UNCLASSIFIED

PRELIMINARY MANUAL

THEORY OF OPERATION

AN/FSQ-7(XD-1, XD-2)
COMBAT DIRECTION CENTRAL

CENTRAL COMPUTER SYSTEM

September 1955

INTERNATIONAL BUSINESS MACHINES CORPORATION

POUGHKEEPSIE, NEW YORK

[redacted]

LIST OF REVISED PAGES ISSUED

INSERT LATEST REVISED PAGES, DESTROY SUPERSEDED PAGES

NOTE: The portion of the text affected by the current revision is indicated by a vertical line in the outer margins of the page.

* The asterisk indicates pages revised, added or deleted by the current revision.

ADDITIONAL COPIES OF THIS PUBLICATION MAY BE OBTAINED AS FOLLOWS: USAF

USAF ACTIVITIES.—In accordance with Technical Order No. 00-5-2.
NAVY ACTIVITIES.—Submit request to nearest supply point listed below, using form NavAer-140: NASD, Philadelphia, Pa.; NAS, Alameda, Calif.; NAS, Jacksonville, Fla.; NAS, Norfolk, Va.; NAS, San Diego, Calif.; NAS, Seattle, Wash.; ASD, NSC, Guam.
For listing of available material and details of distribution see Naval Aeronautics Publications Index NavAer 00–500.

CONTENTS

CONTENTS (cont'd)

CONTENTS (cont'd)

CONTENTS (cont'd)

CONTENTS (cont'd)

CONTENTS (cont'd)

CONTENTS (cont'd)

CONTENTS (cont'd)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (cont'd)

LIST OF ILLUSTRATIONS (cont'd)

LIST OF ILLUSTRATIONS (cont'd)

LIST OF ILLUSTRATIONS (cont'd)

LIST OF ILLUSTRATIONS (cont'd)

LIST OF TABLES

LIST OF TABLES (cont'd)

PH 22-0000

THEORY OF OPERATION

AN/FSQ-7(XD-1, XD-2)
COMBAT DIRECTION CENTRAL

CENTRAL COMPUTER SYSTEM

PRELIMINARY MANUAL

IBM

PART 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 GENERAL

AN/FSQ–7 (XD–1,–2) Combat Direction Centrals are essentially complete data-processing machines designed to handle large amounts of military tactical data associated with a number of aircraft detection and search devices. Because of operational considerations, AN/FSQ–7 (XD–1,–2) Combat Direction Centrals are divided into six separate systems, each of which has its own special function to perform. These systems are shown in figure 1–1 and are listed below:

a. Input System
b. Drum System
c. Central Computer System
d. Output System
e. Display System
f. Power Supply and Marginal Checking System

The function of the Input System is to control the flow of information to the Drum System from radar sites and other AN/FSQ–7 Combat Direction Centrals. The Drum System is a medium-speed, medium-capacity storage device. It stores slowly recurring intermittent data from the Input System and transfers these data to the Central Computer System at a rate more compatible with the extremely swift operating speed of the Central Computer System. It is the responsibility of the Central Computer System to correlate and process these data and to store the results back in the Drum System. In addition, the Drum System provides storage space, in the form of the auxiliary memory fields, to supplement the storage capacity integrated within the Central Computer System. Data which have been processed by the Central Computer System and stored on the drums are further distributed to the Output and Display Systems. The Output System controls the flow of these processed data to the radar and weapons sites, while the Display System presents these data in visual form to human observers for interpretation and control. The Power Supply and Marginal Checking System provides the a-c and d-c voltages needed for electronic operation and also provides the voltages needed for the marginal checking of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals.

As shown in figure 1–1, the Central Computer System occupies a central position relative to the flow of data within AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. Thus, the Central Computer System receives tactical information from the Input System via the Drum System, processes it, and transmits the processed data to the Output and Display Systems via the Drum System. In addition to the transfers between the Central Computer System and the Drum System, information may also be transferred between the Central Computer System and various input-output (IO) units. The IO units consist of auxiliary storage devices which are capable of storing data received from, or transferring data into the Central Computer System.

1.2 FUNCTION

The Central Computer System operates internally in the binary number system and is controlled by means of a stored program. The Central Computer System incorporates an internal memory device capable of storing 8192 33-bit binary numbers and providing rapid access to any one of these numbers. This memory device is termed

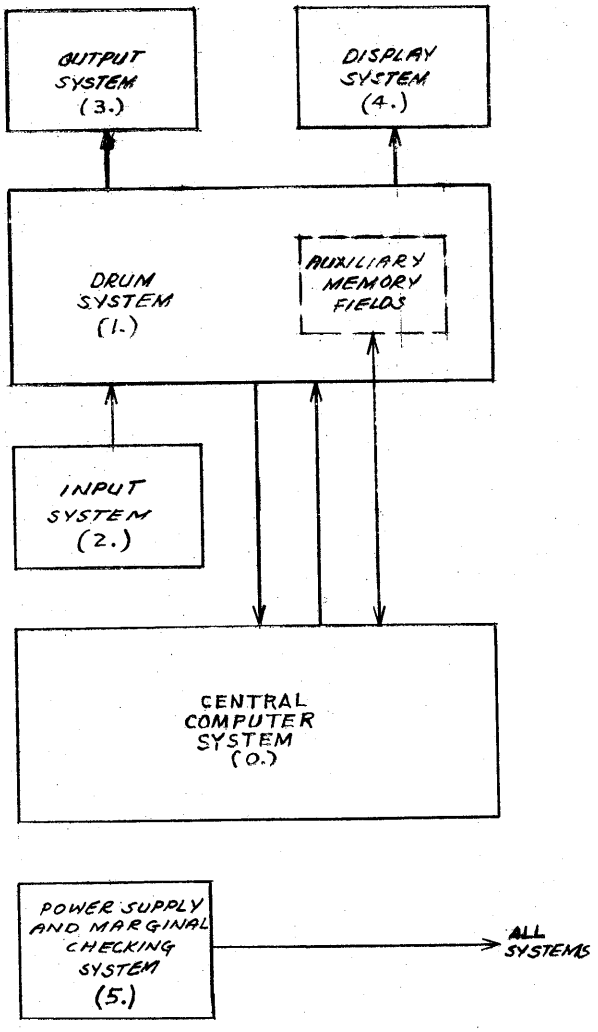


Figure 1–1. AN/FSQ–7 (XD–1, –2) Combat Direction Central, Simplified Block Diagram

magnetic core memory, or more briefly, core memory, and stores all the information which the Central Computer System must use during the execution of its computations and operations.

Information stored in the core memory of the Central Computer System is of two types, instructions and numerical data. An instruction is an order to the Central Computer System which causes it to execute a specific operation such as addition, subtraction, multiplication, etc. The numerical data are the binary numbers representing tactical information and are stored in core memory from the Drum System. Before any processing of this numerical data may be accomplished the Central Computer System must be supplied with a series of instructions describing the exact processing procedure that is to take place. Such a series of instructions is termed a program. It is the function of the Central Computer System to subject the tactical numerical data to processing operations in accordance with the series of instructions, or program, which is stored in core memory. The results of these processing or computational operations are stored back in core memory after the operations are completed and are ultimately transferred to the Drum, Output, and Display Systems, as determined by the program.

1.3 DESCRIPTION

The Central Computer System is functionally divided into six sections or logical elements. Five of these are used for control and calculation, while the sixth is utilized for Central Computer System

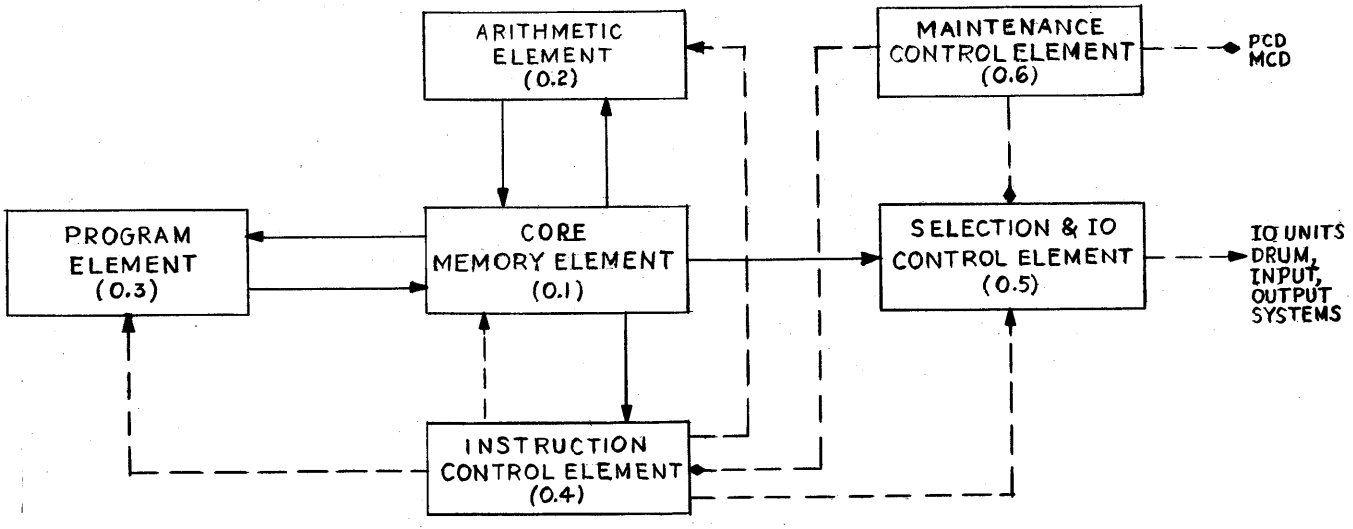


Figure 1–2. Central Computer System, Simplified Block Diagram

maintenance and testing. These six elements are:

a. Arithmetic element
b. Instruction control element
c. Maintenance control element
d. Memory element
e. Program element
f. Selection and IO control element

These elements of the Central Computer System, together with their principal interconnecting control and information paths, are illustrated in figure 1–2.

The instruction control, program, and selection and IO elements act in conjunction with each other to control and co-ordinate the operation of the Central Computer System and associated systems in the execution of the program. This function of co-ordination, or control, may be differentiated into internal control and external control. Internal control refers to the control of operations and calculations taking place within the Central Computer Systems, while external control refers to the control of operations in other associated systems. External control deals principally with the co-ordination of the Central Computer System and the Drum System and IO units so as to effect the transfer of information between them.

The arithmetic element provides those circuits necessary for the execution of the arithmetic computations of the Central Computer System program, while the memory element stores the data involved in these computations and also the program itself. The flow of information associated with these calculations is quite straightforward, in that numerical data are transferred out of the memory element to the arithmetic element, where they are subjected to operations of a strictly mathematical nature. After the computations are complete, the resultant numerical data may be transferred from the arithmetic element back into the memory element, where it is stored for future reference. Thus, it may be seen that the computational operation of the Central Computer System is one of continuously circulating data between the memory and arithmetic elements.

The flow of numerical information and the mathematical operations to which this information is subjected are sequenced and controlled by the instruction control and program elements. These elements provide the circuits necessary for internal control of the Central Computer System. During the exectuion of the Central Computer System program, the instructions of which it is composed are transferred one at a time to the instruction control and program elements. The former element causes the Central Computer System to execute the operations dictated by each instruction as it is received; the latter controls the memory element so as to obtain any additional information required by an instruction, and also sequences the transfer of each consecutive instruction out of core memory.

If information is to be transferred between the Central Computer System and the other systems of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals, the transfer is directed by the program. It is the function of the selection and IO control and program elements to co-ordinate and effect this informational transfer; i.e., to perform the external control function. It should be noted that the program element contains circuits associated with both the internal control functions of the instruction control element and the external control functions of the selection and IO control element. The process of transferring information into or out of the memory element from or to input-output devices is called an input-output (IO) operation. This operation is initiated and co-ordinated, under the control of the Central Computer System program, by the selection and IO control element. The program element contains those circuits which are utilized to provide transfer paths into or out of the Central Computer System, and it also dictates the originating point or designation in the memory element of the information involved in the IO operation. The maintenance control element provides controls for use by Central Computer System personnel to start, shut down, and service the Central Computer System.

CHAPTER 2

BASIC CHARACTERISTICS

2.1 DIGITAL WORDS

All information held in the Central Computer System for control and computation is in binary coded form. This information is handled in groups of 16 or 32 binary digits (bits), such groups being termed binary words. Binary coded numbers as well as instructions and other non-numerical information are usually termed words. The number of bits composing a word is called the word length. The unit of information in the Central Computer System is a 32-bit word composed of two half-words having 16 bits each. These two half-words are distinguished from each other by the terms, right half-word (RHW) and left half-word (LHW). The symbolic layout of a full 32-bit Central Computer System word is shown in table 1–1.

It can be seen that a 16-bit half-word consists of 15 bits numbered from 1 through 15 plus an S bit which is called the sign bit of the word. The S bit has significance only if the half-word contains numerical information. By definition, if the S bit is 0, the number contained in bits 1 through 15 is positive and in true binary form; if the S bit is 1, the number is negative and is in 1's complement form. When referring to a single bit in a 32-bit word, a special notation is employed. For instance, L8 refers to the 8 bit of the left half-word, and R14 refers to the 14 bit of the right half-word.

2.1.1 Instruction Words

An instruction word is a full 32-bit word containing a binary coded command to the Central Computer System which, when decoded and acted upon, will cause the Central Computer System to execute a designated operation. Because of their special importance, the left and right half-words of a full instruction word have been assigned distinctive names. The left half-word is termed the operation portion and the right half-word is termed the address portion. These names are derived from the names of the registers in the Central Computer System in which the left and right half-words are stored while the instruction is being executed.

TABLE 1–1. COMPOSITION OF A WORD

LEFT HALF-WORD																RIGHT HALF-WORD															
S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The operation portion, or left-hand portion of the instruction word, contains a binary coded instruction or command to the Central Computer System describing the operation it is to perform. The S bit is meaningless in the operation portion and bits 1 through 15 may be broken down into groups having their own operational significance. The manner in which the operation portion bits are grouped and the names assigned to these groups is shown in table 1–2.

Bits L1 through L3 are termed the index indicator since they are used to specify one of three index registers. The utilization of an index register, under the control of bits L1 through L3, is called indexing, and provides a means of altering or cycling the Central Computer System program. Indexing is to be discussed in 3.5 of Chapter 3. There are 48 instructions associated with

TABLE 1–2. BIT DESCRIPTION OF LEFT HALF-WORD

Bit	LS	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15
Desig-nation		Index Indicator				Class	Operation Code				Variation		Index Interval			

AN/FSQ-7 (XD-1,-2) Combat Direction Centrals. These 48 instructions are grouped into eight classes. Bits L4 through L6 indicate the class within which a specified instruction is contained. Within a given class, there may be as many as nine distinct instructions. Bits L7 through L10 are used to specify the particular instruction, called the variation, involved in the class determined by bits L4 through L6. The combination of bits L4 through L10 is termed the operation code of an instruction and completely specifies the instruction by the class and variation. The separate instructions within a class are determined by the variations of the class; the combination of class and variation bits serves to completely identify an instruction. Bits L10 through L15 are termed the index interval and are used to provide additional information required by particular instructions. The representation of the index interval varies with the instruction specified by bits L4 through L10. These bits are to be discussed in 3.2, 3.3, and 3.5 of Chapter 3. It should be noted that bit L10 is utilized in both the operation code and the index interval. However, this causes no ambiguity since, by design, the instructions which use the index interval do not require bit L10 for their identification. Thus, when the index interval is utilized, the appropriate instruction is completely specified by only six bits of the operation code instead of the usual seven.

The address portion, or right-hand portion of an instruction word, denotes the location in the memory element or other storage elements of the Central Computer System from which additional information pertinent to the instruction specified by the operation portion may be obtained. This additional information usually takes the form of operands required to execute an instruction specifying a mathematical operation. For example, if the machine is to add, an addend must be obtained. It is the purpose of the address portion to specify the location in the memory element where these operands may be found. There are 8192 core memory locations in the memory element. The memory element consists of two units, each capable of storing 4096 full words. The location of any given word in a core memory unit is designated by its address, which may have a value between decimal 0 and 4095. Thus, the location of a single word in the core memory element may be specified by a binary code designating the core memory unit (No. 1 or No. 2) in which the word is located and the address within that core memory unit where it is stored. Additional words may be stored in the Drum System. The Drum System of AN/FSQ-7 (XD-1,-2) Combat Direction Centrals is divided into 39 sections, called fields. Each field provides 2048 storage locations, any one of which may be specified by a number from decimal 0 through 2047. It is the purpose of the address portion of the instruction word to designate the location of a word in the Central Computer System memory element or on a particular field of the Drum System. In the case where Drum System words are designated, the index interval bits of the operation portion specify on which field of the drums the word may be found and the address portion specifies the exact location of the word on the field. The layout of the address portion bits are shown in table 1-3.

Since the address portion does not contain information of a numerical nature, the S bit is meaningless. Bits R1 through R3 are called the memory unit selector, since they specify the core memory unit in which the desired word may be found. Bits R4 through R15 designate the address, since they specify the exact location in the core memory unit where the desired word is stored. In addition to the Central Computer System core memory units, which are usually designated core memory 1 and 2, there are two other storage units, test memory and the clock register. These are to be discussed in 2.4 of this Chapter. The binary codes of the core memory units designated by the memory unit selector bits and the bits used to specify address locations are shown in table 1-4.

Those bits indicated by dashes in the table are meaningless and may have any value. Since test memory has only 16 memory locations, as shown in 2.4 of this Chapter, the four bits R12 through R15 are sufficient to completely specify the location of a word stored therein. The address

TABLE 1-3. BIT DESIGNATION OF RIGHT HALF-WORD

Bit	RS	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Designation			Memory Unit Selector		Address											

bits are meaningless when the clock register is specified by bits R1 through R3, since this unit possesses only one memory location. Although this discussion is based on the premise that the address register specifies a core memory location where operands may be obtained for use in the Central Computer System proper, it applies equally to the process of storing words or the results of calculations in the core memory location specified by the address portion of the instruction. Thus, the address portion only possesses the property of specifying a particular core memory location. Whether a word will be obtained from this core memory location or stored therein depends upon the operation contained in bits L4 through L10, associated with the address portion of the word.

It may now be seen that an instruction word is a binary coded command to the Central Computer System, describing an operation which is to be performed. If the instruction requires an operand or other additional information, the core memory location where this information may be obtained is specified by the address portion of the instruction word. If the instruction requires that information be stored in the memory element, the address portion of the instruction specifies the core memory location in which the information is to be stored. In the case of instructions which do not require any reference to a core memory unit, the contents of the address portion are usually meaningless. An example of this is the instruction causing the Central Computer System to stop. It is apparent that there would be no new information necessary and none to be stored; the numerical value of the address portion would be meaningless, whatever its contents.

2.1.2 Numerical Words

A numerical word as used in the Central Computer System consists of 16 bits, or one half-word. Thus, a full 32-bit word contains two binary coded numbers, one in the left half-word and one in the right half-word, each having a word length of 16 bits. The 1's complement system of indicating negative numbers is used in the Central Computer System. If the S bit is 1, the number is negative, while if the S bit is 0, the number is positive. For instance, +5/8 would appear in the Central Computer System as the binary number 0.10100 . . . 00 while −5/8 would appear as 1.01011 . . . 11 where the first bit is the S bit and is followed by 15 bits representing 5/8 or its 1's complement.

Since many of the operations involved in the air defense program used in AN/FSQ–7 (XD–1,–2) Combat Direction Centrals involve identical calculations with two quantities, usually X and Y components of position or velocity or radial and angular (R and ϕ) components of position or velocity, the two half-words composing a full 32-bit word are used to represent these two related quantities. The arithmetic element, discussed in 2.4 of this Chapter, is designed to simultaneously process the right- and left-half portions of a full numerical word in an identical manner. Except for a few instructions, the dual sections of the arithmetic element are controlled together and must perform the same operations on both 16-bit words. Thus, the transfer of a numerical word from core memory to the arithmetic element usually involves the transfer of two related quantities, one contained in the left half-word and the other in the right half-word.

All numerical words in the Central Computer System have an absolute magnitude less than unity; i.e., a number X may be anywhere in the range $+1 < X < -1$. Therefore, the binary point (corresponding to the decimal point in decimal numbers) is located between the S bit and the 1 bit. Since the mathematical capacity of the Central Computer System must never be exceeded, it is necessary that all numerical data fed to the Central Computer System be first scaled down to the range where its absolute magnitude is less than unity. Thus, if the multiplication 256 by 33 were to be performed, the two numbers would be fed to the Central Computer System as 0.256 and 0.033 (numbers scaled down by factors of 1/1000). The result of the multiplication by the Central Computer System will be 0.008448, which will be interpreted by the equipment using the results of the calculation as differing from the true product by a factor of 1/1,000,000. Since the Central Computer System is not equipped to handle numbers equal to or greater than unity in absolute magnitude, it contains alarm circuits to indicate if a calculation has resulted in a number beyond its capacity. A

TABLE 1–4. CORE MEMORY DESIGNATION CODES

MEMORY UNIT	MEMORY UNIT SELECTOR CODE	PERTINENT ADDRESS BITS
Core memory No. 1	–00	R4 through R15
Core memory No. 2	–01	R4 through R15
Test memory	01–	R12 through R15
Clock register	11–	Meaningless

number exceeding the capacity of the Central Computer System might appear during an addition or division, but proper programming should eliminate this possibility. This situation is discussed further in 3.3 of Chapter 3.

2.2 MEMORY ELEMENT

The Central Computer System memory element is a high-speed storage device capable of storing 8192 binary words having a word length of 33 bits. Thirty-two of these bits form the standard left and right half-words; the 33rd bit, located to the left of and immediately adjacent to the S bit of the left half-word, is called the parity bit. The parity bit is used to detect errors in the transfer of words into or out of the memory element. The memory element is functionally divided into two identical units, each capable of storing 4096 words; these two units are termed core memory No. 1 and core memory No. 2. The complete memory element is often referred to as magnetic core memory. This name is derived from the fact that core memory stores binary data in the magnetic flux of a permeable toroidal core, which is capable of being magnetized in either a clockwise or counterclockwise direction, corresponding to the storage of a 1 or a 0, respectively.

2.2.1 Magnetic Cores

The idealized hysteresis loop of a typical magnetic core as used in the memory element is shown in figure 1–3. Note that the loop has the shape of a rhombus and possesses a high degree of rectangularity in comparison with the hysteresis loops of other types of magnetic materials. It is upon this property that the storage capabilities of the toroidal core are based. When the flux within the core is in one direction (clockwise, for example) and has a density indicated by point A in figure 1–3, the core is said to contain a binary 1. On the other hand, if the core is magnetized in a counterclockwise direction and the magnetic flux has a density indicated by point E in the figure, the core is said to contain a binary 0. It is apparent that the status of the core (1 or 0) may be changed at any time by applying a magnetomotive force (MMF) of magnitude I in the proper direction. Thus, a MMF of —I will cause a core containing a 1 to change its magnetic state by following the path ABCDE, so that it may be switched from the 1 to the 0 status. Similarly, a core containing a 0 may be switched to the 1 status by applying to it a MMF of +I, causing the flux to follow the path EFGHA.

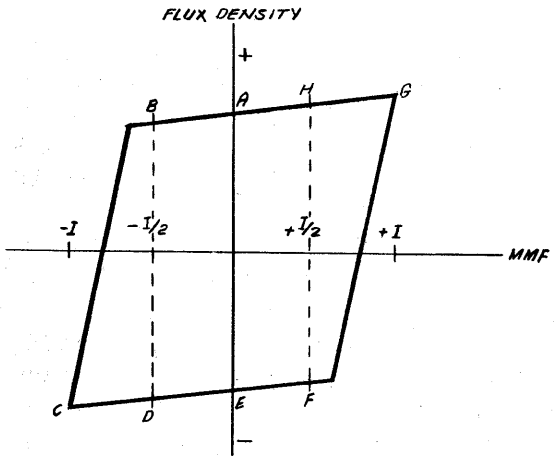


Figure 1–3. Toroidal Core Hysteresis Loop

2.2.2 Word Storage

The method by which the flux state of a core is shifted is shown in figure 1–4. A total of four wires or windings are passed through each core;

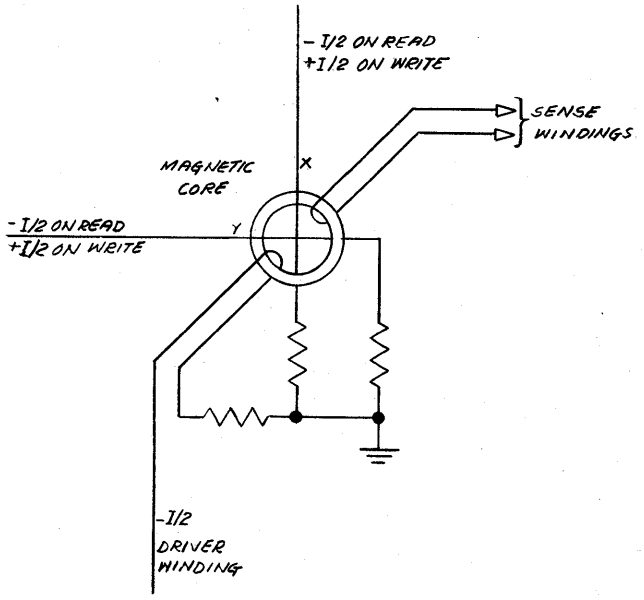


Figure 1–4. Read-Write Core Windings

an X winding, Y winding, sense winding, and digit-plane winding. These four windings allow binary information to be read out of or written into the core by shifting its magnetic state. The status of a core is changed when a full-amplitude current pulse is applied to it. However, the X and Y windings carry only half-amplitude current pulses. When both X and Y half-amplitude pulses pass through a core simultaneously, a full-amplitude pulse is applied which shifts the magnetic state of the core. Therefore, reading information out of a core consists of applying a pulse of —I/2

amplitude to both the X and Y windings passing through the core. A resulting MMF of $-I/2 - I/2 = -I$ which is a full-amplitude pulse, is thus applied to the core. If the core contains a 1, represented by point A in figure 1–3, the magnetic state is changed to 0, represented by point E in figure 1–3, by following the path ABCDE. If the core contains a 0, no change in the magnetic state takes place. The shift of magnetic flux which takes place when a core changes status from 1 to 0 induces a voltage in the sense winding. This induced voltage indicates that the core stored a 1 previous to the read-out of information. If no voltage is induced in the sense winding, the indication is that the core stored a 0 previous to the read-out.

The process of writing into the core (storage of information) follows the same pattern as the read-out, except for an additional step. Half-amplitude $+I/2$ pulses on both the X and Y windings cause a shift in the core status, from point E to point A in figure 1–3, when a 1 is to be written into the core. However, when a 0 is to be written into the core, the magnetic state of the core must remain at point E in figure 1–3; i.e., no status change must take place. The driver winding shown in figure 1–4 applies a half-amplitude $-I/2$ pulse to the core simultaneously with the application of half-amplitude pulses from the X and Y windings. The total current applied is $+I/2 + I/2 - I/2 = +I/2$, which is only a half-amplitude pulse and is not sufficient to cause a change in the magnetic state of the core. Therefore, the sole function of the driver winding is to inhibit the writing process when a 0 is to be stored in the core.

2.2.3 Core Memory Array

The cores and associated windings are arranged into a plane of 64 rows and 64 columns providing a total of 4096 cores. Thirty-three planes and two spares are stacked vertically to form an array. Two such assemblies make up the complete core memory array. Each plane stores one digit (bit) of the word in core memory, so that the 33 planes of the array store a full word plus the parity bit. Since there are two arrays comprised of 4096 groups of 33 cores or core registers for a total storage capacity of 8192 words, the memory element must be provided with some means of selecting the specific core register involved in the read or write process. The manner in which the specific core register is activated is illustrated in figure 1–5, which shows a small portion of the array and the planes. Since the planes are stacked vertically, 32 additional cores should be visualized below each core shown in figure 1–5 to complete a picture of the array. Selection of one of the core memory locations involves the simultaneous selection of one of the 64 rows and one of the 64 columns of a digit plane. Corresponding rows and columns of all 33 digit planes are selected simultaneously. A current of $\pm I/2$ ($-I/2$ for reading and $+I/2$ for writing) is passed through the selected lines. Suppose that these lines are the b line in the X plane and the e line in the Y plane. (See fig. 1–5.) All cores in the 33 digit planes through which the e line passes will have a half-amplitude pulse applied to them. Similarly, all cores in the 33 digit planes through which the b line passes will also have a half-amplitude pulse applied to them. All those cores which are at the junction or intersection of the b and e lines will have a full-amplitude pulse applied to them so that the read or write process may take place for only these cores. All cores which receive a half-amplitude pulse from only one line, either the b or the e line, will not switch their status. The remainder of the cores which have neither an e nor b line passing through them remain unaffected. All cores which are read from or written into are called fully selected cores, while those cores receiving a half-amplitude pulse are termed half-selected cores. When the fully selected cores switch status during the read process, the information contained

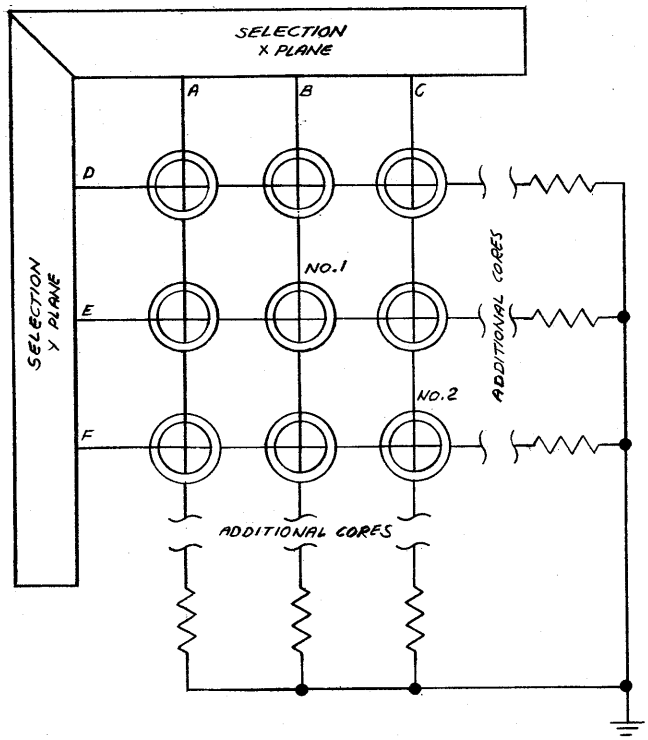


Figure 1–5. Core Selection

in the cores is fed out of core memory via the sense lines to the Central Computer System or to the designated IO devices. In the same manner, information is written into the cores during the write process, except for the activation of the drivers to inhibit a flux change when storing a 0 in the cores.

It is apparent that two cycles are executed when information is transferred out of and into core memory. Core memory first executes a read cycle, during which information is fed out of core memory, and then executes a write cycle for storing information in core memory. The combination of these two cycles forms what is termed a memory cycle. Once core memory is activated it will automatically go through one memory cycle; i.e., read and write, in that order.

2.2.4 Block Diagram

Since core memory No. 1 and No. 2 are identical, only one is shown in figure 1–6. There are only five inputs to the core memory element from the other elements of the Central Computer System. These inputs and the functions they perform are listed in table 1–5.

The core memory element is caused to read or write a word by means of the inhibit sample pulse. If no inhibit sample pulse is received, core memory reads a word out of the selected core register into the memory buffer registers. If the inhibit sample pulse is received, the read-out to the Central Computer System is suppressed and a write cycle is executed, so that the word held in the memory buffer registers is transferred to core memory and stored in the selected core register. The start memory pulse activates core memory. Both the inhibit sample and the start memory pulses are fed to the timing and control section of core memory. (See fig. 1–6.) This section consists principally of a number of delay units which time and distribute the start memory and inhibit sample pulses to the other inputs of core memory. The selection section uses the bits of the address portion (R4 through R15 of the right half-word) of the word to select the proper core register for reading or writing. The output pulse voltage generated in the array during the read process is fed to the sense section and then to the memory buffer registers for distribution and use in either the Central Computer System or one of the various IO devices.

All transfers of information between the Central Computer System (as well as the other systems) and the core memory element take place

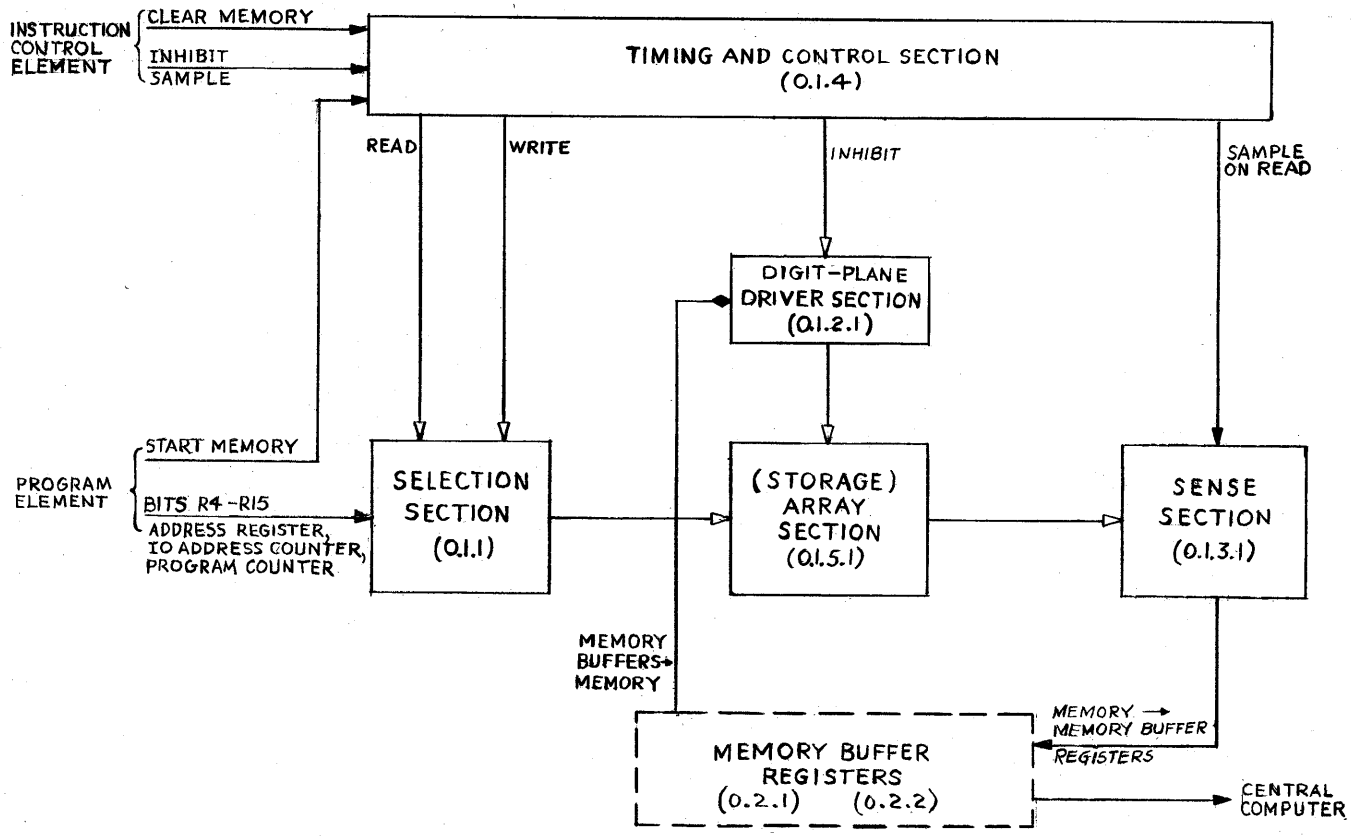


Figure 1–6. Core Memory Element, Simplified Block Diagram

TABLE 1-5. INPUTS TO CORE MEMORY NO. 1 OR NO. 2

INPUT LINE	FUNCTION
Program element to memory address register	Twelve input lines placing 12 binary bits into memory address register to specify the core memory location of a word. The memory address register is in the selection section of the core memory element and its contents are ultimatly decoded to activate the proper core register.
Clear memory	Clears control circuits and registers in core memory in preparation for a new cycle.
Start memory	Causes core memory to automatically sequence through read and write cycles for reading from or writing into a selected core register.
Inhibit sample	Prevents generation of the sample on read pulse to the sense section, which prevents the reading of a word into the memory buffer registers when a write operation is to be performed.
Memory buffer register flip-flop 0 sides	Thirty-three input lines from the 0 sides of the memory buffer register flip-flops. These lines condition digit-plane driver circuits in the digit-plane driver section. This prevents the writing of a 1 into a core in core memory when the corresponding flip-flop in the memory buffer register contains a 0.

through the memory buffer registers. These registers are located in the arithmetic element. However, their operation is associated with core memory. During the read process, the memory buffer registers reecive the information read out of core memory and store it temporarily for distribution to the proper points in the Central Computer System or the other IO devices. When information is to be stored in the core memory element it must first be placed in the memory buffer registers; these serve to write the information into the proper core memory location.

One feature of the use of a magnetic core for storage is that read-out of information in a core is destructive. This was indicated by the fact that when a core was read, it was changed from the 1 to the 0 state. Thus, when a 33-bit core register is read, the information previously contained in the cores is destroyed, and all the cores of the register then contain a 0. However, the original information contained in the core register is then held in the memory buffer registers. If this information is now rewritten into the cores, the overall process becomes nondestructive. This is accomplished by applying current pulses of $+I/2$ to both the X and Y windings and simultaneously driving the digit-plane driver section with an inhibit pulse. The full-amplitude $+I$ pulse applied to the core switches it from the 0 it contains to the 1 status. However, if the memory buffer register flip-flop contains a 0, the digit-plane driver circuit will be conditioned to inhibit the write process in the correspondng core in core memory. This action permits the core to store a 0 by remaining in the 0 state. The full 33-bit word is thus rewritten into the core register from which the previous information was read.

2.3 CONTROL ELEMENTS

There are three control elements in the Central Computer System: the instruction control, program, and selection and IO control elements. The purpose of these three elements is to sequence, co-ordinate, and generally control all the operations in or associated with the Central Computer System. The instruction control element provides command pulses to the Central Computer System in the proper sequence to execute the programmed instructions stored in core memory. The selection and IO control element co-ordinates the operation of the Central Computer System with the other systems of AN/FSQ-7 (XD-1,-2) Combat Direction Centrals during the transfer of words into or out of the Central Computer System. The program element contains those circuits used to sequence the procuring of each instruction or operand required by each instruction. The program element also contains those circuits which direct and sequence the transfer of information into or out of the Central Computer System under the control of the selection and IO control element.

An instruction word contains binary coded information describing the operation which the Central Computer System is to perform, as well as the core memory location where additional information pertinent to the instruction may be obtained. (Refer to 2.1 of this Chapter.) In the

execution of the Central Computer System program, an instruction is read out of core memory into the memory buffer registers and distributed to the three control elements. These control elements decode and distribute the various portions of the instruction word and cause the Central Computer System to execute the operations required. Figure 1–7 shows the distribution of the various portions of the instruction word to the control elements. The instruction word, as received from core memory by the memory buffer registers, consists of a full 32-bit word plus a parity bit. The parity bit is only used to detect an error in the word transferred to or from the core memory registers and is not transferred to any other unit in the Central Computer System. Since bit LS is meaningless to an instruction word, it is not transferred to any other section of the Central Computer System.

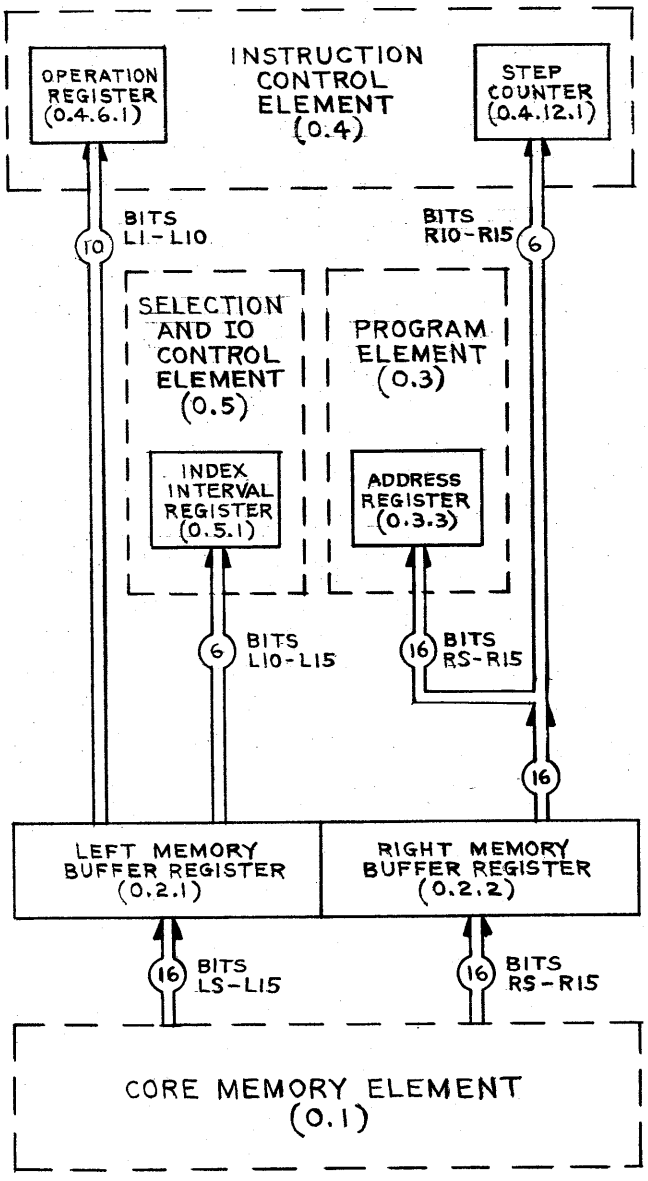


Figure 1–7. Distribution of the Instruction Word Bits

Bits L1 through L10 which designate the index indicator and operation code (2.1 of this Chapter), are transferred to the instruction control element. Here they are decoded and command pulses are generated, causing the Central Computer System to execute the operation specified by the instruction. In addition, the instruction control element also receives bits R10 through R15 from the memory buffer registers. These bits determine how long the Central Computer System is to pause in its prescribed operation while certain special functions are performed. The index interval bits, L10 through L15, are transferred from the memory buffer registers to the selection and IO control element. These bits are used to specify the IO device which is to take part in IO operations. Bits RS through R15, and full right half-word, are transferred to the program element. These bits usually specify the address in core memory from which an operand is to be obtained. The S bit is transferred to the program element although such an address does not require it. This is done in order to make this operation consistent with certain other operations (such as indexing) where the S bit is necessary. The use which each of the three control elements makes of the portions of the instruction word is now considered. It should be noted at this point that although the various portions of an instruction word are distributed to three of the Central Computer System elements, the transfers are all effected simultaneously.

2.3.1 Instruction Control Element

The function of the instruction control element is to decode bits L1 through L10 of the instruction word and generate command pulses to execute the operations designated by the instruction contained in the word. During the execution of the Central Computer System program the instruction word is read out of the memory element into the memory buffer registers. Bits L1 through L10 are then transferred from the memory buffer registers to the operation register. (See fig. 1–8.) The content of the operation register is decoded by the decoding matrices which drive the command generators. The command generators are also driven by timing pulses generated in the pulse generation equipment of the instruction control element. These timing pulses are the reference points with which all Central Computer System

operations are synchronized. The command generators, which consist of a large number of gate tubes, combine the decoded instruction information from the decoding matrices with the timing pulses from the pulse generation equipment. Command pulses are thereby produced which are sent to the other elements of the Central Computer System to execute the operations dictated by the instruction in the operation register.

The pulse generation equipment consists of the time pulse distributor (TPD), the step counter, the divide time pulse distributor, and a 2-megacycle oscillator. The divide time pulse distributor is a special-purpose component of the instruction control element and will not be discussed here, since it is only used during division operations in the arithmetic element. (Refer to Part 2.) The step counter is loaded with bits R10 through R15 of the instruction from the memory buffer registers. These bits determine the length of time that the Central Computer System will pause while certain special repetitive operations are performed in the arithmetic element. Thus, if a specific instruction requires that 15 repetitive operations be performed, the step counter will be loaded with the number 15. The Central Computer System will then halt in its normal operation and begin to execute the repetitive operations. As each operation is completed, the contents of the step counter are reduced by 1. When the 15th operation is executed, the content of the step counter will be reduced to positive zero, indicating that all 15 operations have been completed and that the Central Computer System may continue in its normal operation.

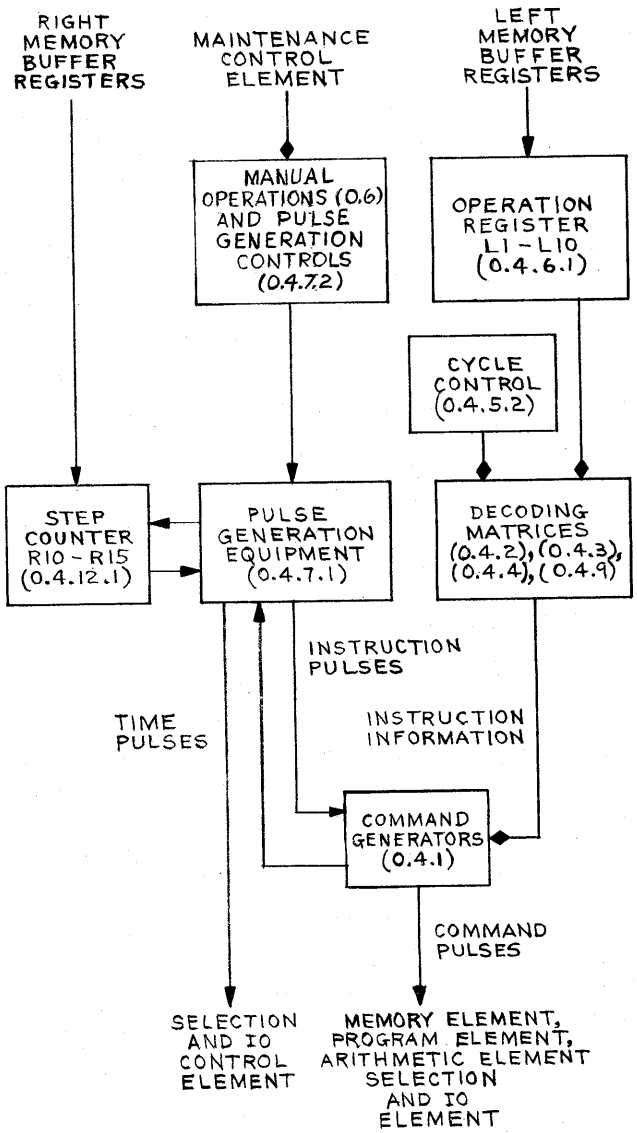


Figure 1–8. Instruction Control Element, Simplified Block Diagram

The 2-megacycle oscillator generates standard (0.1-microsecond duration) pulses at intervals of 0.5 microsecond; these are used to drive the time pulse distributor. This is essentially a 12-stage stepping register which distributes the 2-megacycle pulses from the oscillator sequentially onto each of 12 output lines. Thus each line will be pulsed at intervals of 6.0 microseconds, which is the length of the basic memory and machine cycle. (Refer to 3.1 of Chapter 3.) The time pulse distributor generates two groups of pulses, time pulses (TP) and instruction pulses (IP). The TP pulses are sent to the selection and IO control element, where they assist in the execution of IO operations. The IP pulses are used to drive the command generators; these generators produce the command pulses required to execute the instruction in the operation register.

The decoding matrices consist of four separate matrices: the class cycle matrix, index selection matrix, variation matrix, and instruction matrix. The class cycle matrix decodes bits L4 through L6 of the instruction word, the class bits of the operation code, and combines them with the outputs of the cycle control. The cycle control provides an indication of the status of the Central Computer System at the time the instruction is executed, and also indicates the type of cycle presently being executed. (Refer to 3.1 of Chapter 3.) The index selection matrix decodes bits L1 through L3, the index indicator bits, which describe which index register is to be used with an indexing operation. (Refer to 3.5 of Chapter 3.) The variation matrix decodes bits L7 through L10, the variation

bits of the operation code, which indicate the instruction within the class specified by class bits L4 through L6. The d-c levels from these three matrices are combined and mixed in the instruction matrix which drives the gate tubes of the command generators. Thus, it can be seen that the outputs of the instruction matrix, or in general the decoding matrices, provide a constant indication of the instruction in the operation register and conditions existing in the Central Computer System which might be relevant to that instruction.

The d-c levels from the decoding matrices condition the gate tubes of the command generators. These gate tubes are in turn sensed by the instruction pulses from the pulse generation equipment. Since a different group of gate tubes are conditioned for each instruction in the operation register, the timing pulses will be gated in a different sequence for each instruction. These gated instruction pulses are called commands, since they are sent to all the elements of the Central Computer System and cause the various ordered operations required for the execution of each instruction.

Since all Central Computer System operations are synchronized and initiated by the IP and TP pulses generated by the time pulse distributor, control of these two types of timing pulses provides a convenient means of controlling the overall operation of the Central Computer System. For instance, if it is desired to stop the Central Computer System, all that is necessary is to halt the flow of timing pulses from the TPD. Since there are no timing pulses, the gate tubes of the command generators will not be sensed and no command pulses will be generated; the Central Computer System will be unable to perform any operations and thus will be effectively stopped. The manual operations and pulse generation controls shown in figure 1–8 accomplish this function and contain the TPD controls and manual operations controls of the instruction control element. The TPD controls provide for Central Computer System control of the pulse generation equipment as required by each instruction of the program; the manual controls provide a means by which the Central Computer System may be manually controlled. The manual operations controls of the instruction control element are intimately associated with the maintenance control element.

2.3.2 Selection and IO Control Element

The principal purpose of the selection and IO control element is to co-ordinate the operations of the Central Computer System and the IO units so as to properly effect the transfer of information between the Central Computer System and the IO units. The process of bringing information from an IO unit into the Central Computer System is termed a read operation, while the process of transferring information out of the core memory element to an IO unit is termed a write operation. All words involved in such operations are obtained from or stored in the core memory element. An IO unit may be capable of receiving and storing information from the Central Computer System for future use or be capable of transferring all or part of its information content to the Central Computer System (or both of these functions). The various IO units associated with the Central Computer System may be grouped in the following manner:

a. Drums (consisting of 39 distinct fields)

b. Card machines (consisting of card reader, card punch, and printer)

c. Tape units (consisting of four magnetic tape units)

d. Miscellaneous IO units (special-purpose information inputs)

The index interval bits (L10 through L15 of the operation register) are used to specify which of the 39 fields of the Drum System is to be involved in an IO operation. (Refer to 2.1 of this Chapter.) It is the additional function of these bits to specify the other IO units for an IO operation. These bits are read out of core memory to the memory buffer registers along with the other bits of an instruction word. They are then transferred to the index interval register of the selection and IO control element. (See fig. 1–9.) It should be noted that these bits will have significance only if the instruction with which they are given is an instruction specifyng that an IO unt is to be selected for an IO operation. However, an exception occurs during the indexing process. (Refer to 3.5 of Chapter 3.)

The contents of the index interval register are decoded by a diode decoding matrix whose d-c levels condition a number of gate tubes in the gating controls. In this manner the binary number in the index interval register is represented by a certain configuration of conditioned gate tubes which are sensed by command pulses from the command generators of the instruction control element. If an IO unit is to be selected for an IO operation, the command pulse gated through the gating controls indicates which IO unit has been selected and is used to set up control circuits in

the IO units controls section. In addition, this section receives the TP pulses from the TPD of the instruction control element and receives command pulses from the command generators. The IO units controls section controls the IO operation in co-ordination with control pulses from the selected IO unit in such a manner as to properly effect the transfer of words between the unit and the Central Computer System. Another function of the gating controls is to initiate various operations in certain operate units within and without the Central Computer System under the control of the program. Under these conditions, the index interval bits specify which particular operate unit is to be activated. Command pulses then sense the gate tubes of the gating controls to determine the operate unit specified and the gated pulse is sent to the proper unit to initiate the indicated action. Examples of operate units are display cameras and scan counters in the Display and Drum Systems, respectively, and the mechanical printer and tape unit controls which are associated with the Central Computer System.

An exception to the described operation for selecting an IO unit occurs when the specified IO unit is one of the fields of the Drum System. In this case, the actual decoding of the index interval bits to select the proper field takes place directly in the Drum System instead of in the selection and IO control element. For this reason, a means of transferring the content of the index interval register has been provided. Although decoding for selection purposes takes place in the Drum System, decoding for control purposes is accomplished in the selection and IO control element so that a reading or writing operation may be properly synchronized and executed. During indexing operations it is necessary that the content of the index interval register be transferred to the address register of the program element, and similar transfer facilities have been incorporated to accomplish this.

