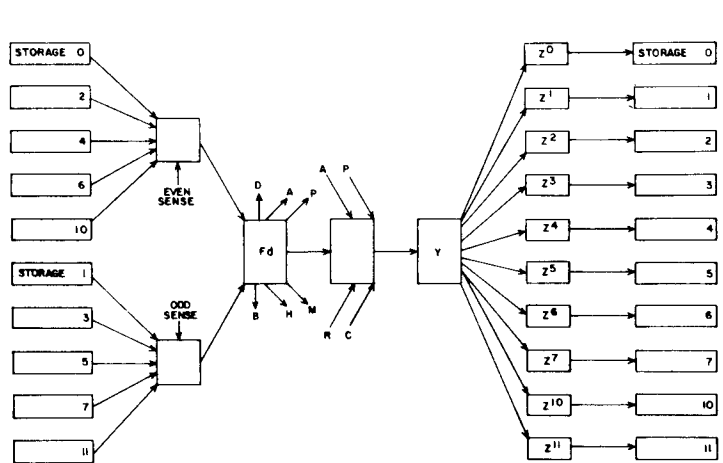
CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR STORAGE SEQUENCE CONTROL TIMING	PRODUCT 6601	
		SIZE C	DRAWING NO. 60119300
		REV. C	SHEET 12
		17	

MEMORY CYCLE PATH

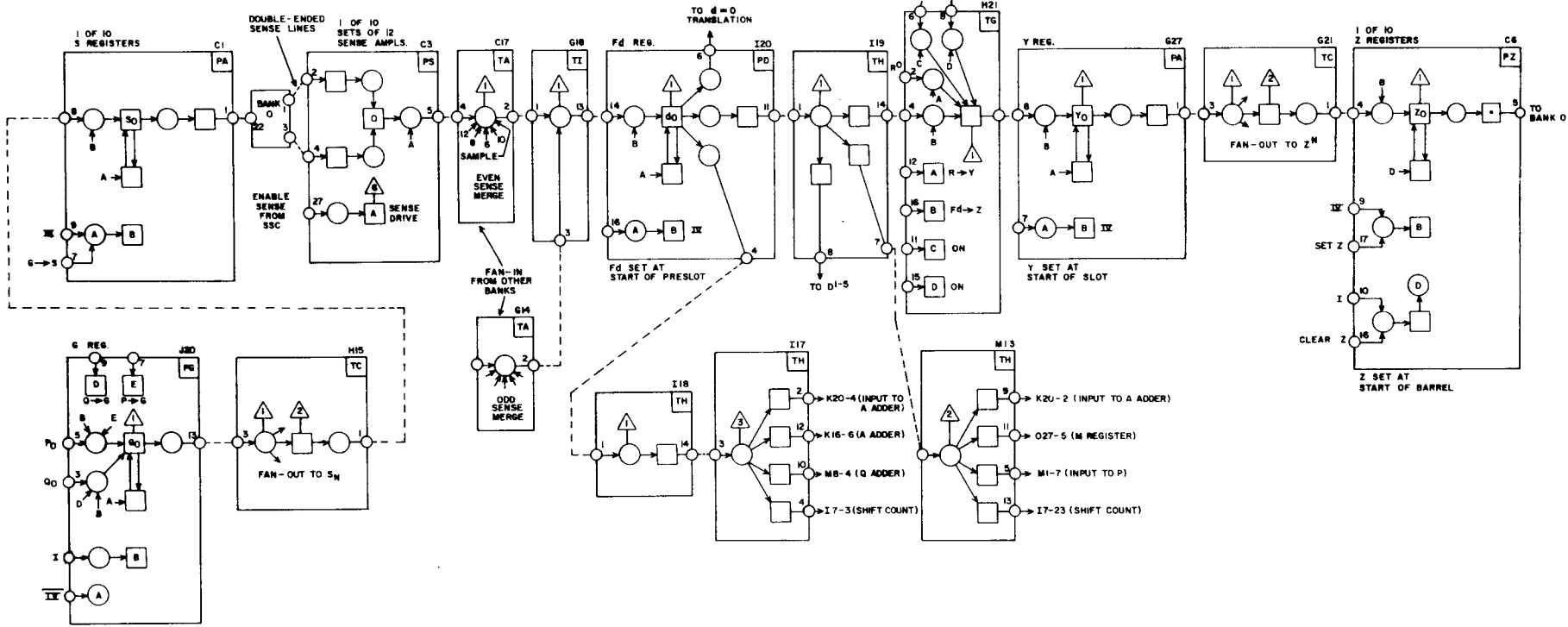
The common memory cycle path used by all processors receives data from the memories via the sense merge. Inputs to the sense merge from the sense amplifiers are a logical "1" (0.2v) when sense is not enabled. When a processor's sense amplifiers are enabled, the outputs of the PS modules are allowed to go to +1.2v for a sensed "0". If the core switches, the sense amplifier output goes to +0.2v "1". The AND combination of logical "1's" from unselected processors, even or odd sense enable, and "1" bits from the selected processor's sense amplifiers sets the word from memory into the Fd register in the memory cycle path.

The memory cycle path sends information to the barrel, I/O channels, translators and central write pyramid and receives information from the barrel, central read pyramid, and I/O channels. Outputs from Fd in the memory cycle path are translated and used to form commands when K = 00X (read next instruction trip).

Information in the memory-cycle path (either the read word or a new word) is fanned out from the Y register to the ten Z registers. The set Z signal from storage sequence control gates the complement of the word to be stored into the proper Z register.

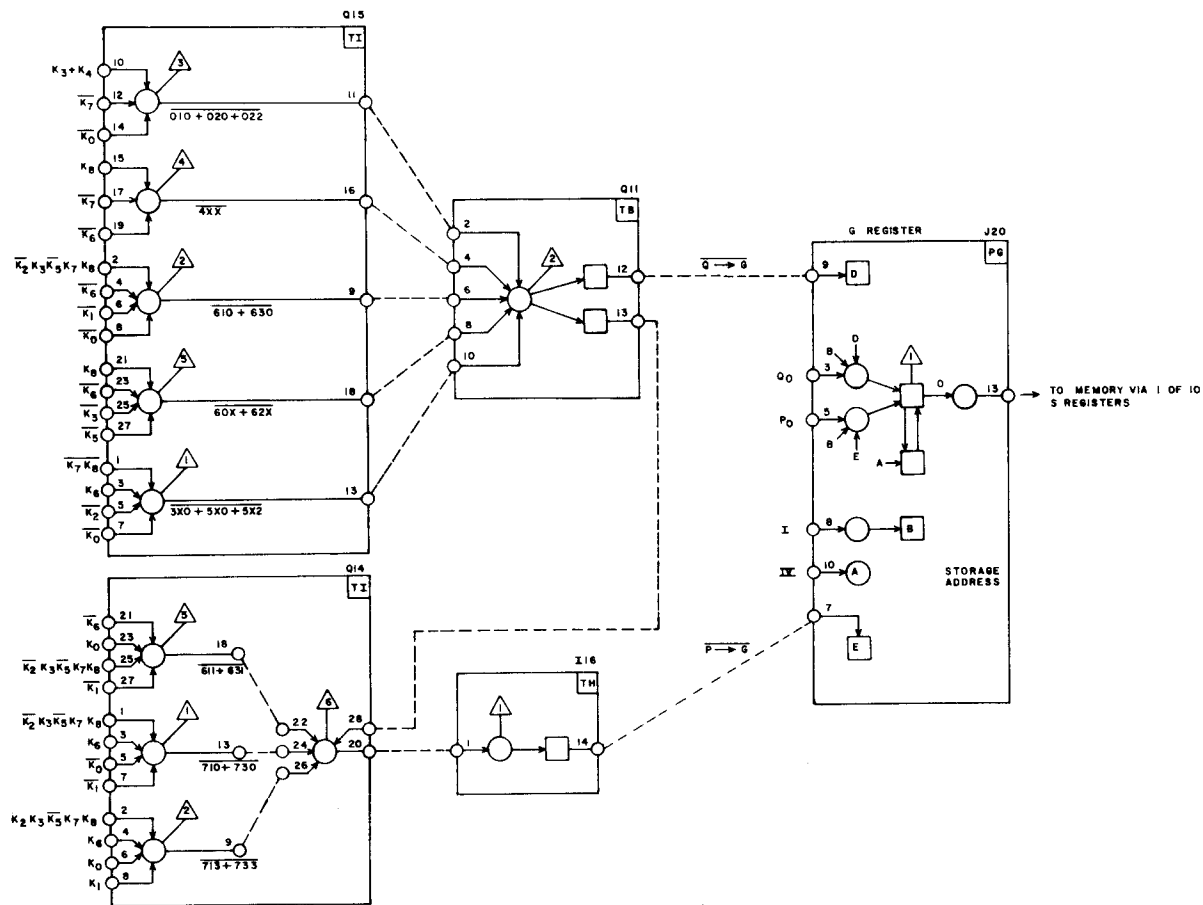


NOTES:
 1. SENSE AMPL. GOES TO "1" WHEN CORE SWITCHES.
 2. Z HOLDS COMPLEMENT OF WORD.
 3. SET STAGES OF Z ENABLE INHIBITS.



94

| | | | | | | | | | | | | | | | | | | | |



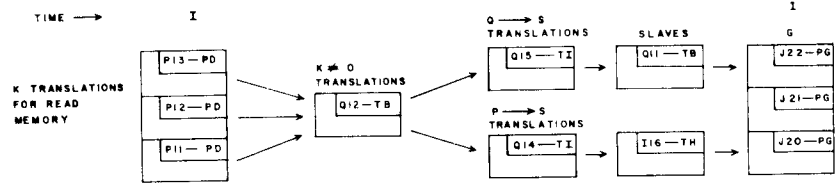
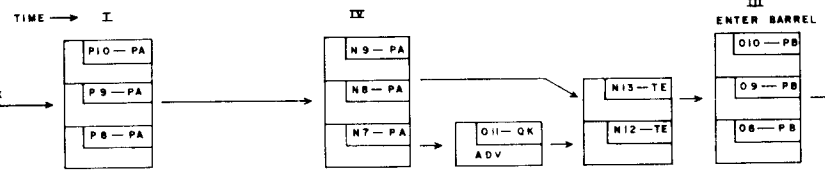
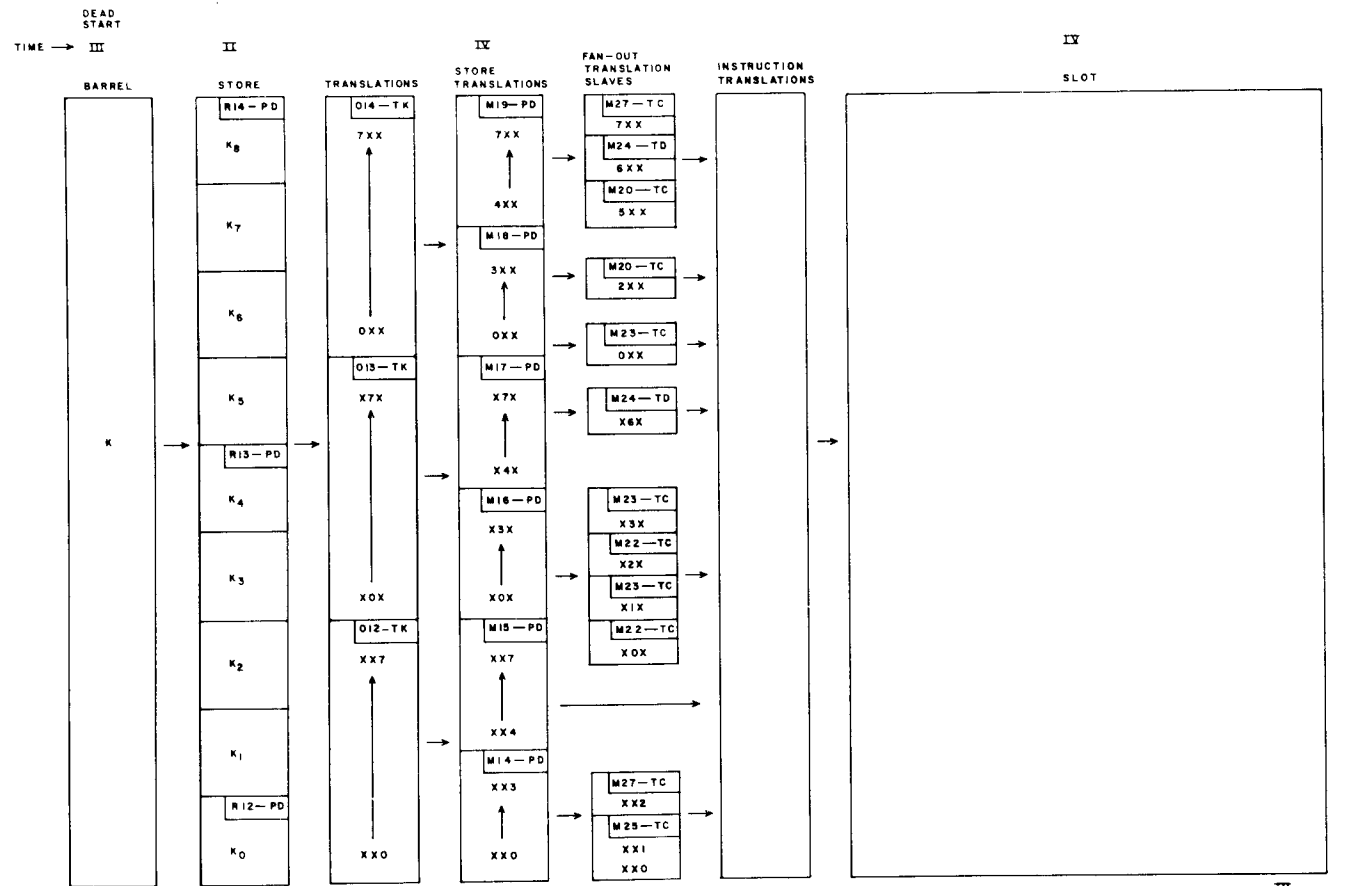
K REGISTER

K in the slot consists of a 3-bit trip counter for the lower three bits and a fan-in for the upper six bits. The advance K signal to the trip counter is enabled by instruction translations. For some instructions, the advance K signal is controlled by signals which indicate status, i.e., the 5X0 trip is skipped by all 5X instructions if $d = 0$, and when $K = 732$, K is advanced only if the I/O channel is empty and active and $A = 1$.