2.3.3 Program Element

The program element consists of those circuits which sequence and control the Central Computer System program and, in addition, contains the transfer and control circuits necessary for the reading or writing of information between the Central Computer System and the IO units. These circuits consist of eight flip-flop registers, four associated with programming operations and four associated with IO operations.

The four registers of the program element which sequence and control the execution of the Central Computer System program in co-ordination with the instruction control element are shown in figure 1–10. The address register contains the address portion of each instruction. This

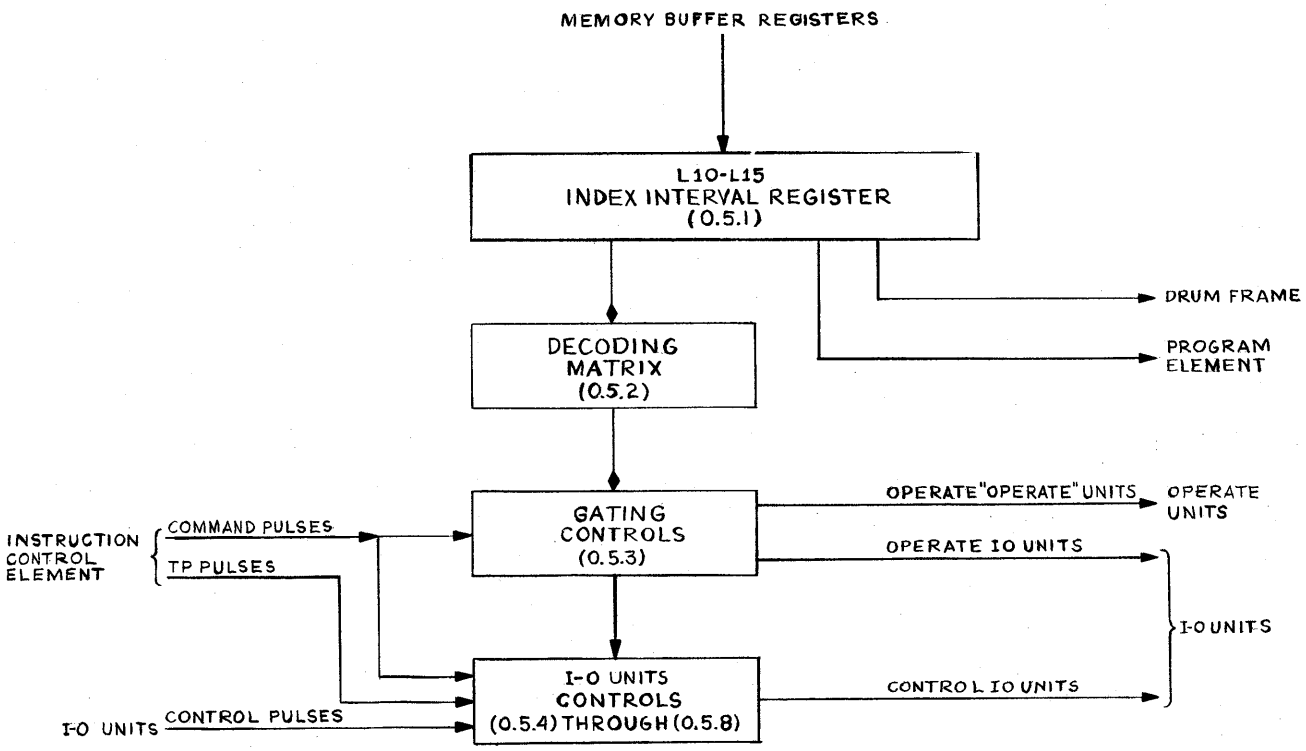


Figure 1–9. Selection and IO Control Element, Simplified Block Diagram

half-word is transferred from the memory buffer registers to the address register as each instruction is read from core memory. The time of transfer is simultaneous to the transfer of the left half-word to the operation register and index interval register of the instruction control and selection and IO control elements, respectively. If the instruction being executed by the Central Computer System requires an operand, the location of this operand in core memory will be specified by the word then in the address register. Under these circumstances, the address is transferred from the address register to the memory address register in the selection section of the core memory element. The manner in which the word stored in the core register specified by this address is read out into the memory buffer registers has been described in 2.2 of this Chapter.

The instructions of a program are stored in sequentially addressed locations in the core memory element. Thus, a program consisting of five instructions would be stored in core memory locations 0000, 0001, 0002, 0003, and 0004 of core memory core No. 1. In a similar manner, a program of N instructions would be stored in core memory locations 0000 through N–1. It is the function of the program counter to control the sequential read-out of each of these instructions from the core memory element into the Central Computer System proper. The program counter is a counting register and its content at any one time will consist of the number of pulses which have been applied to its input line. The program counter is cleared at the beginning of any program so that it contains 0000, which is the address of the first instruction in the program. Its content is then transferred to the selection section (memory address register) of the core memory element so that the instruction contained in this core memory location (0000) is transferred to the Central Computer System proper. During the execution of the first instruction, a command from the instruction control element adds a 1 to the program counter; i.e., steps the counter so that its new content is 0001. The contents of the program counter are again transferred to the core memory element so that the contents of core memory location 0001, which comprise the second instruction in the program, will be read out by the time the first instruction is completed. Each time an instruction is

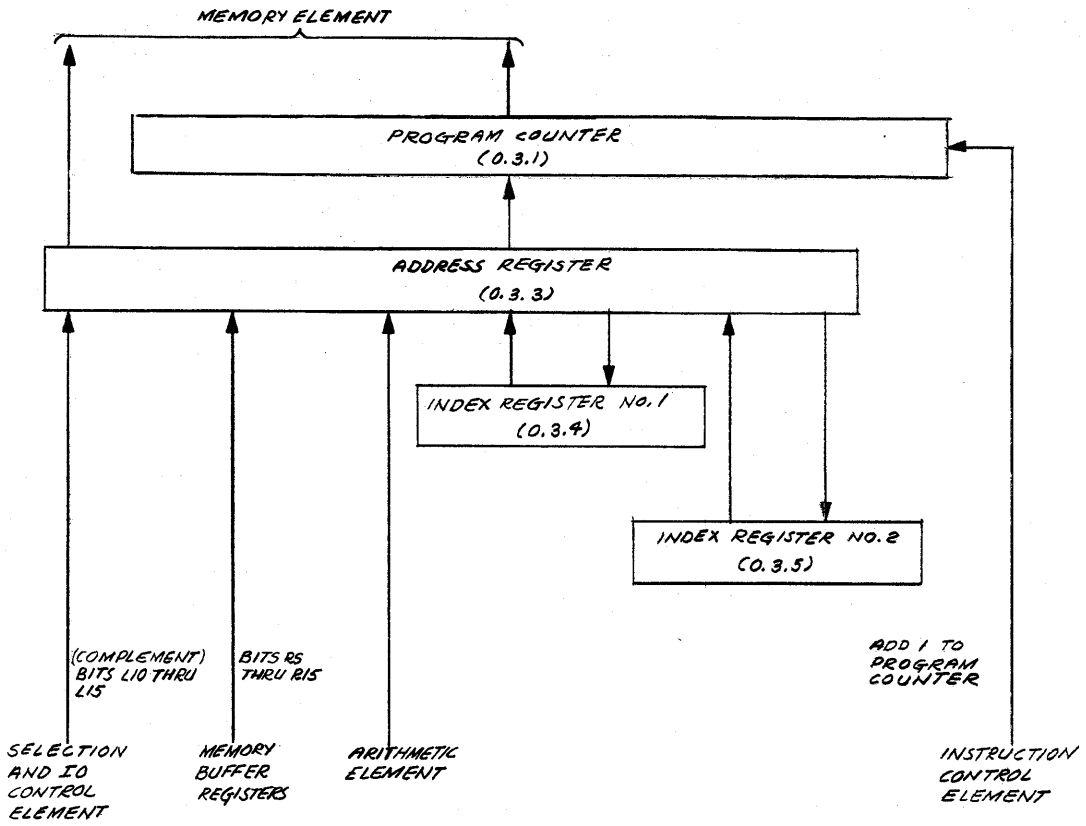


Figure 1–10. Program Element, Programming Circuits, Simplified Block Diagram

being executed, a 1 is added to the program counter and the contents are transferred to the core memory element so that the succeeding instruction in the program may be read out of core memory and executed. Thus, the program counter will always contain the core memory address of the next instruction to be executed.

In some instances, the course that a program is to take is dependent upon the results of calculations performed during the program. An example of this is the *Branch on Minus (BFM)* instruction where the course of the program depends on the results of a subtraction. If the result of the subtraction preceding the *BFM* instruction is positive, the Central Computer System will proceed normally and the next instruction will be obtained from the core memory location following the one that contained the *BFM* instruction. However, if the result of the subtraction was negative, the Central Computer System is not to execute the next instruction in the program sequence but rather take its next instruction from the location specified in the address portion of the *BFM* instruction. This is accomplished in the following manner.

After the subtraction is performed, the instruction in core memory location N will test the sign of the result. In addition, its address portion will contain the number Y. If, during the execution of the instruction, the result is found to be positive, no action will be taken. A 1 will be added to the program counter and its contents (N+1) will be transferred to the core memory element so that the next sequentially located instruction may be obtained. However, if the sign of the result is found to be negative, the program counter will be cleared and the contents of the address register transferred into it. Since the number transferred from the address register to the cleared program counter is Y, the next word to be read out into the Central Computer System for execution will be the instruction in core memory location Y. It is in this manner that the execution of a program is altered, depending upon the results of calculations or existing conditions. Such an operation is termed a branch and the Central Computer System is said to have branched the control of its program from core memory location N to core memory location Y.

It is sometimes necessary to continuously repeat the execution of a selected portion of a program. For instance, it might be necessary to find the square of a number (X) for different values of X. In such a case, the program would be identical but only the operands would change. Suppose that it is desired to compute X^2 for two values of X. The Central Computer System would first execute the program to find X^2, using one of the values as an operand and then repeat the same program using the other value as an operand. Suppose that these two values are stored in core memory location 2000 and 2001 and are called X_1 and X_2, respectively. First, the Central Computer System runs through the program for X^2, obtaining X_1 from core memory location 2000 whenever it is needed, and then runs through another program of the same nature, obtaining X_2 for core memory location 2001 whenever it is needed. Whenever either X_1 or X_2 are required by an instruction the address portion of the instruction will be either 2000 or 2001, respectively. It is apparent that the two programs are identical except that the second program will specify core memory address 2001 whenever the first program specified core memory address 2000.

In order to obviate the necessity of composing and storing programs of this type, which are identical except for operand addresses, the Central Computer System has been equipped with index registers, shown in figure 1–10, which are capable of modifying the address portion of an instruction word. The purpose and action of these index registers will be demonstrated by using the above-described example. In order to obtain the value of X^2 for X_1 and X_2 stored in core memory locations 2000 and 2001, a program is written to first find X_1^2. This program would contain the number 2000 in the address portion of every instruction which required X_1 as an operand. Running this program through the Central Computer System then produces the first answer, X_1^2. The configuration of the circuits of the index registers and address register is such that the contents of the index register may be added to the address register during any instruction. The addition of an index register to the address register takes place during an instruction if bits L1 to L3 of the instruction word contain the code for that index register. (Refer to 2.1 of this Chapter.) If the binary number 1 is placed in index register No. 1 and the program previously executed to find X_1^2 is recycled through the Central Computer System, X_2^2 may be obtained in the following manner. Each time an instruction in the program has an address portion specifying the number 2000, the content of index register No. 1 is added to the address register. Thus, the new content of the address register will now be 2001, which specifies the address in core memory of X_2

instead of X_1. The result will be that X_2 is substituted for X_1 during the second cycling of the program, and the Central Computer System will therefore calculate X_2^2.

The process of modifying an instruction address is called indexing. The foregoing analysis has been elementary, but indexing is discussed in detail in 3.5 of Chapter 3. In addition to the two index registers contained within the program element, the right accumulator register in the arithmetic element is also used as an index register. In order to modify the content of the index registers during cycling loops, the contents of the index interval register may be subtracted from the address register. This is accomplished by transferring the contents of an index register to the cleared address register. The contents of the index interval register in the selection and IO control element are then transferred in complement form to the address register. The circuits of the address register act to subtract the index interval bits from the original number in the address register and temporarily store the difference. Thus, if the index register contains the number Y and the index interval register contains the number Z, the address register will calculate and store the difference, Y–Z. This value is then transferred to the index register from which the original number was obtained. It can be seen that the total operation decreases the contents of any selected index register by an amount equal to the contents of the index interval bits in the index interval register. (Refer to 3.5 of Chapter 3.)

The four registers of the program element which sequence and provide transfer paths for IO operations are shown in figure 1–11, including the address register. Although the address register is not directly associated with the control of IO operations, it provides the means by which information may be placed in the other registers, and is necessary to the logic of the figure. Information is transferred in the form of blocks of binary words between the Central Computer System and the IO units, and the process must be sequenced, synchronized, and generally controlled by the Central Computer System. The operations of synchronization and control are accomplished by the selection and IO control element. However, the sequencing of these IO operations is performed by the circuits of the program element. (See fig. 1–11.)

When words are to be transferred into or out of the Central Computer System, it is necessary to specify which locations in core memory these

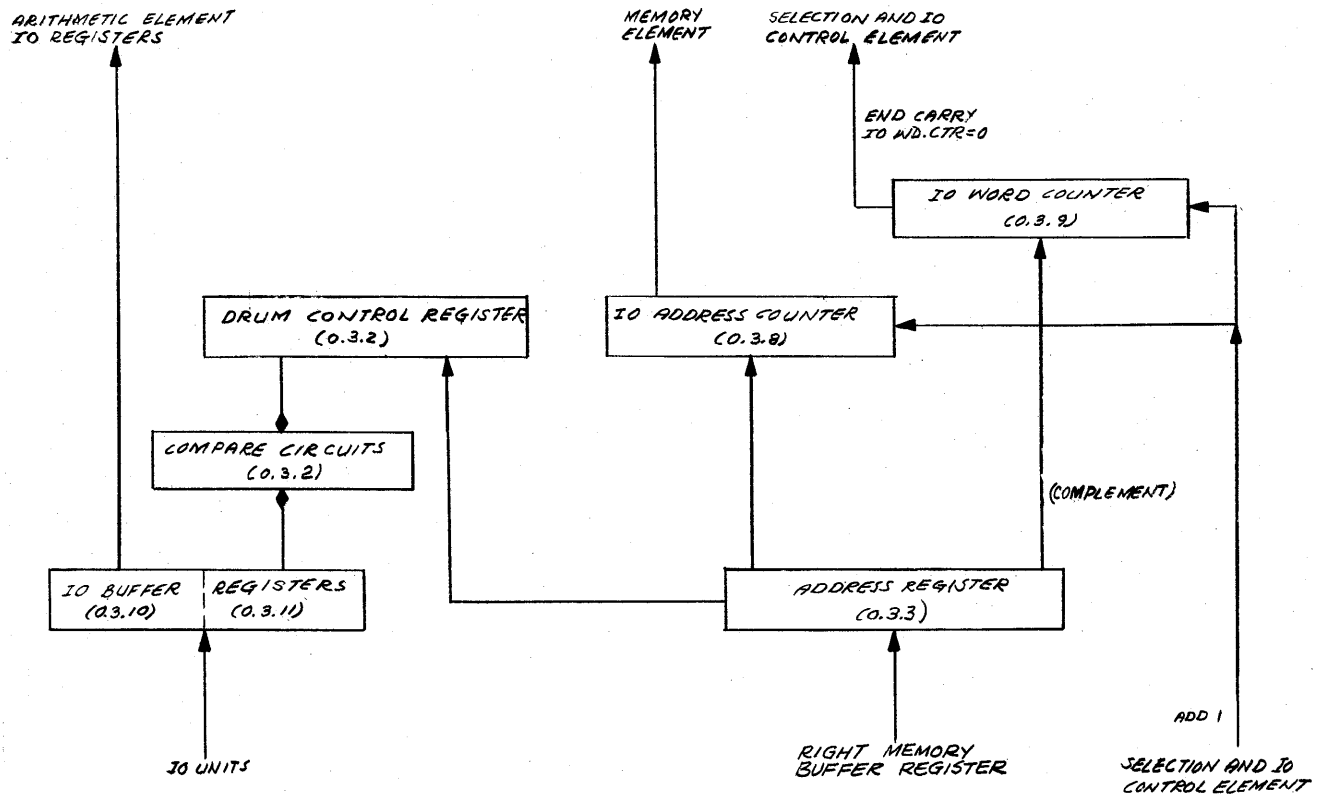


Figure 1–11. Program Element, IO Circuits, Simplified Block Diagram

words are to come from or go to, and also how many words are to be transferred. These functions are accomplished by the IO address counter and IO word counter, respectively. Before an IO operation is initiated, the IO address counter is loaded with the address in core memory from which or to which the first word is to be transferred. Provision is made for the transfer of the content of this counter to the selection section (the memory address register) of the memory element so that the word will be read from or written into the proper core memory location. As each word is transferred between the Central Computer System and the selected IO unit, a 1 is added to the contents of the IO address counter and its new contents are transferred to the core memory element. Thus, the operation of the IO address counter is analogous to that of the program counter. Words involved in an IO operation are associated with sequentially located addresses in core memory and the IO address counter is repeatedly stepped so that it contains the core memory address of the next word to be transferred. The IO word counter performs the function of counting the number of words which have been transferred in an IO operation. The complement of the number of words to be read or written by the Central Computer System is placed in the IO word counter. Each time a word is transferred between the Central Computer System and an IO unit, a 1 is added to the IO word counter. When a number of words equal to the number originally specified by the content of the counter has been transferred, the IO word counter will contain positive zero. At this time it will also generate an end-carry pulse which is sent to the selection and IO control element, causing the IO operation to be halted. The IO address counter is similarly stepped as each word is transferred so that it consistently contains the core memory address associated with the next word to be transferred. Thus, the combination of IO address and word counters acts to sequence and count the successive word transfers during IO operations.

The IO buffer registers provide a transfer path into the Central Computer System for words being read from the Drum, Input, or Output Systems. Thus, if a word is to be read from one of these systems into the Central Computer System, it first will be transferred to the IO buffer registers and then through the IO registers and memory buffer registers of the arithmetic element into the core memory element. It should be noted that only those IO units associated with the Drum, Input, or Output Systems transfer words to the Central Computer System through the IO buffer registers; words read from other IO units are transferred directly through the IO registers contained in the arithmetic element. (Refer to 2.4 of this Chapter.)

The IO buffer registers, acting in co-ordination with the drum control register of the program element, perform the additional function of identifying the location of words, or the words themselves, to be read or written on a selected drum field. Reading from the drums is possible by any one of three modes: address, identification, or status. However, only the address and status modes are applicable to writing. When reading or writing by address, the address on the field of the word to be read or written is placed in the drum control register. This address is an 11-bit binary number contained in bits 5 through 15 and represents the address of one of the 2048 registers on the field. As the drum rotates the address of each word which will shortly pass under the drum read heads is placed in the right IO buffer register. This address comes from the angular position counter of the selected drum in the Drum System and is compared with the specified address in the drum control register. If the two addresses do not compare, no word transfer takes place and the address of the next register to pass under the drum heads is placed in the right IO buffer register. This comparison of each field register address with the address in the drum control register continues until a successful comparison is made. At this time the word contained in the addressed drum register is read into the IO buffer registers of the Central Computer System. When writing, the word contained in the drum write register of the Drum System (previously transferred there from the Central Computer System) is written onto the addressed drum register. In certain cases it is desirable to space the registers on a field in the Drum System on which a word is read or written. This is accomplished by an operation called interleaving, which may be used only when a field is being read or written in the address mode. (Refer to 3.3 of Chapter 3.) One of the requirements of interleaving is that the content of the drum control register be changed by a fixed amount. For this reason, the drum control register has been incorporated with special circuits which allow the addition of either 8, 16, or 64 to it. This provides a means of reading or writing on every 8th, 16th, or 64th register.

The identification mode applies only to reading. Reading by identification is similar to reading

by address, except that certain identification bits in each word on the registers of a selected field are compared with the identification bits which have been placed in the drum control register. Each word passing under the drum heads is loaded into the IO buffer register and its identification bits compared with the bits in the drum control register. If the comparison is unsuccessful the word is cleared out of the IO buffer registers and the next word to pass under the drum heads is placed therein. If the comparison is successful, the word is transferred through the arithmetic element into the core memory element for storage. One other method of reading or writing words on a drum may be utilized. This is called reading or writing by status. When reading or writing a drum field by status, the Central Computer System will read every full register on the field or write a word in every empty register on the field. No comparison is performed in this case.

2.4 THE ARITHMETIC ELEMENT

The arithmetic element contains the registers and circuits of the Central Computer System which are capable of performing arithmetic computations with numerical data as specified by the program. The combination of A registers, adders, accumulator registers, and B registers comprises the arithmetic operational registers. (See fig. 1–12.) Included in the arithmetic element are the clock register, which provides relative time data, the test registers, which provide for auxiliary single word datum storage and marginal checking control, and the IO registers, which provide a transfer path between the Central Computer System and the IO units.

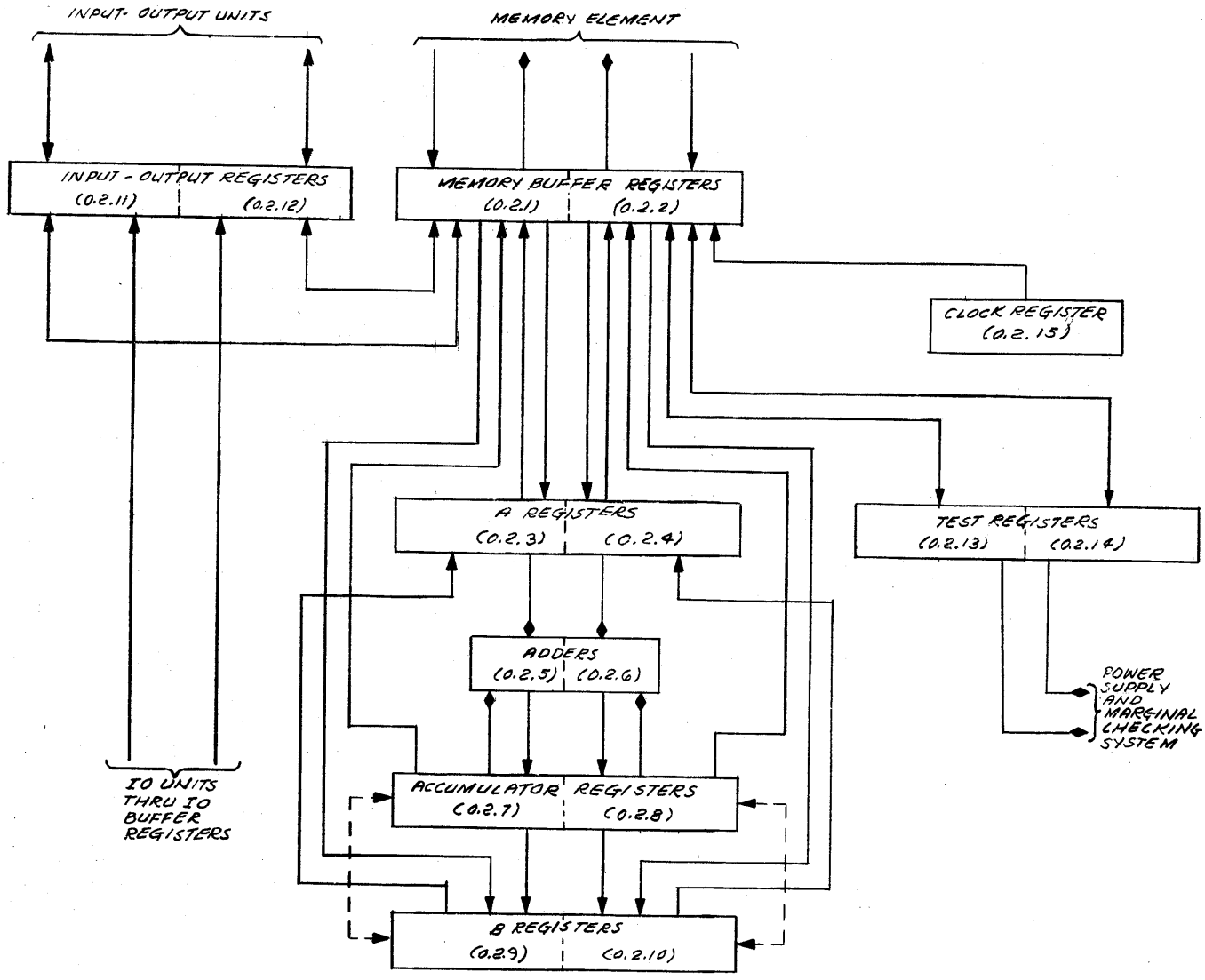


Figure 1–12. Arithmetic Element, Simplified Block Diagram

The memory buffer registers of the Central Computer System are located in the arithmetic element. These registers serve as isolation registers between the Central Computer System registers and the core memory element. All information entering or leaving the core memory element passes through these registers, which serve as a central distribution point to the Central Computer System. (See figs. 1–7 and 1–12.) Numerical data received from core memory to be used by the arithmetic element must be transferred from the memory buffer registers to the A registers, where they will remain long enough to participate in normal arithmetic processes.

2.4.1 Arithmetic Equipment

A full 32-bit numerical word is composed of two binary numbers, each consisting of 15 magnitude bits plus one sign bit. (Refer to 2.1 of this Chapter.) The computational registers of the arithmetic element are designed to process both of these numbers simultaneously. For this reason, the 32-bit A register, accumulator registers, and B registers are each functionally divided into two 16-bit registers. Thus, the A register is composed of the right A register and the left A register, each of which stores one of the two numbers contained in a full Central Computer System numerical word. An identical arrangement exists for the accumulator registers and B registers. In the discussion to follow, arithmetic operations are considered as involving only one number; however, it should be realized that in reality two numbers are being simultaneously processed in an identical manner with both left and right correspondent registers.

The arithmetic element is capable of performing four basic mathematical operations: addition, subtraction, multiplication, and division. During such basic operations, the A registers serve to store the pertinent operands, which would be the addend, subtrahend, multiplicand, and divisor, respectively. These operands may be added from the A registers into the accumulator registers by means of the adders. Thus, if the A registers hold the number X and the accumulator registers hold the number Y, an addition process instituted by an instruction in the Central Computer System program will cause the content of the A registers to be added to the accumulator registers, with the result that the latter registers will contain the sum Y + X. Subtraction is accomplished in a similar manner except that the A register is complemented to obtain the 1's complement of X before the addition is initiated. Thus, addition of the A registers with the accumulator registers will result in the difference, Y — X, being contained in the accumulator registers. The addition and subtraction processes are the fundamental operations from which the multiplication and division processes are derived. During multiplication, for instance, the multiplicand is held in the A register and continuously added into the accumulator registers. This method of multiplication is based on the fact that the multiplication of X by Y may be accomplished by adding X to itself Y times. In a similar manner, the division of Y by X (Y/X) may be accomplished by subtracting X from Y continuously until the remainder becomes negative (assuming that Y was positive to begin with). The quotient is then equal to the number of times the subtraction may be successfully performed. Variations of these methods of multiplication and division (which involve shifting) are used by the Central Computer System to speed operations and are discussed in detail in Part 3.

It can readily be proved that multiplication of two 16-bit numbers by two other 16-bit numbers will produce two 32-bit products. For this and other reasons, the B registers are provided to supplement the accumulator registers in their arithmetic functions. These registers serve to hold the multiplier and provide for storage of one-half of each of the two products which overflows from the accumulator registers during multiplication. Thus, the 32-bit product of two 16-bit numbers is stored in a combined accumulator-B register, the accumulator register containing the first 16 bits of the product and the B register containing the last 16 bits of the product. In order to provide for passage of the bits overflowing from the accumulator into the B register during multiplication, serial transfer paths have been provided between them, as indicated by the dashed lines in figure 1–12. Division operates in the reverse manner. A 32-bit dividend is placed in the combined accumulator-B register and repeated subtraction of the divisor, which is held in the A register, produces a 16-bit quotient in the B register with the remainder after the division consisting of 16 bits in the accumulator register.

Both the accumulators and B registers are capable of performing the arithmetic operation of shifting their content to the left or right. A shift of the content of a register N places to the left or right corresponds to the multiplication or division, respectively, of the number in the register by 2N. Since the results of arithmetic operations performed in the arithmetic element are held in the

accumulator and/or B registers, provision is made for the transfer of their contents to the memory buffer registers and thence into core memory for storage. The contents of the accumulator registers may be transferred directly to the memory buffer registers for storage in core memory. However, the content of the B registers must first be placed in the A registers before storage in core memory. In addition, the B registers may be loaded directly from the memory buffer registers, whereas the accumulator may be loaded only from the A registers through the adders.

In addition to the registers discussed, the arithmetic element contains the IO registers, clock register, and test registers. The IO registers serve as a distribution center for all information entering and leaving the Central Computer System during IO operations. Information entering the Central Computer System through the IO buffer registers, as discussed in 2.3 of this Chapter, is transferred to the IO registers and thence through the memory buffer registers into core memory. Information from IO units which does not enter through the IO buffer registers is received directly by the IO registers and then transferred into the core memory element through the memory buffer registers. When the Central Computer System is writing information from core memory in an IO unit, the IO registers receive the data from core memory and store it until it can be accepted by the selected IO unit. In general, these registers may be considered as buffer stages between the Central Computer System and the IO units. That is, the IO registers are required to take information from low-speed devices (the IO units) and make it available to the Central Computer System at a high speed synchronized with the Central Computer System operations, and vice versa. It should be noted that these transfers are controlled by the selection and IO control element. (Refer to 2.3 of this Chapter.)

2.4.2 Clock Register

AN/FSQ–7 (XD–1,–2) Combat Direction Centrals must be provided with real time information for maintaining chronolgical records and determining accurate time increment measurements. This is accomplished by the clock register, which maintains a count of the time in increments of $\frac{1}{32}$ second, or 31.25 milliseconds. This is accomplished by the counting circuits of the clock register and its associated clock pulse generator. The clock pulse generator generates pulses at intervals of $\frac{1}{32}$ of a second which are used to step the clock register; i.e., add 1 to it. Thus, if the clock register is cleared at the beginning of an operation, the number it contains at any time will be equal to the number of $\frac{1}{32}$'s of a second which have passed since the operation was initiated.

The reason for the existence of the clock register may best be exhibited by means of an example. Suppose it is desired to calculate the velocity of an aircraft in a cartesian co-ordinate (X and Y) system. In order to accomplish this it would be necessary to know the position of the aircraft in terms of its X and Y co-ordinates at two different times. Let it be assumed that the co-ordinates of the aircraft are given by X_1 and Y_1 at a time T_1. The co-ordinates X_1 and Y_1 would be stored in core memory along with the time at which they were observed, which is obtained from the clock register. At some later time T_2, the new set of co-ordinates (X_2 and Y_2) of the aircraft would be obtained and stored in core memory along with T_2 obtained from the clock register. The Central Computer System may now easily calculate the average velocity of the aircraft along the X and Y co-ordinates. Thus, the presence of the clock register in the Central Computer System, which supplies the values for T_1 and T_2, enables AN/FSQ–7 (XD–1,–2) Combat Direction Centrals to make real time calculations applicable to the physical world outside of the system. In order to transfer the time in the clock register to the arithmetic element, where it may be used in calculations, a transfer path has been provided from the clock register to the memory buffer registers. The clock register is selected by bits R1 through R3 of the address register. (Refer to 2.1 of this Chapter.) When the clock register is selected, it will have its contents transferred to the memory buffer registers.

2.4.3 Test Memory

Test memory is the name applied to that unit of the Central Computer System which is capable of storing up to 16 words and transferring them to the memory buffer registers of the arithmetic element. Test menory is essentially a storage device into which data may be manually inserted and from which the Central Computer System is able to read at standard operating speeds. This unit is used for the preliminary testing of the Central Computer System when the core memory element is not available or is malfunctioning, for storage of short programs not exceeding a length of 16 instructions, for reading information into core memory for testing, and for storage of constants used in special programs.

A simplified block diagram of test memory is shown in figure 1–13. The storage devices of test memory consist of a 16-word control panel register, two toggle switch registers, and the test registers (which are also illustrated in figure 1–12). The plugboard and test registers are located in the arithmetic unit and the toggle switch registers are on the maintenance console panel. The control panel is of the type used on the IBM Electronic Document Originating Machine, Type 519. By means of manually inserted jacks, a control panel may be loaded with sixteen 32-bit words. The plugboards may be physically removed from their unit and stored for future use. It is in this manner that a library of short programs may be maintained, obviating the necessity for plugging in a 16 by 32 array of 512 jacks each time the content of the plugboard registers must be changed. The toggle switch registers are composed of two banks of 32 toggle switches each, thus specifying two 32-bit binary words. The test registers, which are composed of 32 flip-flops, may be loaded only by programming from the memory buffer registers. Although test memory has provision for the storage of 19 words, only 16 may be read by the Central Computer System during a program. Hence, test memory has only 16 memory addresses numbered from decimal 0 through 15, and specified by bits R12 through R15 of the address register. (Refer to 2.1 of this Chapter.)

When test memory is selected, these bits are transferred from the program element to the test memory address register shown in figure 1–12. The decoding matrix decodes these bits to determine the test memory location of the desired word. Associated with the designation of the proper core memory location in test memory is a TEST MEMORY ASSIGNED-UNASSIGNED switch on the maintenance console panel. If this switch is in the UNASSIGNED position, memory addresses 0 through 15 specify the 16 registers of the plugboard. However, if this switch is in the ASSIGNED position, memory addresses 0 and 1 will specify the two toggle switch registers and

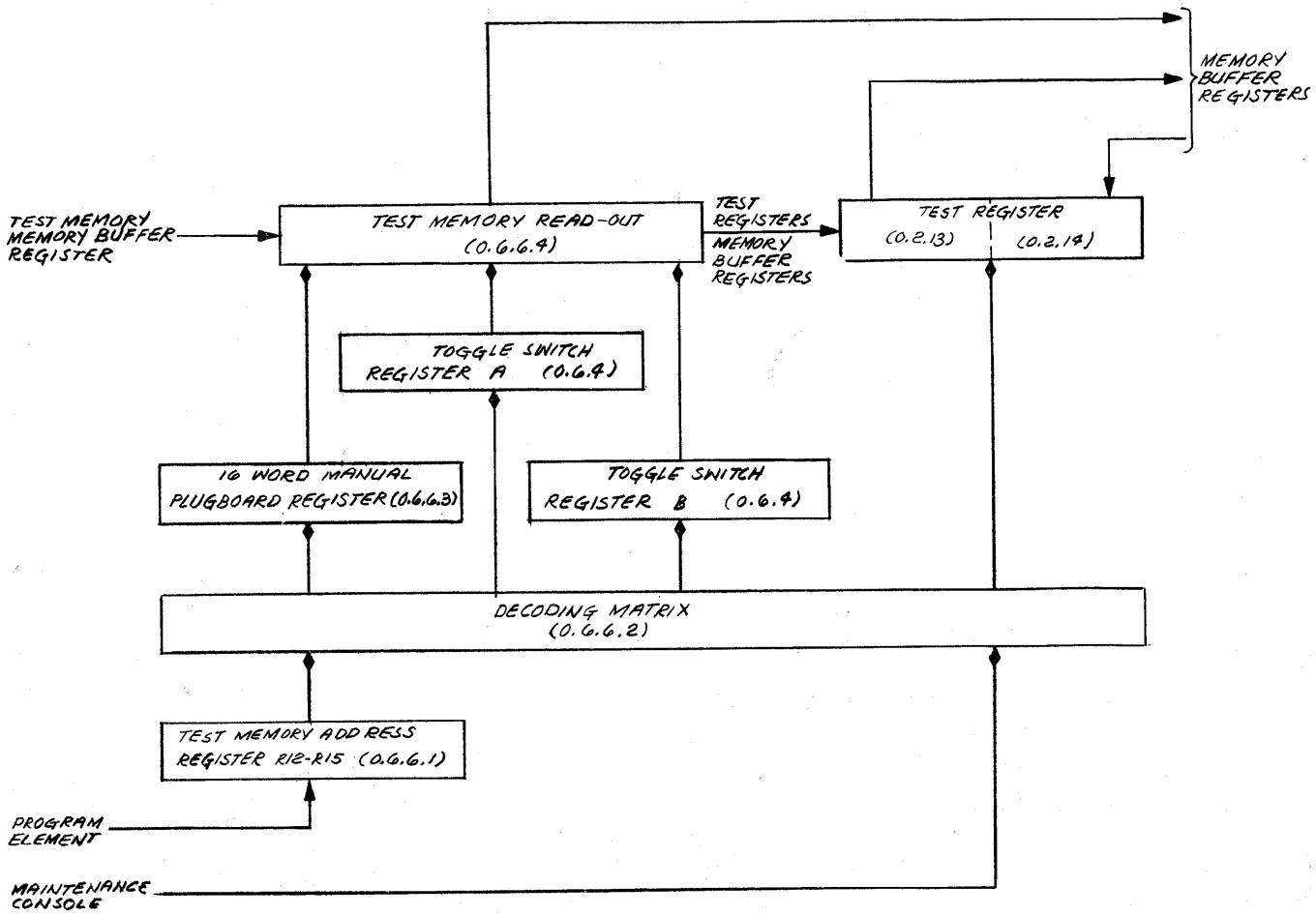


Figure 1–13. Test Memory, Simplified Block Diagram

core memory location 15 will specify the test registers. Thus, 16 words may be read from the plugboard or 13 words may be read from the plugboard and 3 words from the toggle switch and test registers, depending upon the position of the TEST MEMORY ASSIGNED-UNASSIGNED switch. When the core memory location has been determined in accordance with bits R12 through R15 and the switch, a command pulse will be received from the command generators of the instruction control element, transferring the content of the specified test memory register to the memory buffer registers. Once the word is in the memory buffer registers, the Central Computer System handles it in the same manner as if the word had been read out of core memory. If the word is an instruction, it will be executed; if the word is an operand required by a previous instruction, the operand will be used in the specified operation. Essentially, test memory may be considered as supplementing the core memory, but differing in the fact that data may be introduced manually by Central Computer System personnel instead of having to be loaded from the IO units by programming. The exceptions to this are the test registers, which may be loaded only by the Central Computer System. Provision is made for programming marginal checking routines from these registers. The d-c level outputs from the flip-flops of the test register actuate thyraton-controlled relays which initiate the marginal checking operation.

2.5 MAINTENANCE CONTROL ELEMENT

The primary function of the maintenance control element is to keep the Central Computer System operating properly and keep its down time to a minimum. In addition, it provides controls by means of which the Central Computer System is initially loaded with programs and other data which must be introduced externally. The maintenance control element may be broken down into the five functional sections. (See fig. 1–14.) Control and maintenance of the Central Computer System are accomplished by operation of pushbuttons, switches, dials, etc., which are mounted on the panel of a console. This is called the maintenance console and is the focal point for manual control of the Central Computer System and associated equipment by personnel for the purpose of testing AN/FSQ–7 (XD–1,–2) Combat Direction Centrals.

The power controls section of the maintenance control element consists of a number of pushbuttons which control the status of the circuits in the power control and distribution unit (PCD) in such a manner as to apply or remove the various a-c and d-c voltages in the Central Computer System. These controls are only for the convenience of the operator and do not supplement or take priority over the main controls on the PCD unit itself. The marginal checking controls are associated with the maintenance of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals and provide a most powerful means of preventing the system from failing during critical operating periods. This is accomplished by a marginal checking procedure which varies certain critical voltages on the circuits to the limits at which the circuits are expected to operate properly. In doing this, components which have deteriorated to such an extent

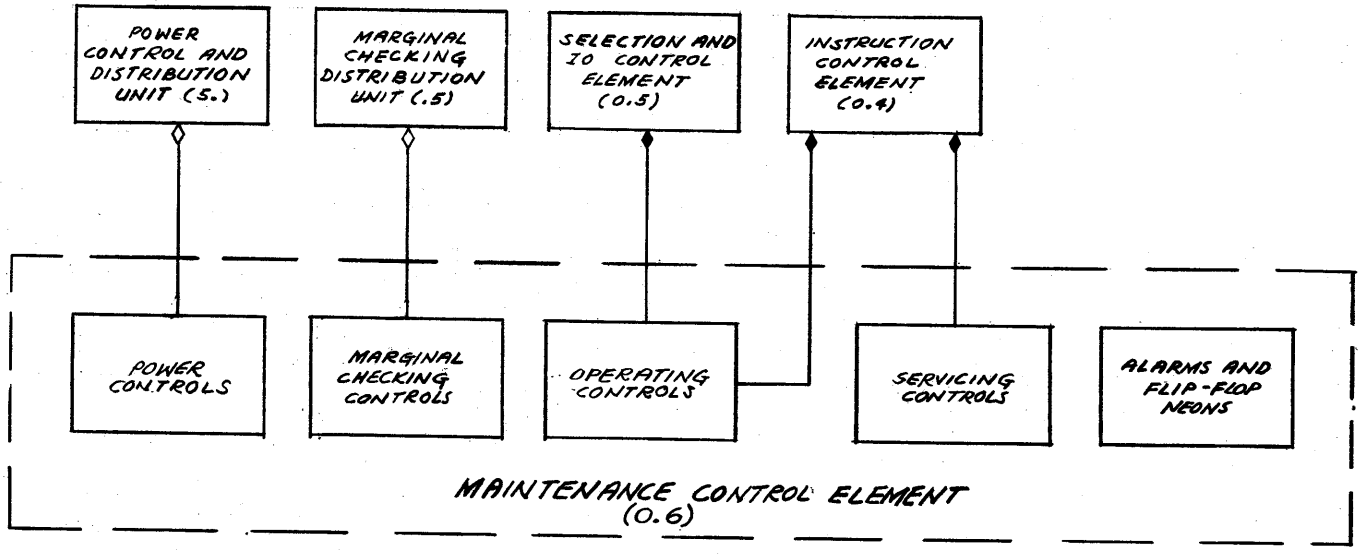


Figure 1–14. Maintenance Control Element, Simplified Block Diagram

that they might be expected to fail in a short time are detected. In conjunction with marginal checking, a general reliability program is used, which is written to put the equipment through an orderly set of operations and to give an indication of faulty operation. The marginal checking controls also provide a means of selecting the circuits to be checked and choosing the amount of voltage variations (excursion voltage) to be applied to them. The actual control and distribution of the marginal checking excursions is accomplished in the marginal checking and distribution unit (MCD), although the operation is controlled remotely by an operator at the maintenance console. Marginal checking may also be initiated by the Central Computer System program through the test registers. (Refer to 2.4 of this Chapter.)

The operating controls section of the maintenance control element contains those pushbuttons and switches which allow the Central Computer System and associated IO units to be manually controlled. This control may take the form of clearing all of the 8192 core registers in core memory, initially loading the Central Computer System with a program from the card machines or the Drum System, stopping or starting the Central Computer System, or directing the future execution of the program. In order to accomplish these functions, the circuits of the operating controls section are tied in with those of the instruction control and selection and IO control elements. Since these elements control the internal and external operations of the Central Computer System and they in turn may be controlled from the maintenance console, it is possible for personnel to manually intervene in the operation of the Central Computer System or institute new operations as desired. The servicing controls section of the maintenance control element provides a means, through pushbuttons and switches, of maintaining the reliability of the Central Computer System in conjunction with marginal checking. These controls allow the testing of circuits which are not readily checked by programming. They provide a means of complementing selected flip-flops in the Central Computer System to see if they are operating properly. In addition, they provide a means of causing the Central Computer System to continuously execute a selected portion of its program or causing it to execute the program one instruction, or portion of an instruction, at a time.

In order to intelligently control the operation of the Central Computer System, the operator must be aware of circuit conditions. For instance, the operator might need to know what numbers are in certain registers, if timing pulses are being generated, or if an IO operation is in progress. An indication of conditions such as these is provided by a number of neon lights on the maintenance console. These neon lights, one for every flip-flop in the Central Computer System, indicate the status of all the registers and control circuits and are mounted on the panel of the maintenance console. In addition to the flip-flop neon lights, a number of alarms are also provided to indicate errors in calculations or the existence of conditions which might be deleterious to the components of the Central Computer System. An example of a typical alarm condition is failure of the air conditioning while a-c or d-c voltages remain applied to the units. This might occur because of a breakdown in the air conditioning system, and is dangerous in that the Central Computer System and other system circuits will seriously overheat without the proper cooling air supply. In this case a light on the panel of the maintenance console will go on and an audible alarm will also be generated. Other alarm conditions are air pressure, low voltages, errors in calculations or word transfers, etc. Audible alarms are provided to supplement the indicating lights in all cases except those which are capable of automatically stopping the Central Computer System.

CHAPTER 3
OPERATIONAL ANALYSIS

3.1 CYCLES AND TIMING

The interval of time required to read or write a word between the core memory registers and the memory buffer registers is 6.0 microsconds and is called a memory cycle. (Refer to 2.2 of Chapter 2.) One memory cycle is required for every word read from or written into core memory. In addition to the memory cycle, the operation of the Central Computer System may be discussed in terms of instruction cycles. An instruction cycle is defined as the time required for the Central Computer System to completely execute one instruction and is usually composed of from one to three memory cycles. The reason for this becomes apparent when the characteristics of the various instructions associated with the Central Computer System are considered. Of the 48 instructions used in AN/FSQ–7 (XD–1, –2) Combat Direction Centrals, 11 involve simple operations such as setting up control circuits or transferring words between two registers; these operations may be completed in 6.0 microseconds or less. Because of their simplicity of operation, these instructions are completed in one memory cycle and are called one-memory-cycle instructions. However, other instructions require that an operand be obtained from the core memory element before they are completed. In instructions of this type, a second memory cycle must be provided during which core memory is referred to, the operand obtained, and the instruction completed. There are 18 instructions which require an additional memory cycle and they are called two-memory-cycle instructions. There are six instructions which require three memory cycles for completion. In these, the instruction is obtained and decoded during the first memory cycle; an operand is obtained and the operation performed during the second memory cycle and the result of the operation then is stored back in the core memory element during the third memory cycle. The memory cycles which compose an instruction cycle have been assigned distinctive names for easy reference. The names and characteristics of these are listed in table 1–6.

TABLE 1–6. TITLES AND CHARACTERISTICS OF MACHINE CYCLES

MEMORY CYCLE	TITLE	CHARACTERISTIC
First	Program Time (PT	Decodes instruction and initiates execution
Second	Operate Time A (OT-A)	Obtains operand and performs operation
Third	Operate Time B (OT-B)	Stores result of operation in core memory

These memory cycles are termined machine cycles when referring to the Central Computer System to distinguish them from memory cycles associated with the core memory element.

Within a machine cycle, Central Computer System operations are synchronized and identified by the time and instruction pulses (TP and IP). (Refer to 2.3 of Chapter 2.) These pulses are generated by the TPD of the instruction control element. There are 12 time pulses in a machine cycle, occurring at 0.5-microsecond intervals. These pulses are numbered in the following manner: TP–0, TP–1, TP–2, TP–3, TP–4, TP–5, TP–6, TP–7, TP–8, TP–9, TP–10, and TP–11. A similar method of numbering is utilized for the instruction pulses, although IP–0 is not needed and is therefore not generated. Although TP pulses are generated simultaneously with IP pulses, they are not used to execute instructions but are sent to the selection and IO control element, where they sequence and synchronize IO operations. A machine cycle (PT, OT–A, or OT–B) begins with instruction pulse 0 (IP–0) and ends with instruction pulse 11 (IP–11), at which time the instruction and time pulses recycle from 0 through 11 for the next machine cycle. (It should be noted

that although IP–0 does not physically exist, the time at which it would normally occur is referred to, since it simplifies the discussion.)

Although a machine cycle begins at IP–0 time, an instruction cycle starts with IP–7 of a program time cycle, which is denoted PT–7. It is at this time that the instruction word is transferred from the memory buffer registers to the Central Computer System elements. (See fig. 1–7.) This is indicated by the charts of figure 1–15, which show the basic machine and instruction cycles. In these charts, the instruction cycles are shown

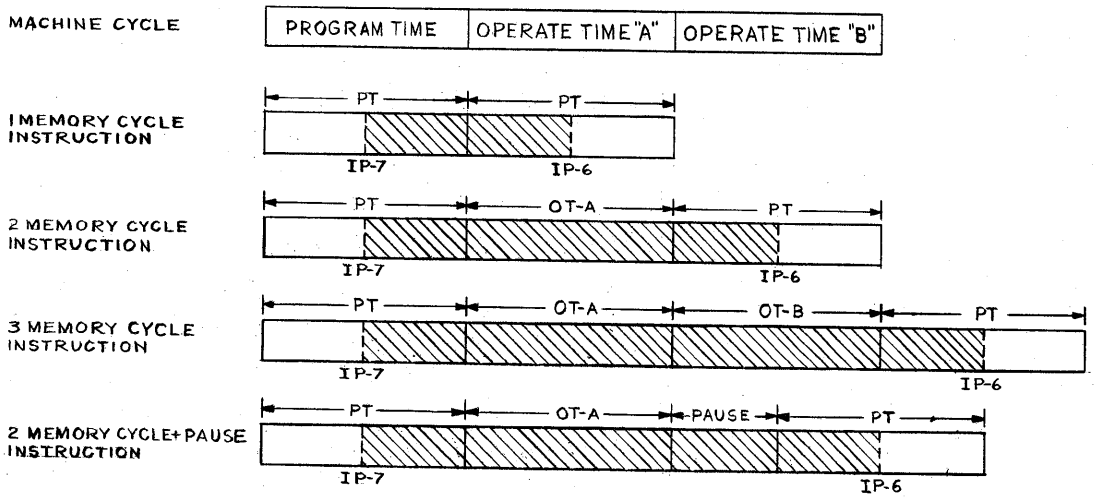


Figure 1–15. Machine and Instruction Cycles

as crosshatched areas within the machine cycles in which they take place. Thus, in a one memory cycle instruction, the decoding process starts at PT–7 and the instruction is completed by PT–6 of the subsequent PT cycle, when decoding of the next instruction is initiated. Similarly, a two memory cycle instruction starts at PT–7, continues through the subsequent OT–A cycle, and is completed by PT–6 of the next PT cycle, when the next instruction will begin to be executed. In some cases (store class), an OT–B cycle replaces the OT–A cycle in order to immediately store a word.

An exception to the usual sequence of machine cycles takes place in the case of 13 instructions of the 48 associated with AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. In these instructions additional time is required to perform a series of repetitious operations such as multiplication, which requires repeated addition, division, which requires repeated subtraction, or shifting, which may require any number of repeated shifts. This additional time is supplied by stopping the generation of IP pulses in the instruction control element, which also stops the operation of instruction and machine cycle sequencing. This is called a pause, since the Central Computer System pauses in its usual sequential operation long enough to complete the repetitious operations. An example of an instruction cycle utilizing a pause is depicted by the last diagram (two memory cycle plus pause instruction) of figure 1–15. This is typical of the multiplication and division operations. In the illustration, decoding of the instruction commences at PT–7; the operand, which in this case is either the multiplicand or divisor, is obtained during the subsequent OT–A cycle. At the end of the OT–A cycle, the generation of instruction pulses is stopped and the Central Computer System goes into a pause during which the repetitious additions or subtractions which compose a multiplication or division are executed. The pause condition then ends and the PT cycle begins, completing the instruction by PT–6.

The operations which take place in the Central Computer System during the memory and machine cycles are illustrated in figure 1–16. The times shown should be considered only as approximations, since they vary from instruction to instruction. The time from IP–0 to IP–6 of a PT cycle is used to complete an instruction begun previously. This time interval is also utilized to bring the new instruction out of the core memory element. The new instruction is placed in the memory buffer registers at or before PT–6 and is transferred to the operation, address, and index interval registers at PT–7. Thus, an old instruction is being completed at the same time that the new instruction is being obtained from the core memory element.

3.2 INSTRUCTIONS

Every automatic operation which the Central Computer System performs is executed as a direct result of an instruction. Basically, there are 48 instructions which AN/FSQ–7 (XD–1,–2) Combat Direction Centrals are capable of executing through the Central Computer System. In comparison with other existing digital computers, this is a large number of instructions and the ability of the Central Computer System to execute these instructions contributes greatly to its versatility. A basic knowledge of the manner in which these instructions are executed will provide an excellent understanding of the underlying principles of the Central Computer System and, for this reason, the following text will be devoted to an operational analysis of each instruction. The discussion given herein is not detailed, since a detailed analysis would not be appropriate to the introductory material contained in this part. Further information of a more specialized nature may be found in Part 5 of PH 45–00002 and Part 2 of PH 22–00001.

The 48 instructions of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals are grouped into eight classes. The instructions are grouped in such a manner that all those instructions which consist of similar basic operations are contained in one class. For instance, one of the eight classes is the add class, within which are grouped nine instructions, each of which involves the basic operation of adding the content of the A registers into the accumulator registers. In a similar manner, within the shift class are grouped only instructions which involve the operation of shifting the content of a register, or registers, a given number of places to the left or right. An exception to this method of grouping exists in the case of the miscellaneous class, within which are grouped those instructions which do not fit the other classes particularly well due to various timing or unusual operational considerations.

Before proceeding to an examination of the 48 instructions which AN/FSQ–7 (XD–1,–2) Combat Direction Centrals are capable of executing, it is well to review some of the points which are pertinent to their execution. These points may be summarized as follows:

An instruction word consists of an operation portion and an address portion composed of bits L1 through L15 and RS through R15, respectively. The operation portion contains the binary code of the instruction to be executed and the address portion contains the address of the core memory location where an operand necessary for the execution of the instruction may be obtained. In some cases, the address portion may contain additional data pertinent to the instruction or may even be meaningless.

An instruction is obtained from the core memory element and placed in the memory buffer registers at or before PT–6. During the first half of the PT cycle the Central Computer System is completing the previous instruction. At PT–7, the various portions of the instruction word are

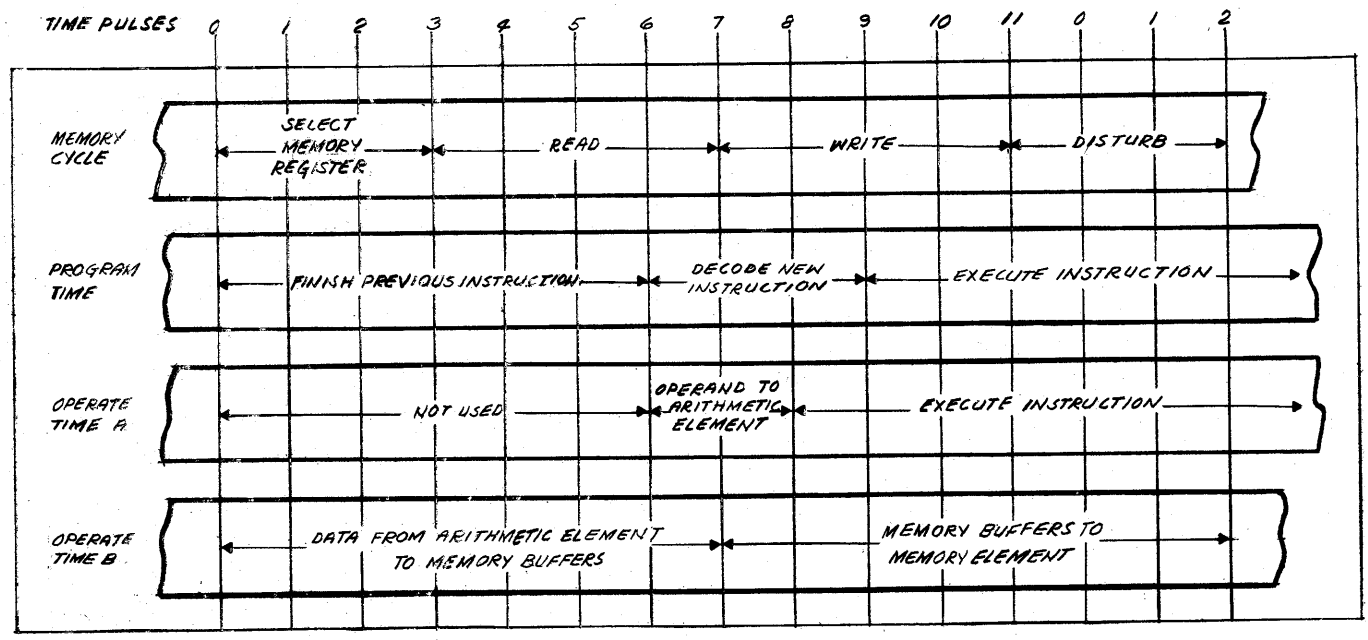


Figure 1–16. Use of Machine Cycles

distributed to their proper registers, shown in figure 1–7 and decoding and execution of the new instruction begins.

When the new instruction is obtained, a 1 is added to the program counter in the program element so that it will continuously contain the core memory address of the next instruction in the program to be executed.

The length of time required to completely execute an instruction is called an instruction cycle and may consist of from one to three machine cycles, depending upon the particular instruction involved. In the case of a few instructions, additional time required to execute repetitious operations is supplied by a pause, during which all other Central Computer System operations are stopped.

During the final PT cycle of an instruction, the content of the program counter is transferred to the memory address register in the core memory element so that the new instruction will be available to the programming circuits and registers by the time the present instruction is completed.

Whenever an instruction obtained from magnetic core memory is to be executed in the Central Computer System, a specific sequence of operations must be initiated. Since this sequence is common to all of the 48 instructions, it is not necessary to repeat it in each of the operation analyses of the instructions to follow. For this reason, the sequence of commands which result in the operations necessary to obtain the instruction from core memory are listed in table 1–7 and are omitted in the discussion of each instruction.

In the discussion of each instruction, both the abbreviation of the instruction and its full name are given. The address portion has a meaning dependent on the instruction with which it is given. If the address refers to a core memory address, the letter x will follow the abbreviation; if the address is a constant, the letter n will be used; if the address is meaningless, two dashes will be used. For example:

a. *Program Stop (HLT)* —

b. *Extract (ETR)* x

c. *Shift Left and Round (SLR)* n

In certain instructions, the index interval (L10 through L15) or certain bits of the index interval may be used. The use of these bits by an instruction will be denoted by a lower-case letter immediately following the instruction abbreviation. Four letters (s, u, o and i) are used. Their significance is:

s: denotes the use of the index interval (L10 through L15) in the *Branch and Index* instruction.

u: denotes the use of the index interval (L10 through L15) for specifying an IO unit, a sense unit, or an operate unit. This will

TABLE 1–7. COMMANDS COMMON TO EVERY INSTRUCTION

TIME	NAME	FUNCTION
PT–0	*Clear memory address register and test memory address register*	Clears registers in anticipation of transfer of core memory address of instruction to be obtained.
PT–1	*Clear memory buffer registers*	Clears memory buffer registers so that they may accept instruction when it is read out of its core memory location.
	Program counter to memory address register	Transfers core memory address of desired instruction to memory address register so that proper core register may be selected; also generates start memory pulse, initiating action of reading desired instruction out of core memory.
PT–6	*Clear address and operation registers*	By this time the instruction will be in the memory buffer registers; the address and operation registers are cleared in anticipation of the transfer of the proper portions of the instruction word to them.
PT–7	*Left memory buffer to operation register* *Right memory buffer to address register*	Transfers the portions of the instruction word from the memory buffer registers to the proper control registers. Decoding and execution of the new instruction begins at this time.
	Add 1 to program counter	Increases content of program counter by 1 so that it contains address of the next instruction in the program to be obtained and executed.

be the case for the *Select, Select Drum, Sense,* and *Operate* instructions.

o: denotes the use of bits L14 and L15 to specify the action the Central Computer System is to take upon the occurrence of an overflow in either accumulator register. This will be the case for only those instructions which may result in an overflow.

i: denotes the use of bits L13 through L15 during the *Read* or *Write* instructions to indicate the method of interleaving on the drum fields.

Examples of the use of the index interval bits notation are:

a. *Branch and Index (BPX)* s x
b. *Add (ADD)* o x
c. *Operate (PER)* u —
d. *Read (RDS)* i n

3.2.1 Miscellaneous Class

Within this class are grouped six instructions which do not fit the other classes particularly well because of eccentricities of timing or operation. The *Program Stop (HLT)* instruction stops the execution of the Central Computer System program and brings all internal operations to a halt. This is accomplished by disabling the TPD of the instruction control element. If an IO operation is in progress (IO interlock flip-flop set) or has been previously programmed and is not yet completed, the execution of the *Program Stop (HLT)* instruction will be delayed until all word transfers into or out of the Central Computer System are effected. The binary code for this instruction consists of all 0's in bits L4 through L10. It is important to note this, since whenever the operation register or a core memory location is cleared or devoid of information, it may be said to contain the *Program Stop (HLT)* instruction.

The *Extract (ETR)* x instruction is used to modify the contents of the accumulator register in accordance with the contents of memory register x. The modification takes the form of setting all bits of the accumulator registers to 0 whose corresponding bits in the memory register are 0; bits in the accumulator registers whose corresponding bits in the memory register are 1 are unaltered. At PT–7 the operation code of the *Extract (ETR)* instruction will be in the operation register and the core memory address of the operand, which describes the manner in which the accumulator registers are to be modified, will be in the address register. During the subsequent OT cycle this address is transferred from the address register to the memory address register, thus selecting the core register in the core memory element within which the operand is stored. By OT–6 the operand will have been read out of the core memory element into the memory buffer registers, from which it is then transferred to the A registers. At OT–9 a logical multiply command pulse is generated by the command generators of the instruction control element; this command performs the desired modification of the words in the accumulator registers. This is accomplished by gating the logical multiply command pulse through the 0 sides of the A register flip-flops into the 0 sides of the accumulator register flip-flops. Thus, all accumulator register bits whose corresponding A register bits contain 0's will be cleared to 0, but all other accumulator register bits will remain unaltered.

The *Operate (PER)* u instruction enables the program to control certain electronic and electromechanical devices in AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. Such devices are called operate units; for example, the card machines associated with the Central Computer System, mechanical tape units, marginal checking controls, scan counter in the Drum System, and display cameras in the Dipslay System. The address portion of the *Operate (PER)* instruction is meaningless, but the index interval (L10 through L15) is used to identify the selected operate unit. (Refer to 2.3.2 of Chapter 2.)

The *Clear and Subtract Word Counter (CSW)* instruction places the 2's complement of the number of words still to be transferred into or out of the Central Computer System during an IO operation in the right accumulator register. This is accomplished by first clearing the right accumulator register and then transferring the content of the IO word counter in the program element to this register. At any time, the IO word counter contains the 2's complement of the number of words to be read or written during an IO operation. (Refer to 2.3.3 of Chapter 2.) Hence, the transfer from IO word counter to right accumulator register by the *Clear and Subtract Word Counter (CSW)* instruction will load the right accumulator register with the 2's complement of the number of untransferred words. The content of the left accumulator register will be unaltered by this instruction but the original content of the right accumulator register will be lost.

The *Shift Left and Round (SLR)* on instruction provides a means by whcih a word of more

than 16 bits, contained in the combined accumulator-B registers, may be properly positioned in these registers and rounded off to 16 bits (including sign). The operation of the *Shift Left and Round (SLR)* instruction may best be exhibited by means of an example. At the end of a division operation as performed by the Central Computer System, as a result of a *Divide (DVD)* or *Twin and Divide (TDV)* instruction, the 16-bit quotients will appear in the B registers, the remainders in bits 1 through 15 of the accumulator registers, and the signs of the quotients will be in the accumulator register S bits. In this case a *Shift Left and Round (SLR)* 15 instruction would be given, where the address portion specifies that the content of the combined accumulator-B registers is to be shifted 15 places to the left. At the end of the PT cycle, during which the instruction is obtained, the Central Computer System goes into an arithmetic pause, during which the 15 repetitious shifting operations are performed. The shifting does not affect the S bits of the accumulators and they will remain unaltered; however, the remainders of the division process, which are in the accumulator registers, will be lost as they are shifted out of the 1 bits of the accumulator registers. At the end of the shifting operations the Central Computer System will resume with a new PT cycle and the quotients will now appear as 17 bits including signs in bits S through 15 of the accumulator registers and the S bit of the B registers. This last bit will be the least-significant digit of the quotient. During this PT cycle the quotients will be rounded off to only 16 bits in the accumulator registers by an examination of the S bits of the B registers. It should be noted that bits R10 through R15, which specify the number of shifts to be executed, are placed in the step counter of the instruction control element. As each shift is completed a 1 is subtracted from this counter. When the content of the step counter is reduced to 0, which will happen when all shifts are completed, the shifting operation is stopped and the Central Computer System will resume its normal operation. Reference should be made to the *Shift Left (DSL)* instruction for further details on the shifting operation.

The *Load B Registers (LDB)* x instruction places the word contained in the core memory location, specified by the address portion, into the B registers of the arithmetic element. During the OT cycle after the *Load B Registers (LDB)* instruction is obtained, the content of the address register is transferred to the memory address register in the core memory element. This will cause the word contained in the core memory location, specified by the address portion of the instruction, to be read out of core memory into the memory buffer registers by OT–6. At OT–7 the word is transferred from the memory buffer registers to the B registers. The prime function of the *Load B Registers (LDB)* instruction is to prepare the Central Computer System for the *Deposit (DEP)* instruction.

3.2.2 Add Class

Within this class are grouped nine instructions which involve the addition of the content of the A registers into the accumulator registers. Since subtraction is accomplished in AN/FSQ–7 (XD–1,–2) Combat Direction Centrals by addition of complements, those instructions which involve an algebraic subtraction are also contained within the add class.

The *Clear and Add (CAD)* x instruction loads the accumulator registers with the content of the core memory register specified by the address portion of the instruction. After the operand, or addend, has been obtained from the core memory element, it is transferred from the memory buffer registers to the A registers. The accumulator registers are then cleared and the command to add the contents of the A registers to the accumulator registers is given. This causes the content of the A registers to be effectively transferred into the accumulator registers, even though the transfer takes place by addition through the adders into the accumulator registers.

The *Add (ADD)* o x instruction causes the content of the core memory register specified by the address portion of the instruction to be added into the accumulator registers. After the execution of the *Add* instruction, the accumulator registers will retain the sum of their original contents plus the addend obtained from the core memory element. This is accomplished in the same manner as for the *Clear and Add (CAD)* instruction, except that the accumulator registers are not cleared before the addition process is begun. Hence, the addition process will add the content of both half-words in the A registers to both half-words in the accumulator registers. The algebraic sums of these half-words appear and remain in the accumulator registers.

The *Twin and Add (TAD)* o x instruction is similar to the *Add* instruction except that the left half-word of the operand, or addend, is not utilized. Instead, the right half-word of the operand is added to the content of both the left and right

accumulator registers. This is accomplished by transferring the content of the right memory buffer register to both the left and right A registers and then initiating the addition process.

The *Add B Registers to Accumulator Registers (ADB)* o x instruction causes the addition of the content of the B registers to the content of the accumulator registers. This is accomplished by transferring the content of the B registers to the A registers and then initiating the addition process. The sum of the original content of the accumulator registers and the content of the B registers is retained in the accumulator registers. No operand is obtained from the core memory element during the execution of this instruction and its address portion is therefore meaningless.

The *Clear and Subtract (CSU)* x instruction loads the accumulator registers with the negative, or 1's complement, of the content of the core memory registers specified by the address portion of the instruction. This is accomplished in the same manner as for the *Clear and Add* instruction except that the operand in the A registers is complemented before the addition process is initiated. At the end of the execution of this instruction, the 1's complement of the content of the specified core memory register is retained in the accumulator registers.

The *Subtract (SUB)* o x instruction causes the content of the memory registers specified by the address portion of the instruction to be subtracted from the content of the accumulator registers and the difference to be stored in the accumulator registers. This is accomplished in the same manner as for the *Add* instruction except that the operand, or subtrahend, in the A registers is complemented before the addition process is initiated. Hence, the addition process adds the 1's complement of the operand, thus subtracting it from the original contents of the accumulator registers. The algebraic difference is retained in the accumulator registers.

The *Twin and Subtract (TSU)* o x instruction is similar to the *Subtract (SUB)* instruction except that the left half-word of the operand, or subtrahend is not utilized. Instead, the right half-word of the operand is subtracted from the contents of both the left and right accumulator registers. This is accomplished by transferring the content of the right memory buffer to both the left and right A registers, complementing the content of the A registers, and then initiating the addition process.

The *Clear and Add Magnitude (CAM)* x instruction loads the positive absolute value of the content of the core memory register specified by the address portion of the instruction into the accumulator registers. Once the operand has been obtained from the core memory element it is transferred to the A registers, where it is complemented if negative; if positive, the operand is unaltered. The accumulator registers are then cleared and the addition process initiated. This causes the contents of the A registers, which is the positive absolute value of the specified core memory register contents, to be effectively loaded into the accumulator registers through the adders.

The *Difference Magnitude (DIM)* x instruction causes the negative absolute value of the content of the core memory register specified by the address portion of the instruction to be added to the positive absolute value of the content of the accumulator registers. This difference is stored in the accumulator. The execution of this instruction is accomplished by complementing the operand in the A registers if positive, complementing the accumulator register content if negative, and initiating the addition process. This causes the differences of the right and left accumulator and A registers to be stored in the accumulator registers.

3.2.3 Multiply Class

Within the multiply class are grouped four instructions which involve repeated addition of the contents of the A registers into the accumulator registers for their execution. Multiplication in the Central Computer System is accomplished by adding the contents of the A registers, the multiplicand, into the accumulator registers whenever there is a 1 in a bit of the multiplier. The content of the accumulator registers is shifted right automatically after each bit of the multiplier is examined and any possible addition is performed. Division in the Central Computer System is accomplished by subtracting the divisor in the A registers from the most-significant part of the dividend in the accumulator registers. As each in a series of subtractions is performed, a 1 is recorded in the B registers for the quotient if the result of the subtraction is positive; a 0 is recorded for the quotient if the result of the subtraction is negative. A detailed discussion of the multiply and divide processes may be found in Part 3.

The *Multiply (MUL)* x instruction multiplies the content of the core memory register specified by the address portion of the instruction by the content of the accumulator registers, and stores the 31-bit products plus sign in the combined accumulator-B registers. Initially, the multiplicand

is obtained from the core memory element and placed in the A registers via the memory buffer registers. The multiplier in the accumulator registers is then transferred to the B registers and a pause is initiated after the current OT cycle. The repetitive additions and shifts which comprise the multiplication process are performed during the pause. At the end of the pause a PT cycle is resumed and the product consists of 31 magnitude bits in the combined accumulator-B registers plus a sign in the sign bit of the accumulator registers. Hence, each half-word of the multiplier and multiplicand produces a complete 32-bit product including sign.

The *Twin and Multiply (TMU)* x instruction is identical in operation to the *Multiply (MUL)* instruction except that the left half-word from the specified core memory register is used as the multiplicand for both half-words of the multiplier in the accumulator registers. This is accomplished by placing this left half-word in both the left and right A registers during the transfer from the memory buffer registers. In the *Twin and Multiply* instruction, the right half-word from the specified core memory register is not utilized, since the left half-word serves as both multiplicands.

The *Divide (DVD)* x instruction divides the contents of the combined accumulator-B registers (31 bits plus sign) by the contents of the core memory register specified by the address portion of the instruction. Prior to the execution of the *Divide* instruction, the dividend must be placed in the combined accumulator-B registers, usually by a preceding *Multiply* or *Twin and Multiply* instructions. After the divisor is placed in the A registers via the memory buffer registers, a pause is instituted and the dividing process is initiated. This divide process is sequenced by the divide time pulse distributor in the instruction control element which times and synchronizes the generation of the commands which perform the division. During the divide process the divisor in the A registers is continuously subtracted from the dividend in the combined accumulator-B registers with a single shift right of the remaining portion of the divisor taking place after each subtraction. As each bit of the quotient is obtained it is placed in the B registers, which are gradually vacated of the dividend by the shift right operations. At the end of the dividing process the quotient appears in the B register as 16 bits and the remainder appears as 15 bits in the accumulator registers. The sign of both the quotient and remainder appears in the sign bit of the accumulator registers. Hence, the division of the two 32-bit divisors in both combined accumulator-B registers by the two half-words from the specified core memory registers produces two quotients and remainders in the two B registers and accumulator registers, respectively.

The *Twin and Divide (TDV)* x instruction is identical in operation to the *Divide* instruction except that the left half-word from the specified core memory register is used as the divisor for both dividends in the combined accumulator-B registers. This is accomplished by placing this left half-word in both the left and right A registers during the transfer from the memory buffer registers. In the *Twin and Divide* instruction, the right half-word from the specified core memory register is not utilized, since the left half-word serves as both divisors.

3.2.4 Store Class

Within the store class are grouped seven instructions, all of which involve the storage of a word from the arithmetic element in the core memory element. Some of these instructions simply store a word in the core memory element while others obtain a word from the core memory element, alter it by specific operations, and store the altered word back in the core memory element.

The *Store (FST)* x instruction stores the 32-bit word in the accumulator registers in the core memory register specified by the address portion of the instruction. This instruction consists of a PT cycle and OT–B cycle. During the OT–B cycle of the *Store* instruction the word in the accumulator registers is transferred to the memory buffer registers. Also, during the OT–B cycle, the address portion of the instruction is transferred from the address register to the memory address register in the core memoy element. This causes the word in the memory buffer registers, originally contained in the accumulator registers, to be stored in the core memory register specified by the address portion of the *Store* instruction.

The *Left Store (LST)* x instruction causes the left half-word in the core memory register specified by the address portion of the instruction to be replaced by the contents of the left accumulator register. This is accomplished by obtaining the word from the specified core memory register during the OT–A cycle and placing it in the A registers. During the subsequent OT–B cycle the contents of the left accumulator register and the right A register are transferred to the memory buffer registers. This causes the original right half-word from the core memory register to

be restored, but replaces the left half-word with the contents of the left accumulator register.

The *Right Store (RST)* x instruction is similar in operation to the *Left Store* instruction except that the content of the right accumulator register replaces the original right half-word in the core memory register. This is accomplished by interchanging the register transfers discussed for the *Left Store* instruction.

The *Store Address (STA)* x instruction replaces the right half-word in the core memory register specified by the address portion of the instruction with the content of the right A register. This is accomplished by first obtaining the word from the specified memory register during the OT–A cycle and storing it in the memory buffer registers. The left A register is then cleared, the content of the left memory buffer register is transferred to the left A register, and the memory buffer registers are cleared. During the subsequent OT–B cycle the new content of the A registers is transfererd to the memory buffer registers and stored in the specified core memory register. As for all OT–A and OT–B cycles, the address portion of the instruction in the address register of the program element serves as the address specification source for the memory address registers of the core memory element.

The *Add One (AOR)* o x instruction adds a binary 1 to the least-significant bit of the right half-word in the core memory register specified by the address portion of the instruction. This is accomplished by obtaining the word from the specified core memory register during the OT–A cycle and placing it in the A registers. By means of the adders, a 1 is then added to the content of the right A register and the sum appears in the right accumulator register. During the subsequent OT–B cycle the altered right half-word is transferred from the right accumulator register to the right memory buffer register and the original unaltered left half-word is transferred from the left A register to the left memory buffer register. The full word is then stored in the specified core memory register.

The *Exchange (ECH)* x instruction exchanges the content of the core memory register specified by the address portion of the instruction and the content of the accumulator registers. This is accomplished by obtaining the word from the specified core memory register during the OT–A cycle and storing it in the A registers. During the subsequent OT–B cycle the contents of the accumulator registers are transferred to the memory buffer registers and stored in the specified core memory register. The contents of the A registers are then added into the cleared accumulator registers through the adders, thus effectively transferring the word from the A registers to the accumulator registers.

The *Deposit (DEP)* x instruction alters the word contained in the core memory register specified by the address portion of the instruction in accordance with the content of the accumulator and B registers. During the execution of this instruction those bits of the core memory register whose corresponding bits of the B registers contain 1 are replaced by the corresponding bits of the accumulator registers. The method by which this alteration of the word in the core memory register is accomplished may be found in Part 2 and Part 3.

3.2.5 Shift Class

Eight instructions are contained within the shift class, each of which involves the shifting of the contents of the accumulator or combined accumulator-B registers to the left or right. (See fig. 1–17.) All of these instructions consist of a single PT cycle and a pause if more than five shifts are to be performed. The number of shifts to be performed during the execution of these instructions is specified by bits R10 through R15 of the instruction word. These six bits are loaded into the step counter, which keeps a count of the number of shifts not yet performed and synchronizes the institution and stoppage of the pause.

The *Shift Left (DSL)* n instruction forms corresponding (left and right) accumulator and B registers into 31-bit shifting registers with the S bit of each B register shifting left into bit 15 of the corresponding accumulator register. The sign bit of each accumulator register is unaffected and the contents of bit 1 of each accumulator register are lost after each shift. At the end of the PT cycle, during which the instruction is obtained, a pause is instituted if more than five shifts are to be performed. During the pause, 2-megacycle pulses are supplied to the arithmetic element, causing the shifts to be performed. When the shifts are completed the succeeding PT cycle is initiated and the instruction is completed.

The *Shift Right (DSR)* n instruction is identical in operation to the *Shift Left* instruction except that the accumulator and B registers are connected so that bit 15 of each accumulator register is shifted right into the sign bit of the corresponding B register. The content of the sign bit of the accumulator registers remains unaltered but is still shifted into the lower-order bits during the

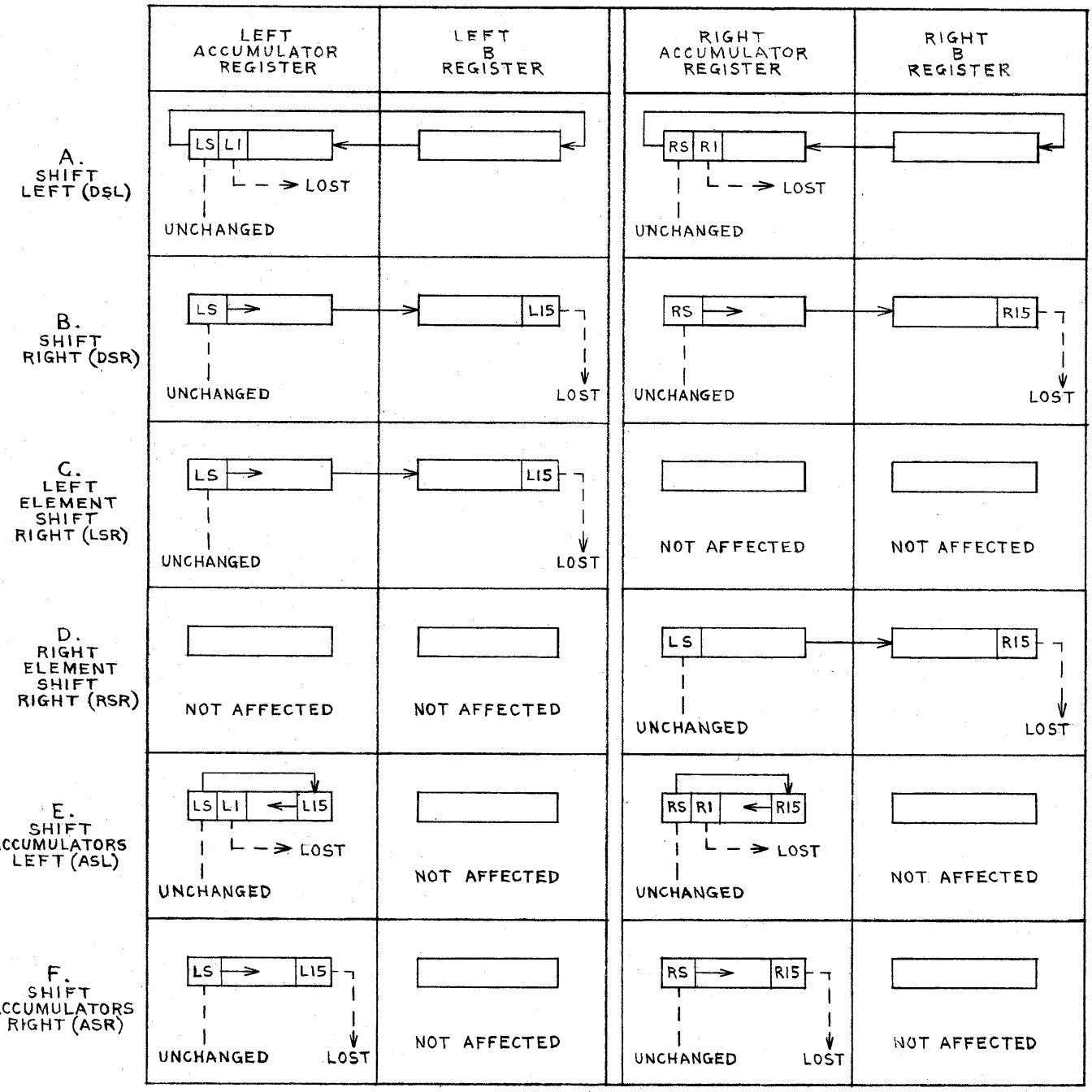


Figure 1–17. Shift Class Register Arrangements, Simplified Block Diagram

shifting operation. Bit 15 of each B register is lost after each shift.

The *Shift Accumulators Left (ASL)* n instruction is identical in operation to the *Shift Left* instruction except for the shifting operation. Each accumulator register is formed into a 15-bit shifting register so that the content of each digit position from 1 through 15 may be shifted into the next left-digit position. The contents of the sign bit of each accumulator register is unaffected, but

the contents of each 1 bit is lost after each shift. Vacated digit positions at the left of each accumulator register are filled with the sign bit of that accumulator register.

The *Shift Accumulators Right (ASR)* n instruction is identical in operation to the *Shift Left* instruction except for the shifting operation. Each accumulator register is formed into a 15-bit shifting register with connections established so that the content of the bit 15 position is lost after each shift and vacated digit positions at the right of each accumulator register are filled by the content of the sign bit, which remains unaltered.

The *Left Element Shift Right (LSR)* n instruction is identical to the *Shift Right* instruction except that only the left accumulator register and left B register are involved in the shifting operation. The right accumulator register and right B register are unaffected by this instruction.

The *Right Element Shift Right (RSR)* n instruction is identical to the *Shift Right* instruction except that only the right accumulator register and right B register are involved in the shifting operation. The left accumulator register and left B register are unaffected by this instruction.

The *Cycle Left (DCL)* n instruction is identical in operation to the *Shift Left* instruction except for the register interconnections established for the shifting operation. Corresponding (left and right) accumulator and B registers are formed into 32-bit shifting registers so that each digit position may be shifted into the next left-digit position. The sign bit of each accumulator register shifts left into bit 15 of the corresponding B register and the sign bit of each B register shifts left into bit 15 of the corresponding accumulator register.

The *Cycle Accumulator Left (FCL)* n instruction is identical in operation to the *Shift Left* instruction except for the register interconnections established for the shifting operation. The left and right accumulator registers are formed into 32-bit shifting registers so that each digit position may be shifted into the next left-digit position. The sign bit of the right accumulator register shifts left into bit 15 of the right accumulator register and the sign bit of the left accumulator register shifts left into bit 15 of the right accumulator register. The B registers are unaffected by the execution of the *Cycle Accumulators Left* instruction.

3.2.6 Branch Class

Within this class are group six instructions, all of which are capable of altering the usual sequential path of the Central Computer System program. In normal operation, the program counter in the instruction control element specifies the address in the core memory element from which the next instruction is to be obtained. However, when certain conditions exist in the equipment it may be desirable to execute a different sequence of instructions stored in a different group of core memory locations in the core memory element. The branch class of instructions provide a means of sensing for various conditions in the Central Computer System which might indicate that the program sequence is to be altered, or that a branch is to be performed. To accomplish this, each branch class instruction senses to determine if a particular condition is present. If the specified condition is absent, the instruction has no effect and the program continues uninterrupted. However, if the specified condition is found to be present, a branch operation is executed. During a branch operation, the contents of the program counter, which are numerically greater by 1 than the core memory address from which the branch class instruction was obtained, are transferred to the right A register in the arithmetic element. The contents of the address register, which specify the core memory address from which the next instruction is to be obtained, are then transferred to the cleared program counter and from there to the memory address registers. This causes the next instruction to be obtained from the core memory address specified by the address portion of the branch class instruction and succeeding instructions to be obtained from sequentially located core memory locations. The original contents of the program counter which are now in the right A register may be preserved in the core memory element by a *Store Address* instruction.

The *Branch and Index (BPX)* s x instruction is used to provide a convenient means of entering and leaving special program iterative loops. (Refer to 3.5 of this Chapter.) During the execution of this instruction the sign bit of the index register specified by bits L1 through L3 of the instruction is sensed. If the content of the specified index register is positive this instruction has no effect and the program continues sequentially. However, if the contents of the specified index register are negative, a special branch and indexing operation takes place. The contents of the specified index register are then reduced

numerically by the contents of bits L10 through L15 of the instruction, which are in the index interval register of the selection and IO control element. In order to accomplish this, the address register is cleared by transfer to the program counter. The contents of the specified index register are then transferred to the cleared address register and the complement of the contents of the index interval register (bits L10 through L15 of the instruction) are transferred to the address register. The index adder of the address register performs the addition of these two numbers and retains the original content of the specified index register less the numerical value of the index interval bits. This number is then transferred back to the specified index register. During this time, the content of the program counter, which is the address portion of the *Branch and Index (BPX)* instruction, is transferred to the memory address registers of the core memory element. As a result, the sequential path of the program is interrupted and the next instruction executed by the Central Computer System is that from the core memory register specified by the address portion of the instruction. It is important to note that the above branching and indexing operations take place only if the original contents of the specified index register are positive; if negative, the *Branch and Index (BPX)* instruction has no effect and the program counter contents are not changed, the branching operation does not take place, and the program sequence continues in normal order.

The *Sense (BSN)* u x instruction enables the Central Computer System to determine the presence or absence of numerous special conditions in the equipment, such as a parity error, overflow alarm, IO unit preparedness, etc. The particular condition whose presence or absence is to be determined is specified by the content of bits L10 through L15 of the instruction, which are stored in the index interval register of the selection and IO control element. If the condition specified by these bits is found to be present (condition met), a branch takes place in the program to the instruction whose core memory address is specified by the address portion of the *Sense* instruction. However, if the condition specified by bits L10 through L15 of the instruction is not present in the equipment, the *Sense (BSN)* instruction has no effect and the Central Computer System program continues sequentially. This instruction utilizes an OT–A cycle to provide additional time for its execution.

The *Branch on Zero (BFZ)* x instruction causes a branch in the program to the instruction contained in the core memory register whose address is specified by the address portion of the instruction if the contents of both left and right accumulator registers are positive or negative zero (0.0000 or 1.1111). In order to provide time for the determination of zero in both accumulator registers, an OT cycle is utilized, although no operand is needed. The method by which zero is determined is discussed in Part 3. If zero is indicated in both accumulator registers, the branch operation takes place.

The *Branch on Minus (BFM)* x instruction examines the sign bits of both accumulator registers in the arithmetic element. If both of these digit positions contain 1, indicating that the words in both left and right accumulator registers are negative, the branch operation takes place.

The *Branch on Left Minus (BLM)* x instruction examines the sign bit of the left accumulator register in the arithmetic element. If this digit position contains a 1, indicating that the word in the left accumulator register is negative, the branch operation takes place.

The *Branch on Right Minus (BRM)* x instruction examines the sign bit of the right accumulator register in the arithmetic element. If this digit position contains a 1, indicating that the word in the right accumulator register is negative, the branch operation takes place.

3.2.7 IO Class

Within the IO class are grouped five instructions, all of which deal with the programming of information transfers between the Central Computer System and other equipment in AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. Every instruction in the IO class is dependent for its execution on the status of the IO interlock in the instruction control element. The IO interlock is turned on whenever the transfer of words into or out of the Central Computer System is initiated, and turned off after all programmed transfers are completed. Since the execution of any IO class instruction interferes with the proper performance of previously programmed IO transfers, an IO class instruction is not executed if the IO interlock is on. Thus, if an IO class instruction appears in the program when the IO interlock is on, its execution and the execution of succeeding instructions is delayed until the IO interlock goes off. If the IO interlock is off when an IO class instruction appears in the program, it is

immediately executed. The delay in program execution which takes place due to an IO class and IO interlock conflict is called an IO pause.

The *Load IO Address Counter (LDC)* n instruction replaces the content of the IO address counter in the program element with the address portion of the instruction. This is accomplished by clearing the IO address counter and transferring the contents of the address register, which are the address portion of the instruction, to the IO address counter. The purpose of this instruction is to load the IO address counter with the first core memory address of a series of core memory locations in the core memory element from which or to which words are to be transferred during succeeding IO operations. This instruction is executed in a single PT cycle.

The *Select Drum (SDR)* u x instruction is used to supply the registers of the Central Computer and Drum Systems with information which is pertinent to impending IO operations. Bits L10 through L15 of the *Select Drum* instruction are used to specify the particular field in the Drum System from which words are to read or upon which words are to be written by the Central Computer System. These bits are stored in the index interval register of the selection and IO control element and are there decoded, causing the control circuits of this element to be set up in the proper manner for the control of the IO operations. These bits are also transferred to the drum selection register in the Drum System and are there decoded, causing the energizing of the read-write heads of the selected field. The address portion of this instruction may contain useful information or be meaningless, depending upon the particular field of the Drum System specified by the index interval register content. In any case, these bits are transferred from the address register to the drum control register of the program element. During IO operations, the address portion of the *Select Drum (SDR)* instruction, which is in the drum control register, is used in conjunction with the right IO buffer register to identify words, or their location on the field, transferred during IO operations. (Refer to 2.3.2 of Chapter 2.) In order to provide time for the decoding and transfer operations, an OT–A cycle is utilized during the execution of this instruction, although no operands are needed from the core memory element.

The *Select (SEL)* u instruction is used to specify which IO unit is to be utilized in conjunction with the Central Computer System during impending IO operations. Bits L10 through L15 of this instruction specify the particular IO unit involved with the Central Computer System in information transfers. These bits are stored in the index interval register of the selection and IO control element and are there decoded, causing the control circuits of this element to be set up in the proper manner for the control of information transfers with the selected (specified) IO unit. The address portion of the *Select (SEL)* instruction is meaningless. In order to provide time for the decoding of bits L10 through L15, an OT–A cycle is utilized, although no operands are needed from the core memory element.

The *Read (RDS)* i n instruction is used to initiate the transfer of words betwen a previously selected IO unit and the Central Computer System. The address portion of this instruction specifies the number of words to be transferred during IO operations from the IO unit to the Central Computer System. This number is loaded into the IO word counter in the program element by transferring the contents of the address register to the IO word counter. Bits L13, L14, and L15 are used to specify the interleaving method to be carried out if an addressable drum is involved in the IO operations. (Refer to 3.3.1 of this Chapter.) The Read instruction causes the IO word transfer to be initiated by channelling a command pulse through the control circuits of the selection and IO control element so that it appears as a start read signal which is sent to the previously selected IO unit. This instruction also sets the IO interlock, indicating that an IO operation is in progress.

The *Write (WRT)* i n instruction is identical in operation to the *Read* instruction except that it causes the generation of a start write signal by the selection and IO control element; this signal is sent to the selected IO unit.

3.2.8 Reset Class

Within the reset class are grouped three instructions, all of which involve the loading of the index registers of the program element with a specific numerical value. These instructions are used to set up the Central Computer System for the performance of a series of iterative loops in the program. Each of the three instructions is completely executed in a single PT cycle.

The *Reset Index Register (XIN)* n instruction causes the content of the index register specified by bits L1 through L3 of the instruction (except for the right accumulator register, for which this instruction is not valid) to be replaced by the address portion of the instruction. This is accomplished by transferring the contents

of the address register to one of the two index registers in the program element.

The *Reset Index Register from Right Accumulator (XAC)* instruction causes the contents of the index register specified by bits L1 through L3 of the instruction to be replaced by the contents of the right accumulator register. This is accomplished by first transferring the contents of the right accumulator register in the arithmetic element to the address register in the program element, and then transferring the contents of the address register to the specified index register.

The *Add Index Register (ADX)* n instruction causes the right A register to be loaded with the sum of the contents of the address portion of the instruction and the contents of the index register specified by bits L1 through L3 of the instruction. This is accomplished by transferring the contents of the specified index register to the address register. The adder circuits of the address register will cause the address register to retain the sum of its original contents (address portion of the instruction) and the contents of the specified index register. The right A register is then cleared and the new contents of the address register are transferred to the right A register. If no index register is specified by bits L1 through L3 of this instruction, the address portion of the instruction is transferred unaltered from the address register to the right A register.

3.3 IO OPERATIONS

An input-output (IO) operation may be defined as the process of transferring a word or words between the core memory element of the Central Computer System and some storage device external to the Central Computer System, called an IO unit. If information is being transferred from the IO unit into the Central Computer System for storage in the core memory element, the process may be further defined as a read, or break-in operation; if information is being transferred to the IO unit from the core memory element, the process may be defined as a write, or break-out operation. Before any operations may be instituted in the Central Computer System, certain basic information must be stored in core memory; i.e., a program, and the numerical data with which the calculations specified by the program are to be performed. This basic information is stored, or loaded into, core memory from an IO unit by a break-in operation. The Central Computer System may then proceed to execute the program with which it has been supplied, perform its calculations, and store the results in core memory. These results are of little value while in the Central Computer System and must be transferred to an IO unit where they can be properly utilized. This transfer is accomplished by a break-out operation.

The institution of any IO operation requires the programming of three instructions; two of these are preparatory and the third is an instruction of execution. The two preparatory instructions are *Load IO Address Counter (LDC)* and *Select (SEL)* or *Select Drum (SDR)*. The *Load IO Address Counter* instruction places into the IO address counter the first location address in magnetic core memory to be involved in subsequent IO operations. The *Select* or *Select Drum* instruction specifies, by means of its index interval (L10 through L15), the IO unit or drum field with which the IO operation is to take place, and also sets up the proper control circuits in the selection and IO control element. The third instruction may be either a *Read (RDS)* or *Write (WRT)* instruction and specifies whether a break-in or break-out operation is to be performed, respectively. The *Read* or *Write* instructions will also cause a start read or start write command pulse to be sent to the selected IO unit, notifying it that it has been selected to be involved in an IO operation. The *Read* or *Write* instruction also sets both IO interlock flip-flops in the instruction control and selection and IO control elements. This provides an indication that an IO operation has been programmed.

When the selected IO unit requires access to core memory in order to transfer a word in or out of the Central Computer System, it requests a break in normal Central Computer System operation. At the end of the current machine cycle during which the break is requested, the Central Computer System ceases operations and all arithmetic and logical processes are brought to a halt for 6.0 microseconds. During this time, which is called a break cycle, the core memory element executes one memory cycle and one word will be either stored in core memory from the IO unit or transferred to the IO unit from core memory. Every transfer of a word into or out of the Central Computer System is executed as a direct result of a break request by the selected IO unit. Since the IO units have a slower data-handling speed than the Central Computer System, break requests come at time intervals greater than a machine cycle. The time between break cycles (as instituted by break requests) is utilized by the Central Computer System to execute its program. Thus, at the end of a current break cycle, the

Central Computer System remains in the IO interlock on state, during which it may perform operations or execute instructions other than those of the IO class or the *Program Stop (HLT)* instruction. The number of words to be transferred between the Central Computer System and the IO unit selected by the *Select* or *Select Drum* instruction is specified by the address portion of the *Read* or *Write* instruction. This number is loaded into the IO word counter and a 1 is subtracted from it as each word is transferred. When the IO word counter content has ben reduced to positive zero, the IO operation is complete, the IO interlock is cleared, and the selected IO unit is disconnected.

One of the prime reasons for the versatility of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals is the unique design and degree of size of the IO equipment. This consists of 50 IO units, which fall into four basic categories. (See table 1–8.) For design reasons, the IO register has been included in the IO units although it does not really exist as a distinct entity. The selection of the IO register provides a convenient means of clearing locations in core memory by writing 0's into the core register to be cleared during a break-in operation. Since the IO register is a highly specialized device, or actually more of an operation, it will not be discussed here and is mentioned only for completeness of discussion.

For the same reasons, the warning lights will be omitted from the discussion to follow, since they are also highly specialized units used to notify personnel at the various consoles in AN/FSQ–7 (XD–1,–2) Combat Direction Centrals of certain Central Computer System conditions or decisions.

3.3.1 Break-In (Read) Operations

A break-in operation involves the transfer of information from a selected IO unit to the Central Computer System for storage in core memory. In general, a break-in operation must be programmed by the following series of instructions:

a. *Load IO Address Counter (LDC)* x
b. *Select or Select Drum (SEL)* u — or *(SDR)* u x
c. *Read (RDS)* n

The *LDC* x instruction replaces the content of the IO address counter in the program element with its address portion specifying the starting address in core memory in which the first word transferred from the selected IO unit is to be stored. The *SEL* u instruction specifies, by its index interval, the IO unit other than a drum field which is to be involved in the break-in operation and also sets up the proper control circuits in the selection and IO control element. If a drum field is to be selected, the *SDR* u x instruction is utilized. Its index interval specifies the particular drum field from which the words to be read are to be

TABLE 1–8. IO UNIT CHARACTERISTICS

IO UNIT	READ	WRITE	REMARKS
Drum fields	Yes	Yes	Reading or writing ability is a property of the field selected; there are 39 separate fields in the Drum System.
Magnetic tape units	Yes	Yes	There are four magnetic tape units, each of which is separately selected.
Card Machines:			
Reader	Yes	No	Reads from IBM punched cards.
Printer	No	Yes	Writes alphanumerically on paper.
Punch	No	Yes	Writes on punched cards.
Manual input switch	Yes	No	Located in Input System.
Burst time counters	Yes	No	There are three burst time counters with a provision for a maximum of eight. These are selected as one unit and are read sequentially until the number of words specified by the *Read* instruction are received. The burst time counters are located in the Output System.
Warning Light System	No	Yes	Writes data into warning light registers whose contents are displayed at the various consoles in AN/FSQ–7(XD–1,–2) Combat Direction Centrals.
IO register	Yes	No	Reads 0's into magnetic core memory for purpose of clearing specified core memory locations.

obtained, and its address portion specifies the address of the register on the field containing the first desired word. This drum address may also be in the form of special identification bits whch identify the word be read from the drum. In either case, the *SDR* instruction transfers its address portion to the drum control register. The final instruction is *RDS* n, where the address portion specifies the number of words to be read from the selected IO unit into the Central Computer System. The complement of this number is transferred to the IO word counter from the address register. In addition, the *RDS* instruction sets up control circuits in the selection and IO control element, sets the IO interlock to indicate that an IO operation has been programmed, and will activate the selected IO unit by sending it a start read signal.

The Central Computer System may now proceed, in the IO interlock on condition, to execute its program in normal fashion. However, if an IO class or *Program Stop (HLT)* instruction appears, the Central Computer System delays its execution by going into a pause until all IO transfers are completed. When the selected IO unit is ready, it transfers its word to the IO registers or IO buffer registers of the Central Computer System and requests a break in the program by sending a break request pulse to the selection and IO control element. At the end of the present machine cycle, the Central Computer System stops all internal operations and executes one break cycle, during which it transfers the word received from the IO unit into core memory.

The core memory location in which the word is stored is specified by the content of the IO address counter, which is transferred to the memory address register of the core memory element at the beginning of the break cycle. At this time the content of both the IO address and word counters are increased by 1. The addition of 1 to the IO address counter provides the method by which successive words received from the selected IO unit are stored in sequentially located core memory locations. Since the IO word counter contains the complement of the number of words to be read, an addition of 1 will effectively reduce its content by 1. Thus, a constant count of the number of words remaining to be transferred is obtained. When the break cycle is completed, the Central Computer System resumes its normal program execution until another break request is received. It then executes a new break-in cycle and stores the next word in core memory. A break cycle is executed as each word is received from the selected IO unit, during which the word is stored in core memory and a 1 is added to the IO address and word counters. When a number of words equal to the number specified by the *RDS* n instruction have been received, the contents of the IO word counter are reduced to positive zero, resulting in the generation of an end-carry pulse. This pulse is sent to the selection and IO control element, notifying it that all necessary word transfers have been completed. The IO unit is then disconnected and the IO interlock cleared; the IO operation of reading a number of words from a selected IO unit is complete and the Central Computer System proceeds with its program.

Perhaps the simplest break-in IO operation is that of reading words from the magnetic tape units. There are four magnetic tape units, providing unlimited storage capacity for special programs. Associated with each word read from the tape units is a parity bit, which is transferred along with the 32 bits of the word coming from the magnetic tape. This parity bit is employed to check the accuracy of the transfer from tape units to Central Computer System. Figure 1–18 shows the circuits and transfers associated with reading from the magnetic tape units.

The operation of reading from a magnetic tape unit is programmed by the usual three IO class instructions. The *LDC* instruction loads the IO address counter with the address of the core memory location in which the first word read from the selected magnetic tape is to be stored. The *SEL* instruction has an index inerval content specifying which of the four magnetic tape units, numbered from 1 through 4, is to be read, and sets up the proper control circuits in the selection and IO control element. The *RDS* n instruction sends a start read signal to the magnetic tape units and loads the IO address counter with its address portion specifying the number of words to be read from the selected tape unit. This instruction also turns the IO interlock on, thus preventing the Central Computer System from executing any instruction (IO class or *(HLT)* which might interfere with the IO operation. During the intervening time between the end of the *RDS* instruction and the time the tape unit is prepared to transfer its first word, the Central Computer System proceeds with its normal program. When the first word has been transferred from the tape word register in the tape unit to the IO registers in the arithmetic element, the tape units send a break request pulse to the selection and IO control element. At the end of its current machine cycle, the Central Computer System stops all arithmetic

operations and executes one break-in cycle, during which the following command pulses are generated and executed:

a. IO address counter to memory address register
b. Inhibit sample
c. Add 1 to IO address and word counters
d. IO registers to memory buffer registers

The transfer at A selects and activates the memory register in the core memory element in which the word from the tape unit is to be stored. The inhibit sample pulse informs core memory that a word is to be written into a core register. The stepping of the IO address and word counters computes the core memory address in which the next word from the tape unit is to be stored and maintains a count of the number of words remaining to be transferred, respectively. The transfer at d places the word from the tape unit in the memory buffer registers so that it can be stored in the selected core memory register. By the time that the break cycle is completed, the word is retained in the proper core register in the core

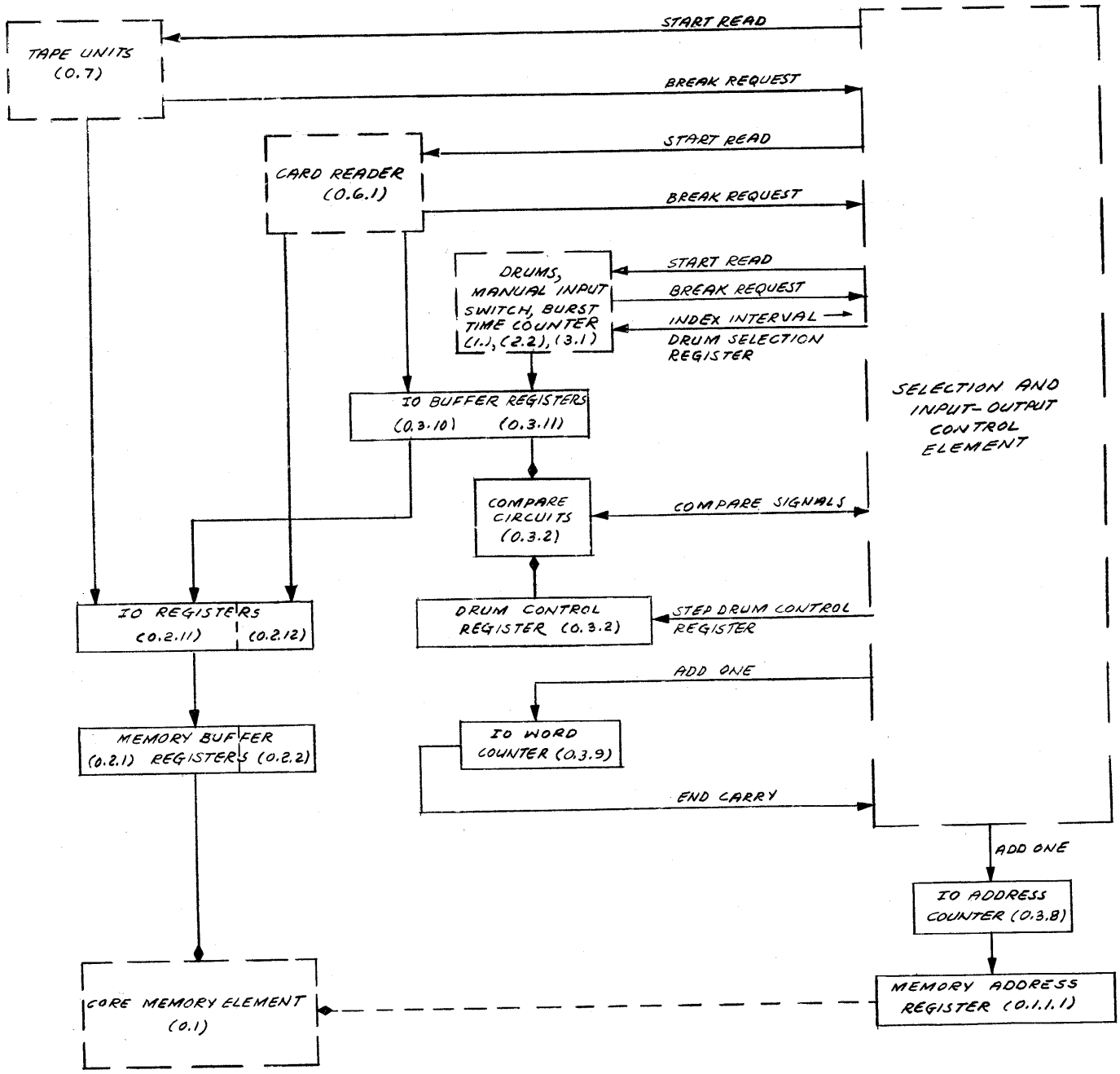


Figure 1–18. Break-In Operation Flow, Simplified Block Diagram

memory element, and the Central Computer System then proceeds with its program until a new break request is received from the tape unit. The second word is then transferred from the IO registers through the memory buffer registers into the core memory location specified by the content of the IO address counter, and the IO address and word counters are stepped. Assuming that the *RDS* instruction specified that five words be read and the *LDC* instruction specified the starting core memory address as octal 2000, five words are read from the selected tape unit and stored in octal memory addresses 2000 through 2004. When the fifth word is transferred, the IO word counter will be stepped to positive zero and an end-carry pulse is generated. This end-carry pulse indicates that the necessary number of words have been read; it is sent to the selection and IO control element, setting up the proper circuits. This causes the IO interlock to be cleared and the tape unit to be disconnected; the IO operation is complete and the Central Computer System then proceeds unconditionally with its program.