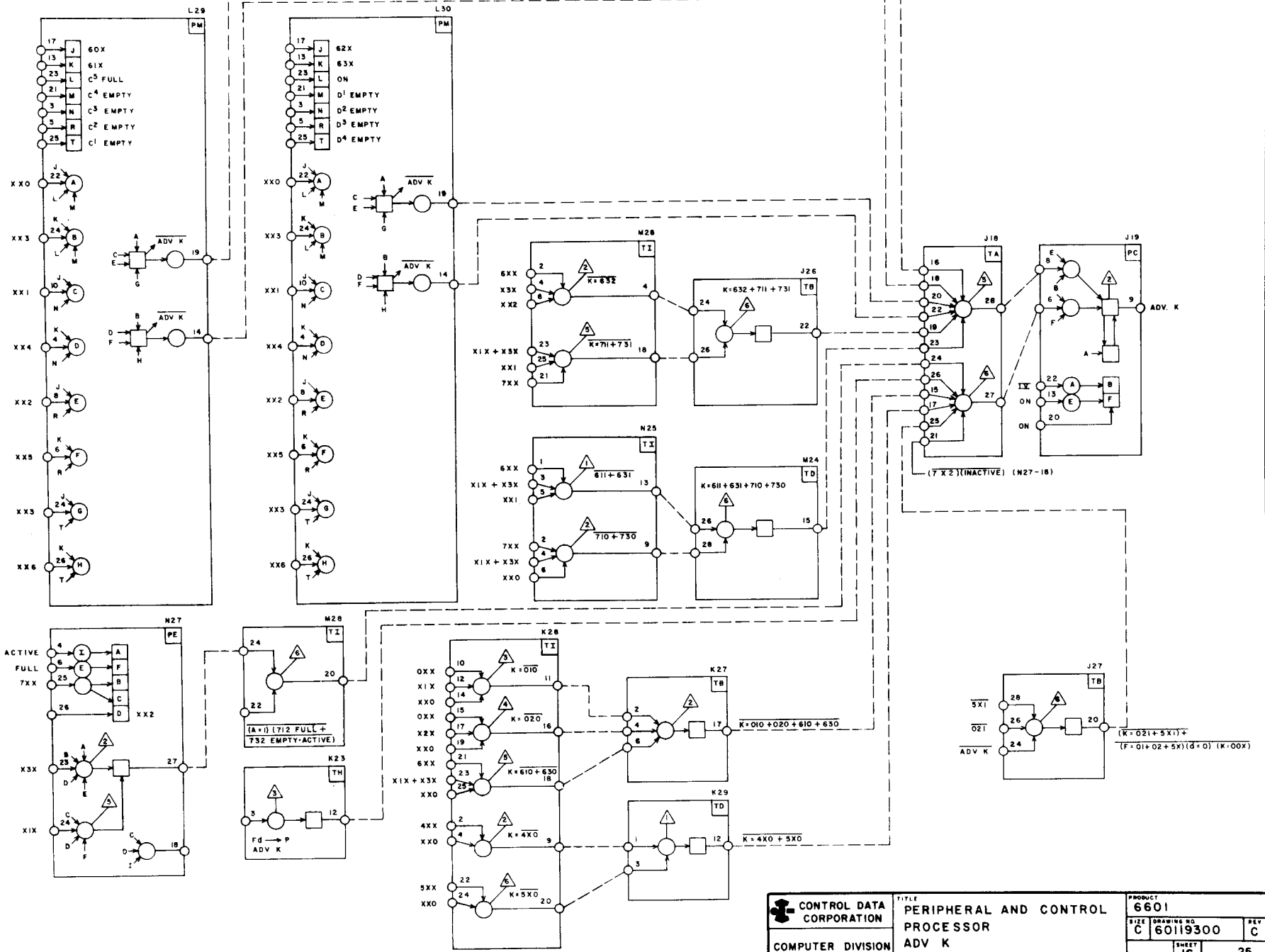
The three-bit trip count controls the sequence of operations for each instruction and is sometimes changed by gates other than the trip counter. For instance, for a central write instruction (63), K is changed from 637

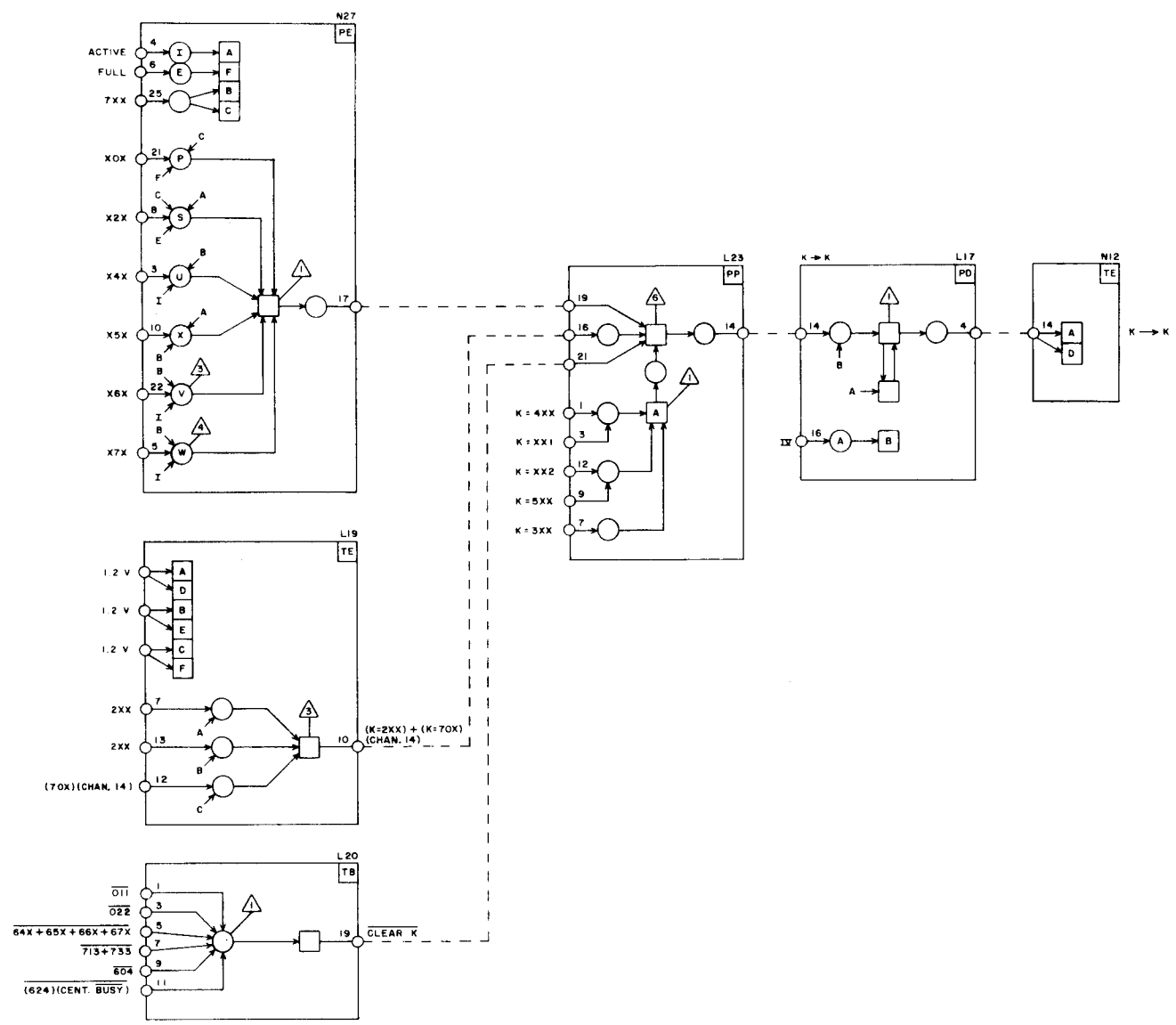
to 633 to repeat the sequence of commands and send another word. When a 63 instruction is completed, K is changed from 637 to 733 to finalize the instruction and obtain the next instruction from storage.

The fan-in to the upper six bits of K allows the instruction code F to be entered into K from storage. The $K \rightarrow K$ path allows another trip around the barrel for the present instruction. The path $K = 340$ is used by replace instructions which automatically use the store instruction 34 to accomplish the store portion of the replace instructions.

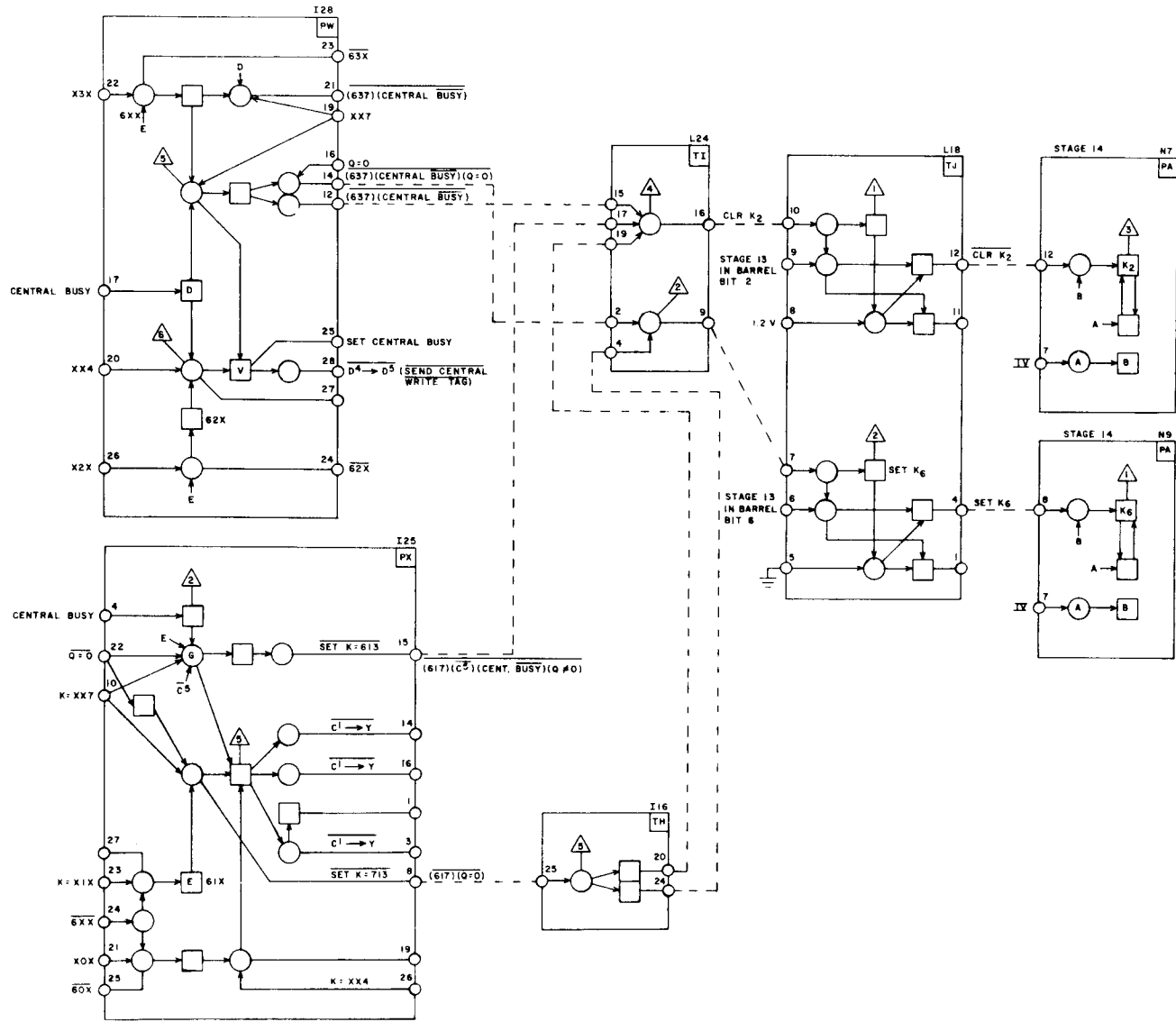


CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR K TRANSLATIONS, GENERAL		PRODUCT 6601
	SIZE C	DRAWING NO. 60119300	REV. C
	SHEET 15	23	

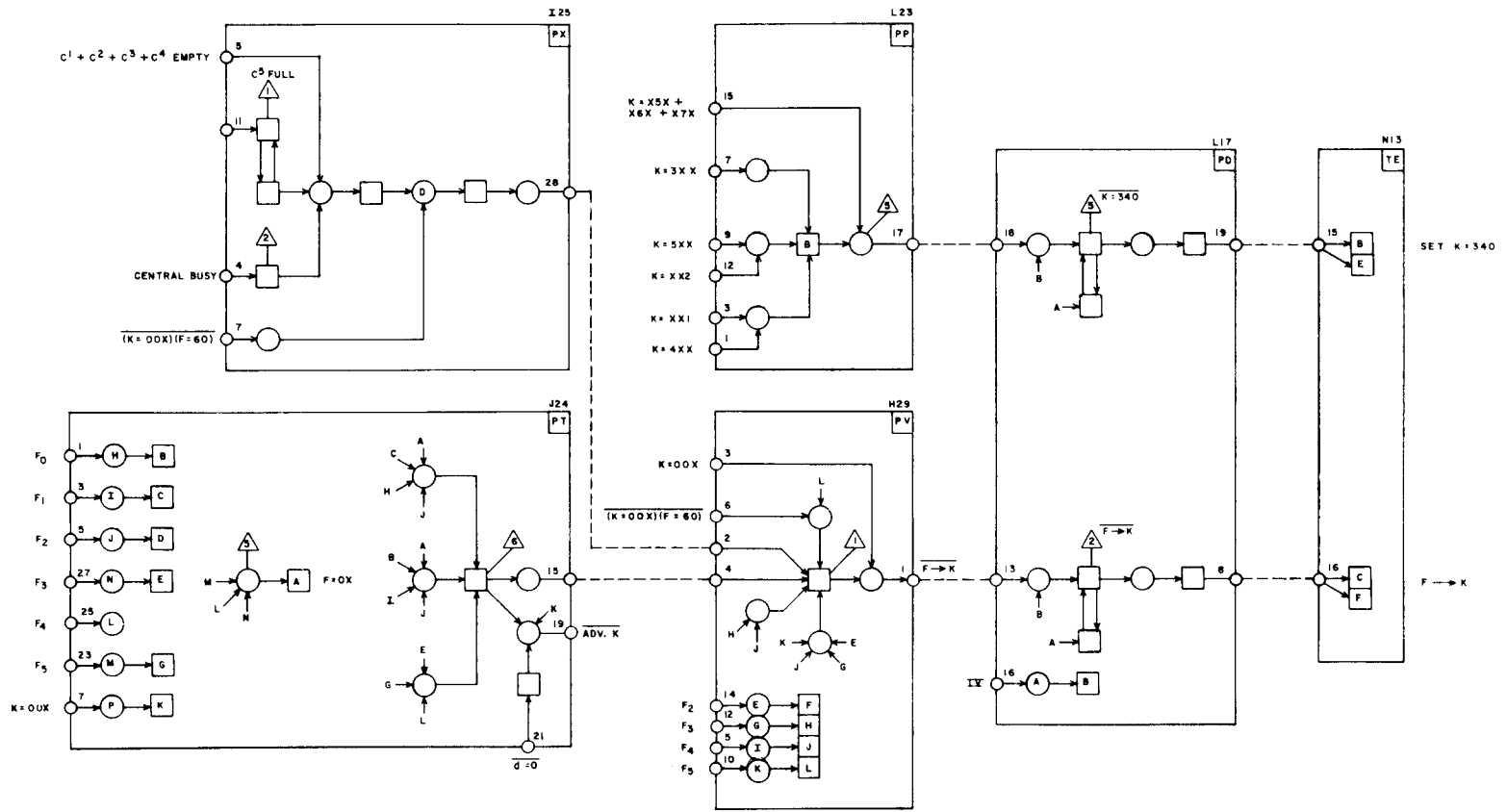


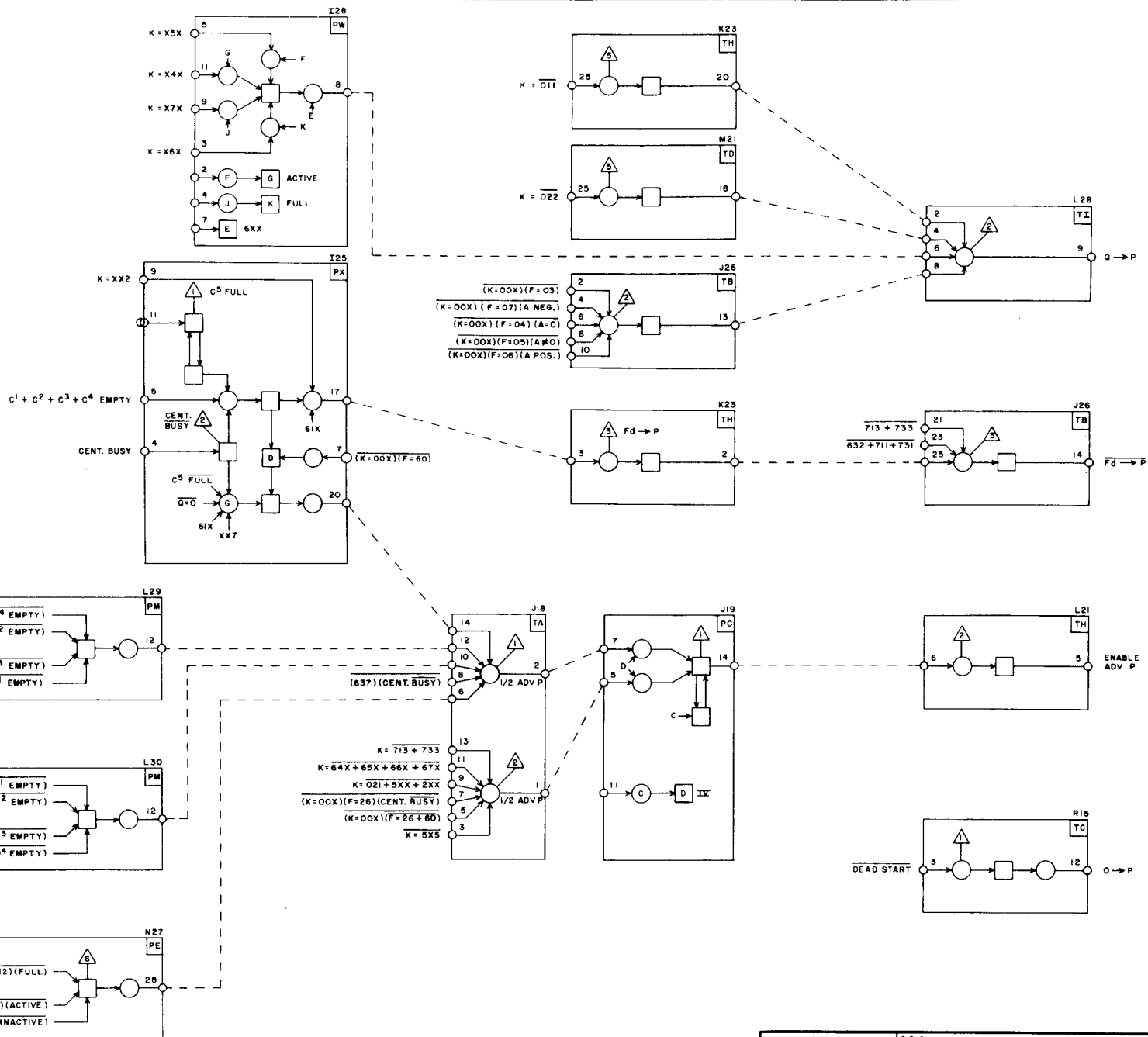


CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE	PRODUCT
	PERIPHERAL AND CONTROL PROCESSOR	6601
	K → K GATE	SIZE DRAWING NO. C 60119300
	15-SHEET 17	REV L 27



CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR CLR K₂, SET K₆	PRODUCT 6601
	SIZE C 60119300	REV L
SHEET 18		REV 29





A ADDER

The A adder is used to execute add, subtract, selective clear, logical product, and logical difference instructions. Parts of the A adder are also used to enter a word into the shift network and gate the result back to the barrel. The quantity in A in the barrel is always complemented when it enters the slot. When no operation on A is called for, (A) is complemented, enters the A adder, is added to zero, and the result is recomplemented at the output. The Add gate on the QD modules is always enabled except when Selective Clear, Logical Product, or Shift commands are enabled.

Add

For an add instruction (A) is complemented and entered into the A input register. The second operand is also complemented and entered into the B input register. The two quantities in the input registers, taken as positive, are added and the sum is re-complemented as it is gated out of the QD modules to the barrel.

Subtract

For subtract instructions, the minuend, (A) is complemented as it enters the adder. The subtrahend is entered into B without being complemented and the two quantities are added as in an add instruction.

Selective Clear

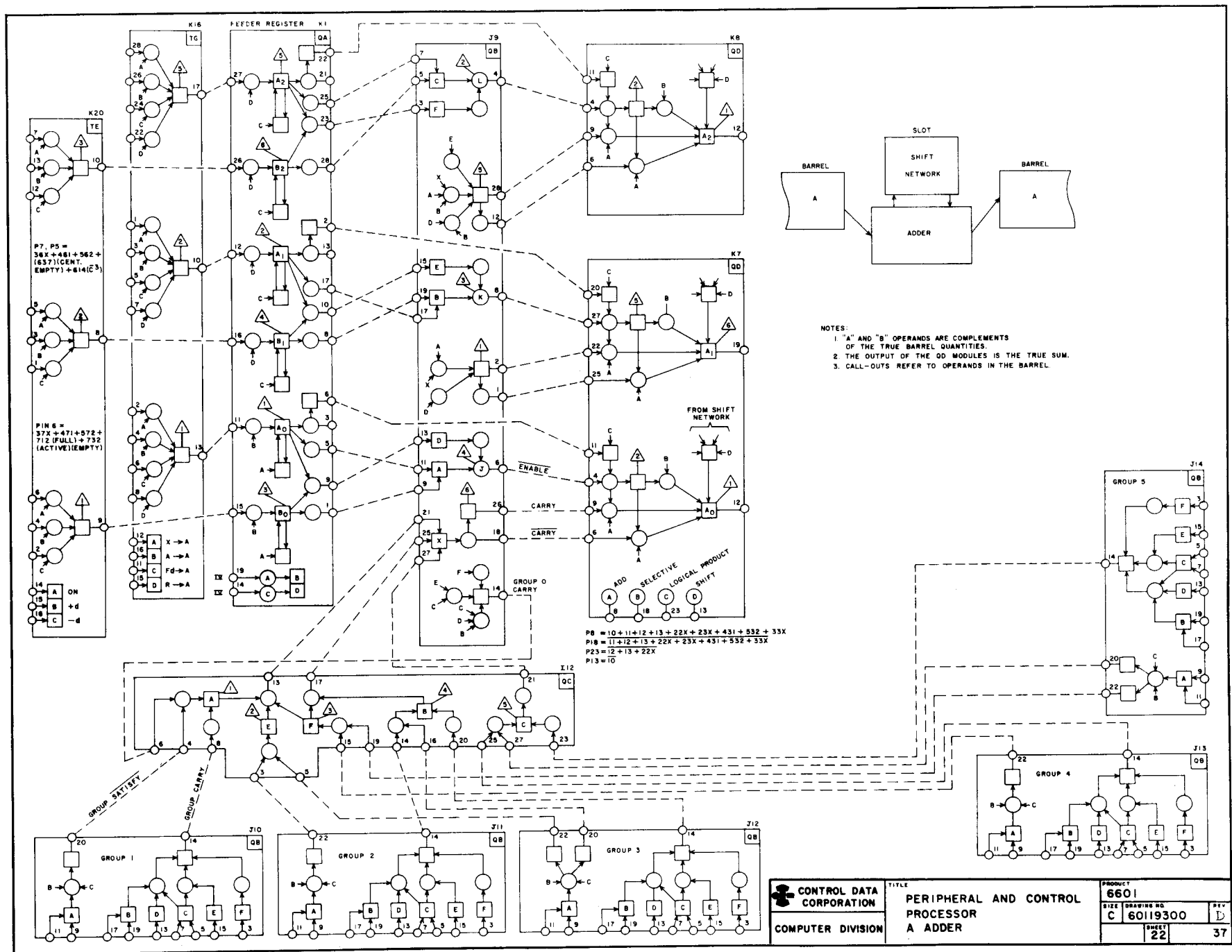
For selective clear, the complement of A and the true value of d are entered into the adder and both the selective and the logical product gates are enabled.

Logical Product

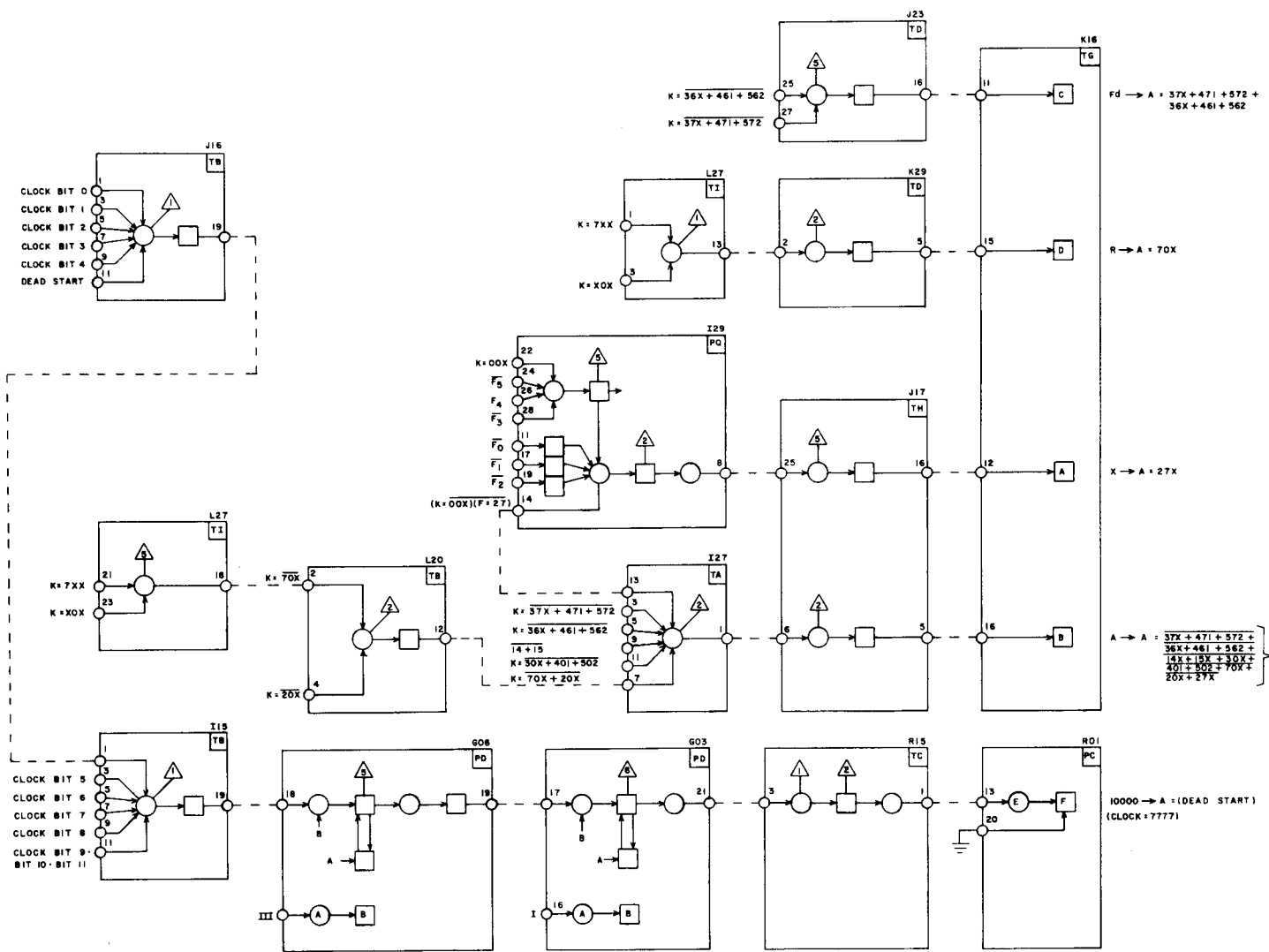
For logical product instructions, both A and d (or dm) are complemented before entering the adder and both the logical product and the selective gates are enabled.

Logical Difference

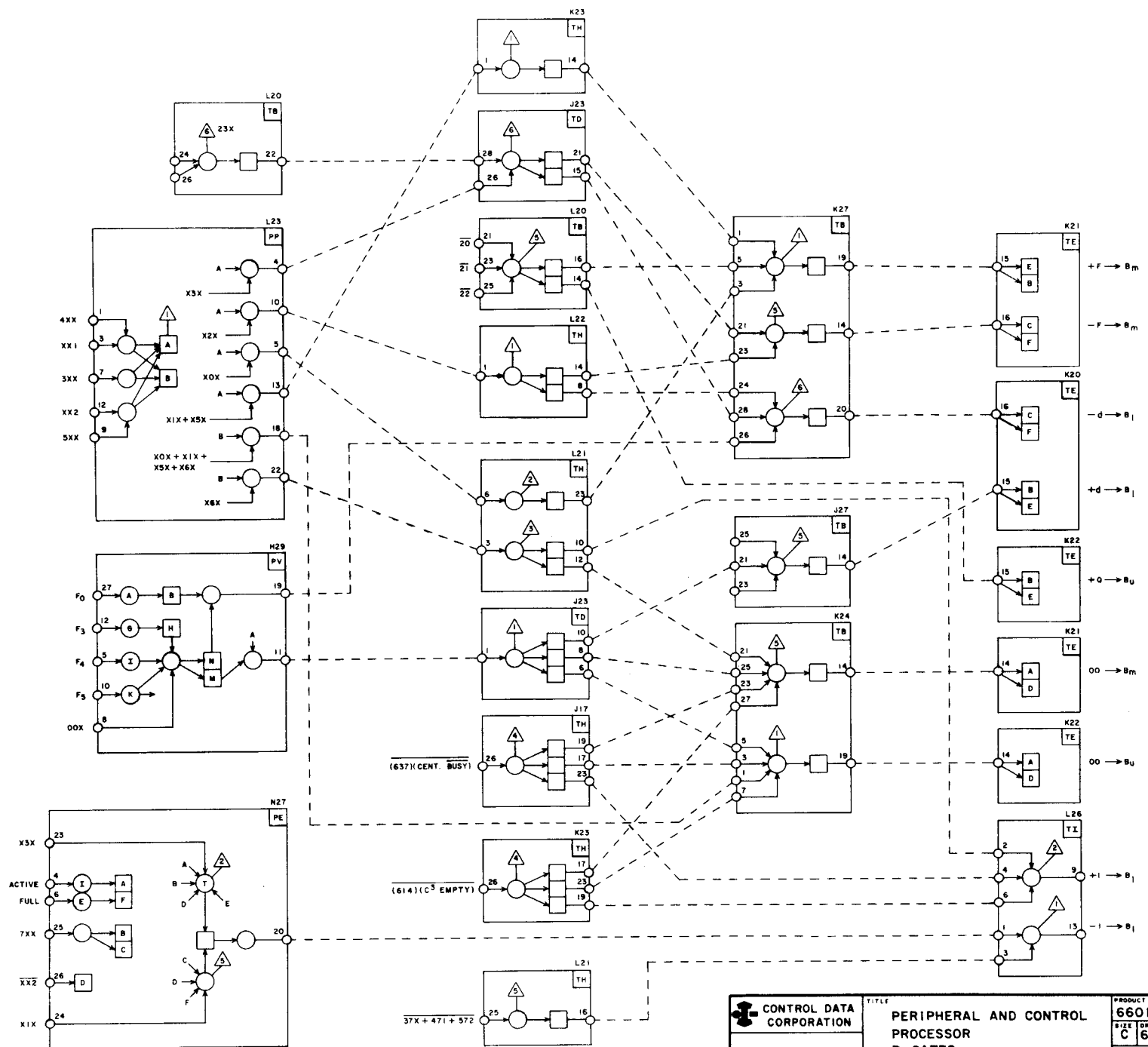
For logical difference instructions, the complement of A and the true value of the second operand enter the adder and only the selective gate is enabled.