The IO operation of reading the words on an IBM punched card by means of the card reader is made complex by the fact the card reader makes two words simultaneously available to the Central Computer System. Each punch card is so arranged that as it passes under the read brushes of the card reader, the 24 words contained on one card are transferred in pairs of two to the IO buffer and IO registers of the Central Computer System. For this reason, the operation of reading from the card reader is different from that of reading from a magnetic tape unit. In general, the standard programmed sequence of instructions necessary to initiate reading is: *LDC, SEL* (card reader), *RDS*. When the *RDS* instruction is executed, a start read signal is sent to the card reader, which causes it to begin feeding the first punch card under the read brushes. In terms of Central Computer System speed, this takes a rather long time and the Central Computer System meanwhile continues to execute its program until a break request is received. When the first two words on the card pass under the read brushes they are transferred to the IO buffer registers and IO registers of the arithmetic element and a break request pulse is sent from the card reader to the selection and IO control element. At the end of its current machine cycle, the Central Computer System stops all arithmetic operations and executes a break-in cycle. The operations which take place during this break-in cycle are:

a. IO address counter to memory address register
b. Inhibit sample
c. Add 1 to IO address and word counters
d. IO registers to memory buffer registers
e. IO buffer registers to IO registers.

The first four operations are identical to the action of storing a word contained in the IO registers in a core memory register. The operation at e acts only to replace the content of the IO registers with the content of the IO buffer registers, which is the remaining word received from the card reader but not yet stored in core memory. After the first break-in cycle is completed, the Central Computer System will immediately execute a second break-in cycle, during which the second word from the card reader, now contained in the IO registers, will be stored in core memory. The operations which take place during this break-in cycle are the same as a through d directly above. Thus, as a result of a single break request from the card reader, the Central Computer System executes two successive break-in cycles and stores the two words received from the card reader in the core memory element. At the end of these two break-in cycles, the content of the IO address counter is greater by two and the content of the IO word counter is less by two.

When the number of words specified by the *RDS* instruction have been stored, the IO word counter is stepped to positive zero and an end-carry is generated. This end-carry causes the IO interlock, which was turned on by the *RDS* instruction, to be cleared, and disconnects the card reader; the IO operation is then complete and the Central Computer System proceeds unconditionally with its program.

The most important and extensive source of data for the Central Computer System is the Drum System. In the operation of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals, the Drum System performs two major functions; it serves as an intermediary between the Central Computer and the Input, Output, and Display Systems, and as an auxiliary memory for the Central Computer System subprogramming data. The former function is represented in figure 1–1. AN/FSQ–7 (XD–1,–2) Combat Direction Centrals equipment is concerned essentially with the problem of processing target data associatd with a number of aircraft detection and search devices. However, the data from these devices must first be put in

a form suitable for use within the Central Computer System. This is accomplished by the Input and Drum Systems; the Input System controls the flow of data from radar sites and other centrals and stores the data on the fields of the Drum System. The Drum System thus stores the slowly recurring tactical data from the Input System and makes it available to the Central Computer System by means of a break-in IO operation at a rate more compatible with the swift operation of the latter. The auxiliary memory fields of the Drum System provide additional storage space for Central Computer System programs, arithmetic tables, subroutines, etc. Thus, these fields provide storage space equal to three times the capacity of core memory. The data on these fields may be transferred to the Central Computer System at any time by a break-in operation.

Reading from the Drum System into the Central Computer System is complicated by the fact that it may be accomplished in any one of three different methods or modes. These are the status, address, and identity modes. In general, each of the fields in the Drum System is read by one particular mode. Thus, the selection of a field from which data is to be read into the Central Computer System automatically determines the mode of operation. An exception to this rule occurs when the Drum and Central Computer Systems are being tested. In this case, a field may be read in a mode dissimilar from its normal mode. However, this case will not be considered in this discussion.

More fields of the Drum System are read in the address mode than by any of the other two modes. The break-in operation of reading from an address-controlled field is programmed by three instructions: *LDC* x, *SDR* u x, and *RDS* i n. The *Load IO Address Counter (LDC)* instruction specifies the starting address in core memory in which the first word read from the selected field is to be stored. The *Select Drum (SDR)* instruction performs three functions. Its index interval specifies which of the fields in the Drum System is to be read and these index interval bits are transferred from the index interval register in the selection and IO control element to the drum selection register in the Drum System. This is done so that the proper field may be activated for reading. The second function of the *SDR* instruction is to specify the address of the register on the field from which the first word is to be read. This is accomplished by bits R4 through R15 of its address portion (2.3.3 of Chapter 2), which are placed in the drum control register of the program element. The third function of this instruction is to set up the proper control circuits in the selection and IO control element so that the reading of the selected field may be properly performed. The *Read (RDS)* instruction sends a start read signal to the Drum System and loads the IO word counter with the number of words to be read from the selected field. Also, bits L13 through L15 specify the mode of interleaving to be executed when reading from the field, if interleaving is desired.

When all three of these instructions have been executed, those registers of the Central Computer System which are involved in the break-in operation contain the following data. That is, the IO word counter contains the number of words to be read from the field; the IO address counter contains the address of the location in core memory in which the first word read is to be stored; the drum control register contains the address of the register on the selected field from which the first word is to be read. After the start read signal is sent to the Drum System, there is a 120-microsecond delay, during which the control and reading circuits in the Drum System are set up and stabilized. This time is utilized by the Central Computer System to continue with its program. After the 120-microsecond delay, the Drum System transfers the contents of the angular position counter (APC) to the right IO buffer register. These contents of the angular position counter in the Drum System are the address of the register on the selected field which is under the read head. A compare pulse (CD–1) is then received from the Drum System; this pulse initiates comparison of the right IO buffer content with that of the drum control register. If the comparison is not successful, a no-compare pulse is generated; this pulse is sent to the Drum System and initiates the loading of the next address on the field to come under the read heads from the angular position counter into the right IO buffer register. This process of loading a drum register address into the right IO buffer register and comparing the address with the address of the desired register in the drum control register continues until a successful comparison is accomplished. At this time, the no-compare pulse is not generated and the Drum System transfers the word contained in the register whose address was successfully compared into the IO buffer register. At the same time, a break request pulse is sent to the selection and IO control element. This break request, which in the case of the Drum System is called an IO buffer

load pulse, transfers the content of the IO buffer registers to the IO registers and requests a break-in cycle.

When a break-in cycle is obtained at the end of the current machine cycle, the word received from the selected field, which is now in the IO registers, is transferred to the memory buffer registers and stored in the core memory location specified by the content of the IO address counter. The contents of the IO address and word counters are also increased by 1. After the first successful comparison and storage of the first word in core memory, the Drum System transfers words on succeeding field registers to the IO buffer registers with no prior comparison of addresses. As each word is received from the Drum System, the Central Computer System executes break-in cycles and stores the word in core memory. Thus, if the sequence of instructions which institutes the reading of a field is:

a. *Load IO Address Counter* (with octal 2000) (LDC) $(2000)_8$
b. *Select Drum* (address 1500) *(SDR)* u $(1500)_8$
c. *Read* (5 words) $(RDS)_5$.....

the contents of the angular position counter are continuously loaded into the right IO buffer register and compared with the octal 1500 in the drum control register. When the two addresses compare, the contents of field register $(1500)_8$ are transferred from the selected field to the IO buffer registers and stored by a break-in cycle in address $(2000)_8$ of the core memory element. Immediately following this operation, the Drum System sequentially transfers the words in field registers $(1501)_8$ through $(1504)_8$ to the Central Computer System and they are stored in addresses $(2001)_8$ through $(2004)_8$ of the core memory element. When the fifth word is transferred and stored, which is the word from field register $(1504)_8$ stored in core memory address $(2004)_8$, the IO word counter is stepped to positive zero and an end-carry pulse generated. This end-carry pulse is sent to the selection and IO control element where it prevents the execution of any more break-in cycles and causes a disconnect pulse to be sent to the Drum System. The IO interlock, which was set by the *RDS* instruction, is also cleared, and the Central Computer System proceeds to execute its program unconditionally, since the break-in IO operation has been completed.

A modification of this manner of reading from an address-controlled field may be performed by a method known as interleaving. When reading by address and interleaving, the Central Computer System reads only words contained in field registers whose addresses differ by a given number, instead of reading words on sequentially addressed field registers. The three interleaving modes on address-controlled fields are:

a. Reading every 8th register on the field
b. Reading every 16th register on the field
c. Reading every 64th register on the field

The mode of interleaving is specified by bits L13 through L15 of the *RDS* instruction. The operation of interleaving may best be described by comparison with the IO operation just discussed. In the previous example, five words were read from an address-controlled field without interleaving. The starting field register address was specified as $(1500)_8$ and the five words were read from field addresses $(1500)_8$ through $(1504)_8$. If bits L13 through L15 of the *RDS* instruction have specified that the mode of interleaving is to be by eight, the first word would have been read from field address $(1500)_8$, since this was specified as the starting field address by the *SDR* instruction. However, successive words would have been read from field addresses $(1508)_8$, $(1516)_8$, $(1524)_8$, and $(1532)_8$. Thus, when interleaving by eight, the Central Computer System reads a word, skips seven registers, reads a second word, skips seven more registers, etc. In a similar manner, the Central Computer System skips 15 or 63 registers when interleaving by 16 or 64, respectively.

Interleaving when reading from an address-controlled field is accomplished by the drum control register. The drum control register is incorporated with special circuits which allow its content to be increased by 8, 16, or 64. (Refer to 2.3.3 of Chapter 2.) The operation of reading from the selected field is the same when interleaving as it is when no interleaving is used, until the time of the first successful comparison between the drum control register and right IO buffer register. When the two addresses compare, the Drum System transfers the word in the successfully compared address to the IO buffer registers and generates an IO buffer load pulse. The Central Computer System then transfers the word through the IO registers into the memory buffer registers and stores it in core memory. At this time, the Central Computer System also generates a dummy no-compare pulse which is sent to the Drum System and prevents it from transferring the words in succeeding registers. Further, the Central Computer System also steps the drum control register at this time. This stepping of the drum control

register is such as to add either 8, 16, or 64 to its original content, depending upon the mode of interleaving specified by the *RDS* instruction. The Drum System then proceeds to load the contents of the angular position counter specifying the address of the field register under the read heads into the right IO buffer register. The Central Computer System compares this address with the new content of the drum control register. When a successful comparison is accomplished, the word in the field register whose address compared properly is transferred into the Central Computer System and stored in core memory. At this time, the drum control register is again stepped and the comparison of the content of the drum control register with the content of the right IO buffer register is re-initiated. These operations of comparison, transfer, storage of a word, and stepping of the drum control register continue until the IO word counter is brought to positive zero and an end-carry pulse is generated. The IO operation of reading from the interleaved registers of a selected field is then complete and the Drum System will be disconnected and the IO interlock is cleared.

Reading from a field in the Drum System which is identification-controlled is similar to reading an address-controlled field. However, in this case words to be read are not located by their address on the field but by special indentification bits contained in the words themselves. The address portion of the *Select Drum* instruction contains these identification bits which are loaded into the drum control register. The bits used for identification may be either RS through R10, R11 through R15, or R14 and R15, depending upon which field of the Drum System is selected. The execution of the *RDS* instruction generates a start read signal which is sent to the Drum System and causes it to load each word which passes under the read heads into the IO buffer registers. The identification bits in the drum control register are then compared with the corresponding bits in the word from the selected field in the IO buffer registers. If the identification bits do not compare the Central Computer System does not accept the word. However, if the identification bits do compare, a break-in cycle is executed, during which the word is transferred from the IO buffer registers and stored in core memory. The Drum System then continues to load words from the registers of the selected field into the IO buffer registers and initiates the comparison operation. Thus, all of those words which pass under the read heads and whose identification bits compare with the identification bits in the drum control register are accepted by the Central Computer System and stored in core memory. The break-in operation of reading from the identity-controlled field may be halted by the IO word counter being stepped to positive zero and generating an end-carry pulse, or by a disconnect pulse from the Drum System, which occurs when all of the 2048 registers on the field have been read and compared. In either case, the IO interlock is cleared and the Central Computer System proceeds unconditionally with its program.

The third method by which data may be read from a field into the Central Computer System is by status control. Associated with each field in the Drum System which is status-controlled are two status channels, the read control status channel and the write control status channel. Since this discussion deals with reading from a field into the Central Computer System, only the former channel will be considered. Each of the 2048 words on a status-controlled field has a corresponding bit in the read control status channel. If this bit is a 0, it indicates that its corresponding register on the field is empty so far as the Central Computer System is concerned. If this bit is a 1, it indicates that the correspondinig register contains a word which has not been previously read by the Central Computer System. Thus, the status channel indicates the status of the registers on the field; i.e., full or empty. When the Central Computer System selects a status-controlled field to be read, the Drum System reads the bits in the read control status channel. When it reads a 1, the Drum System transfers the word contained in the corresponding field register to the IO buffer registers and generates an IO buffer load pulse and compare (CD–1) pulse. The Central Computer System suppresses the comparison operation which is normally instituted by the compare pulse, and a no-compare pulse is not returned to the Drum System. However, the IO buffer load pulse causes a break-in cycle to be instituted during which the word read from the selected field is stored in core memory.

The Drum System continues to transfer every word on the field whose status bit is 1 to the Central Computer System. As each word is read and transferred, an IO buffer load pulse is also generated which causes the word to be stored in core memory. However, since a no-compare pulse was not sent to the Drum System when the first word was transferred, the Drum System does not generate any further compare pulses. Thus, each word

on the selected field whose status bit is 1 is transferred to the Central Computer System and stored in core memory. In addition, a 0 is written over the 1 on the read status control channel as each word is read into the Central Computer System. This indicates that the word has been previously read by the Central Computer System and prevents reading a word twice. It can readily be seen that when reading a status-controlled field, the Central Computer System reads and stores the content of the full registers (as indicated by a 1 in the status channel) but skips all empty registers (as indicated by a 0 in the status channel). The operation is stopped whenever the IO word counter is reduced to positive zero or the status of the 2048 registers on the field has been determined, whichever occurs first. At this time, the IO interlock is cleared and the Drm System is disconnected.

The operations of reading from the manual input switch in the Input System and the burst time counters in the Output System are identical except for the **Select** instruction and are therefore discussed together. When the *Read* instruction is executed, a start read signal is sent to the selected unit. The Central Computer System then continues with its program until the selected unit transfers the first word to the IO buffer registers and generates a break request pulse. The break request pulse transfers the word from the IO buffer registers to the IO registers and requests a break in the Central Computer System program. At the end of the current machine cycle, the Central Computer System halts all internal operations and institutes a break-in cycle, during which the word is transferred from the IO registers through the memory buffer registers and is stored in the location in core memory specified by the content of the IO address counter. While the break-in cycle is being executed, the IO address and word counters are stepped. When the word has been stored, the Central Computer System resumes its program until another word and break request are received from the selected IO unit (manual input switch or burst time counters). After each break request, the received word is stored in core memory during a break-in cycle. Between break-in cycles, the Central Computer System proceeds with its program. When a number of words equal to the number specified by the *Read* instruction have been transferred and stored, the IO word counter is stepped to positive zero, resulting in the generation of an end-carry pulse which causes the IO interlock to be cleared and a disconnect signal to be sent to the selected IO unit. The IO operation of reading from the manual input switch or the burst time counter is then completed and the Central Computer System continues unconditionally with its program.

3.3.2 Break-Out (Write) Operations

A break-out operation involves the transfer of information from the core memory element of the Central Compter System to a selected IO unit. In general, a break-out operation must be programmed by the following series of instructions:

a. *Load IO Address Counter (LDC)* x
b. *Select* or *Select Drum (SEL)* u—or *(SDR)* u x
c. *Write (WRT)* n

The address portion of the *Load IO Address Counter* instruction specifies the address in core memory of the first word to be transferred to the selected IO unit. This address is loaded into the IO address counter in the program element and the succeeding words transferred during the IO operation are obtained from sequentially addressed core memory locations. The *Select* or *Select Drum* instruction performs the same function as in the programming of a break-in operation; i.e., specifying the IO unit or drum field and field register address, respectively. The instruction of execution is the *Write* instruction, whose address portion specifies the number of words to be transferred from core memory to the selected IO unit. This number is loaded into the IO word counter in 1's complement form. The *Write* instruction also sets up the proper control circuits in the selection and IO control element, sets the IO interlock to indicate that an IO operation has been programmed, and activates the selected IO unit by sending it a start write signal.

Perhaps the simplest operation of writing words from the Central Computer System involves writing on a magnetic tape unit. This break-out operation is programmed by the *Load IO Address Counter, Select* (magnetic tape unit) and *Write* instructions. The *Select* instruction sets up the proper control circuits in the selection and IO control element and sends a select signal to the tape units specifying which of the four magnetic tape units is to be written upon. The *Write* instruction sends a start write signal to the tape units which starts the tape into motion. This instruction also sets the IO interlocks, after which the Central Computer System continues with its program in an IO interlock on condition.

The equipment of the Central Computer System which is associated with writing on the magnetic tape units is shown in figure 1–19. When the

magnetic tape unit is ready to write a word, it generates a break request signal which is sent to the selection and IO control element. At the end of its current machine cycle, the Central Computer System institutes a break-out cycle, during which the following command pulses are generated and executed:

a. IO address counter to memory address register
b. Add 1 to IO address and word counters
c. Memory buffer registers to IO registers
d. IO registers to tape word register

The transfer at A causes the word contained in the location of core memory specified by the content of the IO address counter to be read out into the memory buffer registers. The stepping of the IO address and word counters computes the core memory address from which the next word to be written on the tape unit is obtained, and maintains a count of the number of words remaining to be written. The transfer at c transfers the word to be written into the IO registers, from which it is then transferred to the word register in the tape element. Upon receipt of this word, the tape units

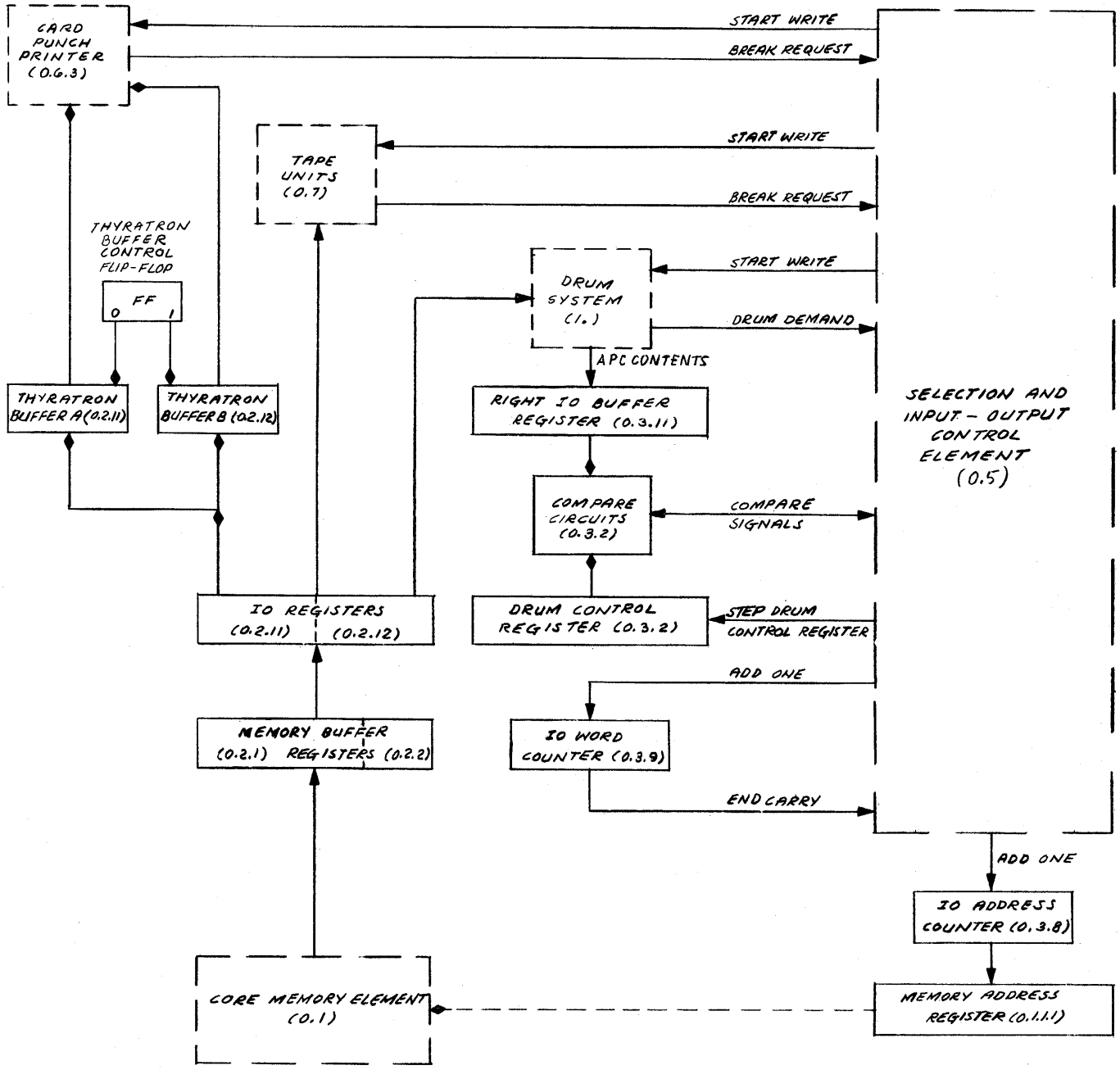


Figure 1–19. Break-Out Operational Flow, Simplified Block Diagram

write it onto the selected magnetic tape unit, where it is permanently stored. At the end of this break-out cycle, the Central Computer System resumes the execution of its program in the IO interlock on condition.

Each time the tape units are ready to write another word they generate a break request signal, causing the Central Computer System to supply the word to be written by a break-out cycle. Also, during each break-out cycle, the IO address and word counters are stepped. Thus, each word written on the magnetic tape unit is obtained from a sequentially addressed core memory location in the core memory element and the IO word counter keeps track of the number of words remaining to be written. When the final word, as specified by the address portion of the *Write* instruction, is transferred, the IO word counter is stepped to positive zero and an end-carry pulse is generated. This causes an IO word counter = 0 signal to be sent to the tape units, notifying them that no more words are to be written. This stops the motion of the selected magnetic tape unit, after which a disconnect pulse is generated and sent to the selection and IO control element. This clears the IO interlocks and the Central Computer System proceeds unconditionally with its program.

The operation of writing on the card machines (card punch or printer) is more involved, since two words are supplied by the Central Computer System for each break request received from the card machines. This is due to the nature of the card machines, which dictates that they write two words at once, whereas the Central Computer System is designed so that it can supply only one word at a time. The operation of writing on the card machines is programmed by the usual series of instructions: *Load IO Address Counter, Select* (card punch or printer), *Write*. The equipment of the Central Computer System which is associated with writing on the card machines is shown in figure 1–19.

During the execution of the *Write* instruction, a start printer or start punch signal is sent to the card machines. When the selected card machine is ready to write the first two words, a break request pulse, which in the case of the card machines is called an index pulse, is sent to the selection and IO control element. This pulse clears the thyratron buffer control flip-flop but does not cause the immediate institution of a break-out cycle. The control circuits of the selection and IO control element causes a delay of about two machine cycles to occur before a break-out cycle is instituted. This allows time for the thyratron buffer registers to stabilize after the clearing of the thyratron buffer control flip-flop. The subsequent break-out cycle obtains the word at the address in core memory specified by the content of the IO address counter and transfers it from the memory buffer registers to the IO registers. Since the thyratron buffer control flip-flop is cleared, only thyratron buffer A is conditioned. Thus, the word transferred into the IO register only activates thyratron buffer A, which energizes one set of printer or punch magnets. This effectively transfers the word to the card punch or printer. During the break-out cycle which supplies this word, the contents of the IO address and word counters are increased by 1 and the thyratron buffer control flip-flop is set from 0 to 1, thus conditioning thyratron buffer B. A delay of another two machine cycles occurs, during which the thyratron buffers are allowed to stabilize.

At the end of this delay period the Central Computer System institutes a second break-out cycle, transferring the second word from the core memory element through the memory buffer registers to the IO registers. This time, however, only thyratron buffer B is conditioned so the remaining set of printer or punch magnets are energized. Thus, as a result of a single break request (index pulse) from the card machine, the Central Computer System executes two break-out cycles, supplying two words to the card printer or punch. The first word transferred passes through thyratron buffer A, while the second word transferred passes through thyratron buffer B. This process of executing two break-out cycles and transferring two words through each thyratron buffer register after each break request from the card punch or printer continues until the IO word counter is counted down to positive zero by the continuous stepping action during each break cycle. Any break requests generated by the card machines after this point are not recognized by the Central Computer System and no more words are transferred. However, the IO interlock is not turned off immediately, since it is necessary to wait for the card machines to reach the end of their cycle. When the electromechanical devices within the card machines have come to a halt, a disconnect signal is generated which then clears the IO interlock and allows the Central Computer System to proceed unconditionally with its program.

Although information is read from the fields of the Drum System into the Central Computer System by any one of three different modes, only

two of these modes (address and status) are applicable to writing. The operation of writing on address- or status-controlled fields in the Drum System is similar to the operation of reading from similarly controlled fields. (Refer to 3.3.1 of this Chapter.) When writing upon an address-controlled field, the address of the register on the selected field into which the first word is to be written is specified by the *Select Drum* instruction, which loads this address into the drum control register of the program element. An address search operation is performed prior to the writing of any words, during which the content of the angular position counter in the Drum System associated with the selected field is placed in the right IO buffer register. The content of the right IO buffer register and the drum control register are then compared. When a successful comparison is performed, the first word is written in the specified register on the field and successive registers on the field are loaded with successive words from the Central Computer System, unless an interleaving operation is specified. When writing upon a status-controlled field, only those registers whose write status channel bit position contains a 0 have a word written into them. Those registers on the field whose write status channel bit position contains a 1 are unaffected by the write operation.

Since the Drum System does not generate break requests until a word has already been written on the surface of the selected field, a variation of the usual word transfer initiation process is necessary when writing upon the fields of the Drum System. In order to supply the first two words to be written upon the selected field, it is necessary that the Central Computer System supply the first two break requests. This initial break request generation is accomplished by the execution of the *Write* instruction in the IO program and is identical for either address- or status-controlled fields. During the execution of the *Write* instruction, a break request is generated and the fact that an additional break cycle is needed is stored in the control circuits of the selection and IO control element. The break request initiated by the *Write* instruction causes a break-out cycle to take place during which the content of the core memory register in the core memory element specified by the IO address counter is transferred to the write register in the Drum System. Due to the control circuits which are set up by the *Write* instruction, a second break request is generated when the first break-out cycle nears completion. This causes a second break cycle to immediately follow the first, during which the second word to be written on the selected field is transferred to the IO registers. Thus, at the end of these two break cycles, the first word to be written is in the write register of the Drum System and the second word to be written is in the IO registers of the Central Computer System. The action of the Central Computer and Drum Systems at this time depends upon whether an address-controlled or status-controlled field was selected by the *Select Drum* instruction.

If an address-controlled field was selected, the control circuits of the selection and IO control element are set up in the proper manner to carry out the writing operation and the drum control register contains the address of the register on the selected field into which the word in the write register of the Drum System is to be written. A delay of 120 microseconds after the reception of the start write signal, generated by the *Write* instruction, must take place before the first word may be written. After this delay, the Drum System loads the contents of the angular position counter (APC) of the selected field into the right IO buffer register. The contents of the right IO buffer register and the drum control register are then compared by means of the compare circuits. If the comparison is not successful, a no-compare signal is sent to the Drum System. This prevents the writing of the word presently held in the write register of the Drum System into the field register presently under the read-write heads. It also causes the transferal of the next address on the field from the angular position counter to the right IO buffer register. The process of loading the right IO buffer register with the content of the angular position counter and comparing with the contents of the drum control register is continued until a successful comparison is accomplished. A no-compare signal is not sent to the Drum System at this time and the content of the write register is written onto the drum surface. Simultaneously, a break request (drum demand) signal is generated by the Drum System and sent to the selection and IO control element. This causes the word in the IO registers to be transferred to the write register, and also causes a new break-out cycle to be initiated, during which the third word to be written is read out of the core memory element and placed in the IO registers.

As each field register passes under the read-write heads, the content of the write register is written onto the drum surface and a break is requested. Hence, each time the Drum System writes a word from the write register onto the drum surface a break request is generated, the

word in the IO registers is loaded into the write register, and a break-out cycle transfers the next word from the core memory element to the IO registers. This process continues with the writing of each word from the Central Computer System in sequentially located field registers on the selected field of the Drum System. When the number of words specified by the address portion of the *Write* instruction is written onto the selected drum field, the IO word counter generates an end-carry signal. This end-carry signal causes the IO interlock to be cleared and the Drum System to be disconnected. The IO process of writing between the Central Computer and Drum Systems is completed and the program continues to be executed unconditionally.

Up to this point, the discussion of address-controlled writing has assumed that interleaving of the words to be written upon the selected field is not employed. Essentially, interleaving of written words on the field by 8, 16, or 64 field registers is accomplished in the same manner just described. However, after the first successful address search comparison between the content of the drum control register and the right IO buffer register (content of the angular position counter), it is necessary to prevent the next word from being written into the next sequential field register. This is accomplished by setting up the control circuits of the selection and IO control element so that a false no-compare pulse is sent to the Drum System after the first word is written. This false no-compare pulse prevents the Drum System from writing the second word onto the drum surface and causes the address search process to be continued. At this time, the drum control register is stepped by a signal from the selection and IO control element. This stepping action causes the content of the drum control register to be increased numerically by 8, 16, or 64, the exact value being the number specified by bits L13, L14, and L15.(Refer to 3.3.1 of this Chapter.) The address search continues until the new content of the drum control register again compares exactly with the content of the right IO buffer register. The writing of the word held in the write register onto the drum surface, the generation of a false no-compare pulse, stepping of the drum control register, and the institution of a break cycle is then repeated. These operations continue until the IO word counter is brought to positive zero, at which time the process of writing on the field in the Drum System is terminated.

If a status-controlled field is selected by the previously programmed *Select Drum* instruction, the Central Computer System writes each word into any empty field register on the selected status-controlled field. This is accomplished principally in the circuits of the Drum System. As the drum rotates, the circuits of the Drum System read each successive bit position in the write status channel of the drum upon which the selected field is located. Whenever a 1 is read from this channel, no action takes place in the Central Computer System and the word in the write register of the Drum System is not written. However, whenever a 0 is read from this channel, the Drum System writes the content of the write register onto the field register corresponding to the bit position read from the write status channel and generates a break request (drum demand) which is sent to the selection and IO control element. This causes a break-out cycle to be executed and the content of the IO registers is transferred to the write register in the Drum System. During the break-out cycle, the word in the core memory register specified by the content of the IO address counter is read out of the core memory element and transferred to the IO registers via the memory buffer registers. This process continues until the IO word counter is brought to positive zero, an end carry generated, the IO interlock cleared, and the Drum System disconnected. Hence, when writing upon a status-controlled field of the Drum System, the Central Computer System writes a word into all registers on the field whose corresponding status channel bit is 0, and leaves unaltered the content of every register on the field whose status channel bit is 1. As for the read process, the status channel bit of 0 is converted to a 1 when a word is successfully written into the corresponding field register. This prevents rewriting by the status method in that field register.

3.4 ARITHMETIC OPERATIONS

It is possible to construct a digital computer which performs only the one basic operation of addition. All other operations (subtraction, multiplication, etc.) are then carried out by additions performed in various ways. Such a computer would have a minumum of components, but would make the programmer's job particularly long and tedious. Furthermore, certain computations would take a long time to be completed. In AN/FSQ–7 (XD–1,–2) Combat Direction Centrals, sufficient basic operations are built in to the Central Computer System to reduce both the programmer's job and the time required for computations to a workable minimum.

3.4.1 Basic Operations

The arithmetic element in the Central Computer System performs six types of basic arithmetic operations. Four of these are the standard operations of addition, subtraction, multiplication, and division. The other two types are also arithmetic in nature. Shift operations are equivalent to multiplication (or division) by powers of 2. Rounding operations are equivalent to discarding non-significant digits in a result, which may also be considered an arithmetic operation. Proper programming will allow the most intricate mathematical computations to be performed by means of the basic operations (and corresponding instructions) provided by the Central Computer System. (Refer to 3.5 of this Chapter.) The logical units which are involved in arithmetic operations are the left and right A registers, accumulators, B registers, memory buffer registers, and adders. Some or all of these units are involved in each arithmetic operation. In most cases, the units in the right arithmetic element operate with different data than the units in the left arithmetic element. In some cases, however, both the left and the right arithmetic elements will have some data in common. For example, in finding the X and Y components of a range, R, one element will use R in finding the X component ($R\cos\theta$), and the other element will use R in finding the Y component ($R\sin\theta$). In these cases, the twin-type instructions (*Twin and Add, Twin and Multiply,* etc.) are used.

The part played by the various logical units in each of the arithmetic operations is described here. The discussion is restricted to one of the arithmetic elements, since both the right and the left elements perform in exactly the same manner. The memory buffer register is used for transferring operands from core memory to either the A register or the B register, and for transferring results of arithmetic operations from the accumulator to core memory. In addition and subtraction operations, the A register and the accumulator both supply one operand each to the adders, and the resulting sum (or difference) appears in the accumulator. Subtraction differs from addition only in that the minuend (in the A register) is complemented before being added to the subtrahend (in the accumulator). Addition of a complemented number is mathematically equivalent to the subtraction of the same number in uncomplemented form.

In multiplication and division operations, the B register is tied to the accumulator to form a 32-bit register. This is necessitated by the fact that the product of two 16-bit numbers will be a 32-bit number, and that, in division, a 32-bit dividend is ordinarily employed. At the beginning of a multiplication operation, the multiplicand is in the A register and the multiplier is in the B register, the accumulator being clear. During the multiplication process, each bit of the multiplier is inspected in turn. If the bit is a 1, the multplicand is added to the contents of the accumulator, and the combined contents of the accumulator-B registers are shifted one place to the right. If the multiplier bit is a 0, the shift to the right occurs, but the contents of the accumulator are not increased by the multiplicand. At the end of the process, the accumulator contains the sign bit and the 15 most-significant bits of the product and the B register contains the less-significant bits of the product. In the process, the multiplier has been lost by the repeated shifts to the right. At the beginning of a division operation, the accumulator holds the sign bit and the 15 most-significant bits of the dividend, the B register holds the less-significant bits of the dividend, and the A register holds the divisor. In the course of the division process, the contents of the combined accumulator-B registers are shifted to the left, and the divisor is repeatedly subtracted from the shifted dividend. At the end of the process, the B register contains the quotient and the accumulator contains the sign bit of the quotient and the magnitude of the remainder. Before the quotient can be transferred to core memory via the memory buffer register it must be placed in the accumulator. This is accomplished by shifting the contents of the combined accumulator-B registers to the left. After 15 shifts to the left, the accumulator will hold the sign bit and the 15 most-significant bits of the quotient.

In both multiplication and division it is necessary to round off the product (or quotient) to 15 significant bits. Rounding off is accomplished by adding 1, 0, or -1 to the 15th bit, depending on the value of the sign bit and of the 16th bit of the product (or quotient). In both multiplication and division, roundoff is obtained by means of the *Shift Left and Round (SLR)* instruction. In division, 15 shifts to the left are specified in the instruction, thus transferring the quotient from the B register to the accumulator; in multiplication, no shifts to the left are specified, since the product is already in the accumulator. It should be noted that a shifting operation may, or may not, be arithmetical in nature. In multiplication and division, the shifting operations correspond

to multiplications (or divisions) by two. In the other cases, shifting is merely a means of transferring data from one place to another.

3.4.2 Calculating

A few simple examples are given to demonstrate how the basic Central Computer System operations are employed in carrying out mathematical computations. As a first example, the sum of four numbers, A, B, C, and D, will be computed. Assume that the four numbers are stored at the addresses decimal 1000 through 1003, and that the sum is to be stored in decimal 1004. The necessary instructions are taken sequentially from addresses starting with decimal 0. Table 1–9 shows the sequence of events.

TABLE 1–9. ADDITION OF FOUR NUMBERS

CORE MEMORY LOCATION	INSTRUCTION		REMARKS
	OPERATION	ADDRESS	
0	*Clear and Add (CAD)*	1000	Accumulator cleared of its original contents, then A added into the accumulator
1	*Add (ADD)*	1001	Accumulator now contains A + B
2	*Add (ADD)*	1002	Accumulator now contains A + B + C
3	*Add (ADD)*	1003	Accumulator now contains A — B + C + D
4	*Store (FST)*	1004	Sum, A + B + C + D, stored in location 1004. Sum also remains in the accumulator

A more extensive program is necessary to evaluate an expression such as Ax^2+Bx+C. Assume again that A, B, C, and x are stored in core memory locations decimal 1000 through 1003. Location decimal 1004 is used both to store the final answer and to serve as temporary storage for intermediate results of the calculation. Instructions are taken sequentially from addresses starting with decimal 0. The program for this calculation is shown in table 1–10.

TABLE 1–10. EVALUATION OF Ax^2+Bx+C

CORE MEMORY LOCATION	INSTRUCTION		REMARKS
	OPERATION	ADDRESS	
0	*Clear and Add (CAD)*	1000	Accumulator cleared of its original contents, then A added into the accumulator
1	*Multiply (MUL)*	1003	Accumulator now contains Ax
2	*Multiply (MUL)*	1003	Accumulator now contains Ax^2
3	*Store (FST)*	1004	Ax^2 temporarily stored in location 1004. Accumulator retains Ax^2
4	*Clear and Add (CAD)*	1001	Accumulator cleared of Ax^2 and B is added into the accumulator
5	*Multiply (MUL)*	1003	Accumulator now contains Bx
6	*Add (ADD)*	1002	Accumulator now contains Bx + C
7	*Add (ADD)*	1004	Accumulator now contains Ax^2 + Bx + C
8	*Store (FST)*	1004	Desired result, Ax^2 + Bx + C, now stored in location 1004. Result also remains in the accumulator

There are ordinarily a number of ways of programming the same computation, particularly if the computation is a complicated one. However, even the simple example just given could have been done in a different and shorter way. The expression to be evaluated, Ax^2+Bx+C, can be factored in the form $x(Ax+B)+C$. While this factorization is trivial from the mathematical

point of view, it is significant from the computational aspect, since it allows a reduction of 53⅓ percent in the length of the program. Table 1–11 shows the shorter program resulting from the use of this factorization. The same core memory locations are assumed for the quantities A,B,C, and x.

TABLE 1–11. EVALUATION OF x(Ax+B)+C

CORE MEMORY LOCATION	INSTRUCTION		REMARKS
	OPERATION	ADDRESS	
0	*Clear and Add (CAD)*	1000	Accumulator cleared of previous contents, then A added in to accumulator
1	*Multiply (MUL)*	1003	Ax now in accumulator
2	*Add (ADD)*	1001	Ax + B now in accumulator
3	*Multiply (MUL)*	1003	x(Ax + B) now in accumulator
4	*Add (ADD)*	1002	x(Ax + B) + C now in accumulator
5	*Store (FST)*	1004	$x(Ax + B) + C = Ax^2 + Bx + C$ now stored in location 1004. Result also remains in the accumulator

3.4.3 Errors

The arithmetic element of the Central Computer System has facilities for detecting and indicating the existence of two types of errors. One type of error occurs when one or more bits are altered in the process of transferring words into or out of core memory. The second type of error occurs when the result of an arithmetic computation is a number whose absolute magnitude is equal to or greater than unity (overflow). The first type of error is detected by means of a so-called parity check, the second type by a so-called overflow check. The detection of any such error leads to the generation of an alarm signal (a parity alarm or an overflow alarm, as the case may be).

Parity, as applied to a binary word, refers to the number of 1's in that word. If there are an even number of 1's, the word has even parity; if the number of 1's in the word is odd, the word has odd parity. Experience has demonstrated that in most of the cases where a binary word has suffered damage in transit, the word's parity is altered. The Central Computer System makes use of this fact in the method by which it checks for errors in the transfer of words into and out of core memory. Before the result of an arithmetic computation is stored in core memory, a parity count is performed in the memory buffer register, and a parity bit is assigned to and stored with the word. This parity bit is 1 for a word of even parity and 0 for a word of odd parity. The first time that this word is read out of core memory, a second parity count is performed in the memory buffer register, and the result is compared with the parity, as indicated by the parity bit of the word. If a change has occurred in the parity, a parity alarm signal is generated. This alarm signal may or may not halt the Central Computer System, depending on the preset condition of a switch on the maintenance console. Words which originate in the Drum System or in the tape element are supplied with a parity bit at their point of origin, not in the memory buffer register.

The Central Computer System is not designed to handle numbers of absolute value equal to or greater than unity. All inputs to the Central Computer System are scaled down to meet this restriction. However, certain arithmetic operations within the Central Computer System could result in an answer greater than unity. These operations are division (when the divisor is of smaller magnitude than the dividend), addition of two numbers of like sign, and subtraction of two numbers of unlike sign. These overflow conditions can frequently be avoided by programmed scale-factoring, if the programmer knows in advance the approximate magnitude of the numbers involved in the calculation. In those cases where overflow cannot be predicted, some means must be provided for either avoiding an overflow condition or detecting and indicating the existence of such a condition, should it occur.

In the case of division, a programmed comparison of the magnitudes of the divisor and of the dividend indicates in advance of the actual division, whether an overflow will or will not result. The course to be followed by the Central Computer System in either event can also be made part

of the program. In those addition and subtraction instructions where an overflow might occur, circuits are provided which automatically detect the existence of an overflow and give an overflow alarm. These circuits inspect the sign bits of the operands and the sign bit of the resulting sum (or difference). From this inspection, the existence of nonexistence of an overflow is readily determined. For example, if two negative numbers are to be added, both operands will have 1's in their sign bit positions. If an overflow results from the addition, the sum will have a 0 in its sign bit position. By means of switch settings or of additional program steps, the programmer can determine what course the Central Computer System is to follow if an overflow does occur. The overflow detecting circuits function only when one of the following instructions is programmed:

a. *Shift Left and Round (SLR)*
b. *Add (ADD)*
c. *Twin and Add (TAD)*
d. *Add One (AOR)*
e. *Add B Registers to Accumulator Registers (ADB)*
f. *Subtract (SUB)*
g. *Twin and Subtract (TSU)*

3.5 PROGRAMMING OPERATIONS

In essence, programming is the method by which the intentions and desires of the operating personnel of AN/FSQ-7 (XD-1,-2) Combat Direction Centrals are expressed in terms which are intelligible to the Central Computer System. This is accomplished by utilizing the 48 instructions which the Central Computer System is capable of executing in such a manner as to direct the operation of AN/FSQ-7 (XD-1,-2) Combat Direction Centrals toward the solution of a problem. A program comprises a group of instructions pertinent to the solution of a problem. The manner in which a program is organized and introduced into the Central Computer System is discussed here. Special programming features of AN/FSQ-7 (XD-1,-2) Combat Direction Centrals are also superficially noted here. For more information on the subject of programming, refer to PH 45-00002.

3.5.1 Program Organization and Execution

Programming begins with the statement of the problem which is to be solved. A method of solving the problem by means of the instructions which the Central Computer System is capable of executing is then formulated. (Refer to 3.4 of this Chapter.) Once the program has been decided upon, it is necessary to translate the instructions into their binary codes, introduce them into the core memory element along with the data or operands which they are to operate upon, and start the Central Computer System so that it may execute the program.

Programs and data are most often initially introduced into the Central Computer System by means of the card reader associated with the maintenance control element. The program is first manually punched onto IBM punch cards in the order in which they are to be executed. This group of cards constitutes a program deck. The data or operands upon which the program is to operate is next punched onto another set of cards. A self-loading card is then prepared. This card contains information which directs the Central Computer System in the loading of the program and data on the other cards. Essentially, this card contains an IO program stating how many words are to be read from the card reader, and the locations in core memory where these words are to be stored. (Refer to 3.3.1 of this Chapter.) Once the self-loading card, program cards, and data cards are in the card reader hopper, the LOAD FROM CARD READER pushbutton on the maintenance console panel is depressed. This causes the Central Computer System to read the words on the first, or self-loading, card in the card reader. This card contains an IO program which is immediately executed and causes the remaining cards to be read from the card reader and stored in the designated locations in core memory. The Central Computer System then proceeds to execute the introduced program and uses the supplied data to arrive at a solution to the problem for which the program was written. The results of the program may then be presented to the operating personnel by causing the solution, or solutions, to be punched onto IBM cards via the card punch or printed on a paper form via the printer.

When operating in defense status, the execution of a program is not quite as simple as just illustrated. The prime reason for this is that the data which the Central Computer System is to handle by means of its program is constantly changing and each new set of data must be processed as it becomes available. For this reason, the program which deals with the defense situation is continuously executed, with each repetition processing new information. Initially, the air defense program is prepared on punch cards. The program is then stored in magnetic core memory via the card reader. Once in the core memory element, the air defense program is written onto the auxiliary memory fields of the Drum System

by means of an IO program. Here the program is retained indefinitely, always available for introduction into the Central Computer System. This introduction of the air defense program may be initiated by depressing the LOAD FROM A.M. DRUMS pushbutton on the maintenance console, which causes the first portion of the program to be loaded into the core memory element and executed. Remaining portions of the defense program are loaded when necessary, as directed by the program portion presently being executed. The data upon which this program operates are brought into the Central Computer System from the Drum System when available and needed. These defense data come predominantly from the Input System. The information is written by the Input System onto the fields of the Drum System, and the information is obtained by IO programs strategically located in the overall program. As the results of the Central Computer System calculations are obtained, they are stored in magnetic core memory and are again transferred to the Drum System by means of IO programs within the overall program. From the Drum System, the computed results are distributed to the Output and Display Systems.

Many programs are often needed for execution in the Central Computer System for testing and various other purposes. Since the number and size of these programs is quite large, it would not be feasible to store them on punched cards. For this reason, diagnostic programs, utility programs, etc., are stored on magnetic tape by means of the tape units. Initially, the programs are prepared on cards, read into the core memory element, and then written onto magnetic tape via the tape element. The magnetic tape provides a convenient, compact storage of often-needed programs. Whenever a program is needed, the magnetic tape upon which it is contained is placed on the reels of the tape units in the tape element. An IO program directing the reading of the magnetic tape is then introduced into core memory by a punched card or is executed directly from the plugboard registers of test memory. The execution of this IO program causes the desired program on magnetic tape to be read into the Central Computer System and executed.

One reason for the versatility of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals is the large number of different types of IO devices with which it is capable of working. These IO devices are coupled to the exceedingly fast magnetic core memory which supplies the registers of the Central Computer System with the instructions and data to be operated upon and executed. The Drum System serves as a medium-capacity, high-speed storage device for all information which must be made immediately available to the Central Computer System. The magnetic tape units provide almost unlimited capacity storage space in a compact form. Although the speed with which information may be written or read from the magnetic tape units is not exceedingly fast, it is still much faster than the card machines and, therefore, the method of storage (compact reels of tape) is more convenient. The card machines are slow but versatile storage devices which act as the intermediary between the registers of the Central Computer System and the operating personnel. With the card machines, information may be manually introduced and information stored in the equipment may be made readily available for human observation. In addition, the large amount of commercial equipment available for handling punched cards provides a convenient means of sorting, punching, reading, and checking the information on the cards.

3.5.2 Indexing

In many instances, numerous sets of data upon which the same set of operations are to be performed are supplied to the Central Computer System. For instance, the Central Computer System might be supplied with five sets of three numbers, with the requirement that multiplication of the three numbers in each set be accomplished to obtain five different products. Essentially, the same series of instructions are necessary to perform the multiplications and store the products for each of the five sets. However, since the numbers to be multiplied must of necessity be stored in different locations in magnetic core memory, the series of instructions which cause the multiplications will have different numerical values for their address portions, which specify the locations of the numbers. The simplest method of performing this type of computation would be to use the following sequence of instructions five times in succession:

a. *Clear and Add (CAD)*
b. *Multiply (MUL)*
c. *Multiply (MUL)*
d. *Shift Left and Round (SLR)*
e. *Store (FST)*

This method would require 25 core memory locations to store the program for multiplying five sets of three numbers each, since the address portions of the above sequence would require different values for each of the five sets of numbers. This

method of performing the same sequence of instructions upon different types of data is wasteful of storage space in the core memory element, and also requires the added work of preparing the instructions, punching them on cards, checking them, etc. The subject discussed in the following text is called indexing, and provides a convenient and rapid method of utilizing the same sequence of instructions on different sets of data having different locations in magnetic core memory.

Essentially, indexing consists of altering the address portion of an instruction before the information contained within the address portion is used to specify an operand to be obtained from magnetic core memory. This is accomplished by the circuits of the address register and the index registers in the program element of the Central Computer System. Whenever so directed by bits L1 through L3 of the instruction word, the numerical content of the index register specified by these bits is added to the address portion of the instruction which is in the address register. This addition operation occurs after the address portion of the instruction has been placed in the address register but before the address portion is transferred out of the address register to its destination, usually the memory address registers of the core memory element. The addition of the content of an index register to the content of the address register in the program element takes place whenever bits L1 through L3 specify an index register (including the right accumulator register, which may be used as an index register). However, it is important to note that not all of the 48 instructions which the Central Computer System is capable of executing may be indexed. Those instructions which may be indexed are listed below:

a. *Load B Registers (LDB)*
b. All instructions in the add class
c. All instructions in the multiply class
d. All instructions in the store class
e. All instructions in the IO class

If an attempt is made to index an instruction other than one of these five, by specifying an index register in bits L1 through L3 of the instruction, no action will take place and the instruction will be executed in normal fashion without indexing.

The method by which a cycling loop is developed when the same sequence of instructions must be used many times in the processing of certain types of data is illustrated here by means of an example. Table 1–12 contains a program performing the same repetitive additions both with and without the indexing feature. The purpose of these two programs is to add together 10 numbers, producing a sum which is stored in magnetic core memory. The 10 numbers are stored in decimal locations 200 through 209 in magnetic core memory and the sum is to be stored in decimal location 300. The column labeled Memory Location gives the locations in magnetic core memory of the instructions of the program.

TABLE 1–12. COMPARISON OF CYCLIC AND NONCYCLIC PROGRAMS

NO INDEXING FEATURE				*WITH INDEXING FEATURE*				
		INSTRUCTION WORD			INSTRUCTION WORD			
PROGRAM STEP	MEMORY LOCATION	OPERATION	ADDRESS	MEMORY LOCATION	INDEX REGISTER	OPERATION	INDEX INTERVAL	ADDRESS
1	100	*CAD*	209	99	1	*XIN*		8
2	101	*ADD*	208	100		*CAD*		209
3	102	*ADD*	207	101	1	*ADD*		200
4	103	*ADD*	206	102	1	*BPX*	1	101
5	104	*ADD*	205		(Branch to 101 if index register 1 was positive)			
6	105	*ADD*	204	103		*FST*		300
7	106	*ADD*	203					
8	107	*ADD*	202					
9	108	*ADD*	201					
10	109	*ADD*	200					
11	110	*FST*	300					

The program shown in table 1–12 without the indexing feature is a straightforward sequence of nine *Add* instructions following a *Clear and Add* instruction. At the end of the execution of the *Add* instruction at program step 10, the sum of the 10 numbers is in the accumulator registers. The *Store* instruction then transfers the sum to the core memory element for storage in core memory location decimal 300. The instruction indexing feature is now considered by an analysis of the program in table 1–12 which does utilize indexing.

The initial instruction of the indexing program is the *Reset Index Register* instruction, in which bits L1 through L3 specify index register No. 1 in the program element and in which the address portion contains a numerical value of eight. This instruction causes the content of the address register (eight) to be transferred to index register No. 1 in the program element. It should be noted that since reset class instructions are not indexable, bits L1 through L3 of the *Reset Index Register* instruction do not cause the instruction to be indexed but rather specify the index register to be affected by the instruction. As will become apparent, the *Reset Index Register* instruction is used to load the specified index register with the number of times that the program is to be recycled.

The next instruction in the program (step 2) is a *Clear and Add* instruction. This instruction causes the number in core memory location decimal 209 to be loaded into the accumulator registers. In this respect, it performs the same function as the *Clear and Add* instruction in the program without indexing. The next instruction in the sequence is an *Add* instruction specifying index register No. 1 and magnetic core memory address decimal 200. However, since an index register is specified, this instruction will not cause the content of core memory location 200 to be obtained. Instead, the content of index register No. 1 (eight) is added to the content of the address register. This causes eight to be added to the address portion of the instruction in the address register so that the new address specified is decimal 208. Immediately following the addition, the content of the address register is transferred to the memory address registers in the core memory element so that the number in core memory location 208 is added to the accumulator registers. The accumulator registers now contain the sum of the two numbers in core memory addresses decimal 209 and 208.

The next instruction (step 4) is a *Branch and Index* instruction having an index interval content of 1 and specifying index register No. 1. (Refer to 3.2.6 of this Chapter.) This intruction causes the recycling of the *Add* instruction in step 3 for the remaining eight additions and modifies the content of index register No. 1 during each recycling operation so that the remaining numbers to be added may be obtained. This instruction first examines the sign bit of index register No. 1. Since it is positive, a branch operation takes place. First, the content of the program counter is transferred to the right A register (the contents of the program counter in this program are ultimately lost, since no *Store Address* instruction is given to preserve it). The content of the address register is then transferred to the cleared program counter so that the next instruction to be obtained is the *Add* instruction at step 3 in core memory location decimal 101, the address portion of the *Branch and Index* instruction. The contents of the index interval register (1) are then subtracted from the content of index register No. 1 so that it will contain seven.

Due to the action of the *Branch and Index* instruction, the *Add* instruction is again executed. This time, however, its address portion is indexed by seven, since this is the new content of the index register. This results in the *Add* instruction causing the addition of the content of core memory location decimal 207 into the content of the accumulator registers. After the execution of the *Add* instruction, the *Branch and Index* instruction is again executed. Since the content of index register No. 1 is still positive, a branch again takes place to the *Add* instruction and the index register contains a six, due to the subtraction of the index interval bits. These iterative cyclings of the *Add* and *Branch and Index* instructions continue until the content of the index register becomes negative, due to the repeated subtractions of the content of the index interval register (bits L10 through L15). At this time the *Add* instruction has been executed nine times, eight times due to the cycling and once due to its original position before the *Branch and Index* instruction.

When the content of index register No. 1 becomes negative, the next execution of the *Branch and Index* instruction does not cause a branch back to the *Add* instruction. Instead, the next instruction executed is the *Store* instruction, which causes the sum of the 10 numbers in the accumulator registers to be stored in core memory location decimal 300. As can readily be seen, the

indexing program requires only five core memory locations for storage of the program, whereas the program without indexing requires 11 core memory locations. If the problem is to add 100 numbers rather than 10, the saving in storage space in magnetic core memory becomes even more marked. This shows the advantage of using the instruction indexing feature to perform repetitive operations. Due to the insertion of the *Reset Index Register* instruction and the repeated execution of the *Branch and Index* instruction, the indexing program requires more time for completion than the nonindexing program. However, this time consideration is more than offset by the ease of compiling the program and the saving in magnetic core memory storage space.

Although the previous discussion has specified index register No. 1, the same results could be accomplished with index register No. 2 by changing bits L1 through L3 of the instruction to specify index register No. 2. The right accumulator register may not be used in this type of program, since the right accumulator register is not affected by the *Branch and Index* instruction. Normally, the right accumulator register is used to modify the address portion of an instruction when the numerical value by which the indexing is to take place is not known at the time the program is compiled. In this event, the value must be calculated and the result of the calculation used in indexing directly from the right accumulator register. If a cycling loop is necessary, the contents of the right accumulator register are transferred to one of the index registers by means of the *Reset Index Register from Right Accumulator* instruction.

3.5.3 Miscellaneous Programming Operations

The process of constructing cycling loops by means of the reset class and *Branch and Index* instructions is perhaps the most prominent feature of programming in AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. Numerous other programming features are also incorporated in the Central Computer System which increases the ease of programming and the versatility of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. Two of the most important programming features are provided by the *Sense* and *Operate* instructions.

The *Sense* instruction provides a means of automatically controlling the path of the program in accordance with conditions existing in AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. The execution of this instruction causes the Central Computer System to sense for the presence of a specified condition. If the specified condition is absent, the *Sense* instruction has no effect and the program continues sequentially, obtaining the next instruction from an address one higher than the location from which the *Sense* instruction was obtained. However, if the specified condition is found to be present, a branch occurs in the program and the next instruction to be executed is obtained from the location in magnetic core memory specified by the address portion of the *Sense* instruction. Typical conditions whose presence or absence may be determined by the *Sense* instrution are that the IO interlock is on; a memory, drum, or tape parity error has occurred; a selected IO unit is not ready for IO word transfers; a switch on the maintenance console is in a particular position, etc.

One important use of the *Sense* instruction is to prevent hangup (indefinite duration of IO pause) in the Central Computer System from taking place by executing an IO program specifying an IO unit which is incapable of transferring words. For instance, assume that there are no IBM punch cards in the hopper of the card reader and an IO program ordering words to be read from the card reader into the Central Computer System is executed. Since there are no cards to be read, no words are transferred. This causes the IO interlock to remain on permanently while the Central Computer System proceeds with its program. If an IO class instruction appears in the program, an IO pause is instituted (since the IO interlock is on) and no further action takes place. Since the IO interlock can never go off (no words are ever transferred from the card reader), the Central Computer System remains in the IO pause condition indefinitely (hangs up). This condition is undesirable, since no computations are performed and the IO pause can only be ended by manual intervention at the maintenance console. In order to prevent a hangup, a *Sense* (IO not ready) instruction may be inserted in the IO program prior to the *Read* instruction. If the IO unit (in this case, the card reader) is ready (cards in hopper), the instruction has no effect, the *Read* instruction is executed, and the IO word transfers are instituted. However, if the IO unit is not ready (no cards in hopper), a branch occurs in the program, thus preventing the execution of the *Read* instruction which initiates the word transfers and turns on the IO interlock. Hence, it can be seen that the *Sense* (IO not ready) instruction provides a means of preventing an IO operation from being programmed with a defective or unprepared IO device.

The *Operate* instruction provides a convenient means of automatically activating certain electromechanical and electronic devices, called operate units, in AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. The execution of this instructions causes an operate signal to be generated by the selection and IO control element; this signal is sent to a specified operate unit, causing it to be activated. Typical actions which may be performed by the *Operate* instruction are: initiation of display cameras in the Display System; alteration of printing or punching format in the card punch or printer; starting and stopping of marginal checking excursions; rewinding the magnetic tape on a selected tape unit, etc. Essentially, the *Operate* instruction provides a means of initiating actions in the equipment of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals as called for by the program.

Four instructions which are analogous in operation and use to the *Sense* instruction are the *Branch on Zero, Branch on Minus, Branch on Left Minus,* and *Branch on Right Minus* instructions. (Refer to 3.2.6 of this Chapter.) These four instructions of the branch class provide a means of causing the Central Computer System program to follow one of two alternate paths in accordance with the results of previous computations. For instance, the program may be made to continue sequentially if the result of a previous computation in the accumulator registers is positive, or the program may be made to branch if the result of the previous computation is negative. This is accomplished by insertion of the *Branch on Minus* instruction immediately following the sequence of instructions which cause the calculation to be performed. In essence, these four instructions allow the Central Computer System to make its own logical decisions as to what to do next, depending on the results of its previous computation.

Aside from those mentioned, other programming features of the Central Computer System exist which augment the versatility and speed of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. However, these features are not discussed here, since all programming features may be found in PH 45–00002. Only those features which are of particular interest and serve to introduce the reader to programming have been included in this Chapter.

PART 2
INSTRUCTION CONTROL ELEMENT
CHAPTER 1
INTRODUCTION

1.1 GENERAL

The instruction control element furnishes the Central Computer System with the commands required to execute all the instructions that the computer is capable of performing. The commands are in the form of gated pulses, and are transmitted in a sequence governed by the instruction in progress. Each command performs a basic operation (transfer, clear, set, complement, etc.). By means of these basic operations, any given instruction is broken down into a series of sequenced steps. A total of 151 instruction control element commands are required to execute the 48 instructions employed in the AN/FSQ–7 (XD–1) Combat Direction Central. In addition to command pulses, this element also furnishes a set of time pulses (TP) which are automatically transmitted at regular intervals.

The outputs of the instruction control element are distributed to the remaining four elements of the Central Computer System. The bulk of the command pulses are sent to the arithmetic element, where they actuate the operations which must be performed on an instruction operand. Commands are also sent to the program element, enabling this element to co-ordinate the scheduling of computer instructions. Time pulses and commands are sent to the selection and IO control element where they control the operation of appropriate IO devices, and enable these devices to communicate with the core memory element. The core memory element receives commands which convey information concerning the selection of memory units and the initiation of memory units and memory cycles.

Throughout the discussion of the instruction control element, it is assumed that the necessary memory cycles are properly synchronized with the commands generated by this element. Therefore, no mention will be made of the processes required to transfer information between the core memory element and the instruction control element. The manual operation function of the instruction control element will not be considered in this preliminary manual.

1.2 BLOCK DIAGRAM ANALYSIS

The basic electronic element used in formulating a command pulse is a gate tube. All gate tubes used for this purpose are termed command generators. As shown in figure 2–1 two signals, a pulse and a d-c signal level, control the action of a command generator. When the d-c level is positive, the positive pulse is transmitted and becomes a positive command pulse. When the d-c level is negative, the pulse is not transmitted by the command generator and no command is issued. The pulses are generated independently of the d-c levels and are applied to the command generators in a repetitive sequence. Since the distribution of these pulses is periodic, they inherently establish a timing sequence.

1.2.1 D-C Level Generation

The function of the d-c level generation equipment is to decode the instruction information supplied to it by the left memory buffer register. This decoding process results in the generation

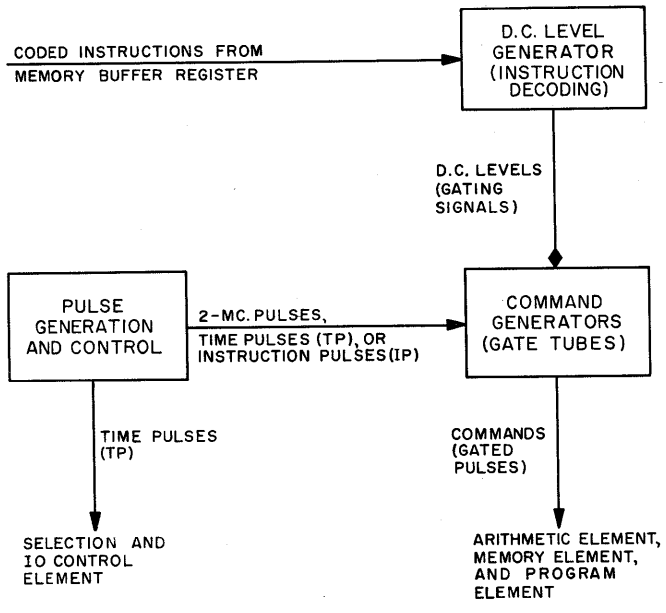


Figure 2–1. Instruction Control Element, Simplified Block Diagram

of a set of d-c levels, which are used to condition those command generators required to execute a specified instruction. The d-c level generation equipment consists of an operation register, several control matrices, a cycle control, a memory unit selection control, and an instruction matrix. These components are interconnected as shown in figure 2–2. The operation register initially receives the operational part of the instruction word (bits L1 to L10) from the left memory buffer register. Information concerning the selection of memory units is received from the program element and is fed directly to the memory unit selection control in the instruction control element.

The information now held in the operation register (bits L1 to L10) is subsequently distributed to the control matrices; i.e., the index selection matrix, the class cycle matrix, and the variation matrix. The index selection matrix determines which of the three available index registers is to be used if an indexing instruction in specified. It also determines the index register whose contents are to be modified or reset during a *Branch and Index* or a reset class instruction. (Refer to table 2–1.) The class cycle matrix determines to which of eight classes a particular instruction belongs, a class being composed of those instructions which are similar in their manner of execution. The output of the class cycle matrix conditions different sets of command generators at different times during a computer cycle. Information concerning computer cycles is supplied to this matrix by the cycle control and is combined with the class information. The individual instructions within a class are called variations of that class. That portion of the instruction which specifies a particular variation is decoded by the variation matrix.

The outputs of the control matrices are fed to the instruction matrix. This matrix consists of eight separate matrices, one for each instruction class. Each class instruction matrix utilizes information from the control matrices to develop d-c levels required for the execution of instructions within that class. Additional d-c levels are supplied directly from the cycle control and class cycle

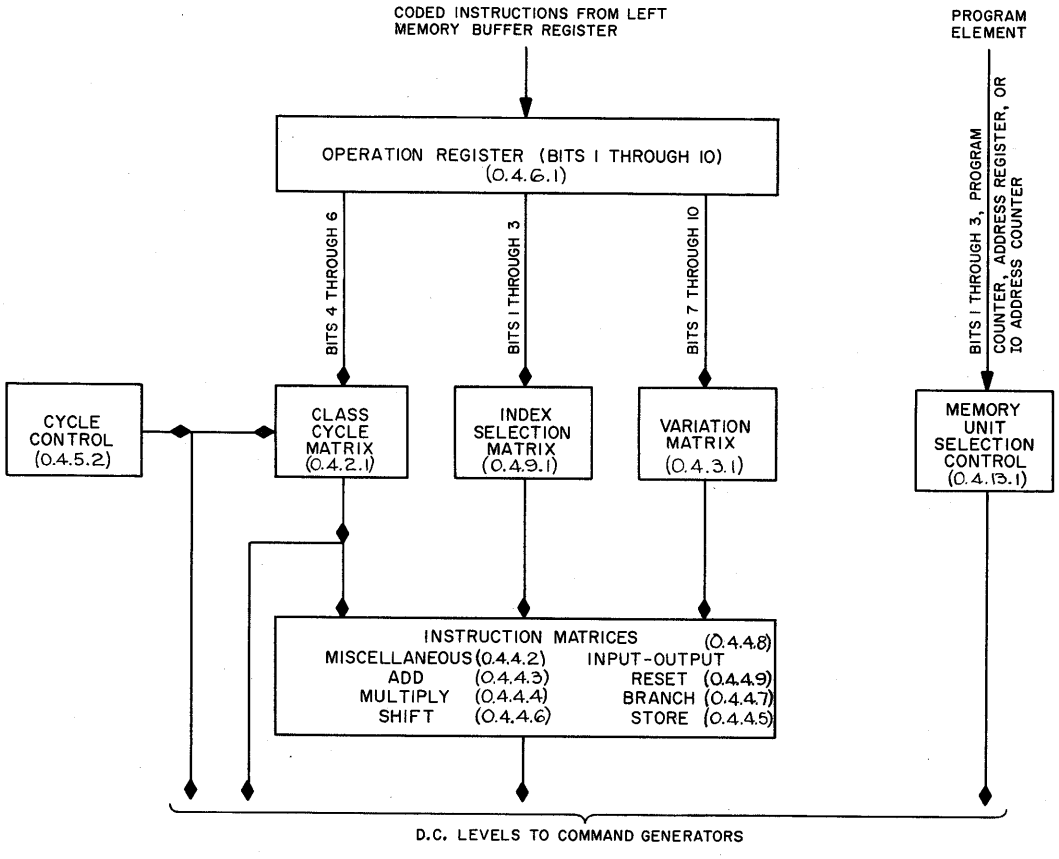


Figure 2–2. D-C Level Generation, Simplified Block Diagram

matrix. Consequently, each instruction within the class generates a different set of d-c levels, although there are certain d-c levels which are common to all of these sets. The d-c levels thus developed are gating signals for the command generators. As described previously, these gating

TABLE 2-1. OPERATION CODES FOR AN/FSQ-7 (XD-1) COMBAT DIRECTION CENTRAL

CLASS	BINARY CODE	VARIATION	BINARY CODE	ABBREV.
Miscellaneous	000	*Program Stop*	0000	*HLT*
		*Extract**	0001	*ETR*
		Operate	001–	*PER*
		Clear & Subtract Word Counter	0100	*CSW*
		Shift Left & Round	0101	*SLR*
		*Load B Registers**	0110	*LDB*
Add*	001	*Clear & Add*	0000	*CAD*
		ADD	0001	*ADD*
		Twin & Add	0010	*TAD*
		Add B Registers to Accumulator Registers	0011	*ADB*
		Clear & Subtract	0110	*CSU*
		Subtract	0111	*SUB*
		Twin & Subtract	1000	*TSU*
		Clear & Add Magnitude	1100	*CAM*
		Difference Magnitude	1101	*DIM*
Multiply*	010	*Multiply*	1010	*MUL*
		Twin & Multiply	1011	*TMU*
		Divide	1100	*DVD*
		Twin & Divide	1101	*TDV*
Store*	011	*Store*	0101	*FST*
		Left Store	0110	*LST*
		Right Store	0111	*RST*
		Store Address	1000	*STA*
		Right Add 1	1001	*AOR*
		Exchange	1010	*ECH*
		Deposit	1100	*DEP*
Shift	100	*Shift Left*	0000	*DSL*
		Shift Right	0001	*DSR*
		Shift Accumulators Left	0100	*ASL*
		Shift Accumulators Right	0101	*ASR*
		Left Element Shift Right	1000	*LSR*
		Right Element Shift Right	1001	*RSR*
		Cycle Left	1100	*DCL*
		Cycle Accumulators Left	1110	*FCL*
Branch	101	*Branch & Index*	001–	*BPX*
		Sense	010–	*BSN*
		Branch on Zero	1000	*BFZ*
		Branch on Minus	1001	*BFM*
		Branch on Left Minus	1010	*BLM*
		Branch on Right Minus	1011	*BRM*
Input-Output*	110	*Load IO Address Counter*	0000	*LDC*
		Select Drum	001–	*SDR*
		Select	010–	*SEL*
		Read	1110	*RDS*
		Write	1111	*WRT*
Reset	111	*Reset Index Register*	1011	*XIN*
		Reset Index Register from Right Accumulator	1101	*XAC*
		Add Index Register	1110	*ADX*

**Indicates that the instruction is indexable.*

signals control the distribution of the pulses sent to the command generators from the pulse generation equipment.

As shown in figure 2–2, the memory unit selection control is an additional source of d-c levels for the command generators. However, the decoding procedure for this information is carried out in the program element. The decoded information, in pulse form, is fed to the memory unit selection control. This control utilizes these pulses to form d-c levels, which are then fed to the command generators.

1.2.2 Pulse Generation and Control

One of the functions of the pulse generation and control equipment is to develop and distribute pulses (instruction pulses) which are subsequently gated by the command generators as described in the previous paragraphs. These pulses are generated during each instruction cycle. They are not generated during break cycles. A second function of this equipment is to develop and distribute another set of pulses (time pulses) which are in synchronism with the instruction pulses but are generated during break cycles as well as during instruction cycles. Both time pulses and instruction pulses can be turned off during certain computer operations (pauses). At these times, the necessary pulse signals are derived from pulses supplied by an oscillator. These pulses are from 20 to 40 volts in magnitude and 0.1 microsecond in width. The base line is at —15 volts.

Figure 2–3 is a simplified block diagram of the pulse generation and control equipment. The source of all pulses is a 2-megacycle oscillator whose output is fed to the time pulse distributor control. The time pulse distributor control directs the flow of the 2-megacycle pulses to the time pulse distributor, as well as to the command generators. Those pulses which are fed to the time pulse distributor are converted by this unit into time pulses and instruction pulses. The two types of pulses are generated identically, each in a group of 12 time-sequenced pulses. The time pulses are then fed to the selection and IO control element and the command generators, and the instruction pulses are fed to the command generators.

To summarize, then, three types of pulses (2-megacycle pulses, time pulses, and instruction pulses) are used within the Central Computer System, and their distribution is essentially controlled by the time pulse distributor control. Signals from two sources, the command generators and the selection and IO control element, regulate the operation of the time pulse distributor control. Commands from the command generators can inhibit the generation of instruction and time pulses. However, the time pulse distributor control substitutes 2-megacycle pulses as an

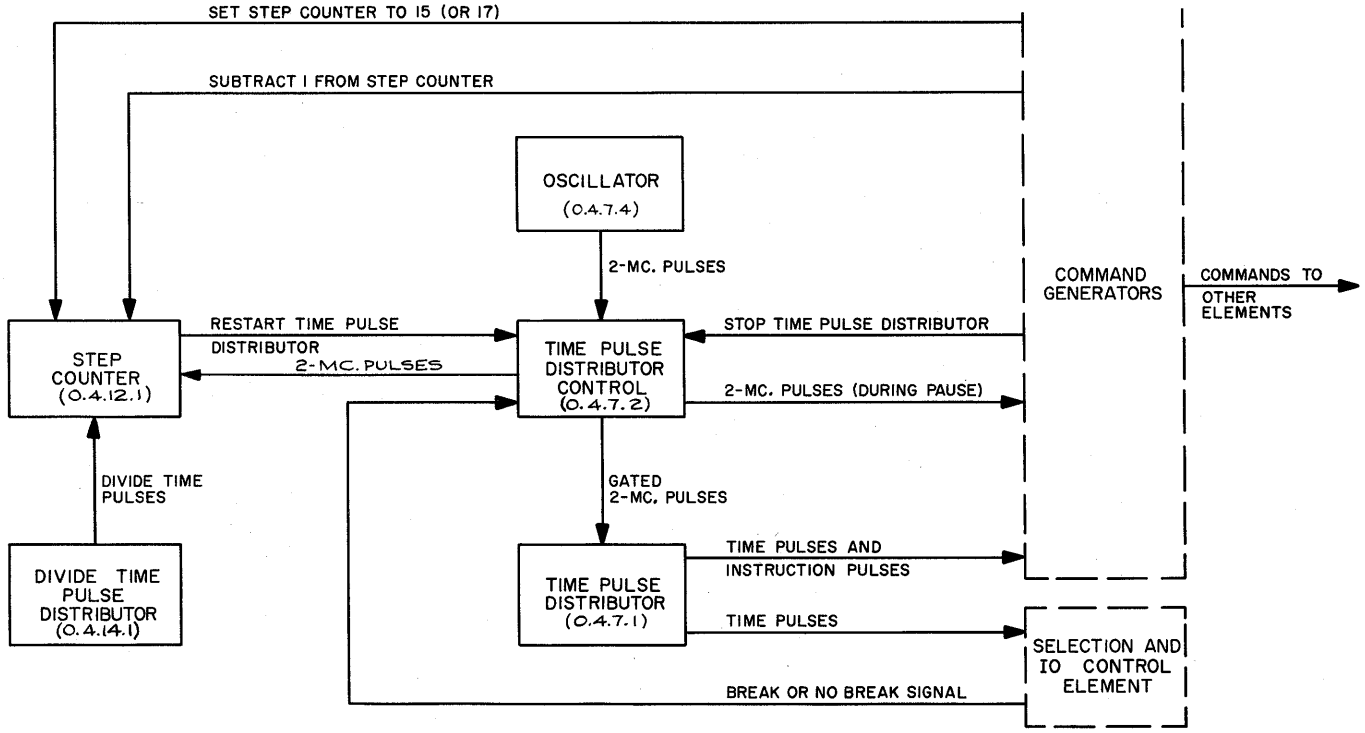


Figure 2–3. Pulse Generation and Control, Simplified Block Diagram

input to the command generators (except during a *Program Stop* instruction). These 2-megacycle pulses continue the instruction previously controlled by the instruction pulses. Such an operation is called a pause and is usually required when iterative operations are to be performed. A pause operation is independent of the computer timing cycles. The duration of a pause varies with the instruction and is controlled by the step counter. An auxiliary control element used with the step counter during the pause time of a *Divide* instruction is the divide time pulse distributor.

The time pulses sent to the selection and IO control element from the time pulse distributor control are used to generate break-in and break-out pulses during IO operation. Information relating to the generation or non-generation of break cycles is relayed to the time pulse distributor control. If an IO device is about to transfer information into or out of the computer (break cycle), the signal from the selection and IO control element stops the generation of instruction pulses, but not that of time pulses. If no break cycle is imminent, the time pulse distributor control allows the operation to proceed without interruption.

1.3 TIMING

Timing of the Central Computer System is accomplished by means of consecutive, 6-microsecond time intervals. Each interval is called a computer cycle, of which there are two types. The first type is used to time core memory operations and is called a memory cycle; the second time is used to time the operations of all other Central Computer System elements, and, is called a machine cycle. The machine cycle itself has three subdivisions; program time, operate time, and operate time B. Program time is that portion of computer time during which instructions are decoded. Operate time and operate time B are those parts of computer time during which operands are manipulated. Break cycles do not directly affect computer timing since they occur asynchronously.

Every program time or operate time cycle, as it occurs, has an accompanying memory cycle associated with it. The pulse generation equipment provides a timing pulse which synchronizes every memory cycle with its associated machine cycle. Figure 2–4 defines the operations which can be performed in core memory during specific portions of the memory cycle.

The time indications given in the figure for the basic memory cycle refer to time pulses. These pulses are synchronous in time with either program time pulses or operate time pulses, depending upon the type of machine cycle with which the basic memory cycle is associated. Three machine cycles, one for program time, another for operate time or operate time A, and a third for operate

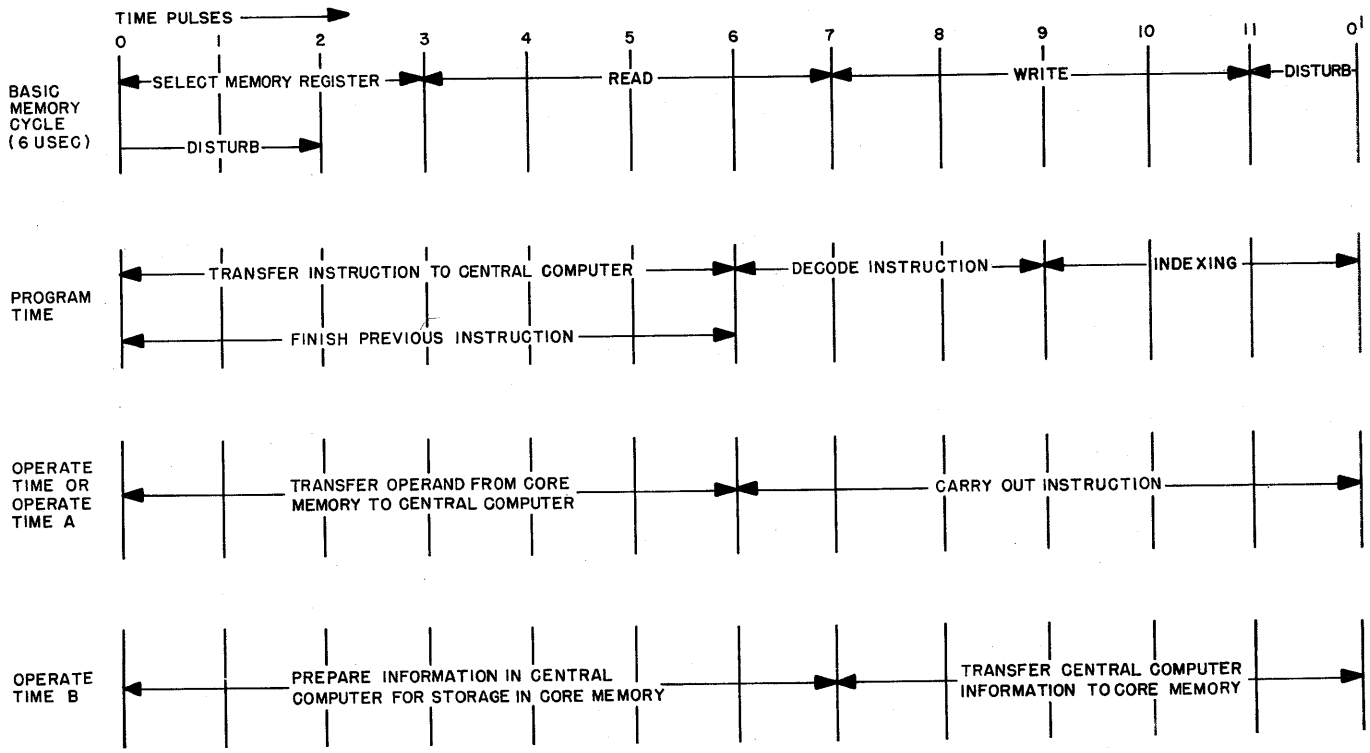


Figure 2–4. Memory Cycle

time B are also included in figure 2–4. The operations that can be performed during any given portion of a machine cycle are ascertained by superimposing the machine cycle in question on the basic memory cycle. For example, an operand is to be transferred from core memory to the Central Computer System between TP–0 and TP–6 of the operate time cycle. Superimposing the operate time cycle on the basic memory cycle shows that this is possible because a portion of the read time in the memory cycle occurs during the time interval between TP–0 and TP–6.

For the present, the 6-microsecond memory cycle has not yet been realized, and three dummy pulses (6A, 8A, and 11A, not shown in the figure) have been made part of the basic memory cycle. The Central Computer System can perform satisfactorily with a 6-microsecond machine cycle, but for the present, must operate with machine cycles which coincide in length with the memory cycle.

The instructions used for the Central Computer System can be separated into three groups according to the manner in which the required instruction time is divided. As shown in figure 2–5, this division is based upon the machine cycles. Note that all of the groups begin at PT_1–7 and end at PT_2–6. This convention was chosen because instruction decoding does not start in any program time cycle until PT–7. The first 3 microseconds (PT–0 to PT–6) are allotted for the operations necessary in transferring the selected instruction from core memory to the Central Computer System.

In the design of the Central Computer System, this has led to the assignment of a dual function to each program time cycle. The time interval between PT–0 and PT–6 has been reserved for completing the previous instruction and, at the same time, transferring the next instruction from core memory to the Central Computer System. This is possible only because the circuits used to complete the previous instruction do not involve those required to transfer the new instruction. The time interval between PT–7 and PT–11 is reserved solely for decoding the instruction in progress. Accordingly, it is assumed in the figure that the first half of the program time cycle (PT_1–0 to PT_1–6) has been completed. Further, the only function of the portion of the PT_2 cycle which is of interest in this discussion is that which completes the instruction decoded in the time interval from PT_1–7 to PT_1–11.

Figure 2–5, part A applies to those instructions which do not require an operand. The PT_1 – PT_2 configuration is used for instructions, such as

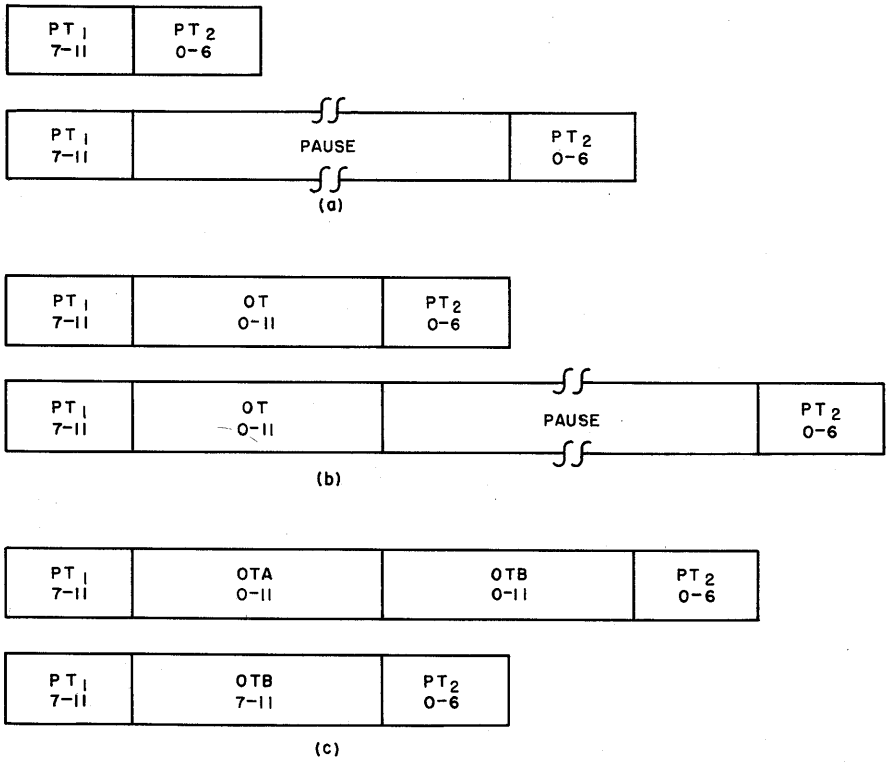


Figure 2–5. Machine Cycles

those comprising the reset class, which require no more than 6 microseconds for their execution. The PT_1-pause-PT_2 configuration is used when, because of iterative operations such as occur during instructions in the shift class, more than 6 microseconds are required for execution of the instruction. Figure 2–5, part B applies to those instructions which require an operand. Manipulation of the operand is performed during the operate time cycle. The PT_1–OT–PT_2 configuration is associated with instructions such as those of the add class which can be executed within the time allowed by two memory cycles. The PT_1-OT-pause-PT_2 configuration is required for those instructions, such as those of the multiply class, which, in addition to requiring an operand, need more than the 6 microseconds allowed by an operate time cycle to complete the manipulations on the operand. The pause portion of the configuration allows this additional time by stopping the computer timing long enough to allow the requisite number of iterative operations to be completed.

Figure 2–5, part C applies to instructions such as those which comprise the store class. The first configuration, PT_1-OTA-OTB-PT_2 is applicable when only half of a memory register must be affected by a store process, while the other half must be preserved. During the OTA cycle, the full word in core memory is read into the Central Computer System so that the half-word which is to remain unaltered can be temporarily stored therein. During this same cycle, the new half-word is prepared for transfer into core memory. The OTB cycle is devoted to writing back into core memory the composite full word, consisting of the unaltered half-word and the new half-word. When a full word must be stored in core memory, the problem of preserving a half-word in core memory does not arise. Consequently the need for the OTA cycle is obviated, and the PT_1-OTB-PT_2 configuration is applicable.

The cycle control establishes the type of machine cycle in which the computer operates in order to execute a particular instruction. The cycle control, in conjunction with the time pulse distributor control, determines whether a program time cycle, operate time, or operate time B cycle is to be used during the execution of a particular instruction. Different circuits are used for different cycles or different combinations of cycles.

Each instruction pulse, of which there are 12 in all, is used to issue a specific command. Every time an instruction pulse is applied to a conditioned command generator, a command is issued. Every time this pulse occurs, the same command is issued, provided the command generator is again conditioned. During program time, for example, a certain set of command generators is conditioned. When IP–1 occurs during program time and pulses these command generators, a number of commands are issued. During operate time, however, different command generators are conditioned. When IP–1 occurs during this time, therefore, a different set of commands is issued. The same condition prevails during operate time B, when a third set of command generators is conditioned, thus issuing a third set of commands when pulsed by IP–1.

Before a pause is initiated, a command sets the step counter to a number which indicates how many repetitive steps are to be performed during the pause. There is one exception to this statement. During a shift, the number of repetitive steps are set into the step counter directly from the address part of the instruction. When the pause is initiated, a command is given to stop the time pulse distributor, turning off the time pulses and the instruction pulses. The 2-megacycle pulses then generate the commands required to finish the pause operation. The 2-megacycle pulses also count the steps performed, subtracting one from the step counter at the completion of every step. When the pause operation is completed, instruction pulses are again sent to the command generators and carry out the ensuing instructions. If, during a pause, a break request is received to either read in or write out information, the time pulses are turned on and the break is executed. The pause, however, continues uninterrupted. When the break is completed, the time pulses are again suppressed and remain so until completion of the pause.

CHAPTER 2
GENERATION OF D-C LEVELS

The function of the instruction decoding units (control and instruction matrices) in the instruction control element is to convert instructions supplied by the left memory buffer register into appropriate d-c levels at the suppressor grids of command generators (gate tubes). When, in accordance with an instruction, the d-c level for a particular command generator is up (positive), this generator will pass pulses applied to its control grid. These pulses are the commands needed to execute a given instruction.

2.1 INSTRUCTION DECODING

The instruction furnished by the left memory buffer register is in the form of 15 binary digits (bits). These bits are arranged according to the following code. Bits L1 through L3 of the operation register are termed the index indicator. They are used to designate one of four possibilities as follows:

a. 001 Index register No. 1
b. 010 Index register No. 2
c. 011 Index register No. 3 (right accumulator)
d. 000 No index register

Bits L4 through L6 of the instruction word identify the class in which a particular instruction appears. All the instructions which the AN/FSQ-7 (XD-1) Combat Direction Central is capable of executing are grouped into eight classes. Each class consists of a number of specific instructions (variations). These variations are determined by the contents of bits L7 to L10 of the instruction word. Table 2-1 lists the variations grouped according to class. The remaining bits of the left half-word (bits L10 to L15) are used to specify the index interval. (One bit, a sign bit, is not used.)

2.2 CLASS AND VARIATION SELECTION

The following units are used for instruction decoding:

a. Operation register, sign bit through bit 10 (0.4.6.1)
b. Cycle control (0.4.5.2)
c. Class cycle matrix (0.4.2.1)
d. Variation matrix (0.4.3.1)
e. Index selection matrix (0.4.9.1)
f. Instruction matrix (0.4.4.0)
g. Memory unit selection control (0.4.13.1)

The memory unit selection control receives information from the program element (address register, the IO address counter, or the program counter). The remaining six units operate on information supplied by the left memory buffer register. Those units which receive their information from the left memory buffer register select a specific class and variation, and are dealt with first. The unit which receives its information from the program element is described last.

2.2.1 Operation Register (0.4.6.1)

The instructions to be executed are sent from the left memory buffer register to the operation register. The flip-flops of the operation register condition AND circuits in the various control matrices (class cycle matrix, variation matrix, and index selection matrix). The AND circuits in the class cycle matrix, in addition to being conditioned by the operation register, are also conditioned by d-c outputs from the cycle control.

2.2.2 Cycle Control (0.4.5.2)

The cycle control employs six flip-flops to perform four functions. Two flip-flops, as directed by command generators, determine the type of machine cycle that is to be performed. As shown in figure 2-6, one of these flip-flops selects program time (0) or operate time (1). The second flip-flop functions only during the operate time cycle and determines whether an operation time A (0) or an operation time B (1) is to be performed. The outputs of these two flip-flops are the only two outputs supplied by the cycle control to the class cycle matrix.

The third flip-flop is termed the branch flip-flop. It is turned on when the *set branch FF on* command (170) is given by a command generator, indicating that during a branch instruction, the condition to perform a branch has been met.

The fourth flip-flop co-ordinates the operation of the selection and IO control element and the

instruction control frame during input-output operations. The IO interlock flip-flop is turned on (1) whenever an IO instruction is being performed. It prevents the Central Computer System from starting a new IO instruction while another IO instruction is in progress. IO break cycles are selected by controlling the outputs of the time pulse distributor, which is discussed in 3.3 of Chapter 3 of this Part.

The fifth and sixth flip-flops are transfer flip-flops. They are turned on whenever the word counter is being stepped. They prevent the transfer of information from the word counter to the right accumulator register while the word counter is being stepped. This condition might occur if a *Clear and Subtract Word Counter* instruction were given before a previously ordered block transfer had been completed. The first of these two transfer flip-flops receives its commands from the selection and IO control element, while the second is controlled directly from a command generator.

2.2.3 Class Cycle Matrix (0.4.2.1)

The class cycle matrix, shown in figure 2–6, mixes, by means of diode AND circuits, the outputs of the class-selecting bits of the operation register, the outputs of the program time-operation time flip-flop, and the A–B flip-flop in the cycle control, to provide 17 outputs. There is one output for each combination of class and cycle required by the computer. All classes of instructions, except the reset class and the shift class, contain at least one instruction that requires an operation cycle in addition to the program cycle. Two operation cycles, designated OTA and OTB, are provided for the store class. The miscellaneous, multiply, and shift classes require a line that is on regardless of the type of cycle that is being performed.

As is shown in figure 2–6, there is one line in each of the miscellaneous, multiply, and shift classes which is independent of both the PT and OT. These lines are made independent of PT or OT so that they may function to initiate, propagate, and end the 2-megacycle operations utilized during pauses.

The outputs of the class cycle matrix are combined with the outputs of the variation matrix in the AND circuits of the instruction matrices. Some of the class cycle matrix outputs are also used to turn on certain commands which are common to all of the instructions within a class. For example, the contents of the left memory buffer register are added to left A register at instruction pulse IP–7 of the operation time cycle for every instruction which is in the add class. Therefore, the ADD-OT output of the class cycle matrix is used to condition the command generator that generates the left memory buffer to left A register command (43). All commands of this type can be distinguished by noting that they supply command generators directly, passing only through OR circuits and no AND circuits in the instruction matrices.

2.2.4 Variation Matrix (0.4.3.1)

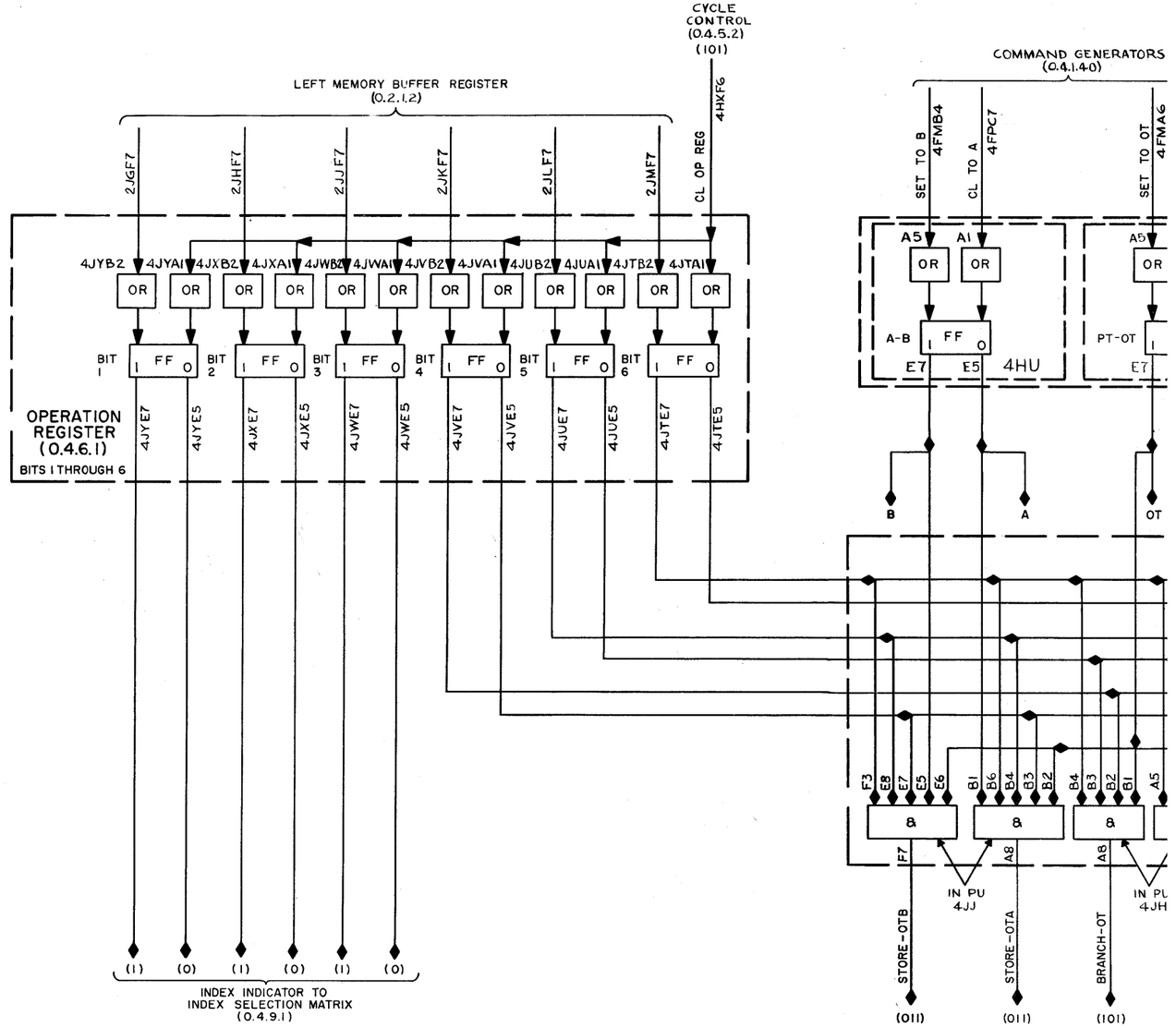
The function of the variation matrix, shown in figure 2–7, is to decode the four variation code bits of the operation register (L7 through L10). The output lines do not, by themselves, define unique variations. In order to obtain a desired variation, they must be combined with a class output line from the class cycle matrix. This combination is effected in the AND circuits of the particular instruction class matrix. Note from the figure that 16 output lines from the variation matrix are identified by four-digit numbers, while two lines are identified by only three digits (001– and 010–). The variations specified by these latter three-digit lines require an index interval. Since the index interval uses L10, the bit normally occupied by the final digit of the instruction variation code, the circuit design allows the three digits to completely specify the variations involving an index interval. The variations are:

a. *Operate* (Miscellaneous Class)	000 001–
b. *Branch and Index* (Branch Class)	101 001–
c. *Sense* (Branch Class)	101 010–
d. *Select Drum* (IO Class)	110 001–
e. *Select* (IO Class)	110 010–

2.2.5 Index Selection Matrix (0.4.9.1)

The index selection matrix, shown in figure 2–8, codes the index indicator bits of the operation register (L1 through L3), thereby selecting the index register whose contents are to be added to the address register during indexable instructions, and also specifies the register to be used during instructions in the reset class. A secondary function of this matrix is to determine which index register is to be inspected on *Branch and Index (BPX)* instructions. The use of an instruction indexing system permits the automatic modification of the address part of certain instructions without altering the basic instruction stored in core memory. A basic instruction is modified by the indexing system immediately before the execution of the instruction.

The index selection matrix has four outputs: the first of these outputs selects index register No. 1 (IX_1), the second, index register No. 2


CYCLE CONTROL (0.4.5.2) (101)
LEFT MEMORY BUFFER REGISTER (0.2.1.2)
2JGF7
2JHF7
2JJF7
2JKF7
2JLF7
2JMF7
CL OP REG
4HKF6
4JYB2
4JYA1
4JXB2
4JXA1
4JWB2
4JWA1
4JVB2
4JVA1
4JUB2
4JUA1
4JTB2
4JTA1
OR
BIT 1
BIT 2
BIT 3
BIT 4
BIT 5
BIT 6
FF
OPERATION REGISTER (0.4.6.1)
BITS 1 THROUGH 6
4JYE7
4JYE5
4JXE7
4JXE5
4JWE7
4JWE5
4JVE7
4JVE5
4JUE7
4JUE5
4JTE7
4JTE5
(1) (0) (1) (0) (1) (0)
INDEX INDICATOR TO INDEX SELECTION MATRIX (0.4.9.1)
COMMAND GENERATORS (0.4.1.4-0)
SET TO B
4FMB4
CL TO A
4FPC7
SET TO OT
4FMA6
A5
A1
A-B
E7
E5
4HU
PT-OT
B
A
OT
F3
E8
E7
E5
E6
B1
B6
B4
B3
B2
B4
B3
B2
B1
A5
8
F7
A8
IN PU
4JJ
IN PU
4JH
STORE-OTB
STORE-OTA
BRANCH-OT
(011)
(011)
(101)

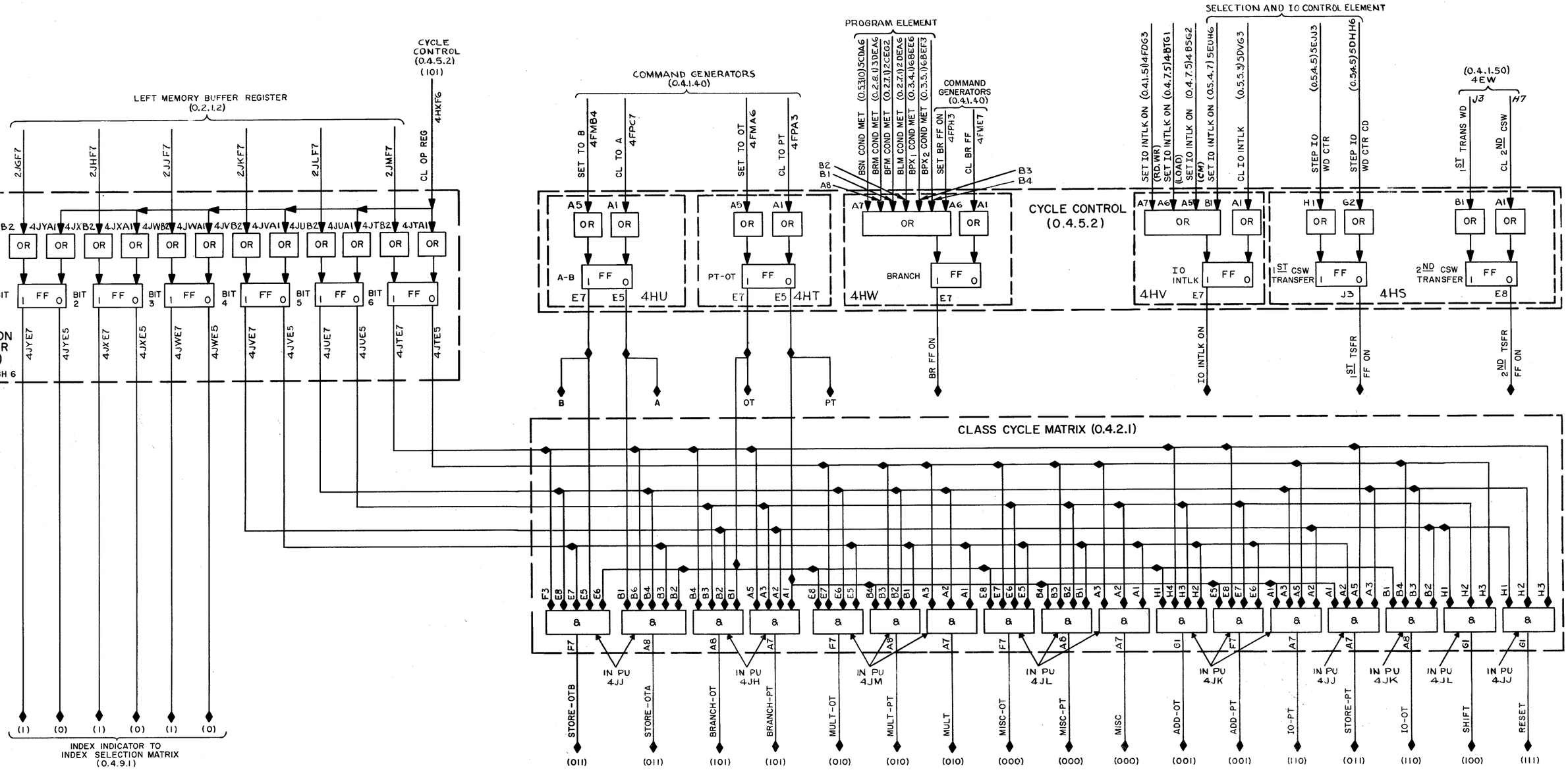


Figure 2–6. Cycle Control and Class Cycle Matrix, Logical Block Diagram

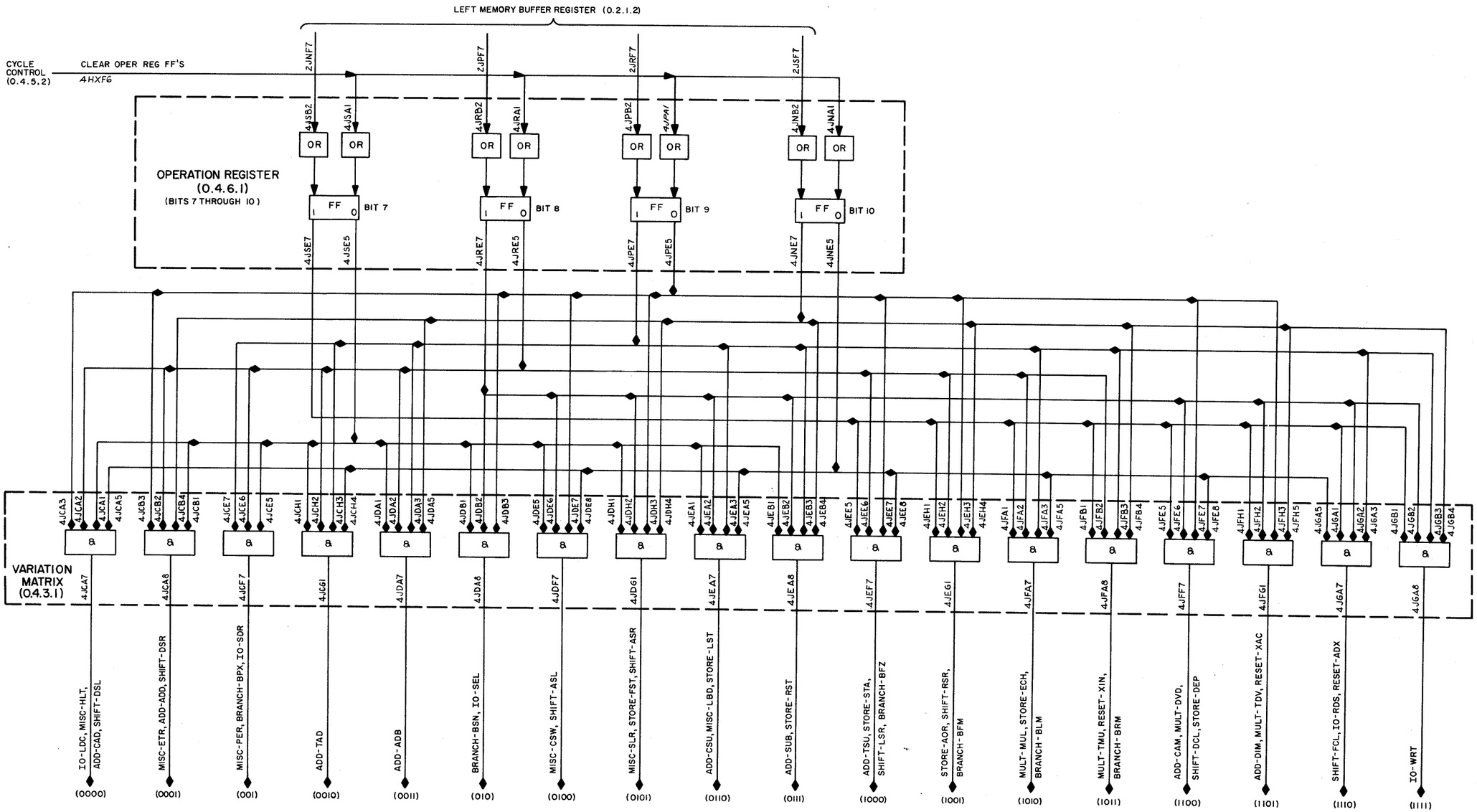


Figure 2–7. Variation Matrix, Logical Block Diagram

(IX_2), the third, the right accumulator register (used as IX_3), and the fourth is used to indicate that no index register is to be used (IX_0). The outputs of this matrix are applied to those class instruction matrices that employ indexable instructions.