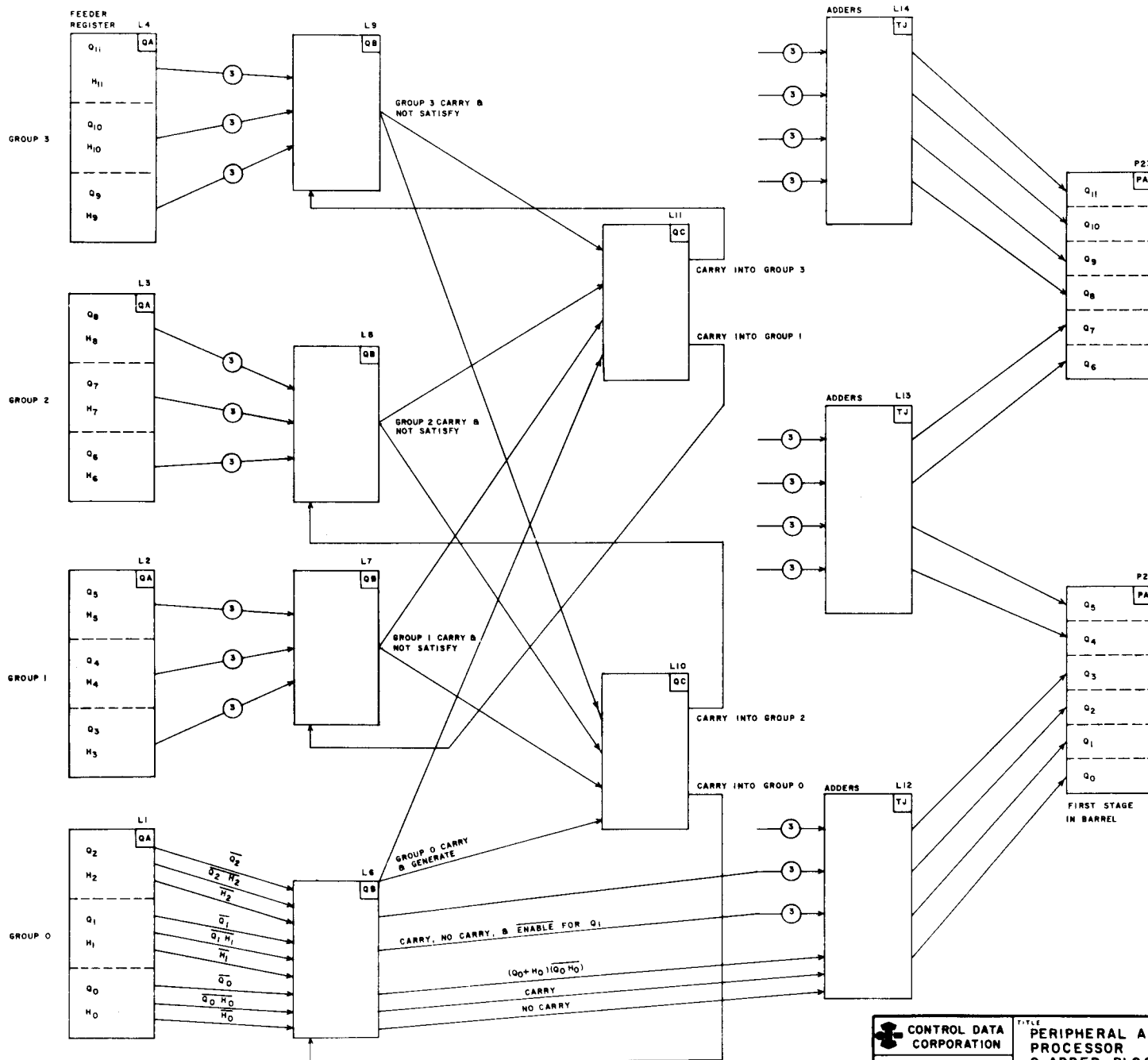


CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR A ADDER	PRODUCT 6601	REV. 1
		SIZE DRAWING NO. C 60119300	SHEET 22
		37	



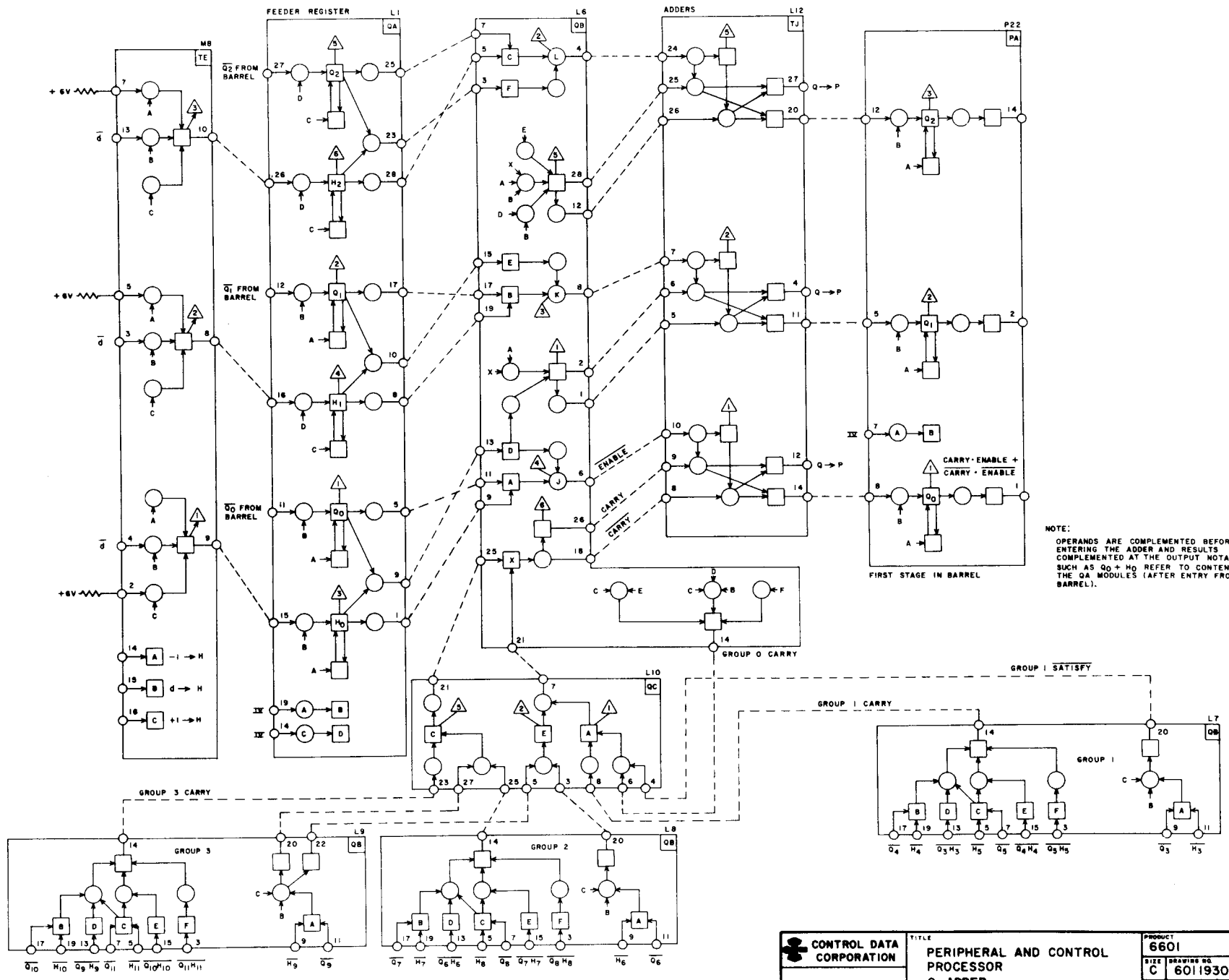
CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR A REGISTER GATES	PRODUCT 6601/04	REV K
		SIZE C	DRAWING NO. 60119300
		SHEET 23	39





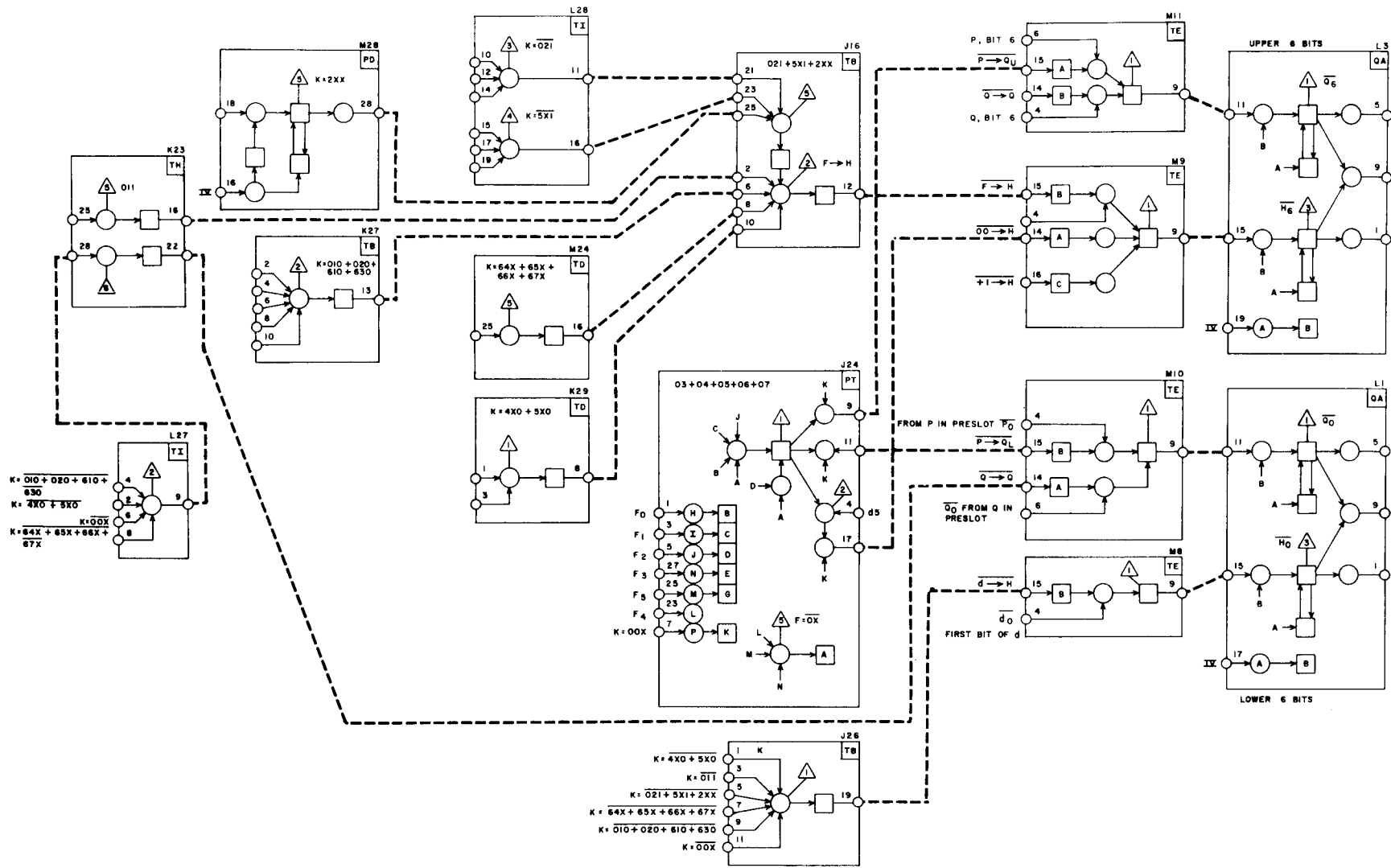
NOTE:
 OPERANDS ARE COMPLEMENTED BEFORE ENTERING ADDER AND RESULT IS COMPLEMENTED AT THE OUTPUT. NOTATIONS SUCH AS $Q_0 + H_0$ REFER TO CONTENT OF THE QA MODULES (AFTER ENTRY FROM BARREL).

CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR Q ADDER BLOCK DIAGRAM	PRODUCT 6601
	FILE DRAWING NO. C 60119300	REV C
SHEET 25		43

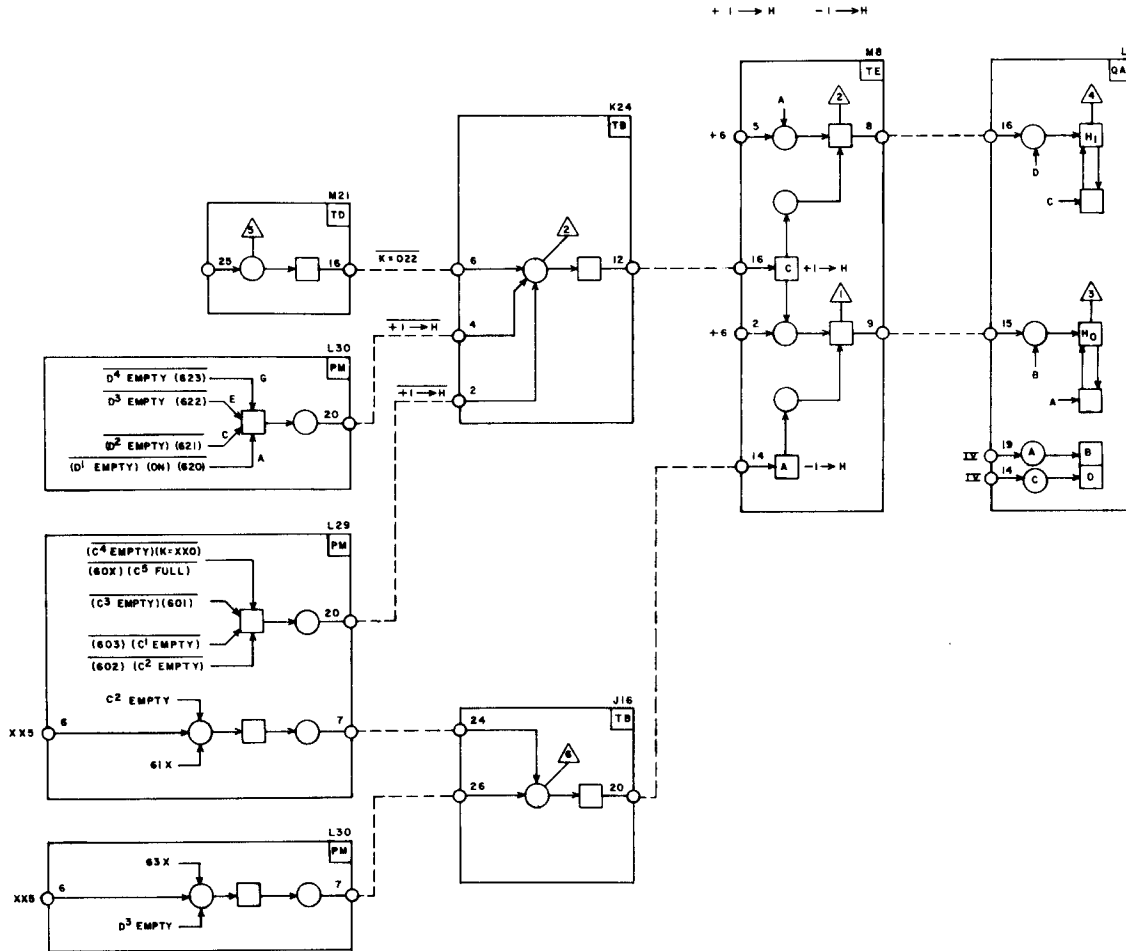


NOTE:
 OPERANDS ARE COMPLEMENTED BEFORE ENTERING THE ADDER AND RESULTS COMPLEMENTED AT THE OUTPUT NOTATIONS, SUCH AS Q₀ + H₀ REFER TO CONTENT OF THE Q_A MODULES (AFTER ENTRY FROM BARREL).

CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE	PERIPHERAL AND CONTROL PROCESSOR Q ADDER
	PRODUCT	6601
	SIZE	DRAWING NO. C 60119300
	REV	D
SHEET		26
REV		45



111



 CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR H GATES		PRODUCT 6601	
	SIZE C	DRAWING NO. 60119300	REV C	SHEET 28
			REV C	SHEET 49

SHIFT NETWORK

The shift instruction (10) provides for shifting the number in A up to 31 places left or right. Left shift is circular with the high order bits re-entering A at the low-order end. Right shift is end-off with low-order bits discarded as they shift out of the A register and with no sign extension. Thus, a left shift of 18 is equivalent to no shift, and a right shift of 18 clears the A register.