2.2.6 Instruction Matrices

The preceding discussion concerning instruction decoding has described the operation of three matrices (class cycle, variation, and index selection) which, in combination with the memory unit selection control discussed in 2.3 of this Chapter, are collectively termed the control matrices. The outputs of the instruction matrix are the d-c levels for the 48 instructions required by the Central Computer System.

When the d-c levels for a particular instruction are up (positive), all commands required to carry out that instruction are turned on by connecting these d-c lines to the proper command generators (gate tubes). Pulses from the time pulse distributor cause the necessary commands to be

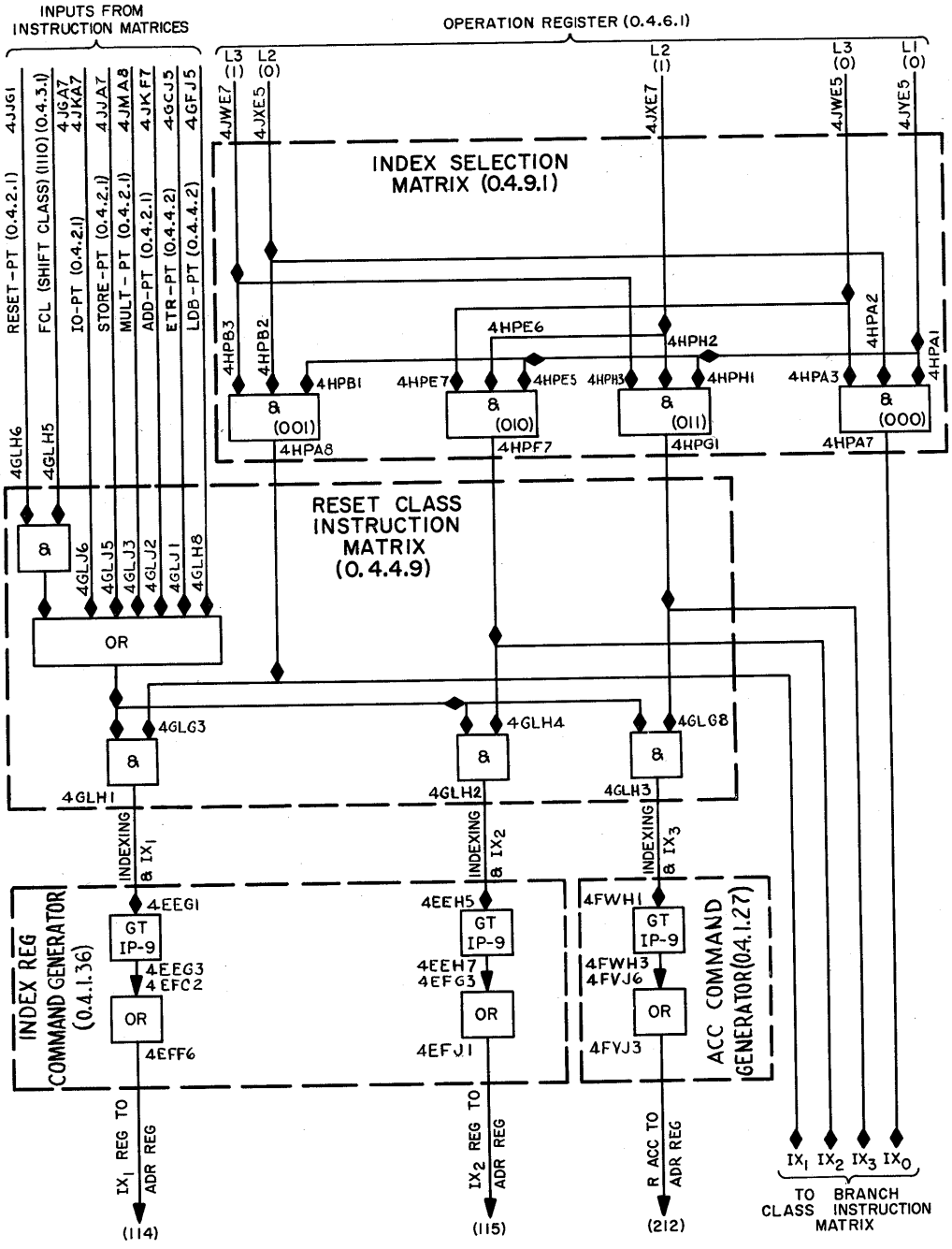


Figure 2–8. Index Selection Matrix, Logical Block Diagram

executed at the proper time and in the proper sequence. In many cases, the same command is required at the same pulse time by several different instructions. In this case only one command generator is provided and several instruction lines are combined in a diode OR circuit so they all can condition the same command generator gate tube.

As indicated above, the instruction matrix is divided into eight smaller matrices, one for each of the eight class instructions; i.e., miscellaneous, add, multiply, store, shift, branch, input-output, and reset. Two sources of information, a logical block diagram and a timing chart, are used in describing the operation of a class instruction matrix. The logical block diagram shows the interconnections of the AND and OR circuits which provide the signal channeling required in each instruction matrix. Eight block diagrams, one for each of the eight instruction classes, are provided for this purpose. The timing chart lists the sequence of commands required to execute a given instruction. It should be noted that the timing charts begin at PT–7 of one program cycle, and end at PT–6 of the succeeding program cycle. This is done to conform with the actual timing schedule used by the instruction control element. Eight timing charts are provided, one for each class. Each timing chart displays all the variations in its class. Table 2–2 lists all the commands generated by the instruction control element. The following descriptions treat the operation of only one element, but it must be remembered that both arithmetic elements are performing identical operations. Distinctions will be made whenever this is not the case.

TABLE 2–2. INSTRUCTION CONTROL ELEMENT COMMANDS

COMMAND NO.	COMMAND NAME
A	*inhibit sample No. 2*
A6	*clock register to right memory buffer register*
C	*clear left test register*
D	*left memory buffer register to left test register*
F	*test memory to memory buffer registers*
G	*clear right test register*
H	*right memory buffer register to right test register*
1	*left accumulator register to left B register*
2	*left accumulator register (2–15) to (1–14)*
4	*left accumulator register 1 to sign*
5	*left accumulator register sign to left B register 15*
6	*left accumulator register sign to 15 register sign*
7	*left accumulator register 15 to left B*
8	*clear left sign control*
9	*left correct sign*
10	*clear left accumulator register*
11	*correct left remainder*
12	*left accumulator register sign to right accumulator register 15*
13	*make left accumulator register positive*
14	*right end carry after add 1*
16	*make left accumulator register and left B register positive*
17	*left accumulator register to left memory buffer register*
18	*ripple left accumulator register right*
19	*complement left accumulator register*
20	*make right accumulator register and right B register positive*
21	*clear left A register*
22	*make left A register positive*
23	*left A register to left memory buffer register*
25	*left logical multiply*
26	*complement left A register*
31	*clear memory address register and clear test memory address register*
33	*inhibit sample No. 1*
39	*left memory buffer register to left B register*
40	*right memory buffer register to right B register*
41	*clear left memory buffer register*
42	*left memory buffer register to operation register*
43	*left memory buffer register to left A register*
44	*left memory buffer register to right A register*

TABLE 2-2. INSTRUCTION CONTROL ELEMENT COMMANDS (cont'd)

COMMAND NO.	COMMAND NAME
47	*parity count*
51	*right memory buffer register to right A register*
52	*right memory buffer register to address register*
53	*clear right memory buffer register*
55	*parity check*
60	*left accumulator conditional shift left*
61	*make left A register and left accumulator signs unlike*
62	*left carry 1*
63	*left end carry*
64	*left carry 0*
66	*record left overflow*
67	*complement left divide connect flip-flop*
69	*right carry 1*
71	*address register to memory address register*
72	*address register to IO address counter*
73	*add 1 to step counter*
74	*set step counter to 17*
75	*set step counter to 15*
76	*address register to right A register*
77	*clear address register and step counter*
78	*address register to index register No. 2*
79	*address register to index register No. 1*
80	*partial quotient*
81	*left B register sign to left accumulator register 15*
82	*left B register (1–15) to (sign–14)*
83	*left partial product*
84	*clear left B register*
85	*left B register (sign–14) to (1–15)*
87	*left round off*
88	*left B register to left A register*
89	*left B register store to left B register sign*
91	*program counter to memory address register*
92	*add 1 to program counter*
93	*program counter to right A register*
94	*clear program counter*
101	*clear operation register*
103	*index interval complement to address register*
104	*sense operate gate tube*
113	*clear index register No. 1*
114	*index register No. 1 to address register*
115	*index register No. 2 to address register*
116	*clear index register No. 2*
121	*right accumulator register to right memory buffer register*
123	*ripple right accumulator register right*
131	*set PT-OT flip-flop to OT*
132	*set A-B flip-flop to B*
134	*pause*
138	*start 2-mc. pulses*
140	*sense for branch*
144	*clear IO registers*
146	*clear drum control register*
147	*clear right IO register*
148	*clear IO address counter*
149	*clear IO word counter*
150	*IO word counter to right accumulator register*
151	*address register to drum control register*
152	*address register complement to IO word counter*
154	*address register to program counter*
155	*deselect pulse*
156	*select pulse*
157	*clear second CSW*
158	*1st or 2nd IO word counter transfer to right accumulator register*
161	*clear PT-OT flip-flop to PT*
162	*test right and left accumulator registers*
163	*clear branch flip-flop*
164	*test index register No. 1 sign for 0*
165	*test the left accumulator register*
166	*test right accumulator sign for 1*
167	*clear A-B flip-flop to A*
170	*set branch flip-flop*

TABLE 2–2. INSTRUCTION CONTROL ELEMENT COMMANDS (cont'd)

COMMAND NO.	COMMAND NAME
174	*test index register No. 2 sign for 0*
180	*PT–6 on read*
182	*PT–6 on read or write*
201	*right accumulator register to right B register*
202	*right accumulator register (2–15) to (1–14)*
204	*right accumulator register 1 to sign*
205	*right accumulator register sign to right B register 15*
206	*right accumulator register sign to 15*
207	*right accumulator 15 to right B register sign*
208	*clear right sign control*
209	*right correct sign*
210	*clear right accumulator register*
211	*correct right remainder*
212	*right accumulator register to address register*
213	*make right accumulator register positive*
214	*left round off sign*
216	*right round off sign*
217	*right accumulator register sign to left accumulator register 15*
219	*complement right accumulator register*
225	*right logical multiply*
226	*complement right A register*
228	*right A register to right memory buffer register*
229	*make right A register positive*
230	*clear right A register*
260	*right accumulator register conditional shift left*
261	*make right A register and right accumulator register signs unlike*
263	*right end carry*
264	*right carry 0*
266	*record right overflow*
267	*complement right divide connect flip-flop*
270	*clear continue flip-flop*
281	*right B register sign to right accumulator register 15*
282	*right B register (1–15) to (sign–14)*
283	*right partial product*
284	*clear right B register*
285	*right B register (sign–14) to (1–15)*
287	*right round off*
288	*right B register to right A register*
289	*right B register store to right B register sign*
290	*sense IO interlock 0*
294	*set IO interlock on*
321	*sense tapes for not ready*
325	*select pulse for drums*

2.2.6.1 Common Commands

Several commands are common to all eight classes. These commands are:

a. 31—*clear memory address register and clear test memory address register*
b. 41—*clear left memory buffer register*
c. 53—*clear right memory buffer register*
d. 42—*left memory buffer register to operate register*
e. 47—*parity count*
f. 52—*right memory buffer register to address register*
g. 55—*parity check*
h. 77—*clear address register and step counter*
i. 91—*program counter to memory address register*
j. 92—*add 1 to program counter*
k. 101—*clear operation register*
l. 131—*set PT–OT flip-flop to OT*
m. 161—*clear PT–OT flip-flop to PT*

Whenever an instruction is being executed, the d-c levels at the command generators are always up for commands 42, 52, 77, 91, 92, and 101 during program time. This is accomplished by using the PT line directly from the cycle control to raise the d-c levels used by the command generators forming these commands. Similarly, command 71 is constantly generated during the operate time of an instruction by utilizing the OT line

from the cycle control. However, certain instructions do not require an operate time cycle. In these cases, the OT line from the cycle control is not energized and command 71 is never generated during these instructions.

Commands 31, 41, 47, and 53 do not use command generators. These commands consist of pulses which are supplied by the time pulse distributor. Command 31 is generated by TP–0; commands 41 and 53 by TP–1; and command 47 by IP–7.

Command 55 is similar to the commands listed in the previous paragraph in that it occurs at every IP–11. However, it is formed by a command generator which is conditioned by the memory unit selection matrix. Therefore, command 55 will be generated only if core memory No. 1 or core memory No. 2 is in use at the time that IP–11 is being distributed to the instruction control element.

The common commands, discussed in the preceding paragraphs, always occur at the indicated times for all instructions. In the discussions of the class instructions, only the *SLR* instruction will include the common commands. For the remaining discussions, it is assumed that the reader will interpose these common commands where necessary.

The cycle control contains the PT–OT flip-flop which determines the mode of operation (PT or OT) in which the Central Computer System is operating. Two commands, 131 and 161, control the condition of this flip-flop. These commands are necessary only if a particular instruction requires an OT cycle. When required, these commands occur during the final instruction pulse of PT and OT, respectively (IP–11). For those instructions not requiring an OT cycle, the flip-flop will always be in PT status. This condition is set by the last previous instruction containing an OT cycle, since command 161 is the final instruction in every OT cycle. The command generator for 131 is conditioned by an OR circuit through which pass signals from all the instructions requiring an OT cycle. The command generator for 161 is conditioned directly by the OT line for the particular class to which the instruction being performed belongs.

Four instruction classes, add, multiply, store, and IO, are indexable. The *Extract* and *Load B Registers* variations of the miscellaneous class fall in this same category. Three AND circuits and an OR circuit in the reset class instruction matrix select the desired register. The OR circuit conditions the three AND circuits which receive their other conditioning inputs from the index selection matrix. Only one of the AND circuits is conditioned in accordance with information originating in the operation register. In the case of the indexable classes, the OR circuit is fed by the program time (PT) line from each of the four classes. Accordingly, the combination of a class PT line and a condition line from the index selection matrix generates the command (114, 115, or 212) which selects the desired index register. The same selection process is used for the *Extract* instruction, but in this instance the OR circuit in the reset class is fed by the *Extract* line (a variation matrix output) rather than the miscellaneous-PT line (a class cycle matrix output).

2.2.6.2 Miscellaneous Class

The miscellaneous class of instructions, figures 2–9 and 2–10, contains those instructions which did not fit the other classes particularly well. The *Shift Left and Round (SLR)* requires one cycle and a pause, and the *Clear and Subtract Word Counter (CSW)* requires one cycle. The *Extract (ETR), Program Stop (HLT), Operate (PER), and Load B Registers (LDB)* variations each require two cycles for execution.

The *Shift Left and Round* instruction is a shift left operation followed by a round operation whenever the necessary conditions are met. Transfer of information from the memory buffer register to the instruction register is performed by common commands 42 and 52 at PT–7. At the same time, common command 92 adds 1 to the program counter, preparing it for the next instruction, and a parity count is performed by common command 47. At PT–9, preparation is made for a shift operation by setting the 2-megacycle flip-flop (138). The shift operation begins at PT–10 by initiation of 2-megacycle command pulses (134). The time pulse distributor is stopped at PT–11, during which time a parity check is effected by common command 55. The number of times to shift is specified by the step counter, set during program time to the value indicated by the address part of the instruction. All commands generated during the pause are the result of combining the MISC line from the class cycle matrix with the 0101 *(SLR)* line from the variation matrix in an AND circuit of the miscellaneous class instruction matrix. The output of this AND circuit conditions four command generators. These command generators receive a continuous train of 2-megacycle pulses for the duration of the pause. These generators provide nine commands (2, 202, 5, 205, 81, 281, 82, 282, and

73) which are simultaneously fed to the arithmetic elements for the duration of the pause. The proper channeling of these commands is then accomplished in the arithmetic elements.

A 1 is subtracted from the step counter for every 2-megacycle operation. When the step counter is reduced to 0, the 2-megacycle shift operation is completed, the time pulse distributor is restarted, and a new program time cycle is initiated at PT–0. At PT–0, the memory address registers and the unit memory selection control flip-flops are cleared (31) in preparation for the next instruction. (The insertion of this instruction is initiated at PT–2 by a start memory pulse. Neither of these latter operations interfere with the continuation of the *SLR* instruction.) To complete the *SLR* instruction, at PT–1 the A register is cleared (21, 230) and both the accumulator register and the B register are complemented (16, 20), provided the sign of the accumulator register is negative. Common commands 41, 53, and 91 also occur at PT–1. At PT–2, a round-off command (87, 287) is executed, provided the sign of the B register is negative. The carry-1 line of the adder is pulsed to begin the addition. The carry-1 operation is followed at PT–5 by a shift operation (60, 260). At PT–6 the B register is cleared (84, 284) and the accumulator register is complemented (214, 216) if its sign control flip-flop is indicating negative. Common commands 77 and 101 also occur at this pulse time, clearing the address and operation registers.

The *Clear and Subtract Word Counter* instruction replaces the contents of the right accumulator register with that of the IO word counter. The required transfer of information is done in one of two ways. If the IO word counter is not being stepped at PT–1, the transfer is effected at PT–1 (150, 158). If the IO word counter is being stepped at PT–1, the transfers occur at PT–5 (158). In the first instance, the purpose of 150 is to decondition 158 so that it does not occur at PT–5. At PT–9, command 157 reconditions 158, and command 210 clears the right accumulator register. These last two commands prepare the Central Computer System for the next instruction.

During the *Extract* instruction, the contents of the accumulator registers are compared with the word in the memory address specified by the address part of the instruction. Accumulator bits in digit positions which have 1's in the corresponding digit positions of the specified word in core memory are unchanged. All other digit positions of the accumulators are reset to 0. The contents of the specified memory register are not changed by this instruction. Since this instruction is indexable, the address part is modified, if required, at PT–9 (114, 115, or 212). At PT–11, the cycle control is set to operate time (131). At OT–1, the A register is cleared (21, 230). The contents of the memory buffer register are transferred to the A register at OT–7 (43, 51). At OT–9, the *logical multiply* command (25, 225) is executed; i.e., if bit X of the A register is 0, bit X of the accumulator register is set to 0; if bit X of the A register is 1, bit X of the accumulator register is left unchanged. The command occurring at OT–11 for this instruction (161) returns the PT–OT flip-flop to PT, and, as is usually the case, the instruction is completed during the first half of the following program time cycle (PT–1 to PT–6).

The *Program Stop* instruction stops the Central Computer System. The address part of this instruction is meaningless. The pause flip-flop is set at PT–10 (134). The pause in this instruction is used only if the IO interlock is on. The IO interlock is automatically sensed at PT–11. If it is on, the pause goes into effect and the succeeding operate time cycle is held up until the IO interlock is off. As soon as the IO break cycle is completed, the operate time cycle is initiated. Command 270, *set stop flip-flop,* is generated at OT–11. On instruction pulse 6 of the following program time cycle, the Central Computer System stops. The 3.0 microseconds from PT–0 to PT–6 are utilized to clear the operation register and to set the contents of the program counter into the memory address register. On restarting, the Central Computer System will resume at PT–7 with the instruction whose address has been held in the memory buffer registers, and the program counter will simultaneously be advanced by 1.

The *Operate* instruction serves to transmit signals from the control element of the computer system to the operate units. The address part of this instruction is meaningless, but the last six bits, L10 through L15, of the left half-word of this instruction identify the selected operate unit. At OT–9, the operate gate tube is sensed (104).

The *Load B Registers* instruction replaces the contents of the B register with the word in the memory address specified by the address part of the instruction. The contents of the indicated memory address and the accumulator register remain unchanged. Prior to OT–6, the word specified by the address part of this instruction is extracted from memory and placed in the memory buffer register. At OT–6, the B register is cleared (84,

MISCELLANEOUS CLASS COMMANDS

(Fig. 2–9.)

COMMAND NUMBER	COMMAND NAME	PLUGGABLE UNIT NO.	LOGICAL REFERENCE NO.
2	*left accumulator register (2–15) to (1–14)*	4GY	(0.4.1.27)
5	*left accumulator register sign to left B register 15*	4FY	(0.4.1.27)
16	*make left accumulator register and left B register positive*	4FV	(0.4.1.27)
20	*make right accumulator register and right B register positive*	4FV	(0.4.1.27)
21	*clear left A register*	4EM	(0.4.1.23)
25	*left logical multiply*	4EM	(0.4.1.23)
31	*clear memory address register and clear test memory address register*	4CE	(0.4.7.1)
39	*left memory buffer register to left B register*	4DX	(0.4.1.21)
40	*right memory buffer register to right B register*	4DX	(0.4.1.21)
41	*clear left memory buffer register*	4CF	(0.4.7.1)
42	*left memory buffer register to operation register*	4DX	(0.4.1.21)
43	*left memory buffer register to left A register*	4CF	(0.4.7.1)
47	*parity count*	4DY	(0.4.1.21)
51	*right memory buffer register to right A register*	4DX	(0.4.1.21)
52	*right memory buffer register to address register*	4DX	(0.4.1.21)
53	*clear right memory buffer register*	4CF	(0.4.7.1)
55	*parity check*	4DX	(0.4.1.21)
60	*left accumulator conditional shift left*	4ER	(0.4.1.25)
71	*address register to memory address register*	4EH	(0.4.1.33)
73	*add 1 to step counter*	4FP	(0.4.1.40)
77	*clear address register and step counter*	4EJ	(0.4.1.33)
81	*left B register sign to left accumulator register 15*	4FJ	(0.4.1.29)
82	*left B register (1–15) to (sign–14)*	4FH	(0.4.1.29)
84	*clear left B register*	4FK	(0.4.1.29)
87	*left round off*	4FH	(0.4.1.29)
91	*program counter to memory address register*	4EH	(0.4.1.33)
92	*add 1 to program counter*	4EH	(0.4.1.33)
101	*clear operation register*	4FF	(0.4.1.51)
104	*sense operate gate tube*	4FD	(0.4.1.51)
114	*index register No. 1 to address register*	4EF	(0.4.1.36)
115	*index register No. 2 to address register*	4EF	(0.4.1.36)
131	*set PT–OT flip-flop to OT*	4FM	(0.4.1.40)
134	*pause*	4FP	(0.4.1.40)
138	*start 2-mc. pulses*	4FN	(0.4.1.40)
150	*IO word counter to right accumulator register*	4EW	(0.4.1.50)
157	*clear second CSW*	4EW	(0.4.1.50)
158	*1st or 2nd IO word counter transfer to right accumulator register*	4EX	(0.4.1.50)
161	*clear PT–OT flip-flop to PT*	4FP	(0.4.1.40)
202	*right accumulator register (2–15) to (1–14)*	4GX	(0.4.1.27)
205	*right accumulator register sign to right B register 15*	4FY	(0.4.1.27)
210	*clear right accumulator register*	4FV	(0.4.1.27)
212	*right accumulator register to address register*	4FV	(0.4.1.27)
214	*left round off sign*	4FT	(0.4.1.27)
216	*right round off sign*	4FT	(0.4.1.27)
225	*right logical multiply*	4EM	(0.4.1.23)
230	*clear right A register*	4EM	(0.4.1.23)
260	*right accumulator register conditional shift left*	4ER	(0.4.1.25)
270	*clear continue flip-flop*	4FP	(0.4.1.40)
281	*right B register sign to right accumulator register 15*	4FJ	(0.4.1.29)
282	*right B register (1–15) to (sign–14)*	4FH	(0.4.1.29)
284	*clear right B register*	4FK	(0.4.1.29)
287	*right round off*	4FH	(0.4.1.29)

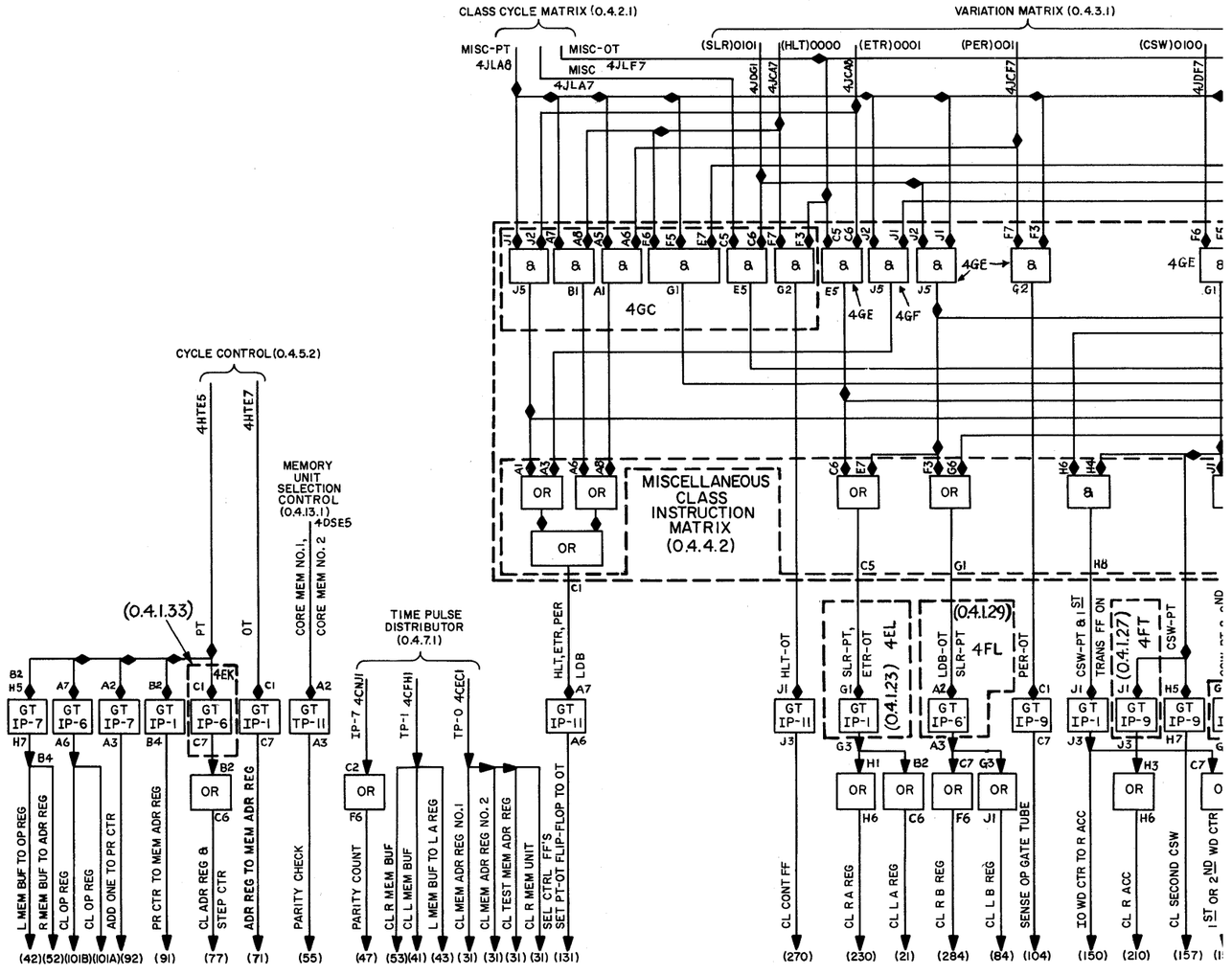


Figure 2–9. Miscellaneous Class, Logical Block Diagram

284). At OT–7, the contents of the memory buffer register are placed in the B register (39, 40).

2.2.6.3 Add Class

The add class, shown in figures 2–11 and 2–12, contains nine instructions, all requiring two cycles to complete. This class is indexable. The normal entrance to the accumulator registers from the A registers is through the adders. The twin-type instructions, *Twin and Add (TAD)* and *Twin and Subtract (TSU)*, involve the transfer of a left half-word in core memory to both accumulator registers, while in the non-twin instructions, the left half-word from core memory is transferred to the left accumulator register and the right half-word to the right accumulator register. The process of subtraction involves addition of complements; consequently, in any add class instruction, the contents of the A registers are added to the accumulator registers, and the sums are placed in the accumulator registers.

In addition to the common commands described in 2.2.6.1 of this Chapter, every instruction in the add class has the following commands in common:

a. Operate time (conditioned by the ADD–PT line from the cycle control):

1. 43—*left memory buffer register to left A register*
2. 21, 230—*clear A register*
3. 64, 264—*carry 0*

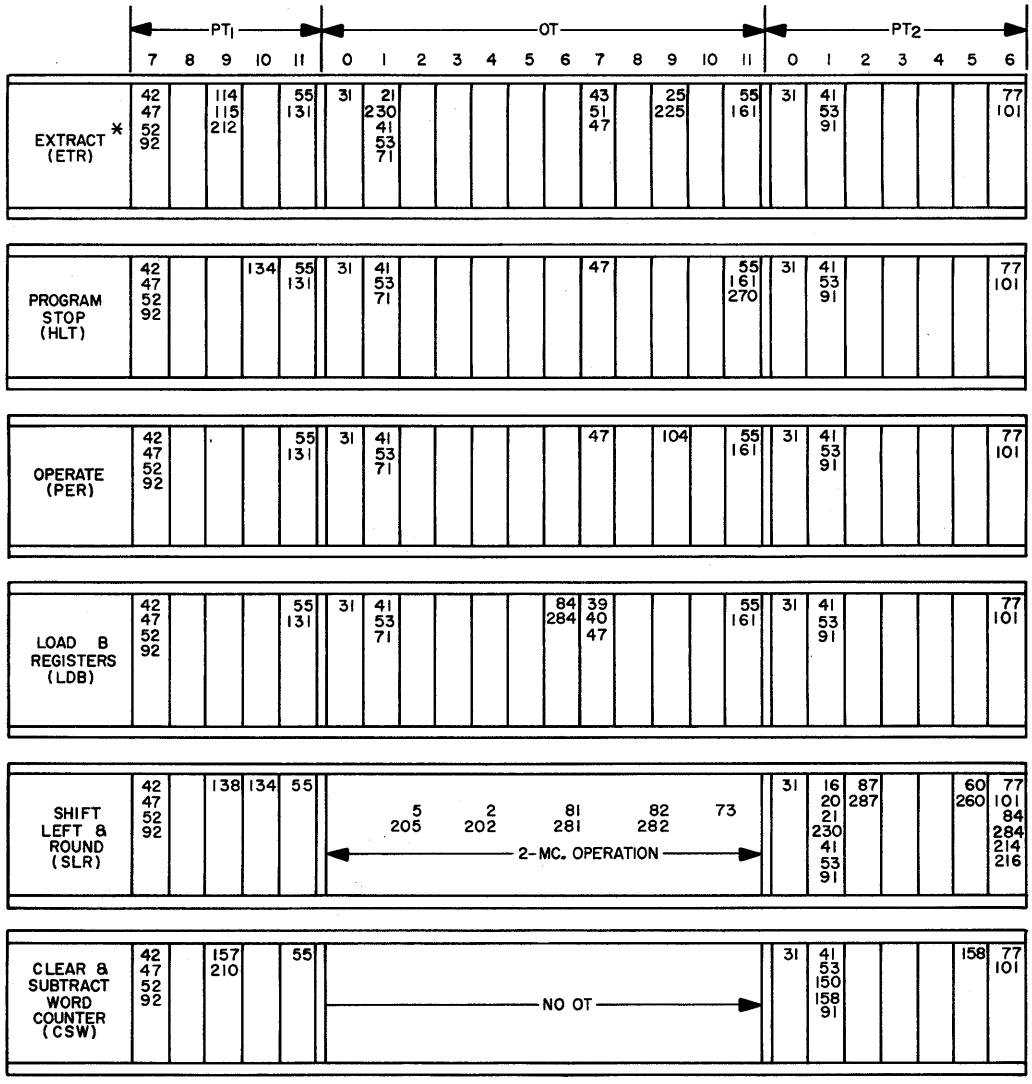


Figure 2–10. Miscellaneous Class, Command Sequence Chart

b. Program time (conditioned by the ADD–OT line from the cycle control):

1. 2, 202—*accumulator register (2–15) to (1–14)*
2. 4, 204—*accumulator register 1 to sign*
3. 21, 230—*clear A register*
4. 60, 260—*accumulator conditional shift left*
5. 63, 263—*end carry*
6. 66, 266—*record overflow*
7. 81, 281—*B register sign to accumulator register 15*
8. 89, 289—*B register store to B register*

The commands listed above and the common commands appear in the timing chart for this class and it is assumed that the reader will interpose them when necessary in the ensuing discussions of the add class instructions. Since the class is indexable, PT–9 is reserved for the indexing command (144, 155, or 212). Since every instruction in the class requires two cycles, command 131 *(set PT–OT flip-flop to OT)* always occurs at PT–11, and command 161 *(clear PT–OT flip-flop to PT)* always occurs at OT–11.

The clear type of add class instructions are *Clear and Add (CAD), Clear and Subtract (CSU),* and *Clear and Add Magnitudes (CAM).* In these instructions the accumulator register is cleared at OT–7, and the operand is extracted from core memory and placed in the A register (41, 53). On the *CAM* instruction, the A register is complemented at OT–9 (22, 229) if the sign bit of the A register is negative. On the *CSU* instruction (and all subtract-type instructions), the following operations take place: the A register is complemented by OT–9 (26, 226), and the *carry 0* command is generated at OT–10 (64, 264). On the *CAM* instruction, the A registers are conditionally made positive by commands 28 and 228, occurring at OT–8.

The operation of the twin-type instructions, *Twin and Add (TAD)* and *Twin and Subtract (TSU),* are similar to the *Add* and *Subtract* instructions, respectively. The operand is extracted from the left memory buffer register and transferred to both the left and right A registers at OT–7 (43, 44). In the subtract instructions, the A register is complemented at OT–9 (26, 226) and, at OT–10, the carry-0 lines to each of the adders are pulsed (64, 264). The A register is cleared at PT–1 (21, 230). At PT–2, the *end carry* command (63, 263) is generated. The end carry correction is followed by a *conditional shift left* command (60, 260) at PT–5. At PT–6, the *record overflow* command is given (66, 266). This pulse examines the left and right auxiliary overflow flip-flops. If either of the flip-flops is in the 1 position (indicating overflow), then the corresponding overflow flip-flop is set to an indicating condition and both auxiliary flip-flops are cleared.

In order to execute the *Add B Register to Accumulators (ADB)* instruction, the contents of the B register are placed in the A register at OT–9 (88, 288). The *carry 0* command (64, 264) to begin addition is generated at OT–10. The execution of commands 43 and 51, which operate on the A register at OT–7, is not blocked since this register is cleared at OT–8 (21, 230) before the contents of the B register are transferred.

The difference between the *Difference Magnitudes (DIM)* instruction and other add-class instructions lies in the condition which must be met before the add process occurs. After extracting the operand from core memory at OT–7 (43, 51), the A register and the sign control flip-flops are complemented at OT–8 (22, 229) if the A register sign is negative. Also at OT–8, the contents of the accumulator register are transferred to the B register (1, 201) which was previously cleared at OT–6 (84, 284). This transfer is made in order to retain the original contents of the accumulator register. At OT–9, the A registers are complemented unconditionally (26, 226). The accumulator register and the sign control flip-flops are complemented provided the accumulator register sign bit is negative (13, 213). These are the operations required before the Central Computer System may proceed with the normal add operation described above.

2.2.6.4 Multiply Class

The multiply class, shown in figures 2–13 and 2–14, contains four instructions. The difference in operation between the twin and non-twin instructions has been described in 2.2.6.3 of this Chapter.

In addition to the common commands described in 2.2.6.1 of this Chapter, every instruction in the multiply class has the following commands in common:

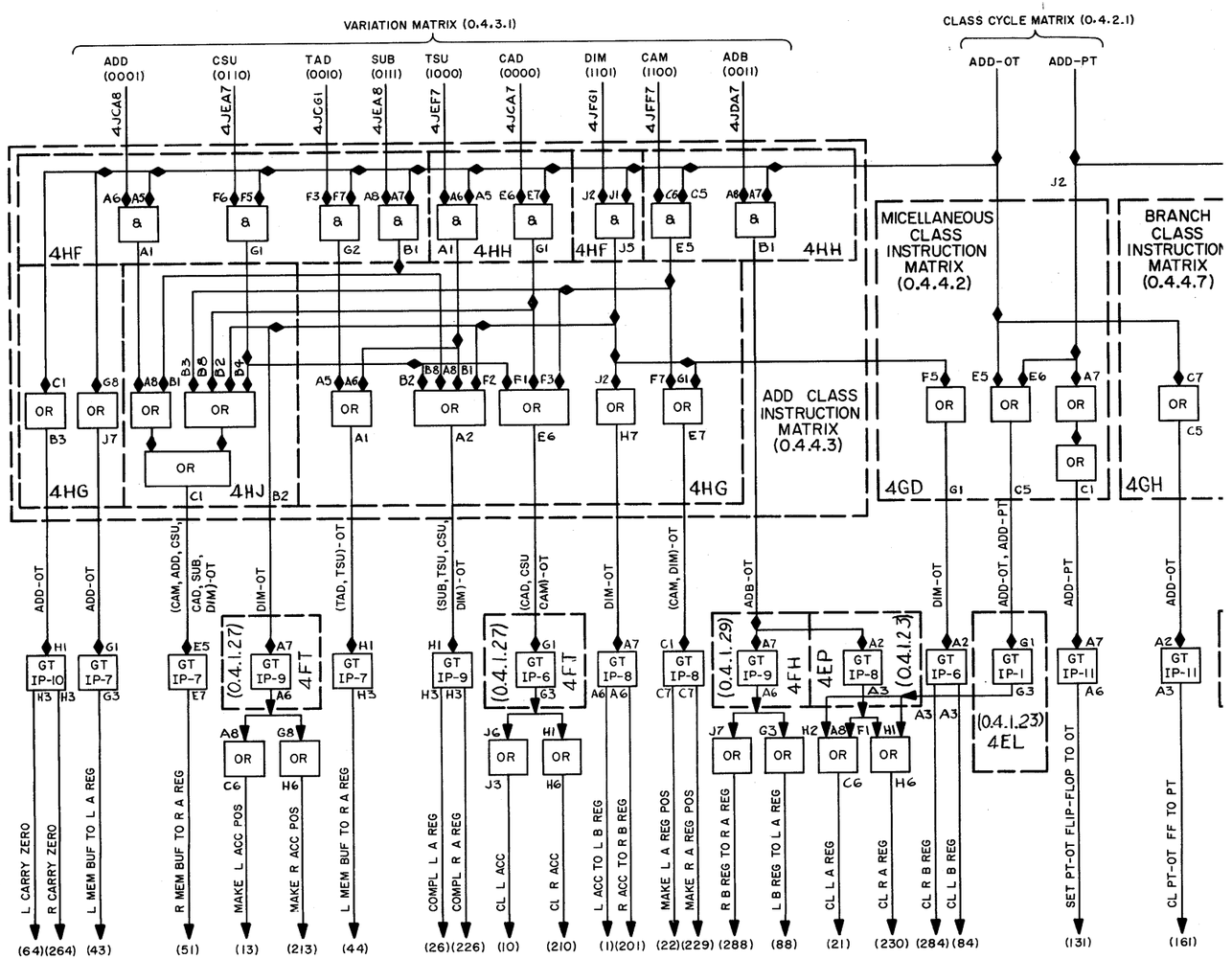
a. Program time (conditioned by the MULT–PT line from the cycle control):

1. 9, 209—*correct sign control*

ADD CLASS COMMANDS

(Fig. 2–11.)

COMMAND NUMBER	COMMAND NAME	PLUGGABLE UNIT NO.	LOGICAL REFERENCE NO.
1	*left accumulator register to left*	4FY	(0.4.1.27)
2	*left accumulator register (2–15) to (1–14)*	4GY	(0.4.1.27)
4	*left accumulator register 1 to sign*	4GY	(0.4.1.27)
10	*clear left accumulator register*	4FU	(0.4.1.27)
13	*make left accumulator register positive*	4FU	(0.4.1.27)
21	*clear left A register*	4EM	(0.4.1.23)
22	*make left A register positive*	4EL	(0.4.1.23)
26	*complement left A register*	4EL	(0.4.1.23)
31	*clear memory address register and clear test memory address register*	4DV	(0.4.1.10)
41	*clear left memory buffer register*	4DY	(0.4.1.21)
42	*left memory buffer register to operation register*	4DX	(0.4.1.21)
43	*left memory buffer register to left A register*	4DX	(0.4.1.21)
44	*left memory buffer register to right A register*	4DX	(0.4.1.21)
47	*parity count*	4DY	(0.4.1.21)
51	*right memory buffer register to right A register*	4DX	(0.4.1.21)
52	*right memory buffer register to address register*	4DX	(0.4.1.21)
53	*clear right memory buffer register*	4DY	(0.4.1.21)
55	*parity check*	4DX	(0.4.1.21)
60	*left accumulator conditional shift left*	4ER	(0.4.1.25)
63	*left end carry*	4ER	(0.4.1.25)
64	*left carry 0*	4ER	(0.4.1.25)
66	*record left overflow*	4ER	(0.4.1.25)
71	*address register to memory address register*	4EH	(0.4.1.33)
77	*clear address register and step counter*	4EJ	(0.4.1.33)
81	*left B register sign to left accumulator register 15*	4FJ	(0.4.1.29)
84	*clear left B register*	4FL	(0.4.1.29)
88	*left B register to left A register*	4FJ	(0.4.1.29)
89	*left B register store to left B register sign*	4FK	(0.4.1.29)
91	*program counter to memory address register*	4EH	(0.4.1.33)
92	*add 1 to program counter*	4EH	(0.4.1.33)
101	*clear operation register*	4FF	(0.4.1.51)
114	*index register No. 1 to address register*	4EF	(0.4.1.36)
115	*index register No. 2 to address register*	4EF	(0.4.1.36)
131	*set PT–OT flip-flop to OT*	4FM	(0.4.1.40)
161	*clear PT–OT flip-flop to PT*	4FP	(0.4.1.40)
201	*right accumulator register to right B register*	4FY	(0.4.1.27)
202	*right accumulator register (2–15) to (1–14)*	4GX	(0.4.1.27)
204	*right accumulator register 1 to sign*	4GX	(0.4.1.27)
210	*clear right accumulator register*	4FV	(0.4.1.27)
212	*right accumulator register to address register*	4FV	(0.4.1.27)
213	*make right accumulator register positive*	4FU	(0.4.1.27)
226	*complement right A register*	4EL	(0.4.1.23)
229	*make right A register positive*	4EL	(0.4.1.23)
230	*clear right A register*	4EM	(0.4.1.23)
260	*right accumulator register conditional shift left*	4ER	(0.4.1.25)
263	*right end carry*	4ER	(0.4.1.25)
264	*right carry 0*	4ER	(0.4.1.25)
266	*record right overflow*	4ER	(0.4.1.25)
281	*right B register sign to right accumulator register 15*	4FJ	(0.4.1.29)
284	*clear right B register*	4FL	(0.4.1.29)
288	*right B register to right A register*	4FK	(0.4.1.29)
289	*right B register store to right B register sign*	4FK	(0.4.1.29)



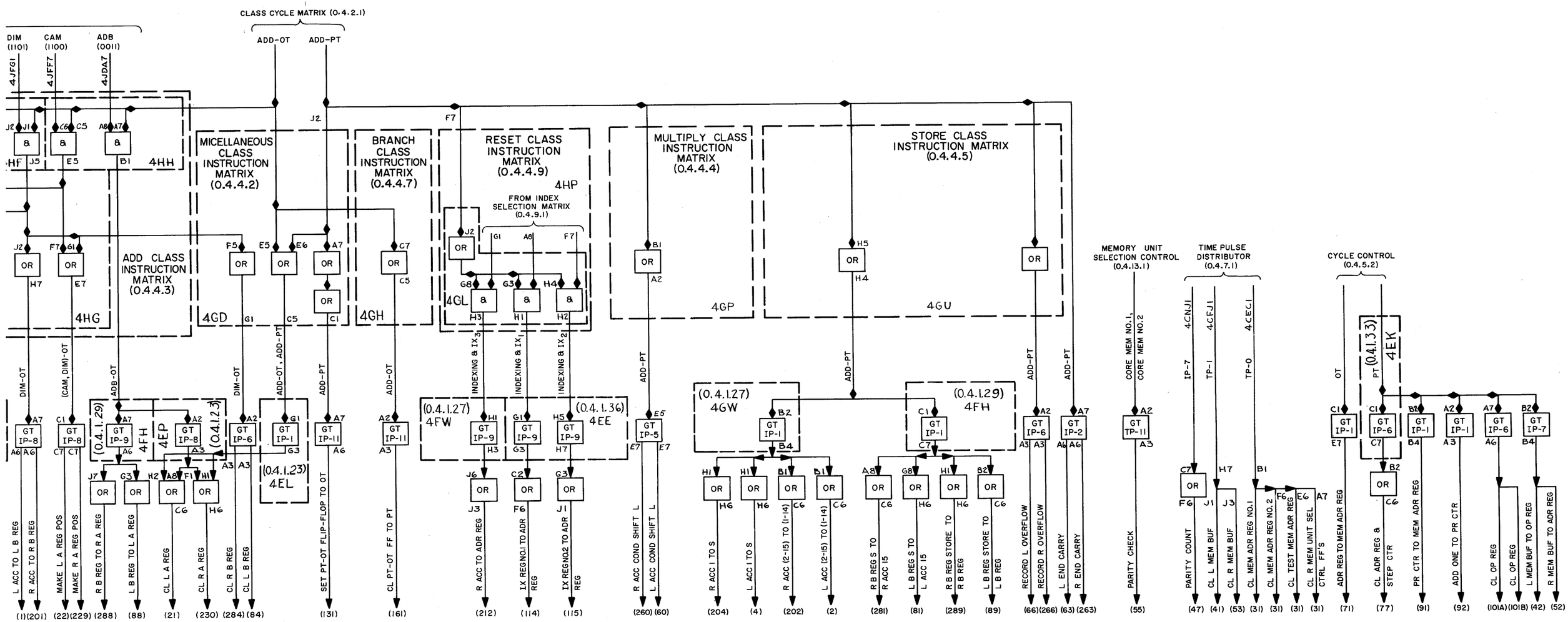


Figure 2-11. Add Class, Logical Block Diagram

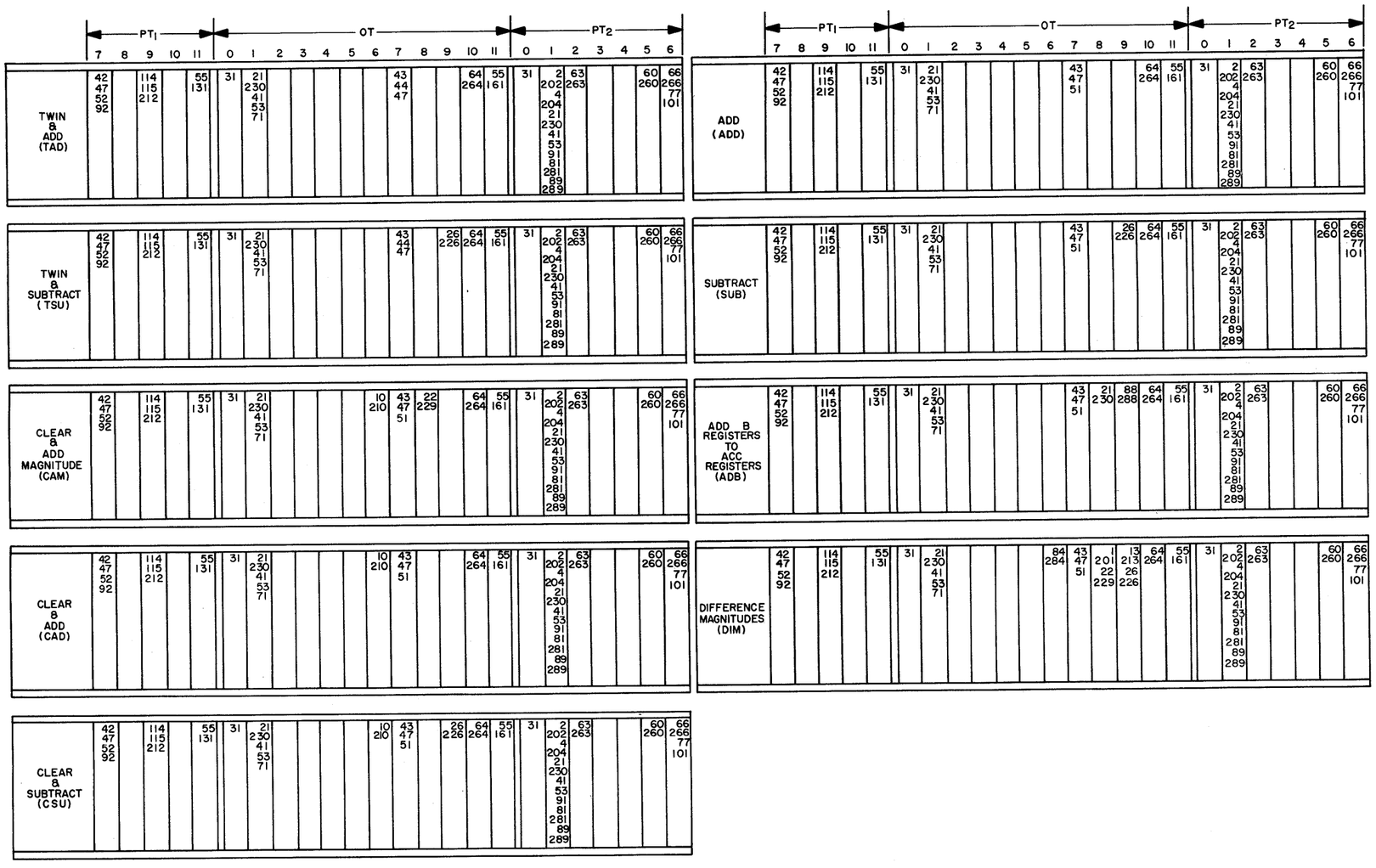
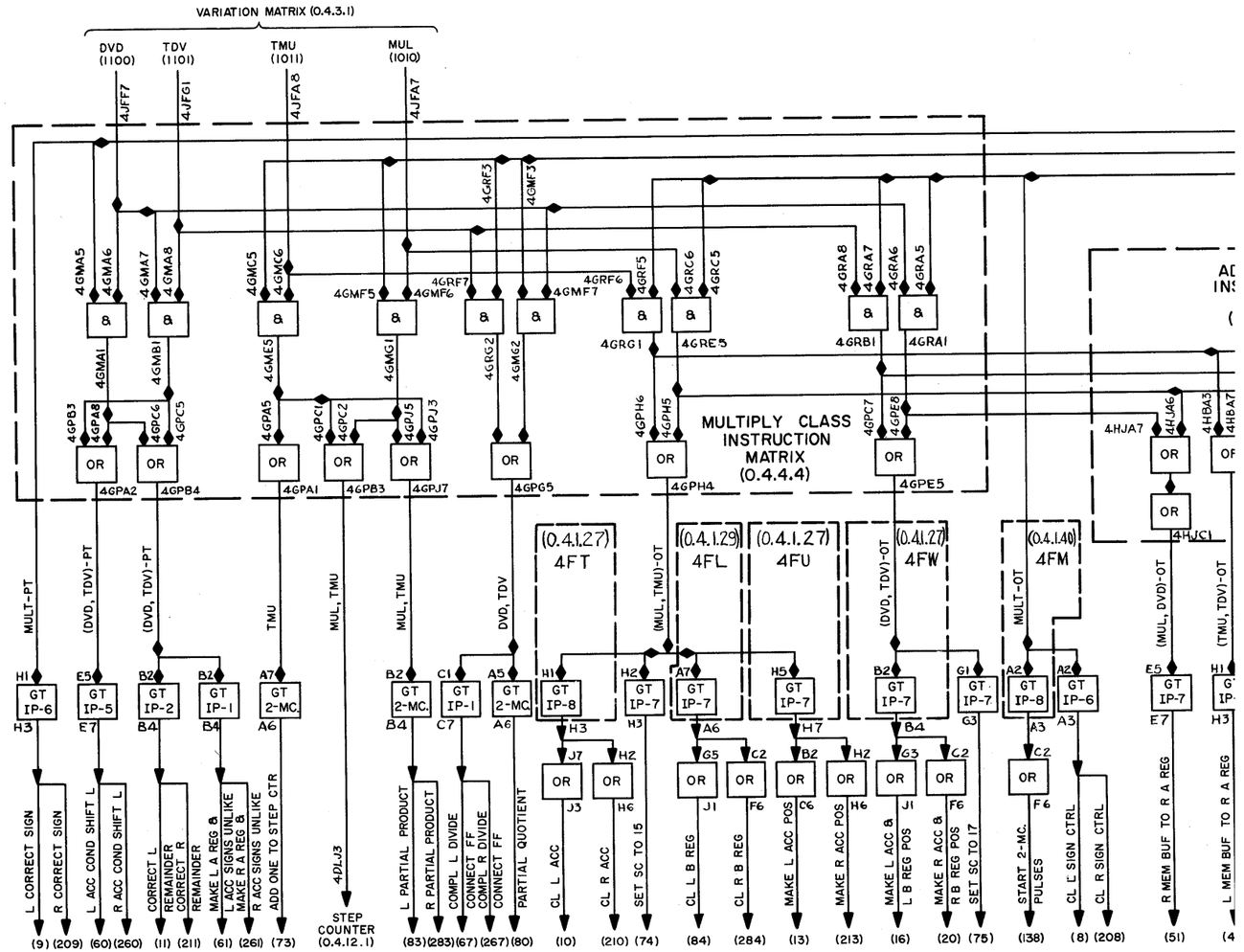


Figure 2-12. Add Class, Command Sequence Chart

MULTIPLY CLASS COMMANDS

(Fig. 2–13.)

COMMAND NUMBER	COMMAND NAME	PLUGGABLE UNIT NO.	LOGICAL REFERENCE NO.
1	*left accumulator register to left B register*	4FY	(0.4.1.27)
8	*clear left sign control*	4ER	(0.4.1.25)
9	*left correct sign*	4GW	(0.4.1.27)
10	*clear left accumulator register*	4FU	(0.4.1.27)
11	*correct left remainder*	4EP	(0.4.1.23)
13	*make left accumulator register positive*	4FU	(0.4.1.27)
16	*make left accumulator register and left B register positive*	4FV	(0.4.1.27)
20	*make right accumulator register and right B register positive*	4FV	(0.4.1.27)
21	*clear left A register*	4EM	(0.4.1.23)
22	*make left A register positive*	4EL	(0.4.1.23)
31	*clear memory address register and clear test memory address register*	4DV	(0.4.1.10)
41	*clear left memory buffer register*	4DX	(0.4.1.21)
42	*left memory buffer register to operation register*	4DX	(0.4.1.21)
43	*left memory buffer register to left A register*	4DX	(0.4.1.21)
44	*left memory buffer register to right A register*	4DX	(0.4.1.21)
47	*parity count*	4DY	(0.4.1.21)
51	*right memory buffer register to right A register*	4DX	(0.4.1.21)
52	*right memory buffer register to address register*	4DX	(0.4.1.21)
53	*clear right memory buffer register*	4DX	(0.4.1.21)
55	*parity check*	4DX	(0.4.1.21)
60	*left accumulator conditional shift left*	4ER	(0.4.1.25)
61	*make left A register and left accumulator signs unlike*	4ER	(0.4.1.25)
67	*complement left divide connect flip-flop*	4ER	(0.4.1.25)
71	*address register to memory address register*	4EH	(0.4.1.33)
73	*add 1 to step counter*	4EP	(0.4.1.40)
74	*set step counter to 17*	4FM	(0.4.1.40)
75	*set step counter to 15*	4FM	(0.4.1.40)
77	*clear address register and step counter*	4EJ	(0.4.1.33)
80	*partial quotient*	4FR	(0.4.1.40)
82	*left B register (1–15) to (sign–14)*	4FL	(0.4.1.29)
83	*left partial product*	4FK	(0.4.1.29)
84	*clear left B register*	4EH	(0.4.1.33)
91	*program counter to memory address register*	4EH	(0.4.1.33)
92	*add 1 to program counter*	4FF	(0.4.1.51)
101	*clear operation register*	4EF	(0.4.1.36)
114	*index register No. 1 to address register*	4EF	(0.4.1.36)
115	*index register No. 2 to address register*	4FM	(0.4.1.40)
131	*set PT-OT flip-flop to OT*	4FP	(0.4.1.40)
134	*pause*	4FN	(0.4.1.40)
138	*start 2-mc. pulses*	4FP	(0.4.1.40)
161	*clear PT-OT flip-flop to PT*	4FY	(0.4.1.27)
201	*right accumulator register to right B register*	4ER	(0.4.1.25)
208	*clear right sign control*	4GW	(0.4.1.27)
209	*right correct sign*	4FV	(0.4.1.27)
210	*clear right accumulator register*	4EP	(0.4.1.23)
211	*correct right remainder*	4FV	(0.4.1.27)
212	*right accumulator register to address register*	4FU	(0.4.1.27)
213	*make right accumulator register positive*	4EL	(0.4.1.23)
229	*make right A register positive*	4EM	(0.4.1.23)
230	*clear right A register*	4ER	(0.4.1.25)
260	*right accumulator register conditional shift left*	4ER	(0.4.1.25)
261	*make right A register and right accumulator register signs unlike*	4ER	(0.4.1.25)
267	*complement right divide connect flip-flop*	4ER	(0.4.1.25)
283	*right partial product*	4FL	(0.4.1.29)
284	*clear right B register*	4FK	(0.4.1.29)



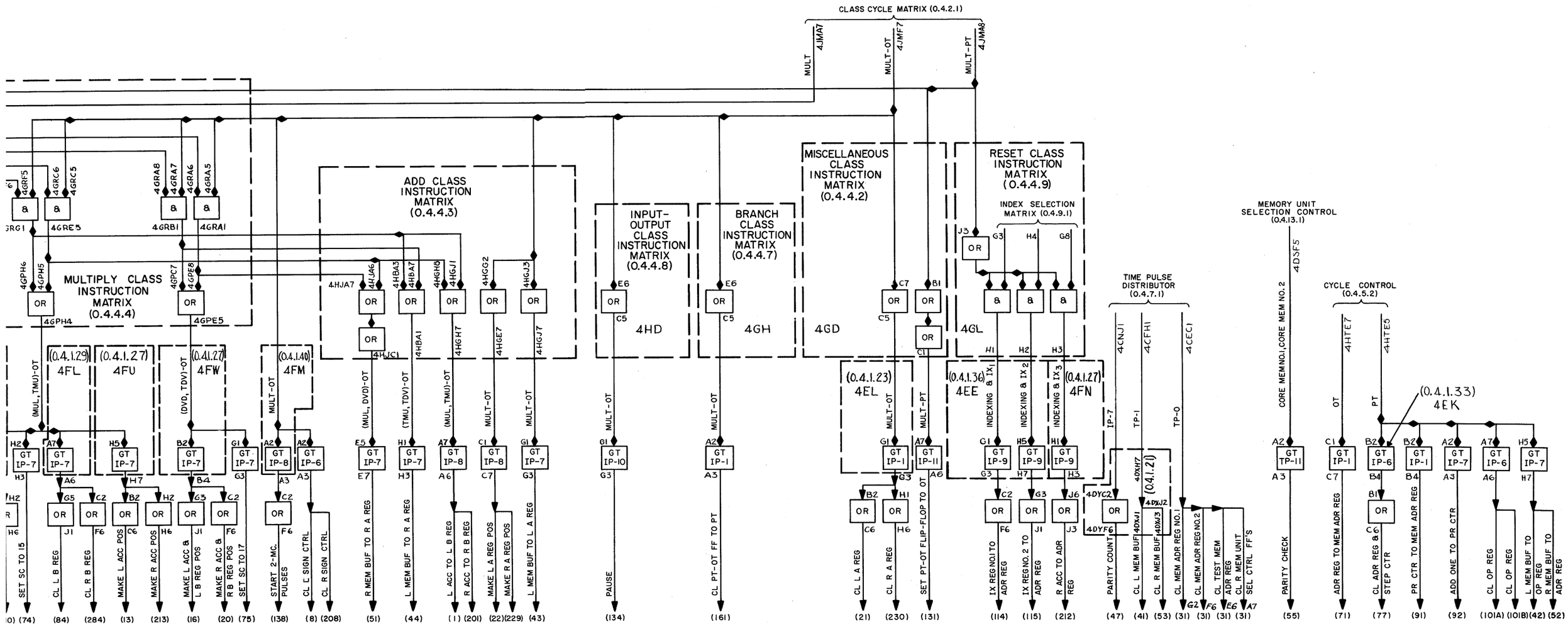


Figure 2-13. Multiply Class, Logical Block Diagram

b. Operate time (conditioned by the MULT–OT line from the cycle control):

1. 8, 208—*clear sign control*
2. 21, 230—*clear A register*
3. 22, 229—*make A register positive*
4. 43—*left memory buffer register to left A register*
5. 134—*pause*
6. 138—*start 2-megacycle pulses*

The commands listed above and the common commands appear in the timing chart; it is assumed that the reader will interpose them when necessary. Since the class is indexable, PT–9 is reserved for the indexing command (114, 115, or 212). Since every instruction requires two cycles, command 131 *(set PT–OT flip-flop to OT)* always occurs at PT–11, and command 161 *(clear PT–OT flip-flop to PT)* always occurs at OT–11. Each instruction requires 2-megacycle operation so that command 138 *(start 2-megacycle pulses)* is always generated at OT–8 and command 134 *(pause)*, which sets the pause flip-flop, is always generated at OT–10.

In the multiply-type instructions *(MUL, TMU)*, the multiplicand is extracted from core memory at OT–7 (43, 51) and placed in the A register. Also at OT–7, the step counter is set to

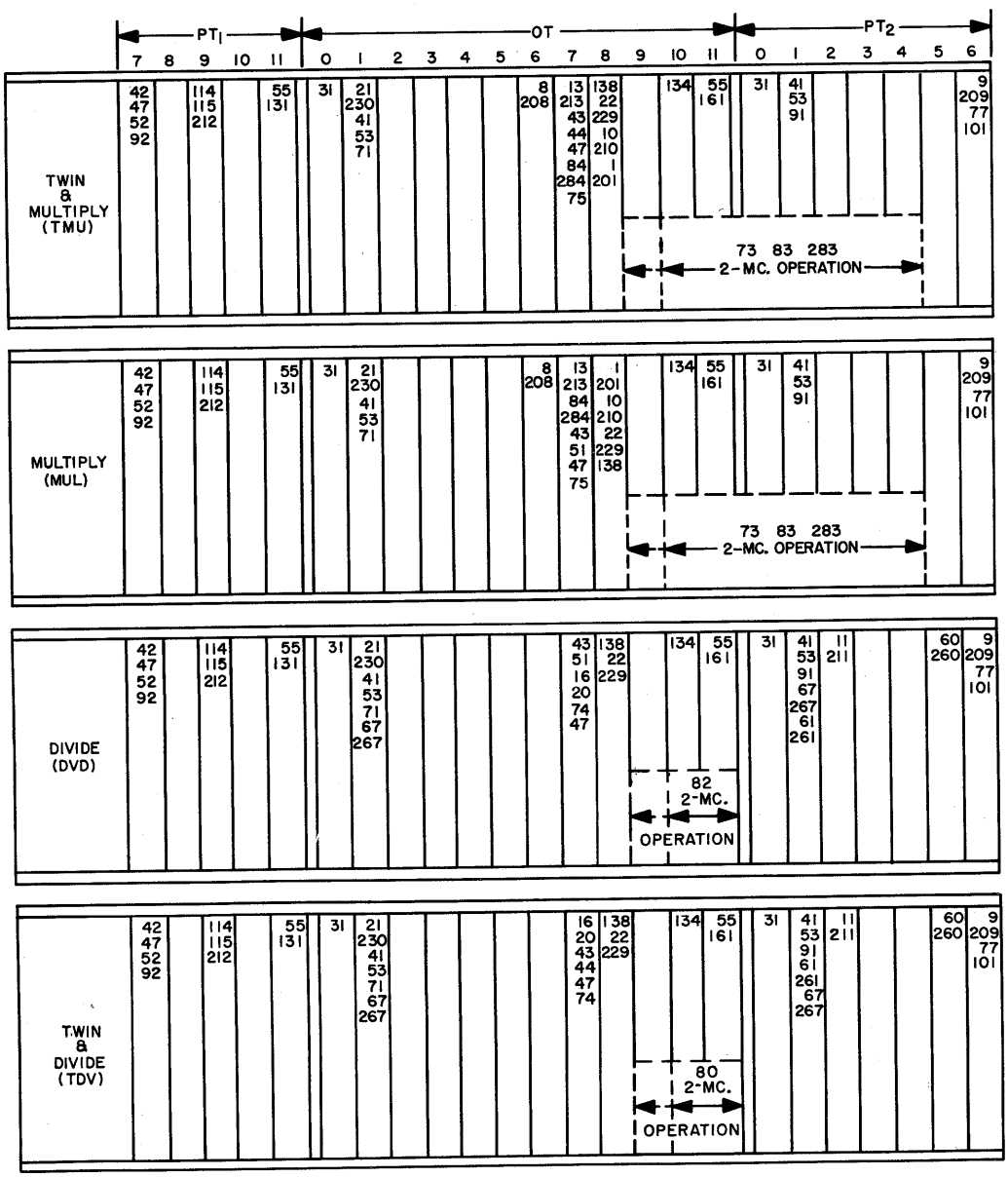


Figure 2–14. Multiply Class, Command Sequence Chart

15 by command 75 to indicate that 15 multiplication steps must be performed before the instruction is completed. The multiplier is held in the accumulator register and is examined for its sign. If the sign flip-flop equals 1, the accumulator register and the sign control flip-flop are complemented (13, 213). At OT–8, the contents of the accumulator register are placed in the B register (1, 201), and the accumulator register is cleared (10, 210). The multiplicand which was placed in the A register at OT–7 is examined for sign. If the sign equals 1, the A register is complemented at OT–8 (22, 229). Thus, by OT–9, both multiplicand and multiplier are positive and the sign of the product is stored in the sign control flip-flop. Now 2-megacycle operation is started. At OT–11, the time pulse distributor is stopped and the PT–OT flip-flop is set to PT (161). For each 2-megacycle operation, the step counter is reduced by 1. As soon as the step counter equals five, the time pulse distributor is restarted. This initiates the PT cycle of the next instruction.

The remaining 2-megacycle operations required to complete the multiplication instruction utilize part of this next program time (2½ microseconds). The end of 2-megacycle operation is indicated by the step counter being reduced to 0. This overlap operation saves time since the time interval between PT–0 and PT–5 is utilized. By PT–5, multiplication is completed, and the step counter controls are cleared. The command at PT–6 (9, 209) pertains to the correction of sign. In this operation, the sign control flip-flop is examined. If the flip-flop indicates 1, the accumulator register and B register are complemented and the sign control flip-flop is cleared. The accumulator and B registers enter into the complementing process because these registers together hold the single 32-bit product.

On divide-type instructions, the dividend is contained in the combined accumulator and B registers. At OT–7, the dividend is examined for sign (16, 20). If the accumulator register sign bit is 1, the accumulator register and the sign control flip-flop are complemented. At OT–7, the step counter is set to 17 by command 74, and the divisor is placed in the A register (43, 51). At OT–8, the sign of the divisor is examined (22, 229). If the A register flip-flop indicates a 1, the A register and the sign control flip-flop are complemented. The 2-megacycle operation is started at OT–9. A basic divide cycle consists of five steps which are controlled by the divide time pulse distributor. The step counter is reduced by 1 for each divide cycle.

When the step counter is reduced to 1, the time pulse distributor is restarted and program time of the next instruction is initiated. The commands at PT–2 (11, 211) and PT–6 (9, 209) deal with the correction of the remainder and correction of signs, respectively. To correct the remainder, the A register sign flip-flop is examined. If this flip-flop indicates 0, the carry 0 line is pulsed. This is followed at PT–5 by a shift left command (60, 260). Final correction of signs is described as part of the multiply instructions.

2.2.6.5 Store Class

The store class, shown in figures 2–15 and 2–16, contains seven instructions, each of which is indexable. All the instructions, except the *Store (FST)*, require three cycles to complete. The *Store* instruction requires two cycles but is unlike other instructions in this category in that program time is followed by OTB instead of OTA.

In addition to the common commands described in 2.2.6.1 of this Chapter, every instruction in the store class has the following commands in common:

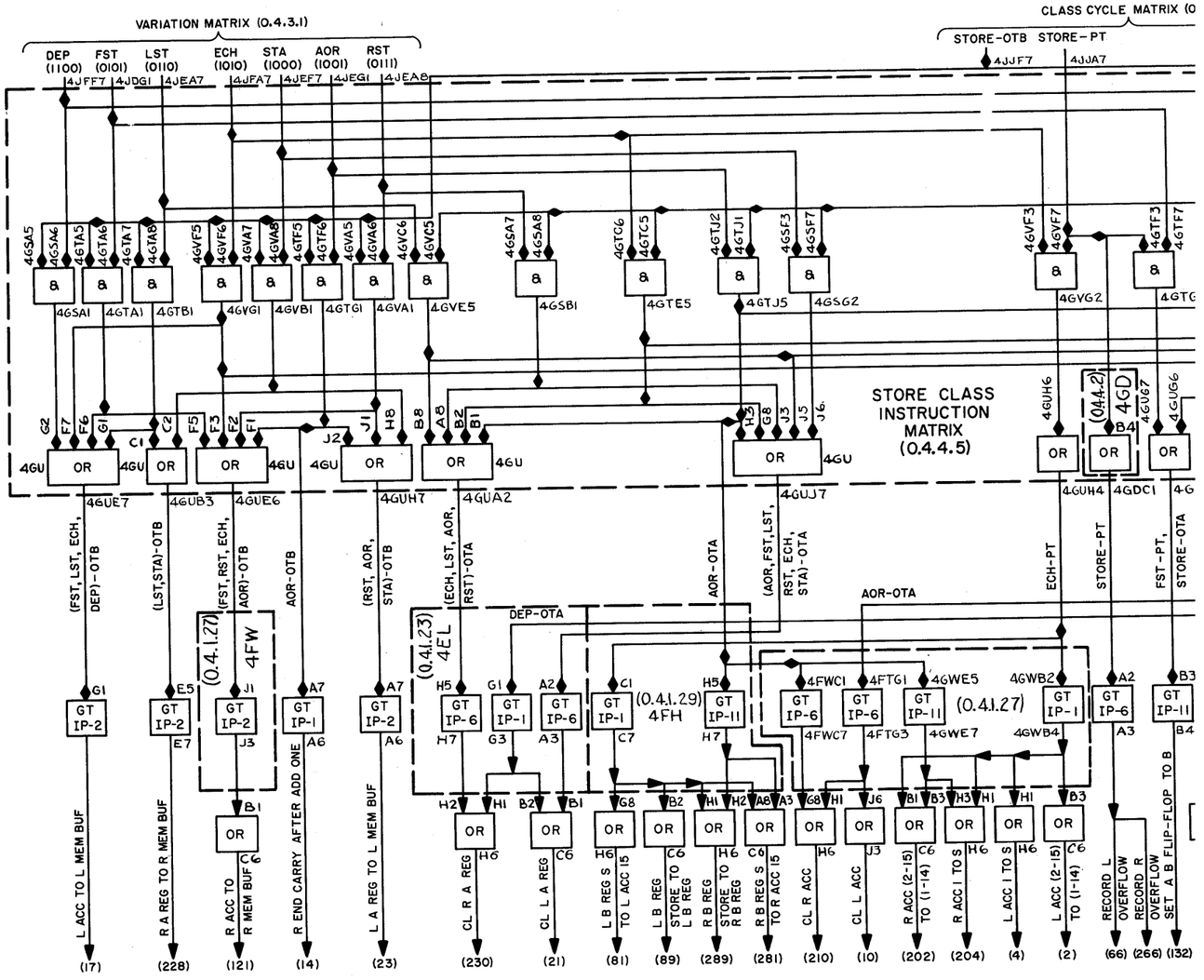
a. Store PT (conditioned by STORE–PT line the cycle control):
   1. 66—*record left overflow*
   2. 266—*record right overflow*
b. Store OTA (conditioned by the STORE–OTA line from the cycle control):
   1. 43—*left memory buffer register to left A register*
   2. 132—*set A–B flip-flop to B*
c. Store OTB (conditioned by the STORE–OTB line from the cycle control):
   1. 47—*parity count* (also a common command occurring at IP–7)
   2. 161—*clear PT–OT flip-flop to PT*
   3. 167—*clear A–B flip-flop to A*

The commands listed above and the common commands appear in the timing chart; it is assumed that the reader will interpose them when necessary in the ensuing discussion of the store class instructions. Since the class is indexable, PT–9 is reserved for the indexing command (114, 115, or 212). Since every instruction in the class, with the exception of *Store (FST)* requires three cycles, command 131 *(set PT–OT flip-flop to OT)* always occurs at PT–11, command 132 *(set A–B flip-flop to B)* always occurs at OTA–11, and commands 161 *(clear PT–OT flip-flop to PT)* and 167 *(clear A–B flip-flop to A)* always occur at OTB-11. The program times for all instructions are

STORE CLASS COMMANDS

(Fig. 2–15.)

COMMAND NUMBER	COMMAND NAME	PLUGGABLE UNIT NO.	LOGICAL REFERENCE NO.
2	*left accumulator register (2–15) to (1–14)*	4GY	(0.4.1.27)
4	*left accumulator register 1 to sign*	4GY	(0.4.1.27)
10	*clear left accumulator register*	4FU	(0.4.1.27)
14	*right end carry after add 1*	4EU	(0.4.1.25)
17	*left accumulator register to left memory buffer register*	4FW	(0.4.1.27)
19	*complement left accumulator register*	4FU	(0.4.1.27)
21	*clear left A register*	4EM	(0.4.1.23)
23	*left A register to left memory buffer register*	4EL	(0.4.1.23)
25	*left logical multiply*	4EM	(0.4.1.23)
31	*clear memory address register and clear test memory address register*	4CE	(0.4.7.1)
41	*clear left memory buffer register*	4CF	(0.4.7.1)
42	*left memory buffer register to operation register*	4DX	(0.4.1.21)
43	*left memory buffer register to left A register*	4DX	(0.4.1.21)
47	*parity count*	4DY	(0.4.1.21)
51	*right memory buffer register to right A register*	4DX	(0.4.1.21)
52	*right memory buffer register to address register*	4DX	(0.4.1.21)
53	*clear right memory buffer register*	4CF	(0.4.7.1)
55	*parity check*	4DX	(0.4.1.21)
64	*left carry 0*	4ER	(0.4.1.25)
66	*record left overflow*	4ER	(0.4.1.25)
69	*right carry 1*	4ES	(0.4.1.25)
71	*address register to memory address register*	4EH	(0.4.1.33)
77	*clear address register and step counter*	4EJ	(0.4.1.33)
81	*left B register sign to left accumulator register 15*	4FJ	(0.4.1.29)
88	*left B register to left A register*	4FJ	(0.4.1.29)
89	*left B register store to left B register sign*	4FK	(0.4.1.29)
91	*program counter to memory address register*	4EH	(0.4.1.33)
92	*add 1 to program counter*	4EH	(0.4.1.33)
101	*clear operation register*	4FF	(0.4.1.51)
114	*index register No. 1 to address register*		
115	*index register No. 2 to address register*		
121	*right accumulator register to right memory buffer register*	4FV	(0.4.1.27)
131	*set PT–OT flip-flop to OT*		
132	*set A–B flip-flop to B*	4FM	(0.4.1.40)
161	*clear PT–OT flip-flop to PT*	4FP	(0.4.1.40)
167	*clear A–B flip-flop to A*	4FP	(0.4.1.40)
202	*right accumulator register (2–15) to (1–14)*	4GX	(0.4.1.27)
204	*right accumulator register 1 to sign*	4GX	(0.4.1.27)
210	*clear right accumulator register*	4FU	(0.4.1.27)
212	*right accumulator register to address register*		
219	*complement right accumulator register*	4FU	(0.4.1.27)
225	*right logical multiply*	4EM	(0.4.1.23)
228	*right A register to right memory buffer register*	4EL	(0.4.1.23)
230	*clear right A register*	4EM	(0.4.1.23)
264	*right carry 0*	4ER	(0.4.1.25)
266	*record right overflow*	4ER	(0.4.1.25)
281	*right B register sign to right accumulator register 15*	4FJ	(0.4.1.29)
288	*right B register to right A register*	4FK	(0.4.1.29)
289	*right B register store to right B register sign*	4FK	(0.4.1.29)



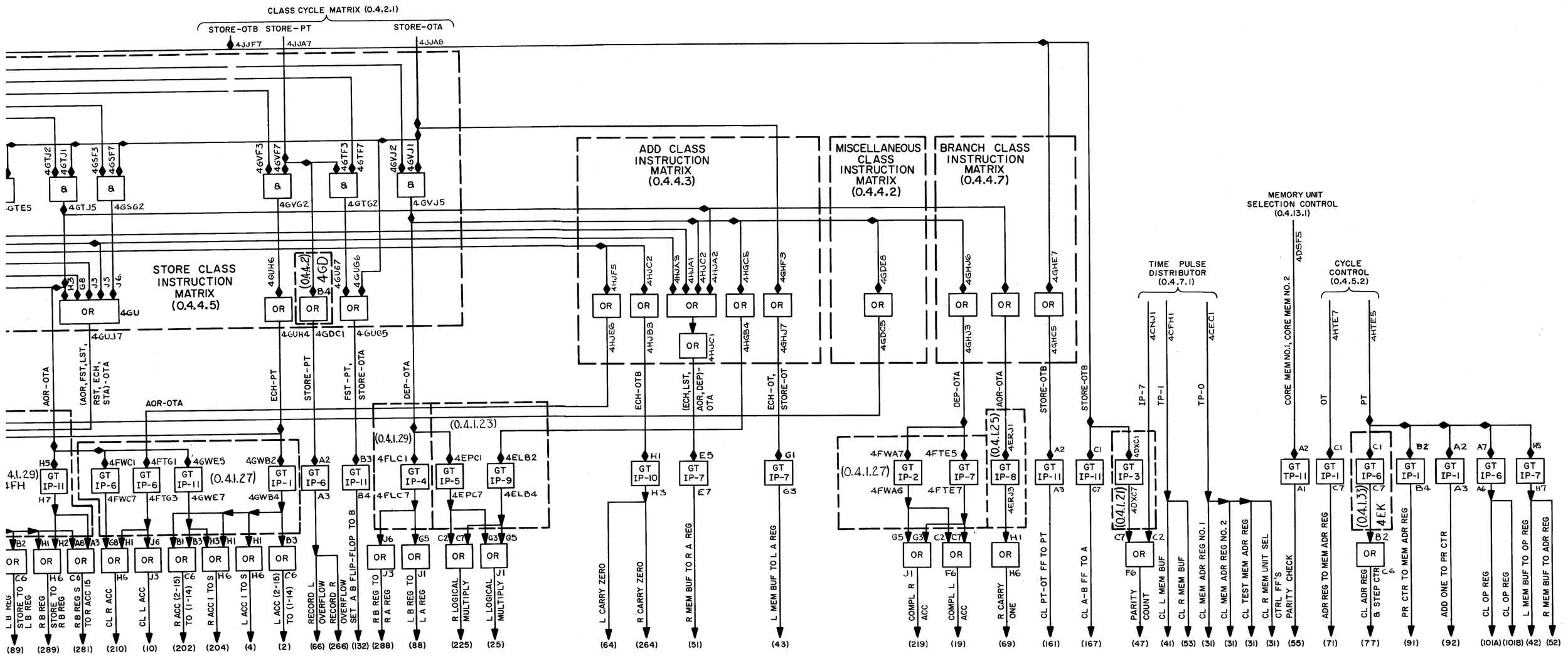


Figure 2-15. Store Class, Logical Block Diagram

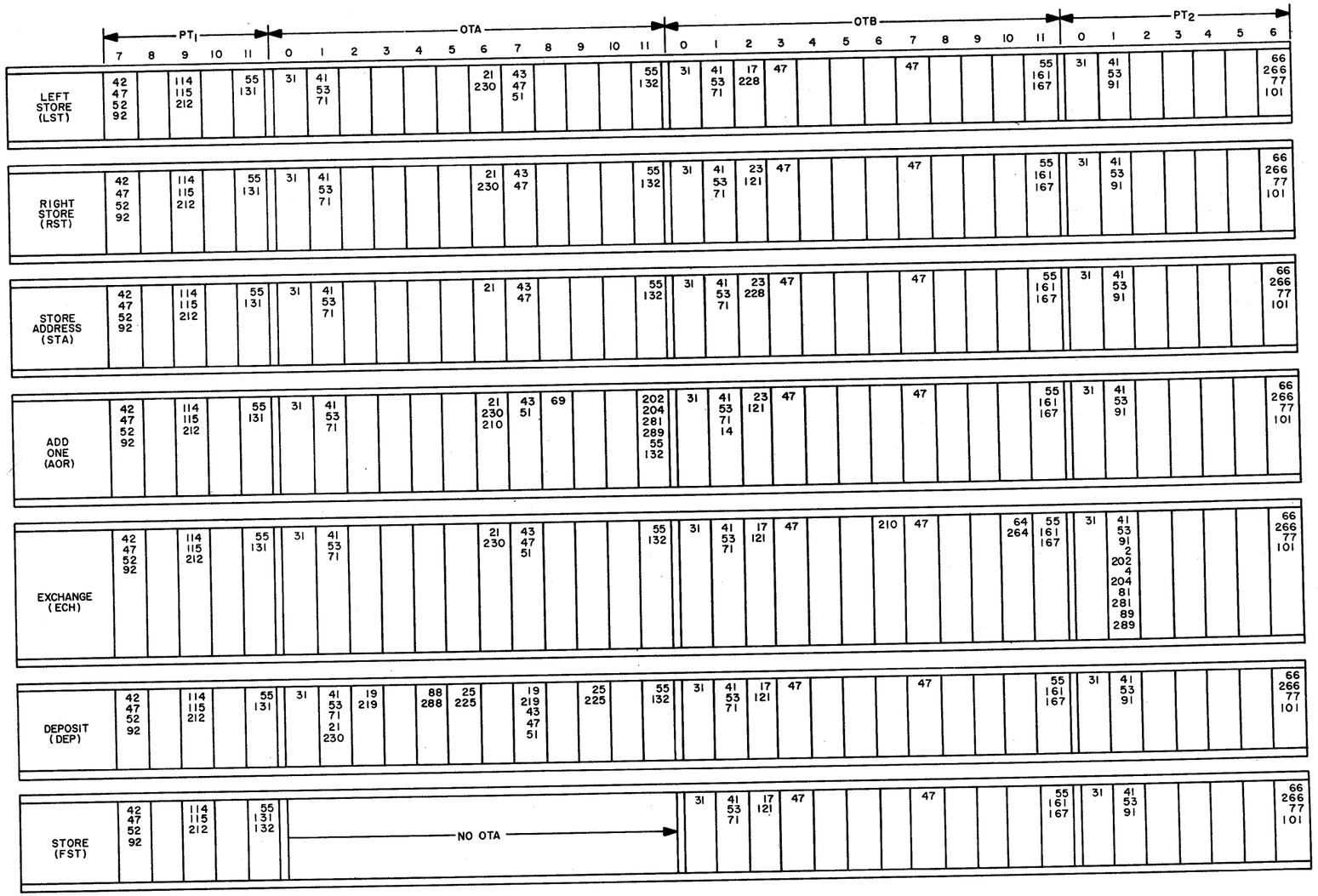


Figure 2–16. Store Class, Command Sequence Chart

similar with two exceptions. In the store instructions, command 132 is executed at PT–11 (in addition to common commands 55 and 131). In the *Exchange (ECH)* instruction, additional information transferring is performed at PT–1 (2, 202, 4, 204, 81, 281, 89, 289).

When a *Left Store (LST)* instruction is performed, the contents of the left accumulator register are stored in the left half of the memory address specified by the address part of the instruction. The contents of the accumulator registers and the right half of the indicated memory address are not changed, but the original contents of the left half of the indicated memory address are lost during this operation. This instruction clears the A register at OTA–6 (21, 230). At OTB–2, the contents of the right A register are transferred to the right memory buffer register (228), and the contents of the left accumulator register are transferred to the left memory buffer register (17). The contents of the right A register represent the half-word originally stored in core memory. This half-word was transferred from the right memory buffer register to the right A register at OTA–7 (51) to prevent the destruction of this word because of an IO break between the OTA and OTB cycles.

In the *Right Store (RST)* instruction, the contents of the right accumulator register are stored in the right half of the memory address specified by the address part of the instruction. The contents of the accumulator registers and the left half of the indicated memory address are not changed, but the original contents of the right half of the indicated memory address are lost during the operation.

The execution of the *Right Store* instruction is similar to that of the *Left Store* instruction described above, except that at OTB–2, the contents of the left A register are transferred to the left memory register (23), and the contents of the right accumulator register are transferred to the right memory buffer register (121).

During a *Store Address (STA)* instruction, the contents of the right A register are stored in the right half of the memory address specified by the address part of the instruction. The contents of the accumulator registers and the left half of the specified memory address are not changed by this operation. The original contents of the right half of the specified memory address are lost in this operation. The execution of this instruction is similar to the *RST* instruction, except that at OTB–2 the contents of the A registers are transferred to the memory buffer registers (23, 228). In addition, command 230 does not occur at OTA–6.

The purpose of the *Add 1 (AOR)* instruction is to add a 1 to the least significant digit of the right half of the specified memory address. The modified half-word is returned to its original memory address. The right accumulator register contains the modified half-word at the end of this operation. The left half-word in the specified memory address and the contents of the left accumulator register are not changed.

To execute the *Add 1* instruction, the accumulator register and the adder circuits must be used. Consequently, at OTA–6 the right accumulator register is cleared (210). At OTA–8, after the word has been placed in the A registers, the right carry 1 line is pulsed (69) to cause a 1 to be added to the contents of the right A register and the right accumulator register. This sum is placed in the right accumulator register, but displaced one position to the right. Hence, at OTA–11, a shift left is executed. In the following cycle, a *right end carry after add 1* command (14) is executed if the right carry flip-flop is equal to 1. In the particular case in which an end carry would occur, it is only necessary to set the 15th bit of the right accumulator register to 1. An actual addition for this correction is not required. The right carry flip-flop is cleared at the completion of this sequence of commands, as 1 has been added to the least significant digit of the number obtained from the specified memory address. At OTB–2, the contents of accumulator register are transferred to the memory buffer register (23, 121).

In the *Exchange (ECH)* instruction, the contents of the specified memory address and the contents of the accumulator registers are interchanged. In the execution of this instruction, the contents of the accumulator register are transferred to the memory buffer registers at OTB–2 (17, 121). Previous to this time, it was necessary to transfer the original contents of the specified memory address to the A register via the memory buffer register. After the original contents of the accumulator register are transferred to the memory buffer register at OTB–2, the accumulator register is cleared at OTB–6 (10, 210). At OTB–10, the carry 0 lines are pulsed (64, 264), causing the addition of the contents of the A register and the cleared accumulator register.

The *Deposit (DEP)* instruction is used to change the contents of certain bits of the word in the memory address specified by the address portion of this instruction. The contents of these bits are replaced by the contents of the corresponding bits of the accumulator register. The bits involved are determined by the B register. Every B register bit containing a 1 designates the corresponding bit of the accumulator register which is to be placed in the memory address.

In the execution of the *Deposit (DEP)* instruction, the A register is cleared at OTA–1. At OTA–2, the contents of the accumulator register are complemented (19, 219); at OTA–4, the contents of the B register are transferred to the A register (88, 288); and at OTA–5, the *logical multiply* command (25, 225) is executed. (See description of *Extract* instruction in the miscellaneous class.) The result of the *logical multiply* command is now in the accumulator register. The new word is extracted from core memory at OTA–7 (19, 219) and placed in the A register. This transfer combines this second half-word with the previous information in the A register. At OTA–7, the accumulator register is complemented once again (19, 219), followed by another *logical multiply* command at OTA–9 (25, 225). Thus, the contents of the B register determine which bit positions of the specified word in core memory are to be replaced by the corresponding bit positions of the accumulator register. The contents of the B register are not affected, but the contents of the accumulator register are transferred to the memory buffer register for storage in core memory.

The *Store (FST)* instruction requires that the contents of the accumulator register be transferred to the memory buffer register for storage in core memory. The contents of the accumulator register are unchanged; however, the original contents of the indicated memory address are lost. This instruction does not utilize an OTA cycle, going directly from PT to OTB. Otherwise the execution of this instruction is identical to that of the *Deposit* instruction.

2.2.6.6 Shift Class

The shift class, shown in figures 2–17 and 2–18, contains eight instructions, each of which is similar to either the *Shift Left* or the *Shift Right* instruction. None of the instructions in this class require an operate time cycle. The execution of the instructions in this class requires the accumulator register and the B register to be connected in several ways. Both registers are capable of shifting their contents one position to the right or one position to the left. In the accumulator register, the bit in position 1 is lost when left shifts are executed. This bit is not lost on cycle instructions. Similar statements apply to the bit in position 15 for the other instructions of this class.

In all these instructions, the address part of the instruction specifies the number of times to shift; a maximum of 64 shifts is provided. At PT–10, the shift operation is started. For every 2-megacycle operation, the step counter, which contains the number of times to shift, is reduced by 1. At PT–11, the time pulse distributor is stopped, and only 2-megacycle shift command pulses are now available. When the step counter is reduced to 5, the time pulse distributor is restarted and the next program time cycle is initiated. When the step counter is reduced to 0, 2-megacycle command pulses are stopped since the shift operation is completed. If the specified number of shifts is five or less, the time pulse distributor need not be stopped.

2.2.6.7 Branch Class

The branch class, shown in figures 2–19 and 2–20, contains six instructions. The *Sense (BSN)* and *Branch on Zero (BFZ)* instructions require two cycles; *Branch on Minus (BFM), Branch and Index (BPX), Branch on Left Minus (BLM),* and *Branch on Right Minus (BRM)* instructions each require one cycle.

In addition to the common commands described in 2.2.6.1 of this Chapter, every instruction in the branch class has the following commands in common:

a. Branch PT (conditioned by the BRANCH–PT line from the cycle control):
   1. 21, 230—*clear A register*
   2. 163—*clear branch flip-flop*

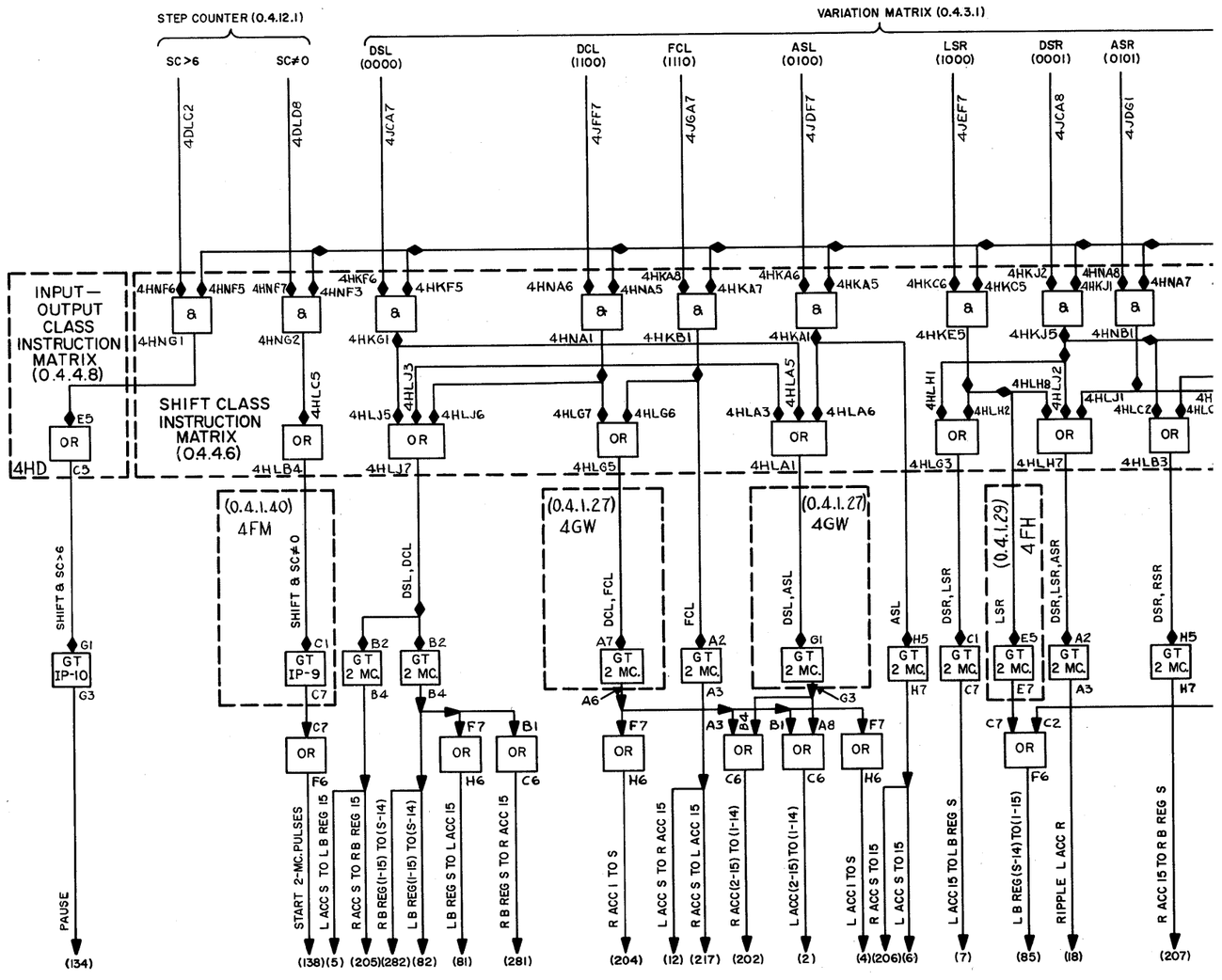
The commands listed above and the common commands appear in the timing chart; it is assumed that the reader will interpose them when necessary in the ensuing discussion of the branch class instructions.

The execution of the *Branch on Minus, Branch on Left Minus,* and *Branch on Right Minus* instructions are very similar; only the initial conditions to be met are different. In these instructions, normal execution from sequential addresses is interrupted if the numbers in the left and right accumulator registers are both negative, or if either one is negative. At PT–9, the A register is cleared and the appropriate accumulator register sign bits are examined (21, 230). If the appropriate sign bits are equal to 1, the branch

SHIFT CLASS COMMANDS

(Fig. 2–17.)

COMMAND NUMBER	COMMAND NAME	PLUGGABLE UNIT NO.	LOGICAL REFERENCE NO.
2	*left accumulator register (2–15) to (1–14)*	4GY	(0.4.1.27)
4	*left accumulator register 1 to sign*	4GY	(0.4.1.27)
5	*left accumulator register sign to left B register 15*	4FY	(0.4.1.27)
6	*left accumulator register sign to 15*	4GW	(0.4.1.27)
7	*left accumulator register 15 to left B register sign*	4FT	(0.4.1.27)
12	*left accumulator register sign to right accumulator register 15*	4FY	(0.4.1.27)
18	*ripple left accumulator register right*	4FT	(0.4.1.27)
31	*clear memory address register and clear test memory address register*	4CE	(0.4.7.1)
41	*clear left memory buffer register*	4CF	(0.4.7.1)
42	*left memory buffer register to operation register*	4DX	(0.4.1.21)
47	*parity count*	4DY	(0.4.1.21)
52	*right memory buffer register to address register*	4DX	(0.4.1.21)
53	*clear right memory buffer register*	4CF	(0.4.7.1)
55	*parity check*	4DX	(0.4.1.21)
73	*add 1 to step counter*	4FP	(0.4.1.40)
77	*clear address register and step counter*	4EJ	(0.4.1.33)
81	*left B register sign to left accumulator register 15*	4FJ	(0.4.1.29)
82	*left B register (1–15) to (sign–14)*	4FH	(0.4.1.29)
85	*left B register (sign–14) to (1–15)*	4FJ	(0.4.1.29)
91	*program counter to memory address register*	4EH	(0.4.1.33)
92	*add 1 to program counter*	4EH	(0.4.1.33)
101	*clear operation register*	4FF	(0.4.1.51)
123	*ripple right accumulator register right*	4FY	(0.4.1.27)
134	*pause*	4FP	(0.4.1.40)
138	*start 2-mc. pulses*	4FN	(0.4.1.40)
202	*right accumulator register (2–15) to (1–14)*	4GX	(0.4.1.27)
204	*right accumulator register 1 to sign*	4GX	(0.4.1.27)
205	*right accumulator register sign to right B register 15*	4FY	(0.4.1.27)
206	*right accumulator register sign to 15*	4GW	(0.4.1.27)
207	*right accumulator 15 to right B register sign*	4FY	(0.4.1.27)
217	*right accumulator register sign to left accumulator register 15*	4FY	(0.4.1.27)
281	*right B register sign to right accumulator register 15*	4FJ	(0.4.1.29)
282	*right B register (1–15) to (sign–14)*	4FH	(0.4.1.29)
285	*right B register (sign–14) to (1–15)*	4FJ	(0.4.1.29)



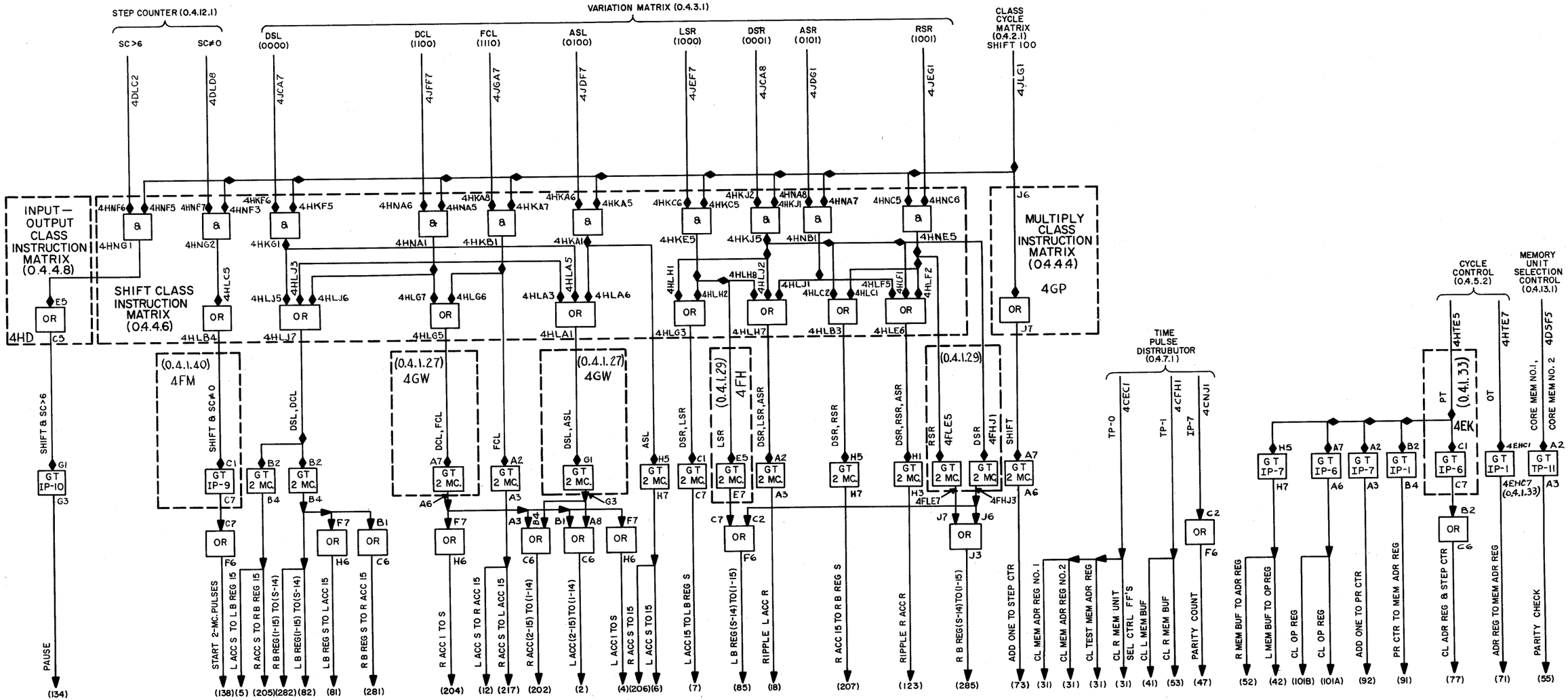


Figure 2–17. Shift Class, Logical Block Diagram

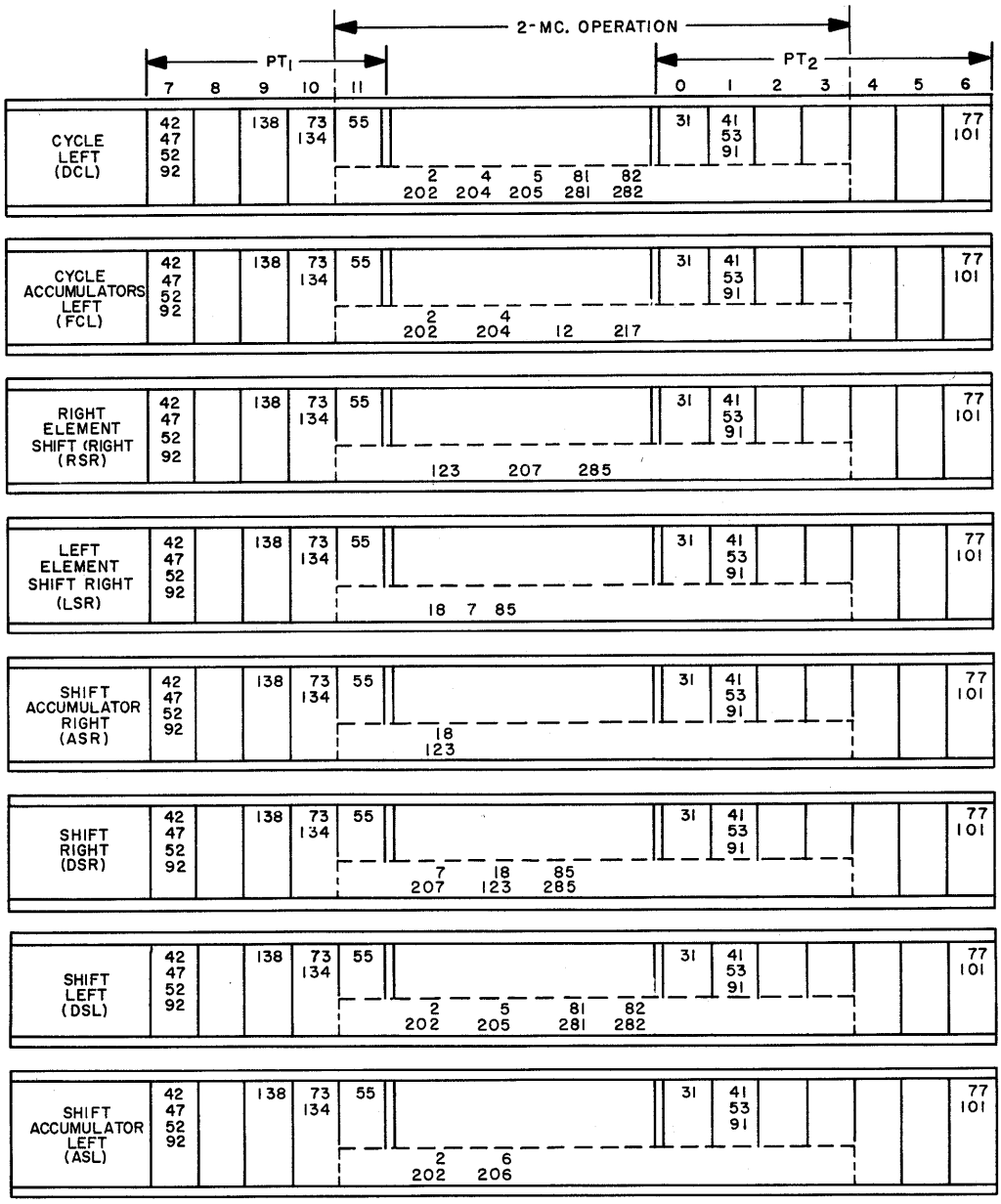
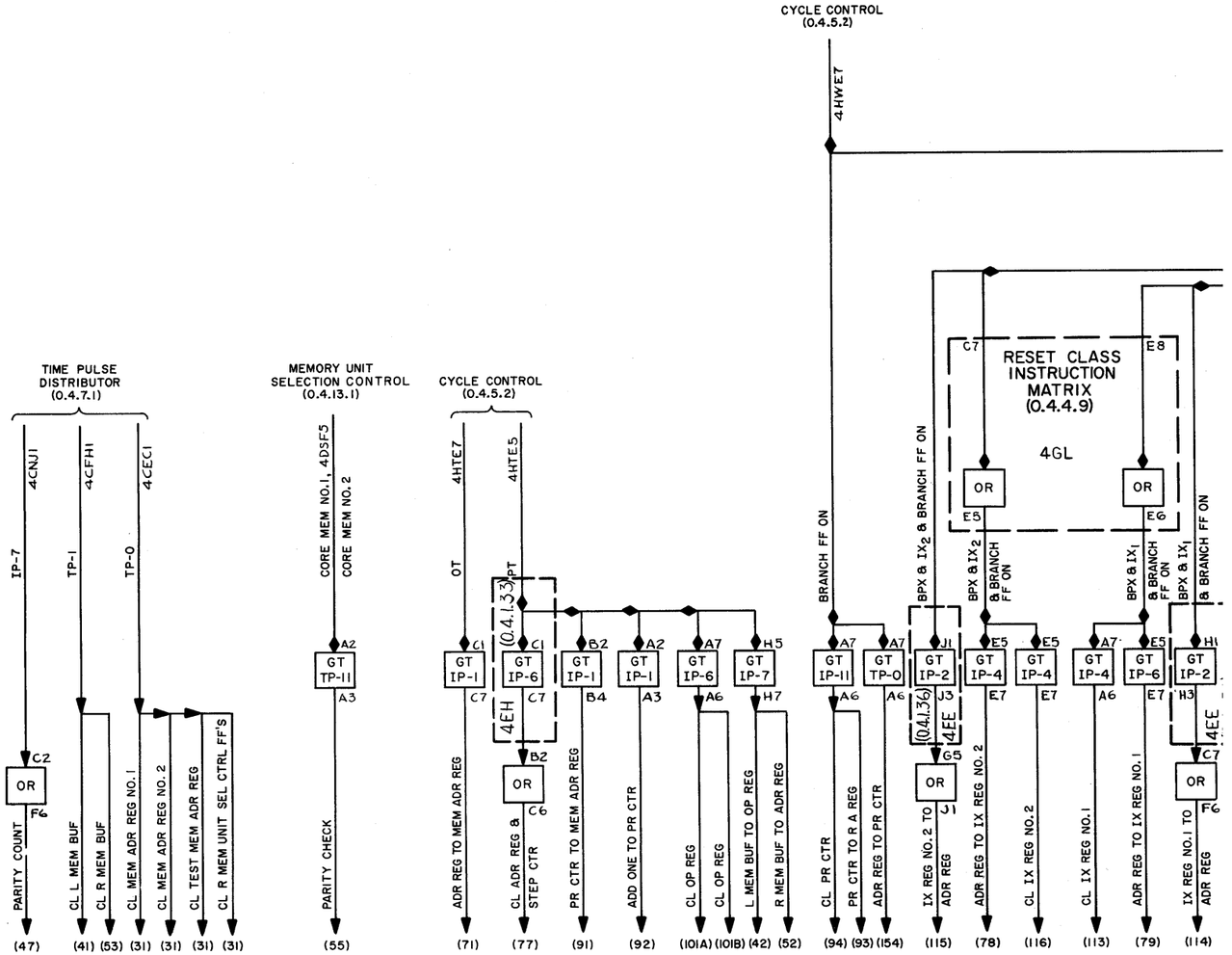


Figure 2-18. Shift Class, Command Sequence Chart

BRANCH CLASS COMMANDS

(Fig. 2–19.)