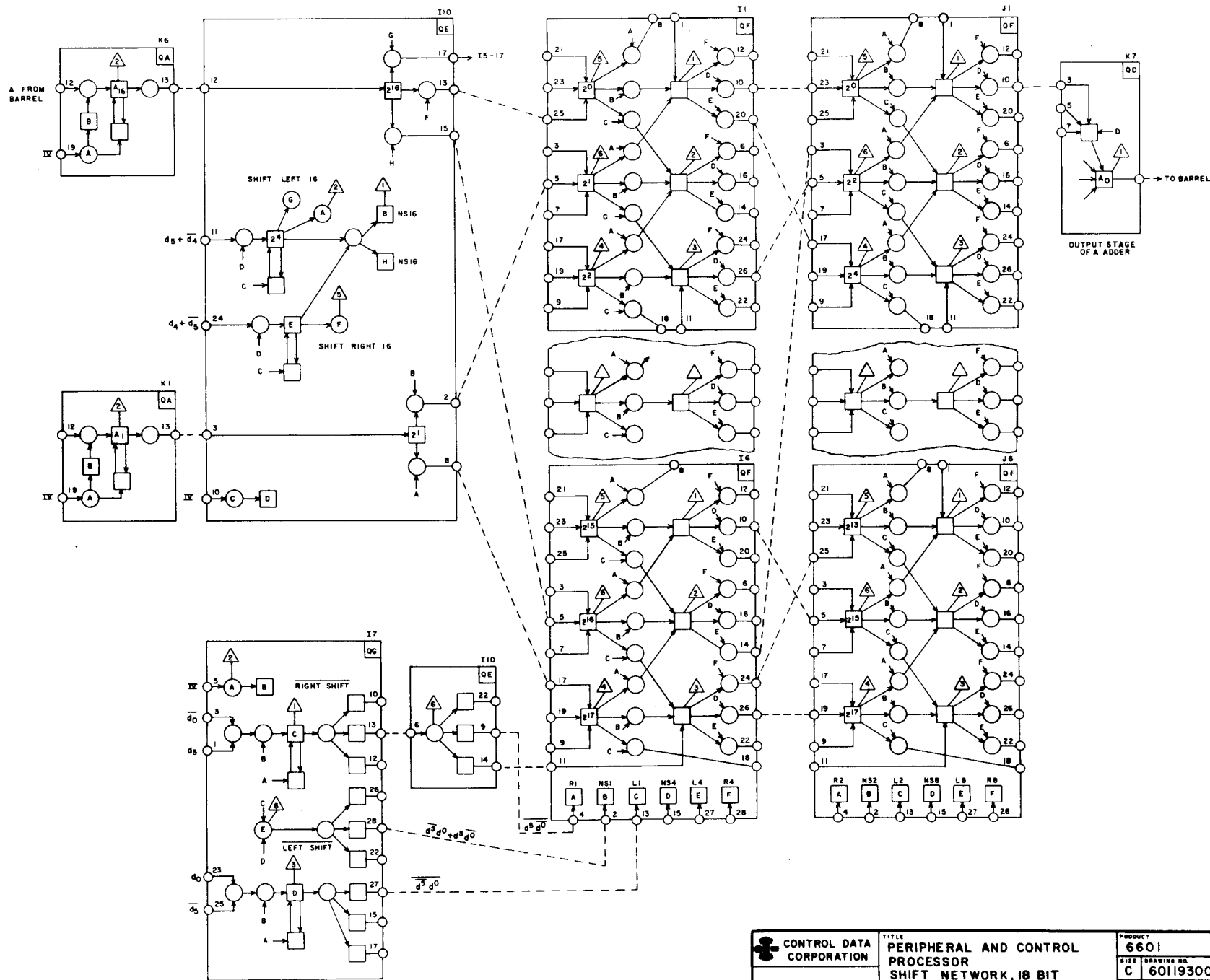
The shift network is a static network. The content of A enters the register at time IV, each bit follows a path established by static translations of the 6-bit shift count in d, and the result re-enters A in the barrel at the next time IV. The input to the shift network comes from the A input register in the A adder (the content of that register, which is the complement of A, is re-complemented before entering the shift register). The output of

the shift network is gated back to the barrel by way of the output modules (QD) of the A adder. Note that the quantity in A is always shifted but the result is gated to the barrel only when the current instruction is a shift.

If d is positive ($00-37_8$) the shift is left and the shift count is the content of d. If d is negative ($40-77_8$) the shift is right and the shift count is the complement of the number in d.

At the first stage of the shift network, d_4 and d_5 are tested to determine whether the shift is greater or less than 16 and whether it is right or left. If the shift is 16 or greater, a shift of 16 is made at this point and the result then enters the rest of the shift network.

Bits d_0-d_3 are tested with d_5 to set up paths through the rest of the network.



 CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR SHIFT NETWORK, 18 BIT	PRODUCT 6601
	SIZE DRAWING NO. C 60119300	REV D
	SHEET 29	51

COMMUNICATION WITH CENTRAL MEMORY
AND
CENTRAL PROCESSOR

The peripheral and control processors may communicate with the central processor and central memory in several ways. They may read the central processor's program address, tell the central processor to jump to given central memory address for its next instruction, or read from or write into central memory.

CENTRAL PROGRAM MONITOR

The 18-bit central processor program address is sent to the central program monitor register on chassis 1 every minor cycle. A Read Program Address instruction (27) sends the central address to the A register. Thus the progress of a central program may be monitored by any peripheral and control processor.

Exchange Jump, Central Read, and Central Write instructions all use the content of A as a central memory address. (A) is unconditionally sent to address control in the central processor every minor cycle. This quantity is recognized and used as a central memory address only if accompanied by a Central Read, Central Write, or Exchange Jump signal.

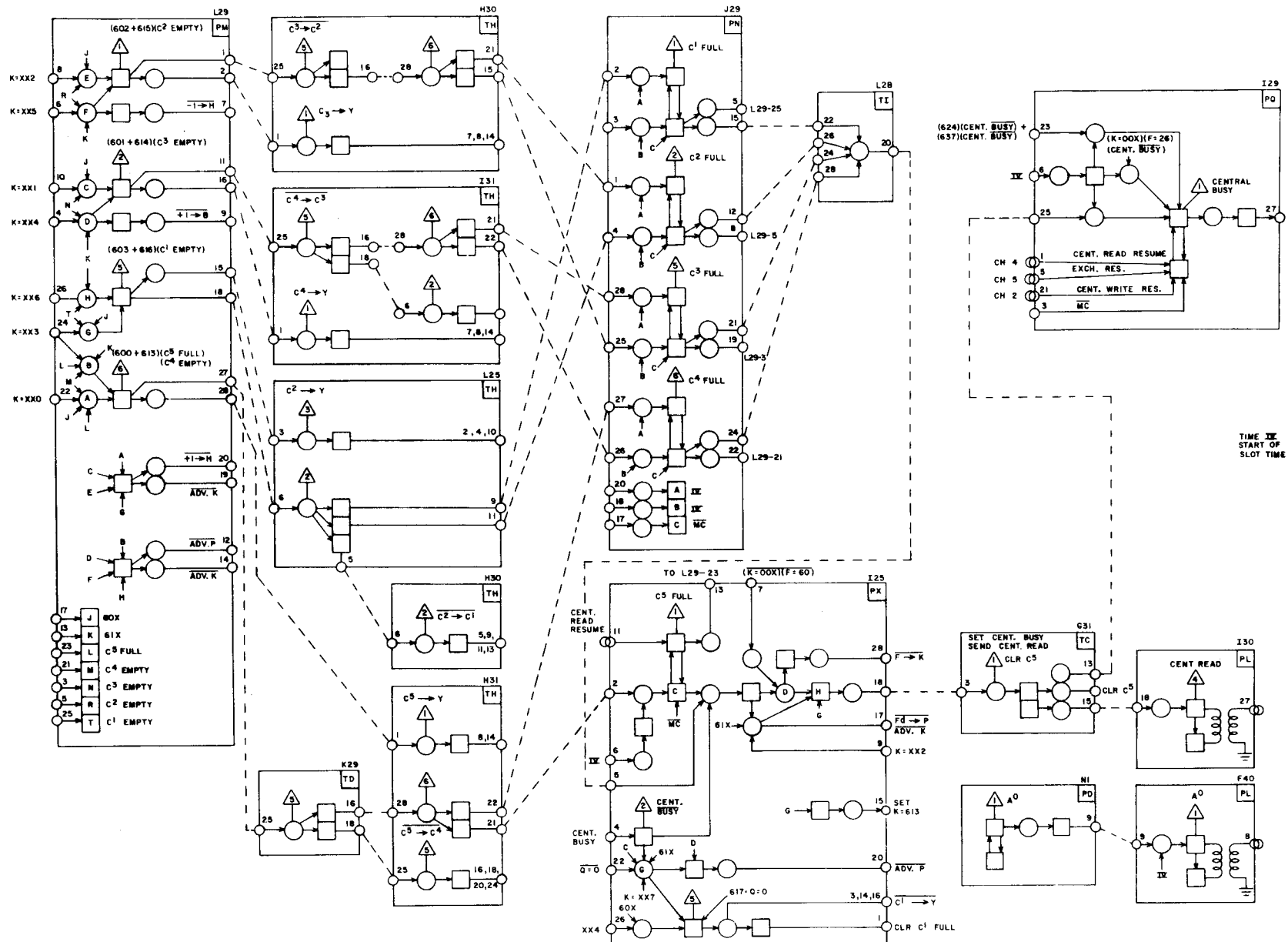
The Central Busy FF indicates when a reference to central is in progress. A central Busy condition prevents initiating a central reference until one in progress is completed.

EXCHANGE JUMP

An exchange jump instruction is used to command the central processor to stop the program it is executing and go to a central memory location specified by the instruction. An exchange jump may be issued by any peripheral and control processor so long as the Central Busy FF is clear. The instruction sends an Exchange Jump signal to the central processor and sets the Central Busy FF. The Exchange Jump signal tells the central processor to recognize the 18-bit address sent from the peripheral processor and to perform an exchange jump. After the central processor has performed the exchange jump and started a new program it sends a Resume signal which clears the Central Busy FF to allow another central reference. If a peripheral and control processor tries to issue an Exchange Jump instruction while the Central Busy FF is set, the processor must wait until the previous central reference is completed and the Central Busy FF is cleared.

Peripheral and Control
Processors

Pub. No. 60119300
Rev. C Page 52



CONTROL DATA CORPORATION
COMPUTER DIVISION

TITLE
PERIPHERAL AND CONTROL PROCESSOR
CENTRAL READ CONTROL

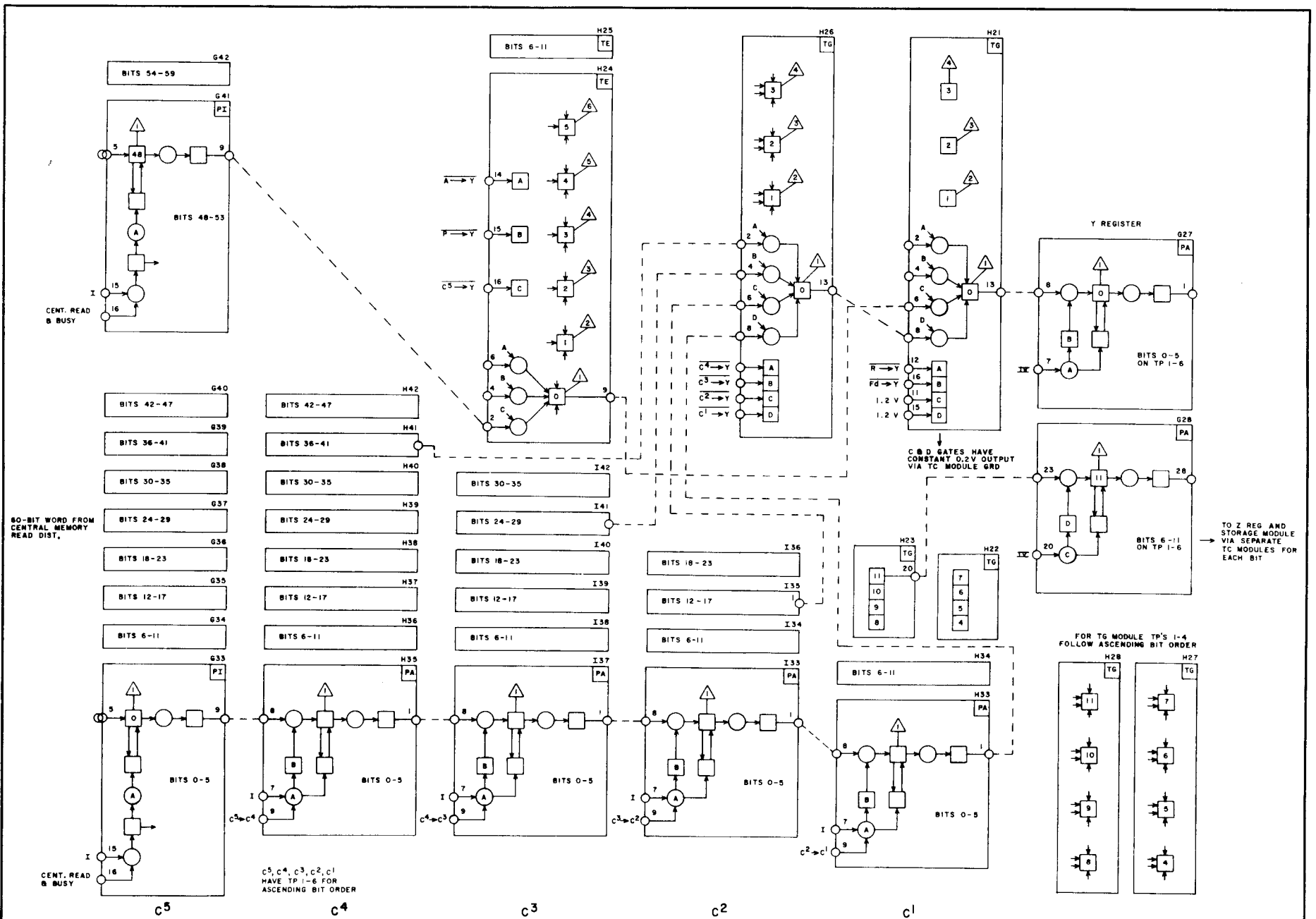
PRODUCT
6601
SIZE **C** DRAWING NO. **60119300** REV **D**
SHEET **30** OF **53**

CENTRAL READ

The Central Read instruction allows a peripheral and control processor to obtain one word (60-bits) or a block of words from Central Memory. The instruction sends a Central Read signal to central address control enabling it to use the 18 bit quantity from A as a central memory address. At the same time, the Central Busy FF is set to inhibit other references to central until the read word is received. When a 60-bit word is sent by central to the Central Read Pyramid, it is accompanied by two control signals; ~~Read Resume~~ ^{Read Resume} which clears the Central Busy FF and a signal which sets the C^5 Full FF. Each rank of the Central Read Pyramid C^1 - C^5 has an associated Full/Empty FF used to control the flow of data through the pyramid. C^5 Full and C^4 Empty enables the processor doing the read instruction to send the upper 12 bits of C^5 to memory and the lower 48 bits to C^4 . Subsequent steps in the Central Read instruction step the central word down through the pyramid and store the rest of the central word as 12-bit peripheral words. Each step in this storage procedure

requires that the next lower rank in the pyramid be empty before a transfer is made. No Central Read instruction may be issued until the C^5 Full FF and Central Busy FF are clear. However, as many as ^{four} central memory words, in different stages of disassembly, may be in the Central Read Pyramid at one time. A read instruction for which the proper full and empty conditions are not met must wait until previous instructions progress further and conditions are met.

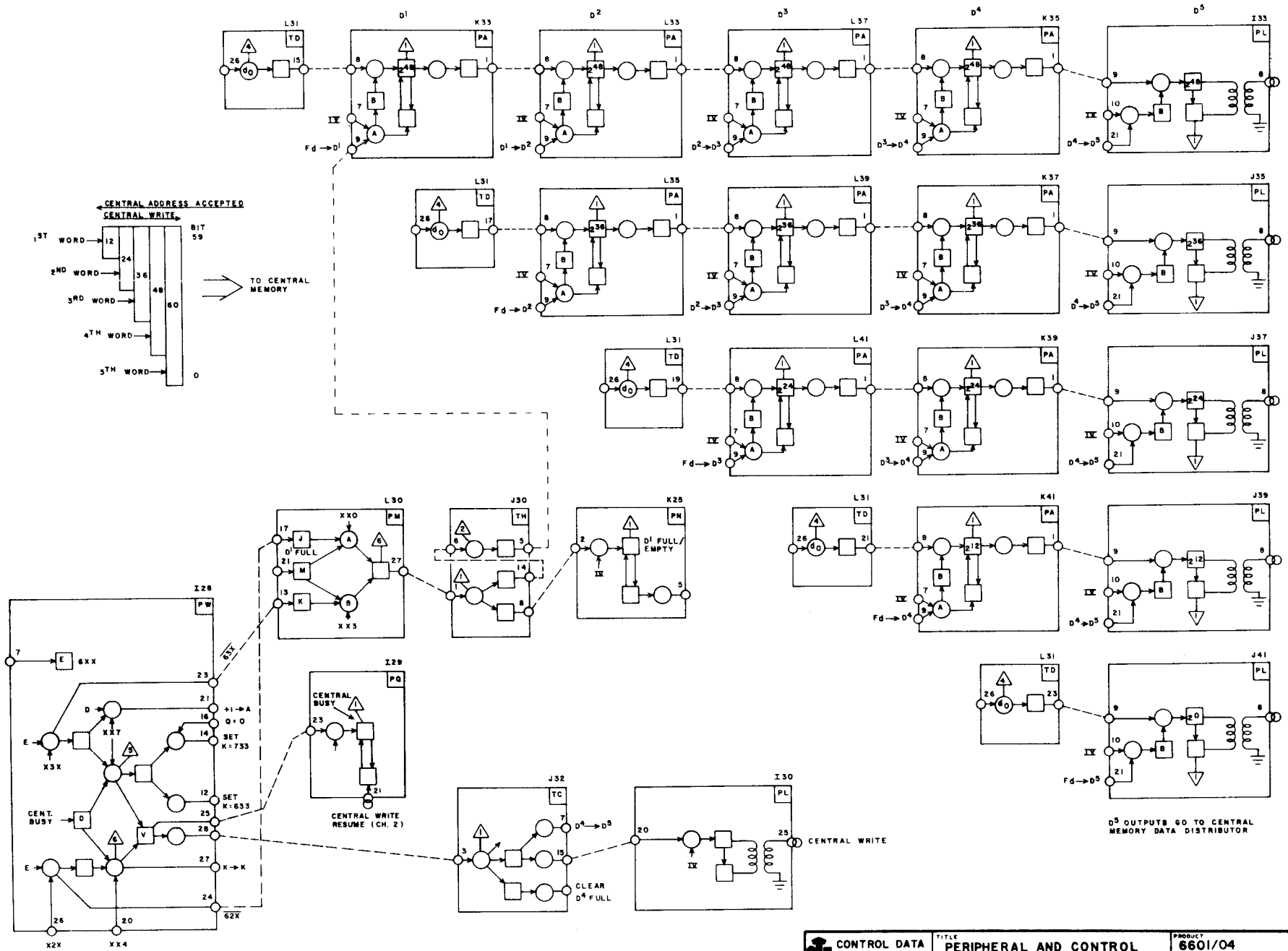
A 60 instruction reads only one central memory word and stores it as five peripheral words. A 61 instruction reads a block of words specified by (d). In either instruction the first central memory address is specified by (A). For a 60 instruction, d specifies the peripheral address at which the upper 12 bits of the peripheral word are stored; the next lower 12 bits go to d + 1, etc. For a 61 instruction, (d) gives the number of central words to be read and m is the address for the upper 12 bits of the first central word.



CENTRAL WRITE

Central Write instructions send one 60-bit word or a block of 60-bit words to Central Memory. Each 60-bit word sent to Central Memory is assembled in the Central Write Pyramid from five 12-bit peripheral words. A Central Write instruction assembles a 60-bit word and sends the word and a Central Write signal to central address control and sets the Central Busy FF. The Central Write signal enables central address control to accept the 60-bit word and store it at the

address specified by (A). When the word has been stored, an accept signal is sent back to clear the Central Busy FF. Up to four Central Write instructions may be in progress at one time with portions of four different words in D^1 - D^4 . D^5 is an output network only and cannot store a word. The first 12-bit word goes to D^1 and will be the upper 12 bits of the 60-bit word. When a second 12-bit word goes to D^2 , D^1 is also sent to D^2 . When the fifth word goes to D^5 , the 48 bits in D^4 are also sent to D^5 and the 60-bit word is sent to central.



CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR CENTRAL WRITE PYRAMID	PRODUCT 6601/04
	SIZE C 60119300	REV K

INPUT/OUTPUT

Each of the 12 independent data channels can handle 12-bit words at a maximum rate of one word every major cycle (equivalent to a 1 megacycle rate). Each channel has an Active/Inactive FF and a Full/Empty FF which indicate channel status to the processors. Any channel may be used by any processor, but the external equipment assigned to a channel is wired in and may be assigned to another channel only by changing cable connections.

The lines of a data channel are:

<u>Input</u>	<u>Output</u>
Data or Status Reply (12 bits)	Data or Function word (12 bits)
Active	Active
Inactive (Disconnect)	Inactive
Full	Full
Empty	Empty
	MC

In addition, two clock signals are available to external equipment: a 1 mc clock and a 10 mc clock. The clock pulses are 25 nsec wide, as are all data and control signals (except master clear). Controllers for each external equipment (or group) perform the conversion between the 6600 pulse signals and the signals required by I/O devices.

A data channel may be used for communication between processors if it is selected for input by one processor and for output by another. The status of data channels may be sensed by instructions 64-67: jump to m if channel d active, etc.

MASTER CLEAR

A Master Clear (MC) signal is generated only by the Dead Start circuit. MC removes all equipment selections (except Dead Start) and sets all channels to the Active and Empty condition (ready for input). MC is a

1 usec pulse which is repeated every 4096 usec while the Dead Start switch is on.

DISCONNECT (75)

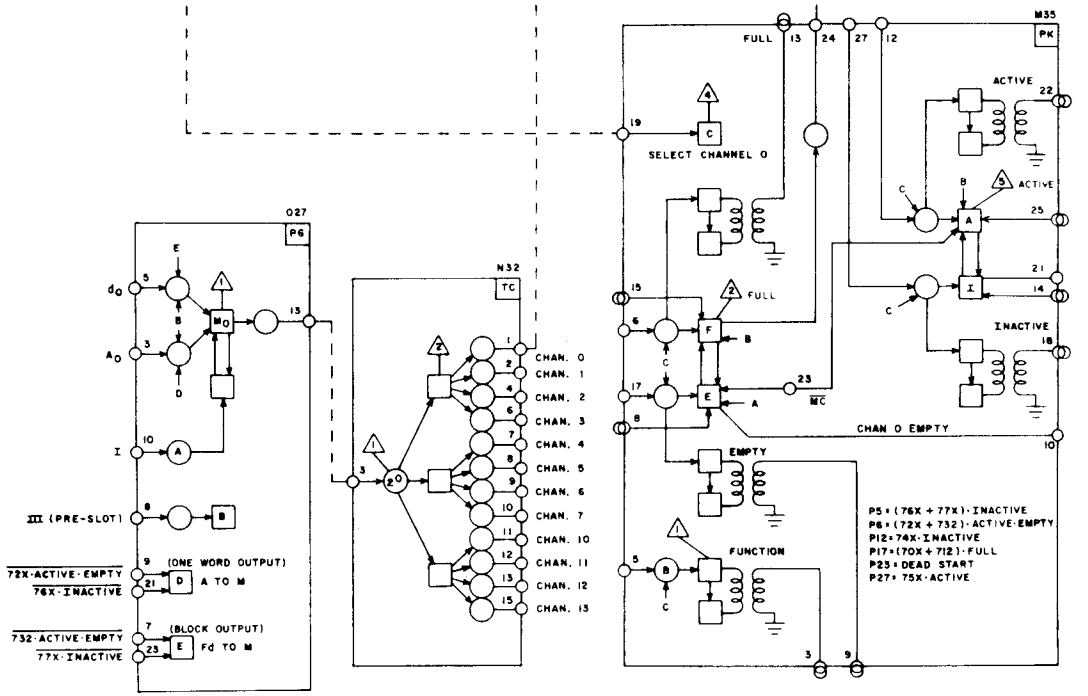
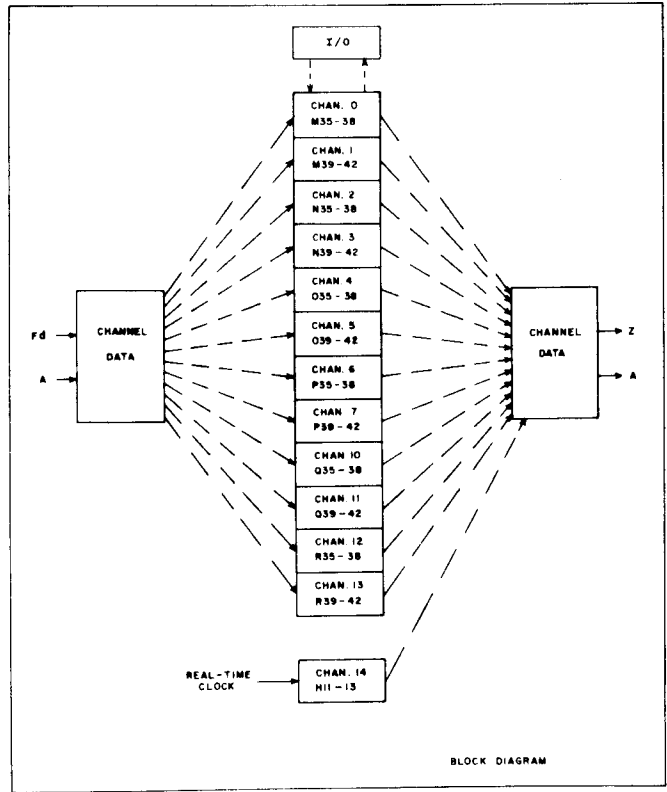
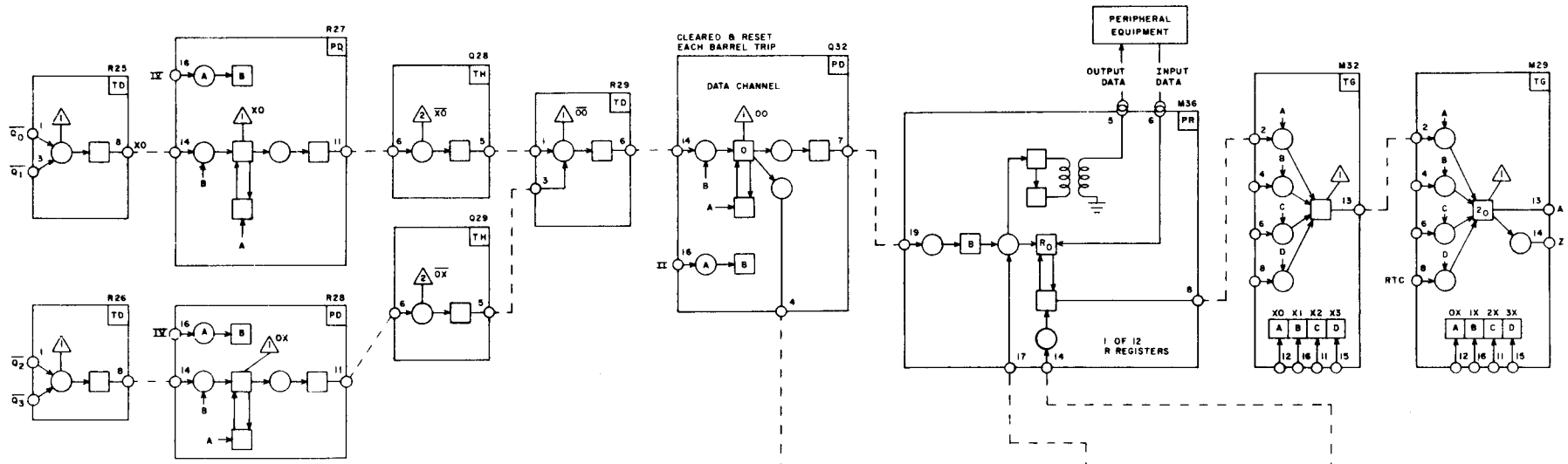
A disconnect instruction clears the channel Active FF if it is set and sends an inactive pulse to the equipment on that channel. If a disconnect instruction is given for a channel which is already inactive, the processor which issued the disconnect will "hang up" and will not be able to continue until the channel is activated by another processor (or by MC).

FUNCTION (76 or 77)

A function instruction sends a 12-bit function code (from A or Fd) on the data lines and sends a Function signal. It also sets the Active and Full FFs for the channel but does not send Active and Full pulses. Upon receipt of the function code, the external equipment sends an Inactive (disconnect) signal, clearing the Active FF in the data channel which in turn clears the Full FF. If a Function instruction is given for an active channel, the processor will hang up until the channel is deactivated.

ACTIVATE (74)

An Activate instruction sends an Active signal on the channel and sets the Active FF if the channel is inactive. If an Activate instruction is given for a channel which is already active, the processor which issued the instruction will "hang up" until the channel is inactivated by another processor or by an Inactive (disconnect) signal from an external equipment on the channel.



DATA INPUT SEQUENCE

An external device sends data to the processor by way of the controller in the following manner:

- 1 The processor places a function word in the channel register and sets the full flag and the channel active flag. Coincidentally, it sends the word and a function signal to all controllers. The function signal tells all controllers to sample the word and identifies the word as a function code rather than a data word. The code selects a controller and a mode of operation. Non-selected controllers clear, leaving only the selected one turned on.
- 2 The controller sends an inactive signal to the processor indicating acceptance of the function code. The signal drops the channel active flag which in turn drops the full flag and clears the channel register.
- 3 The processor sets the channel active flag and sends an active signal to the controller which signals the device to start sending data.
- 4 The device reads a word and then sends the word to the channel register with a full signal which sets the channel full flag.
- 5 The processor stores the word, drops the full flag, and returns an empty signal indicating acceptance of the word. The device clears its data register and prepares to send the next word.
- 6 Steps 4 and 5 repeat for each word transferred.
- 7 At the end of the transfer, the controller clears its active condition and sends an inactive signal to the processor to indicate end of data. The signal clears the channel active flag to disconnect the controller and the processor from the channel.
- 8 As an alternative, the processor may choose to disconnect from the channel before the device has sent all of its data. The processor does this by dropping the active flag and sending an inactive signal to the controller which immediately clears its active condition and sends no more data, although the device may continue to the end of its data record or cycle (e.g., a magnetic tape unit would continue to end of record and stop in the record gap).

STATUS REQUEST

A status request is a special one word data input transfer in which an external device indicates a ready or error condition to a processor.

- 1 The processor places a function word in the channel register and sets the full flag and the channel active flag. Coincidentally, it sends the word and a function signal to all controllers. The function signal tells all controllers to sample the word and defines

the word as a function code rather than a data word. The code selects a controller and places it in status mode. Non-selected controllers clear, leaving only the selected one turned on.

- 2 The controller sends an inactive signal to the processor indicating acceptance of the status function code. The signal drops the channel active flag which in turn drops the full flag and clears the channel register.
- 3 The processor sets the channel active flag and sends an active signal to the controller which signals the device to send the status word.
- 4 The controller sends the status word to the channel register with a full signal which sets the channel full flag.
- 5 The processor stores the word, drops the full flag, and returns an empty signal indicating acceptance of the word.
- 6 The processor drops the channel active flag to disconnect the channel and sends an inactive signal to the controller to disconnect it.

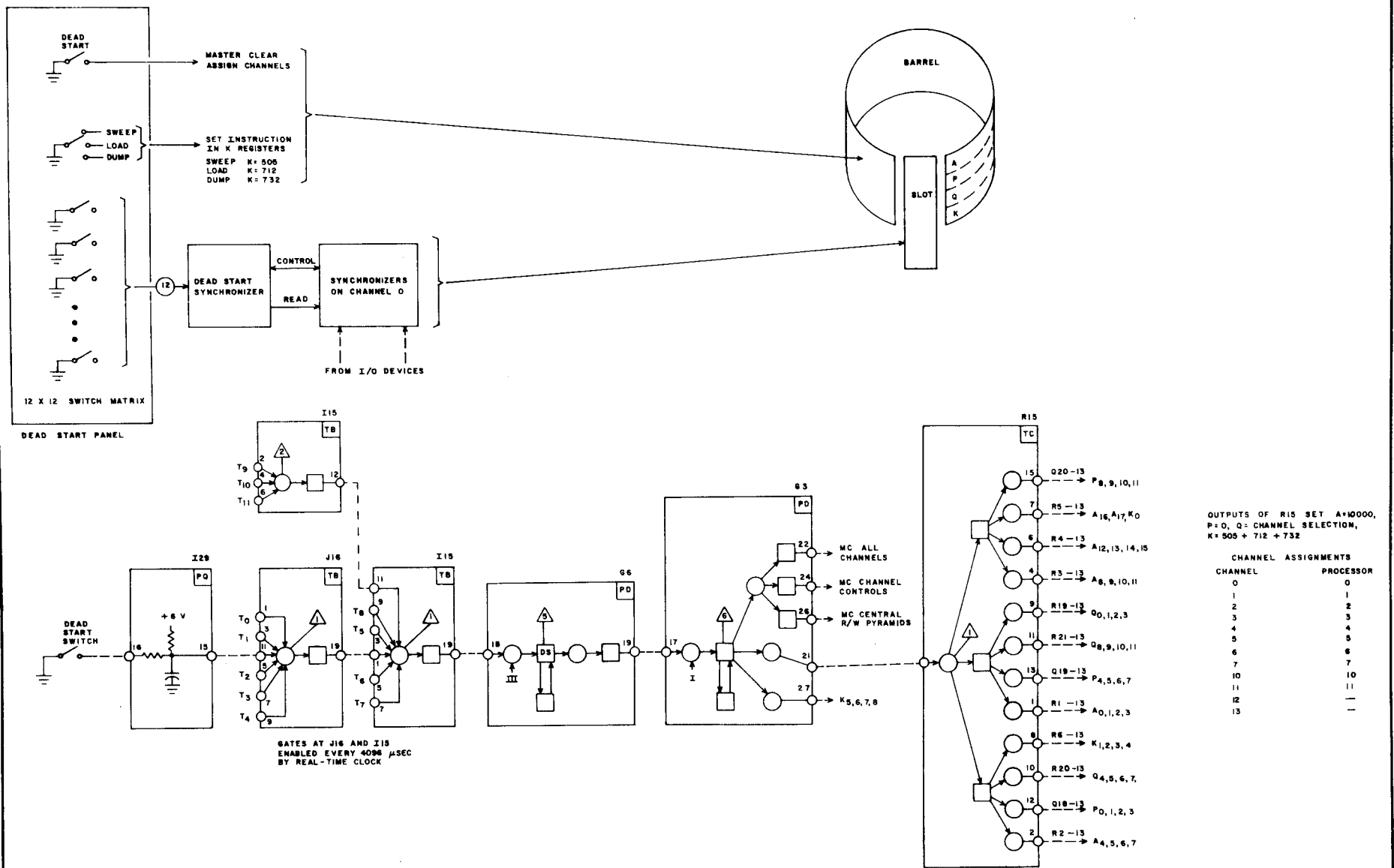
DATA OUTPUT SEQUENCE

The processor sends data to an external device in the following manner:

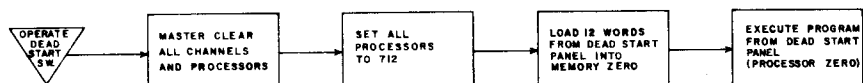
- 1 The processor places a function word in the channel register and sets the full flag and the channel active flag. Coincidentally, it sends the word and a function signal to all devices. The function signal tells all controllers to sample the word and identifies the word as a function code rather than a data word. The code selects a controller and a mode of operation. Non-selected controllers clear, leaving only the selected one turned on.
- 2 The controller sends an inactive signal to the processor, indicating acceptance of the function code. The signal drops the channel active flag which in turn drops the full flag and clears the channel register.
- 3 The processor sets the channel active flag and sends an active signal to the controller which signals the device that data flow is starting.
- 4 The processor places a data word in the channel register and sets the full flag. Coincidentally, it sends the word and a full signal to the controller.
- 5 The controller accepts the word and sends an empty signal to the processor where it clears the channel register and drops the full flag.
- 6 Steps 4 and 5 repeat for each processor word.
- 7 After the last word is transferred and acknowledged by the controller empty signal, the processor drops the channel active signal to the controller to turn it off.

Peripheral and Control
Processors

Pub. No. 60119300
Rev. C Page 60



DEAD START LOGIC ON CHASSIS I



DEAD START

Dead Start is a system used to initially start the computer, dump the contents of the peripheral and control processor memories to a printer or other output device, or sweep memory without executing instructions.

The Dead Start panel contains a 12x12 matrix of toggle switches, a Sweep-Load-Dump switch and a Dead Start switch. It also contains memory margin switches which are used for maintenance checks.

LOAD

To initially load programs and data, the Sweep-Load-Dump switch is put in the Load position. The matrix of toggle switches is set to a 12-word program (up = "1", down = "0"). When the Dead Start switch is turned on, a 1 usec Dead Start pulse:

- 1 Assigns to each peripheral and control processor the corresponding I/O channel.
- 2 Sets all channels to Active and Empty
- 3 Sets K for all processors to 712 (Input)
- 4 Sends a MC on all channels
- 5 Sets P for all processors to zero. (A is then set to 10000₈ in the barrel)

The Dead Start pulse is repeated every 4096 usec while the Dead Start switch is on. To start the machine, the DS switch is normally turned on momentarily, then off. Recycling of the DS pulse is controlled by the Real Time Clock; the pulse is formed by ANDing DS switch in the ON position with 10 bits of Real Time Clock.

When the Dead Start controller on channel 0 receives the MC sent by Dead Start, it sends a Full pulse but no data. When processor 0 receives the Full, it stores the content of the channel 0 input register (All zeros) in location 0000 and sends an Empty pulse to the Dead Start controller. The Dead Start controller then acts like an input device, sending twelve 12-bit words from the switch matrix which processor 0 stores in locations 0001-0014₈. After the last word, the Dead Start

controller sends a disconnect which causes processor 0 to exit from the 712 instruction. Processor 0 reads location 0000, adds one to its contents and goes to 0001 for its next instruction. It then executes the 12-word (or less) program which normally is a control program to load information and begin operation. The other processors are still set to 712 (waiting to input when their channels become full) and may receive data from processor 0 via their assigned I/O channels.

SWEEP

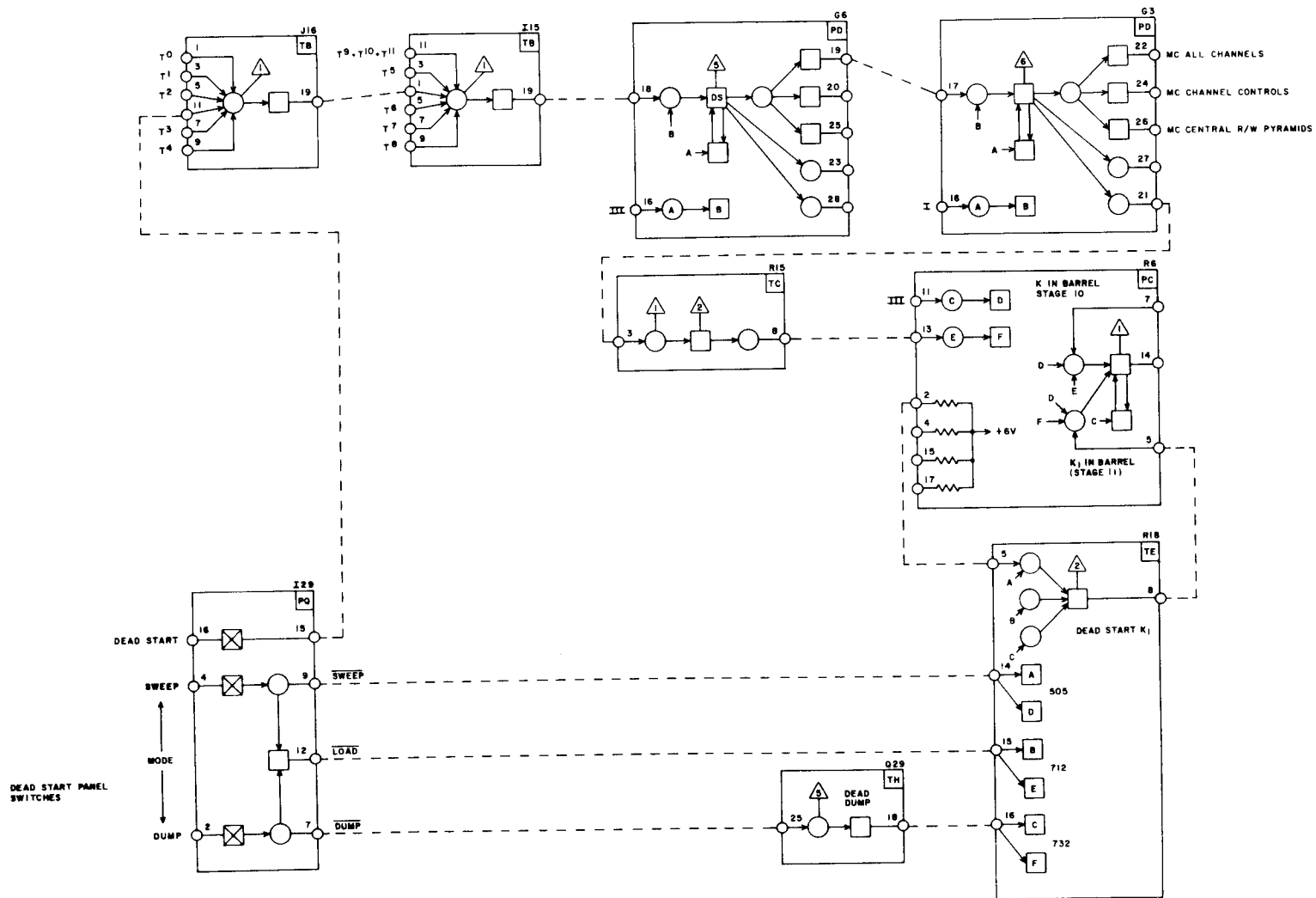
If the DS switch is operated with the Sweep-Load-Dump switch in the Sweep position, all processors are set to a 505 instruction and P registers set to 0000. Since the 50 instruction doesn't require 5 trips around the barrel there is no logic to clear or advance K from 505. The 50X translation of K, causes all processors to sweep through their memories; reading and restoring without executing instructions. This is a maintenance routine and may be used to check the operation of memory logic.

DUMP

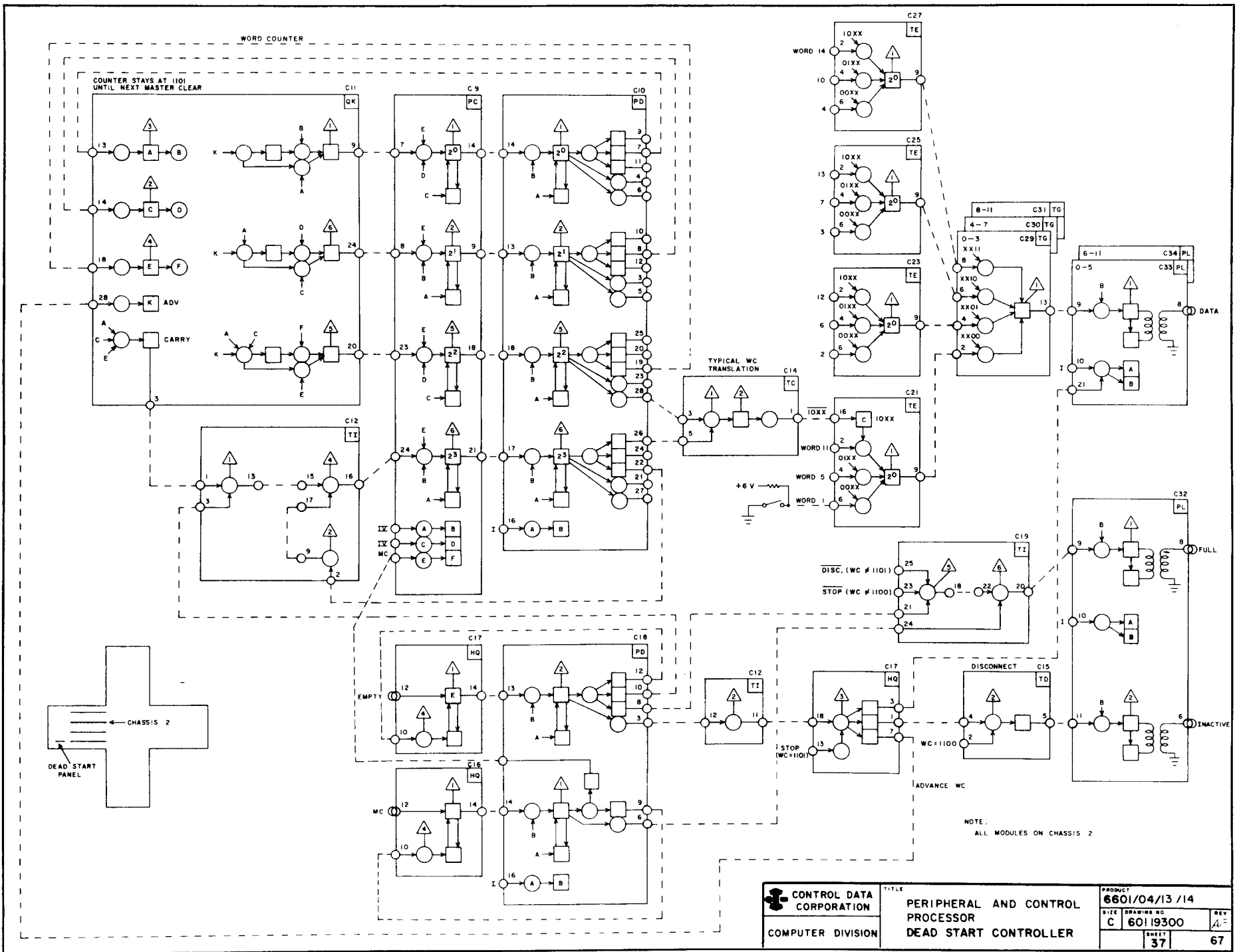
Dead Start with the Sweep-Load-Dump switch in the Dump position:

- 1 Sets all processors to 732.
- 2 Sends MC on all channels.
- 3 Holds channel 0 Active and Empty.
- 4 Assigns each processor to its corresponding I/O channel.
- 5 Sets all A and P registers to 0.

All processors sense the Empty and Active condition of their assigned channels, output the content of their address 0000, set their I/O channels to Full, and wait for an Empty. All processors advance P by one and reduce A by one ($A = 7776_8$). Channel 0, which is assigned to processor 0, is held by Empty by the Dump switch. Processor 0 therefore cycles through the 732 instruction until $A = 1$ and then goes to memory location 0001 for its next instruction. Processor 0 has sent its entire memory content on channel 0 although no I/O device was selected to receive it. Processor 0 is now free to execute a dump program which must have been previously stored in memory 0 (beginning at location 0001).



CONTROL DATA CORPORATION COMPUTER DIVISION	TITLE PERIPHERAL AND CONTROL PROCESSOR	PRODUCT 6601/04
	712, 732, 505 → K (DEAD START)	SIZE DRAWING NO C 60119300
		SHEET 35
		63



STUDENT EXERCISE

INTRODUCTION

It is the intention of Control Data Institute to give as much effective training to the trainee as possible in time allotted. The exercise was written to implement the training presentation and to increase the effectiveness of course material. The purpose of the exercise is threefold:

1. To provide a method of self study and self testing for the trainee to find weaknesses and guide his understanding of the presented material.
2. To provide helpful information sheets and drawings of material that is not presented in other reference sources.
3. To provide a good review source for the trainee, both while in the training situation and later in his respective assignment.

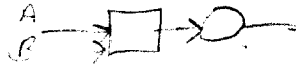
The student is to be cautioned that the exercise is an auxiliary reference source and is to be used for problem solving and is not a primary source of information.

EXERCISES: BUILDING BLOCK

- REFERENCES:
1. Control Data Institute 6600 Training Manual - Appendix B
 2. Printed Circuits Manual, Volume 3

PROBLEMS:

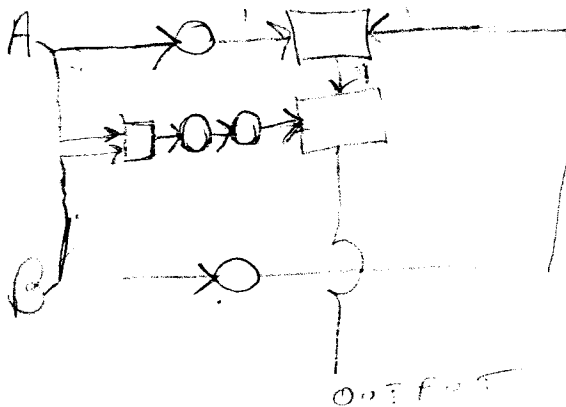
1. The logic levels for the 6000 Logic circuits are:
"1" =
"0" =
2. Draw a logic circuit that will form the "and" of A and B

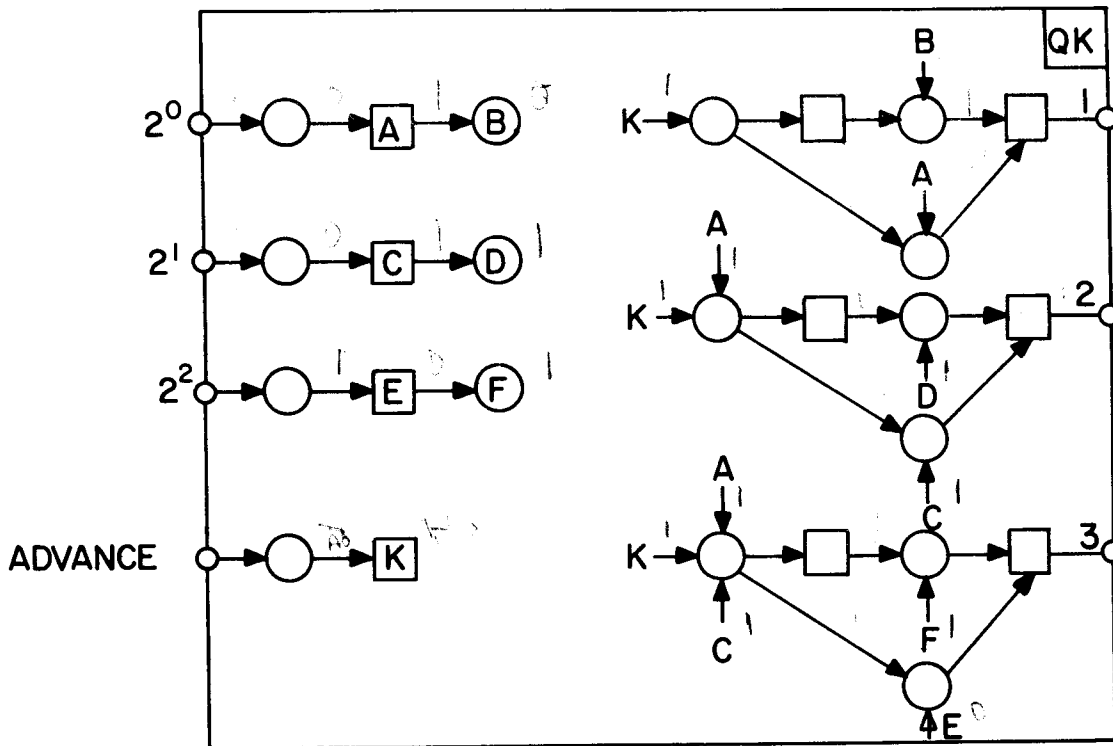


3. Draw a logic circuit that will form the "OR" of A and B



4. Draw a logic circuit that will output a "1" if A and B are equivalent.





a) Translate the previous module for 1's out of pins 1, 2, & 3

$$1 = \overline{\text{ADVANCE}} \cdot 2^0 + \overline{\text{ADVANCE}} \cdot 2^0$$

$$2 = \overline{2^0 + \overline{\text{ADVANCE}} \cdot 2^1} + \overline{2^1 \cdot 2^0 \cdot \text{ADVANCE}}$$

$$3 = \overline{2^2 \cdot 2^1 + \overline{\text{ADVANCE}} + 2^0} \cdot \overline{2^2 \cdot 2^0 \cdot 2^1 \cdot \text{ADVANCE}}$$

b) If $K = 1$ what is the function of the circuit?

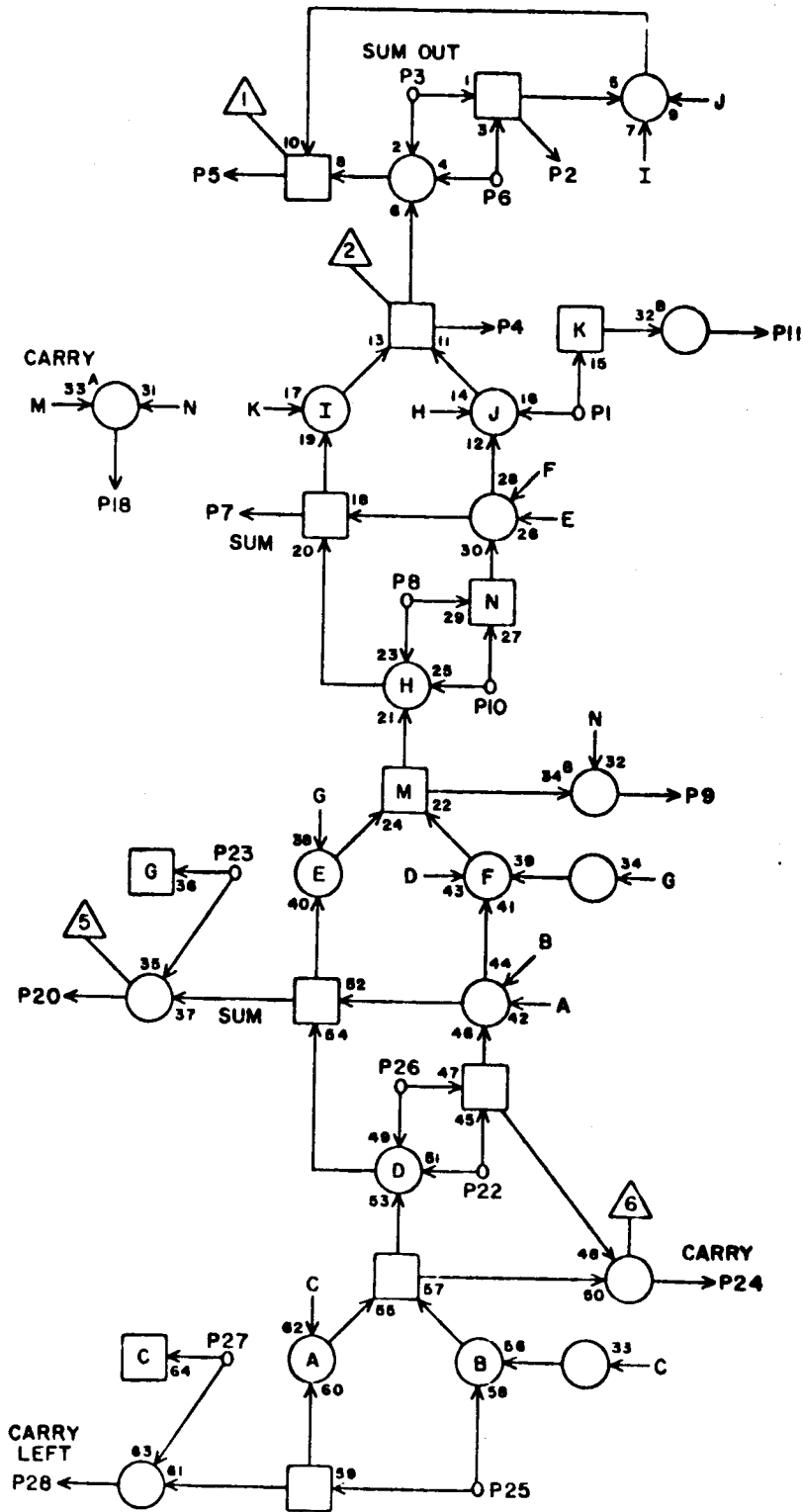
Answer: delay line

c) If $K = 0$ what is the function of the circuit?

d) If inverter D is ban and always outputs a 1 what will the module output if 011_2 is fed into it?

$K = 1$ 110

$K = 0$ 011



1	$B_1 + X + 1$		
2			
3	$\overline{P_2}$		
4			
5			
6	$\overline{C_3}$		
7			
8	$\overline{P_1}$		
9			
10	$\overline{C_2}$		
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22	$\overline{C_1}$		
23	$B_1 + X + 2$		
24			
25	$\overline{P_5}$		
26	$\overline{P_6}$		
27	$B_1 + X$		
28			

ACK PIN LG

CIRCUIT SPECIFICATION 1027000

(1127)

5. Translate the following terms for "1's" on the MC Module with the given inputs:

A =

B =

C =

D =

E =

F =

G =

H =

I =

J =

K =

L =

M =

N =

Pin 28 =

Pin 24 =

Pin 20 =

Pin 9 =

Pin 4 =

Pin 7 =

Pin 11 =

Pin 18 =

Pin 2 =

Pin 5 =

EXERCISE: PPU INSTRUCTIONS

REFERENCES: 6600 Reference Manual

PROBLEMS:

1. The PPU instruction that monitors the progress of the C.P. is?
2. During a 63 central write instruction, which register holds the block length?
3. During a 63 instruction after Trip 3 through the slot
 - a) A is checked for being equal to 1
 - b) The 73 Instruction is at location 0000
 - c) The 73 Instruction location +1 is at 0000
 - d) None of the above
4. During Trip 1 through the slot of A 35 inst.
 - a) The 6 bit address goes to Q
 - b) The 6 bit address goes to P
 - c) The 6 bit address goes to K
 - d) None of the above
5. During a Trip 1 through the slot of A23 inst.
 - a) The logical product of Qfd goes to A
 - b) F goes to K
 - c) Q holds the lower 6 bits of the second operand
 - d) Fd holds the second operand

6. After Trip 3 through the slot on a 02 inst.
 - a) P contains the contents of M
 - ~b) K has been advanced twice
 - c) P contains $Q + 1$
 - d) Q contains $M + (d)$

7. During a 06 instruction, if A is negative
 - a) $P + d$ is transferred to P
 - b) The instruction requires two trips
 - c) $P + 1$ is transferred to P
 - d) The instruction hangs up

8. During a PPU output instruction (73XX) Memory location 0000_8 is read when the K count equals
 - a) 73.5
 - b) 732
 - c) 733
 - d) 734

EXERCISE : Storage Sequence Control & Memory Cycle

- REFERENCES: 1. 6600 Customer Engineering Diagrams Vol. 1
Pages 14-21
2. Chassis 1 Tabs

PROBLEMS:

1. Each PPU Memory is always referenced each minor cycle.
(True or False)
2. During the restoration cycle in the PPU the correct sequence is
 - a) mem → Fd → Y → Z → Mem
 - b) Mem → Y → Fd → Z → Mem
 - c) mem → Fd → Z → Y → Mem
 - d) mem → Fd → Y → Mem
3. What is read by the PPU from its memory when no storage reference is required? 9
4. If storage 9 is just bringing up read drive
 - a) Storage 5 will set write immediately
 - b) Storage 5 will set Z 50 ^{50ns} ~~ns~~ later
 - c) Storage 7 has just enabled sense
 - d) Storage 5 has set Z
5. Processor 6 is gated to rank 6 in the barrel at Time III. What is the relationship of the remaining processors in the barrel?
 - a) 0 _____ d) 2 _____ g) 2 _____ j) 2 _____
 - b) 1 _____ e) _____ h) _____
 - c) 2 _____ f) 2 _____ i) 2 _____

Which ranks are duplicated?

6. A PPU executes the following program

0010 = 5012

0011 = 3003

0012 = 0302

0013 = 1407

0014 = 1406

0015 = 0300

TP2 on I20 is a constant 1 output

a) What will A equal when the program hangs up? 000

b) What will P = when the program hangs up? 15

7. Is there any time that an instruction that will not effect a P - G or a Q - G transfer? If there is, which one (s) and why?

Q → R

00

03

↓

07

10

↓

17

20

↓

23

26

↓

30

33

↓

37

40

only see storage P → Q
Q → R for loading P → A
P → Q for transfer

EXERCISE : A, P, Q & K in the Barrel

REFERENCES: 1. 6600 Customer Engineering Diagrams

PROG L

PROBLEMS: 1. What is the function of the Set K = 340 gate in the PPU?

*Read R A → Z
Store A into R
A → Z*

2. What is the function of the F → K Transfer and at what time does it occur?

*to put contents of R into K register
on instruction R → K (NIB F → K)*

3. At the end of an RNI sequence (K = 00X) The contents of K will always contain the F portion of the next instruction.
(True or False)

EXERCISE : A & Q ADDERS

REFERENCES: 6600 Customer Engineering Diagrams

PROBLEMS : 1. A one output from inverter E of L10 on the ^QA Adder indicates what condition of the adder?

Zero or carry

2. During the execution of a LDC (20) instruction, the 18 bit final contents of A are formed, using which of the following gating conditions?

Answer

a) $+Q \rightarrow Bu$, $Fd \rightarrow A$, $00 \rightarrow Bm$ and finally A adder to A in the barrel.

Answer

b) $+Q \rightarrow Bu$, $+F \rightarrow B$, $+d \rightarrow B$ and finally A adder to A in the barrel.

c) $+Q \rightarrow Bu$, $-F \rightarrow B$, $-d \rightarrow B$ and finally A adder to A in the barrel.

d) $+Q \rightarrow Bu$, $A \rightarrow A$, $0 \rightarrow Bm$ and finally A adder to A in the barrel

EXERCISE : SHIFT NETWORK

REFERENCES: 6600 Customer Engineering Diagrams

PROBLEMS :

1. A = 000007. After executing a 1015 shift instruction
A = 070000. A possible cause of this trouble is

a) I6 inverter B is a constant 1

b) I7 -TP3 is a constant 1

c) I6 inverter C is a constant 0

d) I7 - TP1 is a constant 0

EXERCISE : READ/WRITE PYRAMIDS

REFERENCES: 6600 Customer Engineering Diagrams

- PROBLEMS:
1. What is the function of pin 20 on module L29 in the central read control
 - a) Advances K from G12 → G13
 - b) Controls the number of words in a block read inst.
 - c) Advance the address of the next word from CM
 - d. Advance the address of the PPU memory
 - e) Blocks a 1 input to Q, to get Q = 0 needed to store the program count in memory location 0000.
 2. How many processors may share the read pyramid from central memory at the same time.

4

EXERCISE : DATA CHANNEL

REFERENCES: 6600 Customer Engineering Diagrams

PROBLEMS:

1. When is the first empty signal sent from the data channel to the I/O Device on a read operation.

just Transfer from 7200 71 A. 70X

EXERCISE : DEAD START

REFERENCES: 6600 Reference Manual
Chapter 6

- PROBLEMS:
1. During a head start "dump" operation, which of the following is always true
 - a) The contents of all the processors are dumped on the disc
 - b) Processor 0 dumps its core contents onto the I/O device on Channel 0
 - c) The dead start panel holds a constant empty on channel 0 until its contents are received by the I/O device.
 - d) The program in Processor 0 is executed beginning at address 0001

Figure 1

0	0000
1	1410
2	7307
3	0005
4	7507
5	0000
6	7707
7	3060
10	7707
11	3020
12	7407
13	7107
14	0000

2. If the program in Figure 1 is in the dead start panel, the maximum record length that can be read from tape 0 into processor 7 is
 - a) 10000 words
 - b) 7770 words
 - c) 7776 words
 - d) 7777 words

3. In Figure 1, address 5 is 0000 because
- a) A pass instruction is required after a disconnect
 - b) Processor 0 must halt after transferring data to Processor 7
 - c) Because the output to Processor 7 must be stored in address 0
 - d) Processor 7 must know where to begin executing its memory contents after Processor 0 disconnects it

COMMENT SHEET

6000 ~~LINE PRINTER EQUIPMENT TRAINING MANUAL~~

Publication Number 011568

FROM: Name: _____

Address: _____

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

CONTROL DATA INSTITUTES

3255 Hennepin Avenue So.
MINNEAPOLIS, MINNESOTA
55408

5630 Arbor Vitae Street
LOS ANGELES, CALIFORNIA
90045

3717 Columbia Pike
ARLINGTON, VIRGINIA
22204

CONTROL DATA
COMPUTER TRAINING SCHOOL
66 West 12th Street
NEW YORK, NEW YORK
10011

60 Hickory Drive
Bear Hill Industrial Park
WALTHAM, MASSACHUSETTS
02154

Exchange Park Garden Mall
DALLAS, TEXAS
75235

23775 Northwestern Highway
SOUTHFIELD, MICHIGAN
48075

Bockenheimer Landstr. 10
6000 FRANKFURT /M.
GERMAN FEDERAL REPUBLIC

CONTROL DATA

C O R P O R A T I O N