COMMAND NUMBER	COMMAND NAME	PLUGGABLE UNIT NO.	LOGICAL REFERENCE NO.
2	*left accumulator register (2–15) to (1–14)*	4GY	(0.4.1.27)
4	*left accumulator register 1 to sign*	4GY	(0.4.1.27)
13	*make left accumulator register positive*	4FU	(0.4.1.27)
19	*complement left accumulator register*	4FU	(0.4.1.27)
21	*clear left A register*	4EM	(0.4.1.23)
31	*clear memory address register and clear test memory address register*	4CE	(0.4.7.1)
41	*clear left memory buffer register*	4CF	(0.4.7.1)
42	*left memory buffer register to operation register*	4DX	(0.4.1.21)
47	*parity count*	4DY	(0.4.1.21)
52	*right memory buffer register to address register*	4DX	(0.4.1.21)
53	*clear right memory buffer register*	4CF	(0.4.7.1)
55	*parity check*	4DX	(0.4.1.21)
62	*left carry 1*	4ES	(0.4.1.25)
69	*right carry 1*	4ES	(0.4.1.25)
71	*address register to memory address register*	4EH	(0.4.1.33)
77	*clear address register and step counter*	4EJ	(0.4.1.33)
78	*address register to index register No. 2*	4EK	(0.4.1.33)
79	*address register to index register No. 1*	4EH	(0.4.1.33)
81	*left B register sign to left accumulator register 15*	4FJ	(0.4.1.27)
89	*left B register store to left B register sign*	4FK	(0.4.1.27)
91	*program counter to memory address register*	4EH	(0.4.1.33)
92	*add 1 to program counter*	4EH	(0.4.1.33)
93	*program counter to right A register*	4EK	(0.4.1.33)
94	*clear program counter*	4EK	(0.4.1.33)
101	*clear operation register*	4FF	(0.4.1.51)
103	*index interval complement to address register*	4FD	(0.4.1.51)
113	*clear index register No. 1*	4EE	(0.4.1.36)
114	*index register No. 1 to address register*	4EF	(0.4.1.36)
115	*index register No. 2 to address register*	4EF	(0.4.1.36)
116	*clear index register No. 2*	4EE	(0.4.1.36)
131	*set PT-OT flip-flop to OT*	4FM	(0.4.1.40)
140	*sense for branch*	4FD	(0.4.1.51)
154	*address register to program counter*	4EK	(0.4.1.33)
161	*clear PT-OT flip-flop to PT*	4FP	(0.4.1.40)
162	*test right and left accumulator registers*	4GY	(0.4.1.27)
163	*clear branch flip-flop*	4FM	(0.4.1.40)
164	*test index register No. 1 sign for 0*	4EE	(0.4.1.36)
165	*test the left accumulator register*	4FY	(0.4.1.27)
166	*test right accumulator sign for 1*	4FY	(0.4.1.27)
170	*set branch flip-flop*	4FP	(0.4.1.40)
174	*test index register No. 2 sign for 0*	4EE	(0.4.1.36)
202	*right accumulator register (2–15) to (1–14)*	4GX	(0.4.1.27)
204	*right accumulator register 1 to sign*	4GX	(0.4.1.27)
213	*make right accumulator register positive*	4FU	(0.4.1.27)
214	*left round off sign*	4FT	(0.4.1.27)
216	*right round off sign*	4FT	(0.4.1.27)
219	*complement right accumulator register*	4FU	(0.4.1.27)
230	*clear right A register*	4EM	(0.4.1.23)
281	*right B register sign to right accumulator register 15*	4FJ	(0.4.1.27)
289	*right B register store to right B register sign*	4FK	(0.4.1.27)
321	*sense tapes for not ready*	4FD	(0.4.1.51)



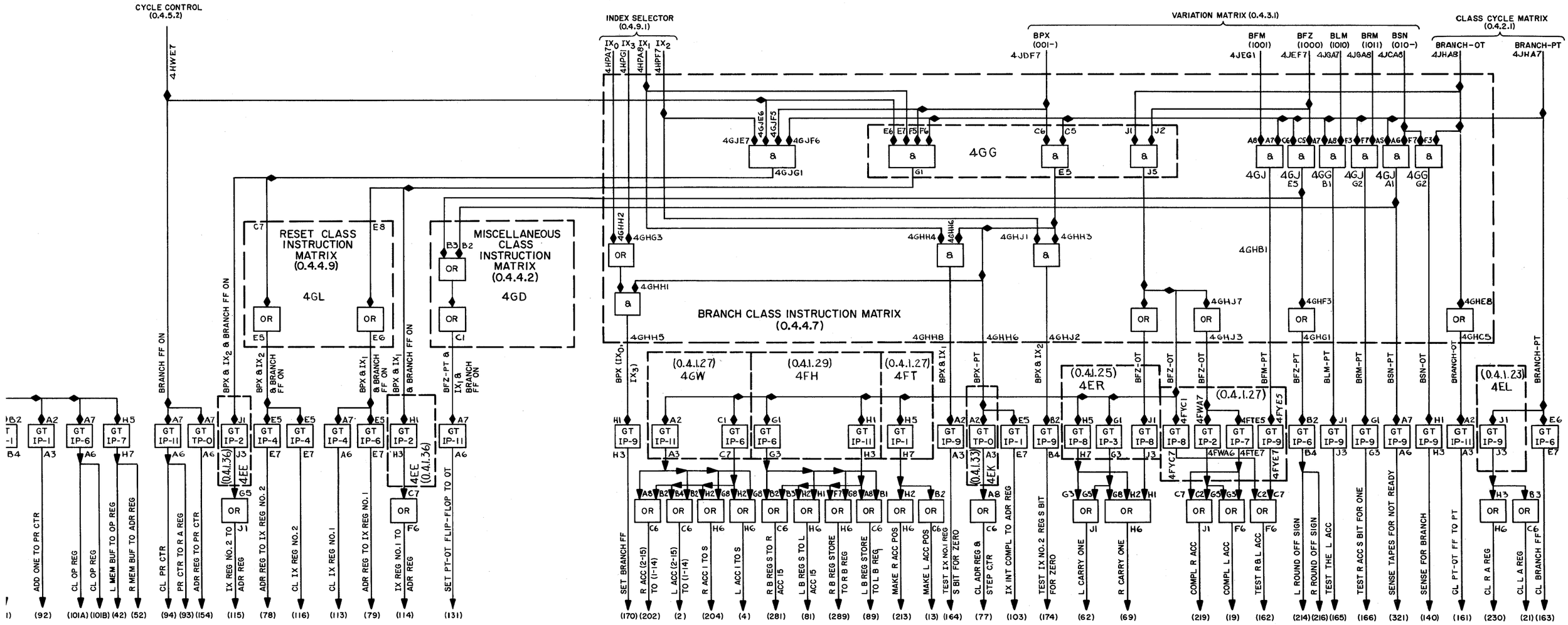


Figure 2–19. Branch Class, Logical Block Diagram

flip-flop is also set to 1, and the contents of the address register are transferred to the program counter on the following PT–0 pulse. Thus, the program counter is set to the address of the next instruction to be executed. The branch flip-flop is cleared at PT–6 (163). If the sign conditions have not been met, the branch flip-flop remains in a cleared condition, and the contents of the program counter are not altered.

The *Sense* instruction enables the control element of the Central Computer System to determine the status of the computer sense units. The last six bits (L10 through L15) of the left halfword of this instruction identify the sense unit selected. This instruction is used to sense such conditions as the status of the IO interlock, the status of condition lights and sense switches, and overflow conditions of the accumulator register. An unconditional branch of control is executed if an electrical signal is present at the terminal of the selected sense unit.

At OT–9, the *sense for branch* command (140) is generated, which causes the status of the sense unit to be examined. If the necessary conditions are met, the branch flip-flop is set to an indicating condition and, at OT–11, the program counter is transferred to the right A register (93), previously cleared (21, 230) at PT–9. The program counter is cleared at OT–11 (94). At PT–0, the contents of the address register are transferred to the program counter (154). The branch flip-flop is cleared at PT–6 (163).

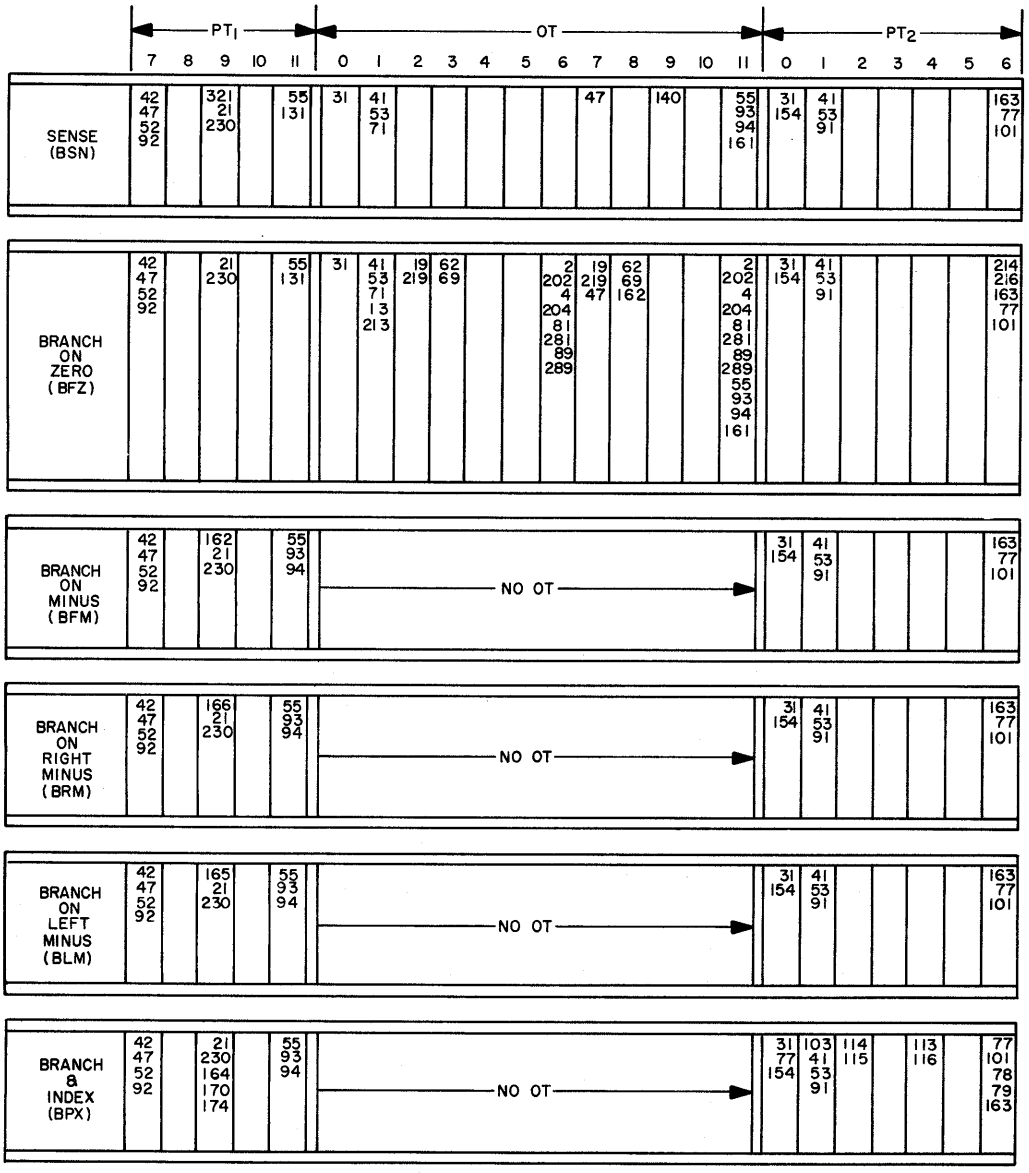


Figure 2–20. Branch Class, Command Sequence Chart

The *Branch on Zero* instruction causes the A register to be cleared unconditionally, the contents of the program counter to be transferred to the right A register, and the address of the next instruction to be placed in the program counter, provided the contents of the accumulator registers are plus or minus 0. At PT–9, the A registers are cleared (21, 230). At OT–1, the sign bit of each accumulator register is examined. If the sign bit of either accumulator register is 1, this register and the sign control flip-flop are both complemented. At OT–2, the accumulator registers are complemented again (19, 219). This makes both accumulator registers negative. At OT–3, the carry 1 lines are pulsed (62, 69) and a carry will result if all of the bits of the accumulator register are 1; i.e., negative 0.

This action is followed at OT–6 by a correction shift left operation (2, 202, 4, 204, 81, 281). The restoring operation begins at OT–7 and is conditional on the status of the sign control flip-flop of the respective accumulator register. If a sign control flip-flop is 0, the corresponding accumulator register is complemented at OT–7 (19, 219). At PT–8, the sign bits of the accumulator register are examined (162). If both sign bits are equal to 1, the branch flip-flop is set to an indicating position. Also at this time the carry 1 lines are pulsed (62, 69) and followed by the *shift left* commands at OT–11. If the branch flip-flop is in an indicating condition, the contents of the program counter are transferred to the right A register, and the program counter is cleared at OT–11 (94).

At PT–0, the contents of the address register are placed in the program counter (154). At PT–6, the sign control flip-flops are examined; if either flip-flop is in the 1 condition the corresponding accumulator register is complemented (124, 216). Both sign control flip-flops are cleared at PT–6.

The execution of the *Branch and Index* instruction is contingent upon the sign of the specified index register. At PT–9, the sign bit of the specified index register is examined (164 or 174). If it is equal to 0, the branch flip-flop is set to an indicating condition, and the right A register is cleared. At PT–11, the contents of the program counter are transferred to the right A register (93), and the program counter is cleared (94). At PT–0 of the next program time cycle, the contents of the address register are placed in the program counter (154), and the address register is cleared (77). The new contents of the program counter provide the address of the next instruction to be executed. At PT–1, the complement of the index interval is transferred to the address register (103), and the contents of the specified index register are added to the contents of the address register at PT–2 (114 or 115). The process of addition, including the propagation of the carry, is completed by PT–6. Thus, the contents of the specified index register are reduced by the value specified by the index interval. If no index register or if the right accumulator register (which may under certain circumstances be used as an index register) had been specified, no branch will take place. In this case, the contents of the index interval are meaningless.

2.2.6.8 Input-Output Class

The IO class, shown in figures 2–21 and 2–22, contains five instructions, all of which are indexable. These instructions are associated with the execution of a break-in or a break-out cycle when the Central Computer System makes memory available to a selected IO unit. The execution of the instructions of this class is contingent upon the IO interlock being in an off condition. If the IO interlock is in an on condition, the execution of these instructions is delayed until the interlock is cleared. The *Load Address Counter, Read,* and *Write* instructions require one memory cycle, and the *Select* and *Select Drums* instructions require two memory cycles.

The *Load IO Address Counter (LDC)* instruction is used to indicate the address of the first core memory location involved in a subsequent read or write operation. Initially, the IO interlock is sensed. If the interlock is in an on condition, the pause flip-flop is set to an indicating condition at PT–10 (134). The time pulse distributor is stopped by the pause condition. During the pause condition, sensing for an IO break is made at a 2-megacycle rate. When the IO interlock is cleared, indicating completion of the IO break previously in progress, the Central Computer System proceeds with the instruction. At PT–2, the IO address counter is cleared (148), and at PT–3, the contents of the address register are transferred to the IO address counter (72). The contents of the address register at this time indicate the address in the core memory from which, or to which, the first word of the following IO operation will be transferred.

The *Select (SEL)* instruction indicates the IO unit, other than drums, which is to function during the following IO break operation. The IO unit is identified by the last six bits, L10 through L15, of the operation register. These are the same

bits which are used as the index interval on other instructions. Two cycles are required for the execution of this instruction.

The execution of the *Select* instruction is contingent on the IO interlock being in an off condition. The interlock is sensed at OT–11. If the interlock is in an off condition, the PT–OT flip-flop is set to PT (161), and the Central Computer System proceeds with the instruction. If the interlock is in an on condition, the PT–OT flip-flop is set to PT, and the pause flip-flop is set to the pause condition, which causes the break request flip-flop to be sensed at a 2-megacycle rate. During the execution of the *Select* instruction, a deselect pulse is generated at PT–1 (155). This pulse resets the selection flip-flops associated with the previously selected IO units. At PT–3, the contents of the address register are transferred to the drum control register (151), which had been cleared at PT–2 (146). This transfer is permitted to make the *Select* instruction consistent with the *Select Drums* instruction. At PT–5, the *select pulse* command is executed (156). This command examines the PerSelBsn matrix to see which IO unit has been selected. As a result of this operation, the selection flip-flops associated with the desired IO unit are set to an indicating condition.

The *Select Drums* instruction is similar to the *Select* instruction. It is used to select a particular drum field associated with the Drum System of the AN/FSQ–7 (XD–1) Combat Direction Central. The distinction between selecting drums and IO devices other than drums is necessary because the total number of addresses required by the IO units exceeded 64. The execution of this instruction is contingent upon the IO interlock being in an off condition. At PT–3, the contents of the address register are transferred to the drum control register (151) which was cleared at PT–2 (146). The drum control register now contains the starting address of the selected drum field. This command is also executed in those drum operations which require no starting address. At PT–5, the contents of the index interval are transferred to drum selection register located in the drum frame (325). The necessary conditions for the execution of a break-in or break-out operation are provided by the circuits of the selection and IO control element.

The *Write (WRT)* instruction provides the necessary conditions to effect the transfer of information from core memory in the Central Computer System to the selected output unit. The execution of the instruction is contingent on the IO interlock being in an off condition. At PT–1, the IO interlock is set to its on condition (294), and the IO word counter is cleared (149). At PT–2, the contents of the address register are transferred to the IO word counter (152), and the IO register is cleared (144, 147). The IO word counter now indicates the number of words to be transferred. At PT–3, the IO word counter is sensed (290). If the contents of the counter are equal to 0, the IO word counter status flip-flop is set equal to 0 and the IO interlock is cleared. At PT–6, the write flip-flop is set to an indicating condition (182), and the index interval bits are sensed for the interleave mode.

The execution of the *Read (RDS)* instruction is similar to that of the *Write* instruction. The only difference between the two is that in the *RDS* instruction, the read flip-flop, instead of the write flip-flop, is set to an indicating condition at PT–6 (180). Again the control of the actual reading operation is performed by circuits of the selection and IO control element.

2.2.6.9 Reset Class

The reset class, shown in figures 2–23 and 2–24, contains three instructions, all of which deal with the index register specified by the index indicator, bits 1 through 3 of the operation register. The instructions are not indexable and require one cycle to complete.

The *Reset Index Register (XIN)* instruction sets the specified index register to the value indicated by the address part of the instruction. At PT–4, the specified index register is cleared (114 or 115). At PT–6, the contents of the address register are transferred to the specified index register (78 or 79).

The *Reset Index Register from the Right Accumulator Register (XAC)* instruction sets the specified index register to the value indicated by the contents of the right accumulator register. The address register is cleared at PT–9 (77). The specified index register is cleared at PT–4 (113 or 116) and the contents of the address register are transferred to the specified index register at PT–6 (78 or 79). In the execution of this instruction, the contents of the right accumulator register are transferred to the address register at PT–10 (212).

In the *Add Index Register (ADX)* instruction, the contents of the index register specified by the index indicator (bits L1 through L3) in the operation part of the instruction are added to the address part of the instruction, and the sum is

placed in the right A register. If bits L1 through L3 are 0, this instruction has the effect of placing its address in the right A register. At PT–9, the contents of the specified index register are added to the contents of the address register (114, 115, 212). At PT–10, the carry operation is started (the adder circuit used here is not the same as that used in the arithmetic element). At PT–3, the right A register is cleared (230), and at PT–5, the modified contents of the address register are transferred to the right A register (76). At this point the value specified by the contents of the address register is the sum of the address specified by the *ADX* instruction and the contents of the specified index register. If no index register is specified, then the original contents of the address register are transferred unmodified to the right A register.

2.3 MEMORY UNIT SELECTION

The selection of a memory unit is determined by the first three bits (R1 through R3) of the right half of the instruction word. This information is first decoded in the program element and the resulting pulse information is sent to the memory unit selection control in the instruction control element. As illustrated in figure 2–25, one of four possible memory units is selected by this control: core memory No. 1, core memory No. 2, the test memory, or the clock register. No more than one memory unit can be selected at one time.

The memory unit selection control obtains information from one of three possible program element registers: the address register, the program counter, or the IO address counter. If one of the core memories or the test memory has been specified, any one of the three sources can originate the information. If the clock register has been specified, the program counter cannot be the information source because, since the clock register is not an instruction address, it can never be specified by the program counter.

The pulse from the program element sets the control flip-flop associated with the memory unit selected. The flip-flop will condition a set of command generators. The decoding process is identical with that employed for decoding the class and variation bits of the operation register. Inputs from the selection and IO control element and the cycle control determine the type of cycle which is in effect at the time any command is issued.

In addition to the instruction pulses, which usually inspect the command generators, break pulses BI–3, BI–4, and BO–5 are applied to these generators. In this instance, the function of the BI–3 pulse is to prevent the memory buffer registers from writing into core memory during a break-in cycle. The BI–4 pulse transfers the contents of the memory buffer register if test memory has been selected. The BO–5 pulse reverses the direction of this latter transfer if a break-out cycle specifying test memory is in effect.

INPUT-OUTPUT CLASS COMMANDS

(Fig. 2–21.)

COMMAND NUMBER	COMMAND NAME	PLUGGABLE UNIT NO.	LOGICAL REFERENCE NO.
31	*clear memory address register and clear test memory address register*	4CE	(0.4.7.1)
		4CF	(0.4.7.1)
41	*clear left memory buffer register*	4DX	(0.4.1.21)
42	*left memory buffer register to operation register*	4DV	(0.4.1.21)
47	*parity count*	4DX	(0.4.1.21)
52	*right memory buffer register to address register*	4CF	(0.4.7.1)
53	*clear right memory buffer register*	4DX	(0.4.1.21)
55	*parity check*	4EH	(0.4.1.33)
71	*address register to memory address register*	4EH	(0.4.1.33)
72	*address register to IO address counter*	4EJ	(0.4.1.33)
77	*clear address register and step counter*	4EH	(0.4.1.33)
91	*program counter to memory address register*	4EH	(0.4.1.33)
92	*add 1 to program counter*	4FF	(0.4.1.51)
101	*clear operation register*	4EF	(0.4.1.36)
114	*index register No. 1 to address register*	4EF	(0.4.1.36)
115	*index register No. 2 to address register*	4FM	(0.4.1.40)
131	*set PT-OT flip-flop to OT*	4FP	(0.4.1.40)
134	*pause*	4EW	(0.4.1.50)
144	*clear IO registers*	4EW	(0.4.1.50)
146	*clear drum control register*	4EW	(0.4.1.50)
147	*clear right IO register*	4EW	(0.4.1.50)
148	*clear IO address counter*	4EW	(0.4.1.50)
149	*clear IO word counter*	4EH	(0.4.1.33)
151	*address register to drum control register*	4EH	(0.4.1.33)
152	*address register complement to IO word counter*	4FD	(0.4.1.51)
155	*deselect pulse*	4FD	(0.4.1.51)
156	*select pulse*	4FP	(0.4.1.40)
161	*clear PT-OT flip-flop to PT*	4FD	(0.4.1.51)
180	*PT–6 on Read*	4FD	(0.4.1.51)
182	*PT–6 on Read or Write*	4FV	(0.4.1.27)
212	*right accumulator register to address register*	4EW	(0.4.1.50)
290	*sense IO interlock 0*	4EW	(0.4.1.50)
294	*set IO interlock on*	4FF	(0.4.1.51)
325	*select pulse for drums*		

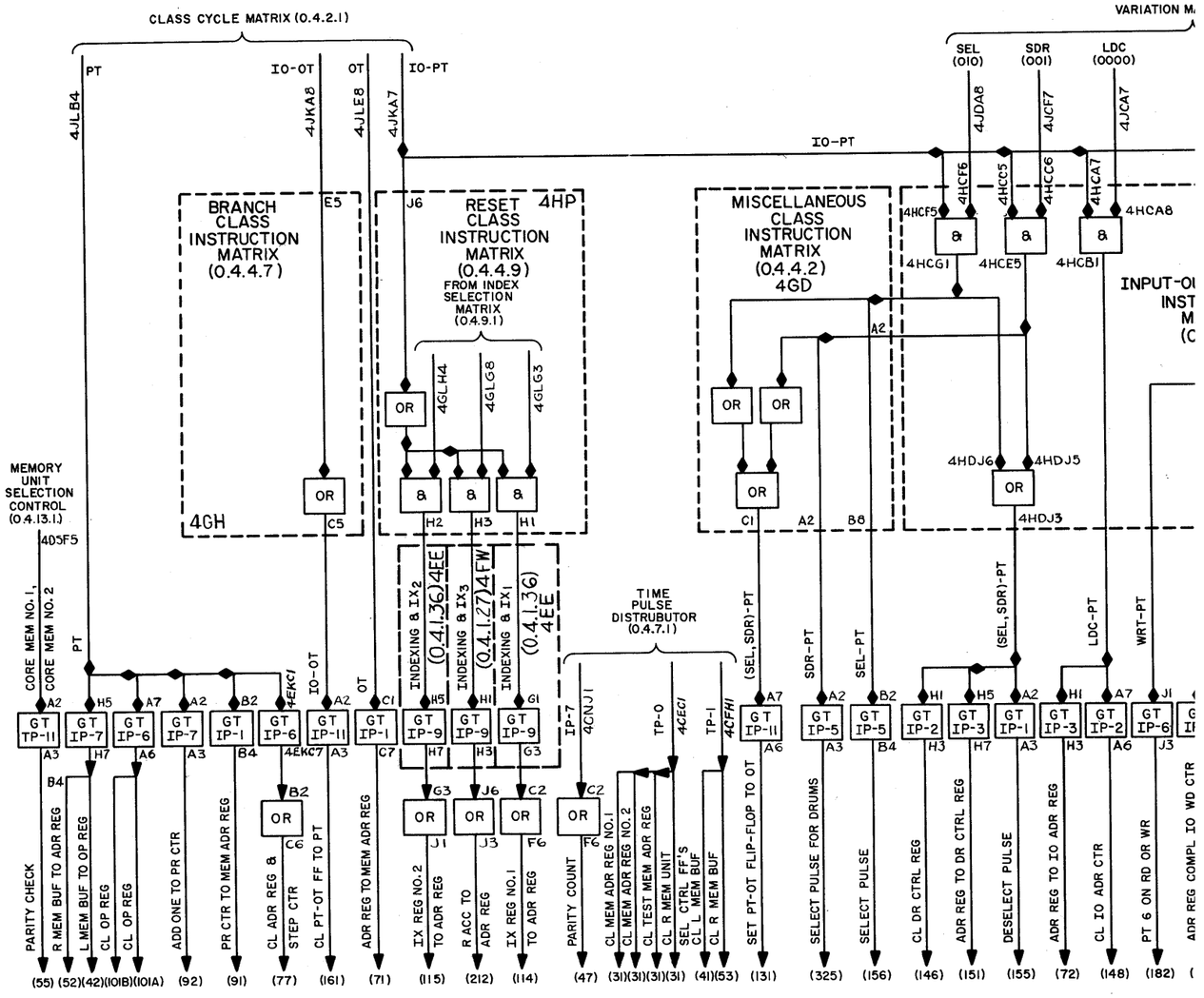


Figure 2–21. Input-Output Class, Logical Block

PUT-OUTPUT CLASS COMMANDS

(Fig. 2–21.)

COMMAND NAME	PLUGGABLE UNIT NO.	LOGICAL REFERENCE NO.
ıory address register and clear test memory		
ɔss register	4CE	(0.4.7.1)
memory buffer register	4CF	(0.4.7.1)
ıry buffer register to operation register	4DX	(0.4.1.21)
unt	4DV	(0.4.1.21)
ıory buffer register to address register	4DX	(0.4.1.21)
:t memory buffer register	4CF	(0.4.7.1)
eck	4DX	(0.4.1.21)
egister to memory address register	4EH	(0.4.1.33)
egister to IO address counter	4EH	(0.4.1.33)
ess register and step counter	4EJ	(0.4.1.33)
counter to memory address register	4EH	(0.4.1.33)
rogram counter	4EH	(0.4.1.33)
ation register	4FF	(0.4.1.51)
ister No. 1 to address register	4EF	(0.4.1.36)
ister No. 2 to address register	4EF	(0.4.1.36)
' flip-flop to OT	4FM	(0.4.1.40)
	4FP	(0.4.1.40)
egisters	4EW	(0.4.1.50)
n control register	4EW	(0.4.1.50)
t IO register	4EW	(0.4.1.50)
ıddress counter	4EW	(0.4.1.50)
vord counter	4EW	(0.4.1.50)
egister to drum control register	4EH	(0.4.1.33)
egister complement to IO word counter	4EH	(0.4.1.33)
ulse	4FD	(0.4.1.51)
se	4FD	(0.4.1.51)
OT flip-flop to PT	4FP	(0.4.1.40)
Read	4FD	(0.4.1.51)
Read or Write	4FD	(0.4.1.51)
:mulator register to address register	4FV	(0.4.1.27)
:nterlock 0	4EW	(0.4.1.50)
erlock on	4EW	(0.4.1.50)
se for drums	4FF	(0.4.1.51)

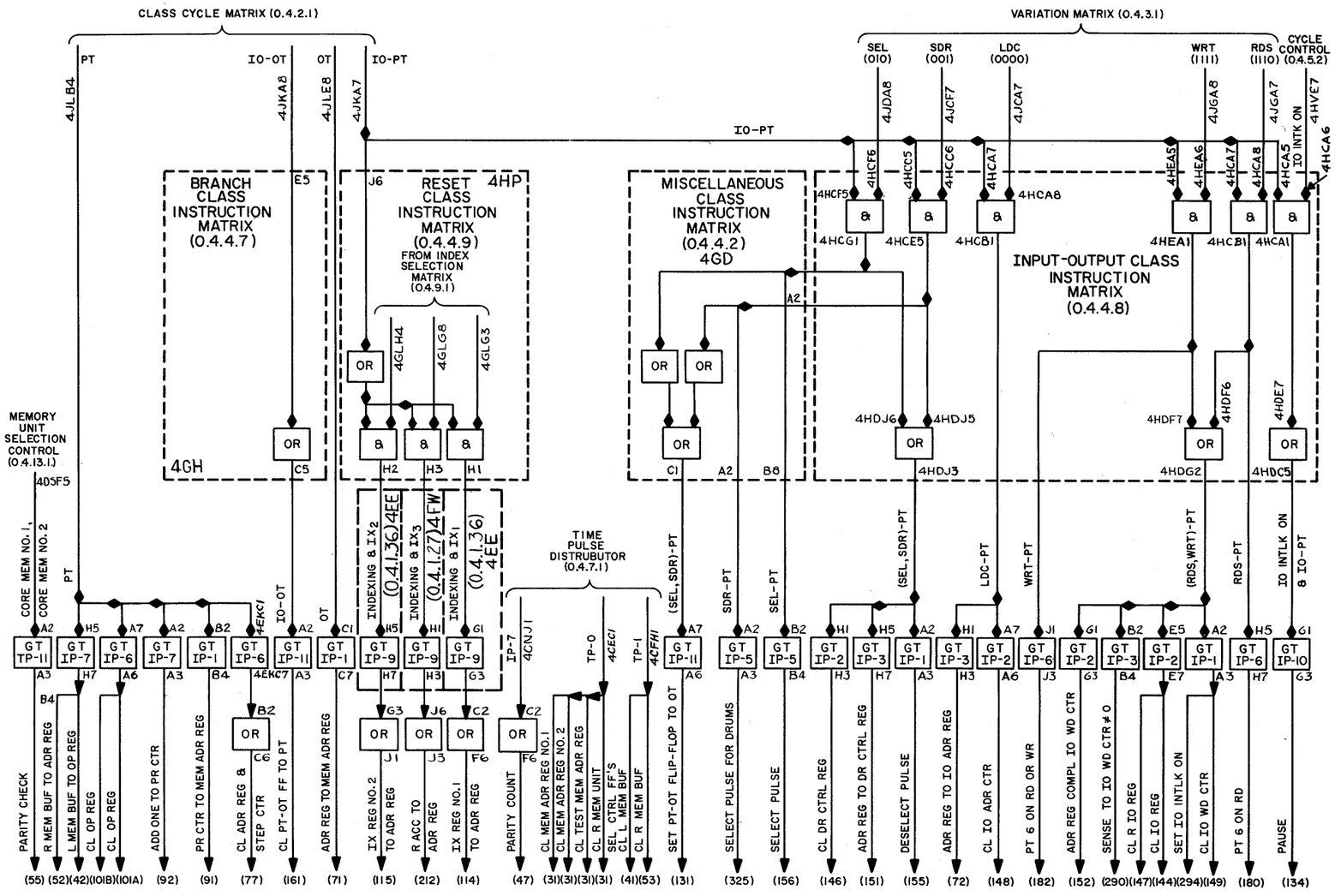


Figure 2–21. Input-Output Class, Logical Block Diagram

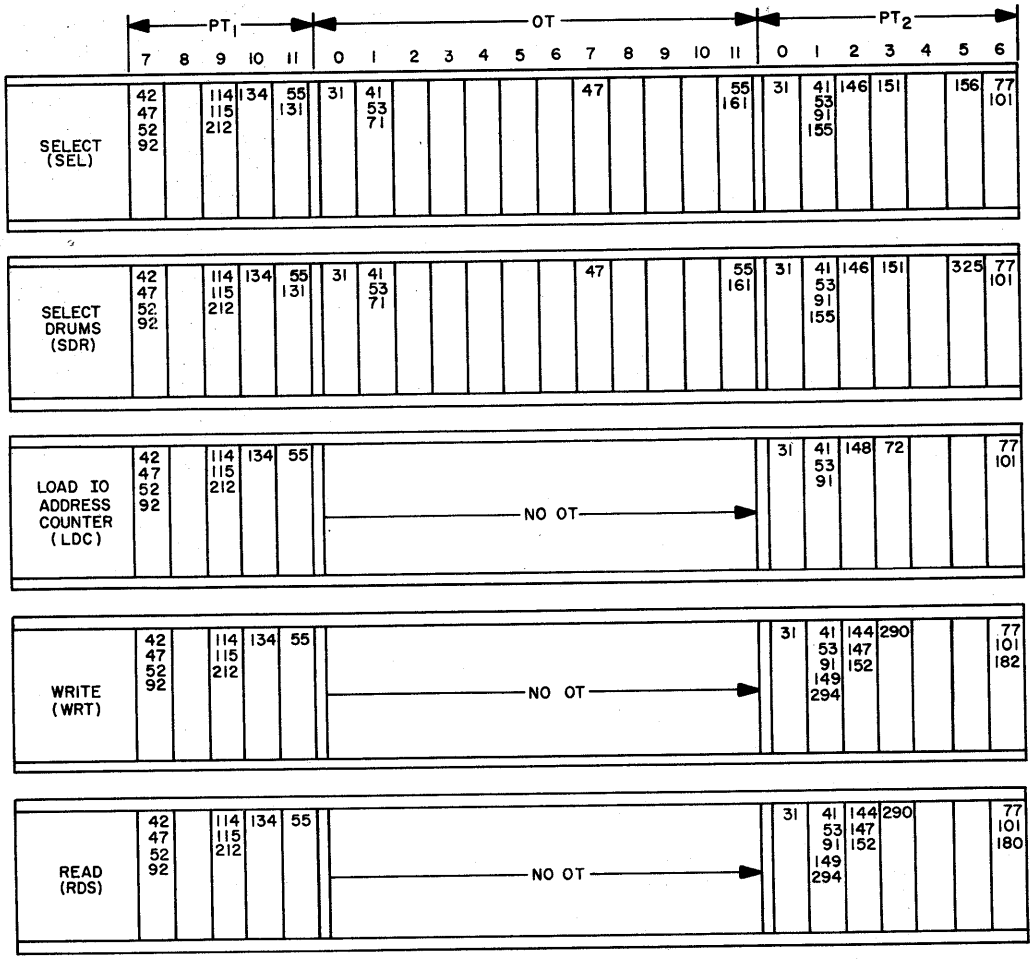


Figure 2-22. Input-Output Class, Command Sequence Chart

RESET CLASS COMMANDS

(Fig. 2–23.)

COMMAND NUMBER	COMMAND NAME	PLUGGABLE UNIT NO.	LOGICAL REFERENCE NO.
31	*clear memory address register and clear test memory address register*	4DV 4DY	(0.4.1.10) (0.2.1.1)
41	*clear left memory buffer register*	4DX	(0.4.1.21)
42	*left memory buffer register to operation register*	4DY	(0.4.1.21)
47	*parity count*	4DX	(0.4.1.21)
52	*right memory buffer register to address register*	4DY	(0.2.2.1)
53	*clear right memory buffer register*	4DX	(0.4.1.21)
55	*parity check*	4EH	(0.4.1.33)
76	*address register to right A register*	4EJ	(0.4.1.33)
77	*clear address register and step counter*	4EK	(0.4.1.33)
78	*address register to index register No. 2*	4EH	(0.4.1.33)
79	*address register to index register No. 1*	4EH	(0.4.1.33)
91	*program counter to memory address register*	4EH	(0.4.1.33)
92	*add 1 to program counter*	4FF	(0.4.1.51)
101	*clear operation register*	4EE	(0.4.1.36)
113	*clear index register No. 1*	4EF	(0.4.1.36)
114	*index register No. 1 to address register*	4EF	(0.4.1.36)
115	*index register No. 2 to address register*	4EE	(0.4.1.36)
116	*clear index register No. 2*	4FV	(0.4.1.27)
212	*right accumulator register to address register*	4EM	(0.4.1.23)
230	*clear right A register*		

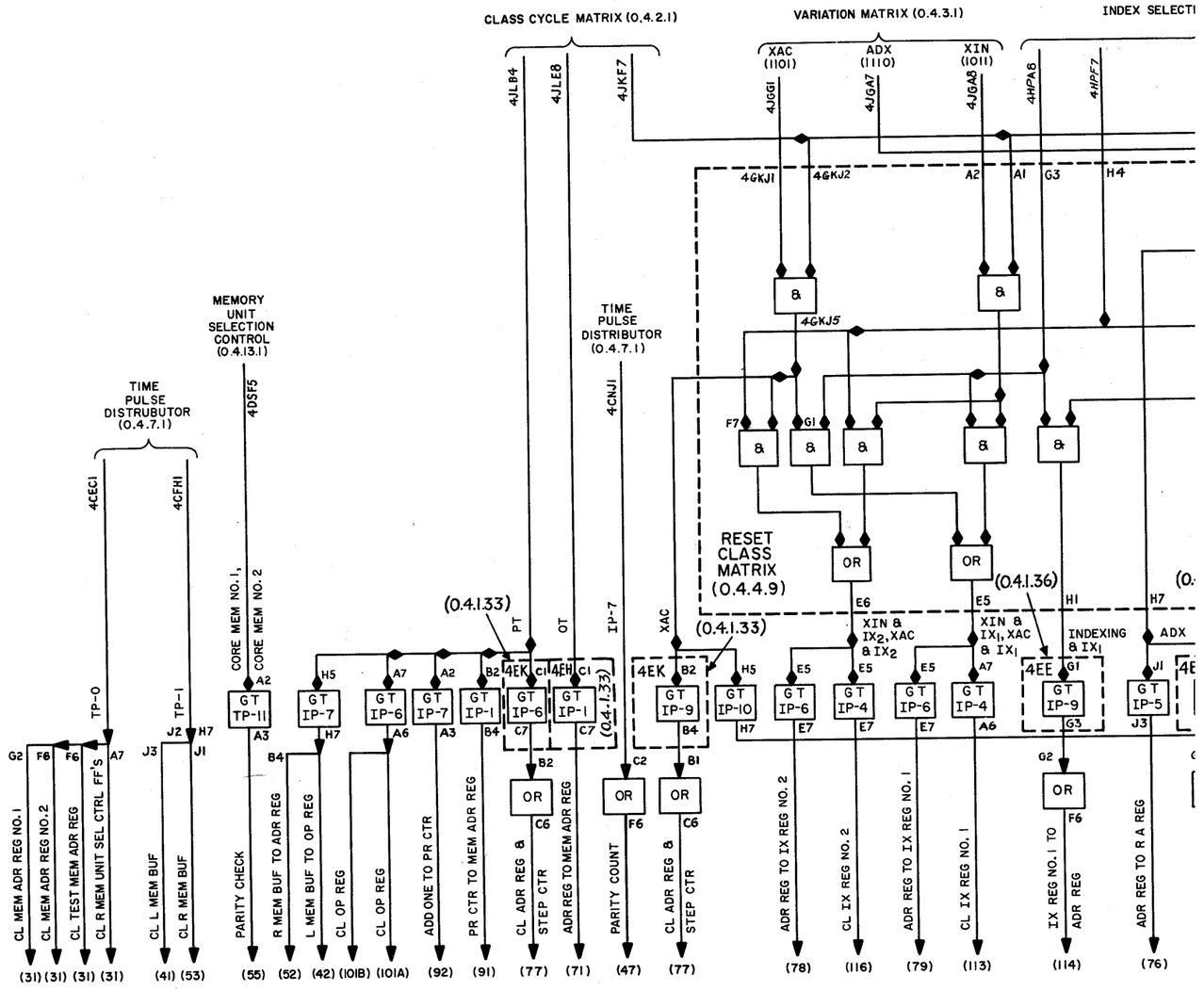


Figure 2–23. Reset Class, Logical Block Diag

RESET CLASS COMMANDS

(Fig. 2–23.)

COMMAND NAME	PLUGGABLE UNIT NO.	LOGICAL REFERENCE NO.
r address register and clear test memory		
register	4DV	(0.4.1.10)
mory buffer register	4DY	(0.2.1.1)
buffer register to operation register	4DX	(0.4.1.21)
	4DY	(0.4.1.21)
y buffer register to address register	4DX	(0.4.1.21)
emory buffer register	4DY	(0.2.2.1)
	4DX	(0.4.1.21)
ter to right A register	4EH	(0.4.1.33)
register and step counter	4EJ	(0.4.1.33)
ter to index register No. 2	4EK	(0.4.1.33)
ter to index register No. 1	4EH	(0.4.1.33)
nter to memory address register	4EH	(0.4.1.33)
ram counter	4EH	(0.4.1.33)
on register	4FF	(0.4.1.51)
egister No. 1	4EE	(0.4.1.36)
r No. 1 to address register	4EF	(0.4.1.36)
r No. 2 to address register	4EF	(0.4.1.36)
egister No. 2	4EE	(0.4.1.36)
ulator register to address register	4FV	(0.4.1.27)
l register	4EM	(0.4.1.23)

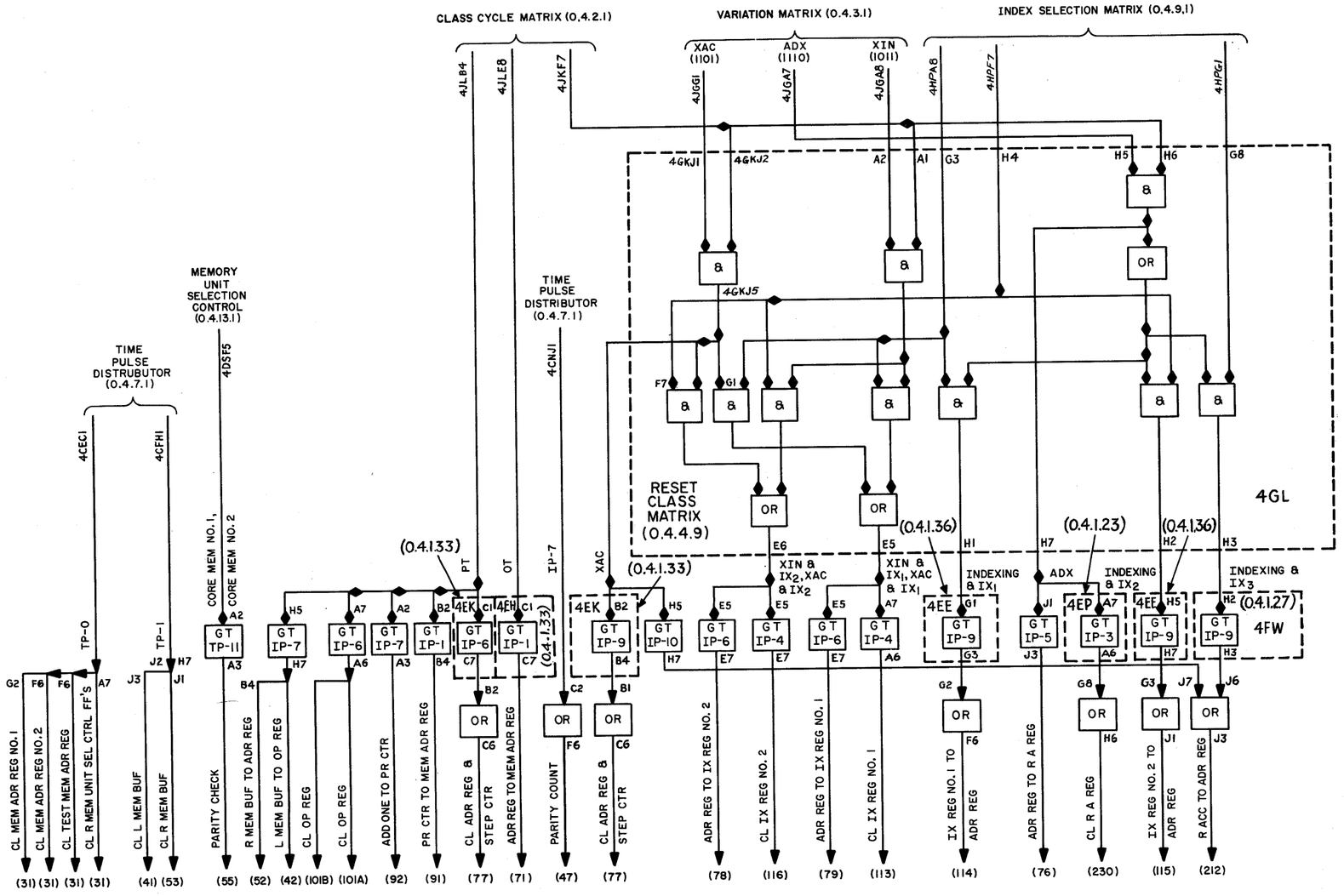


Figure 2–23. Reset Class, Logical Block Diagram

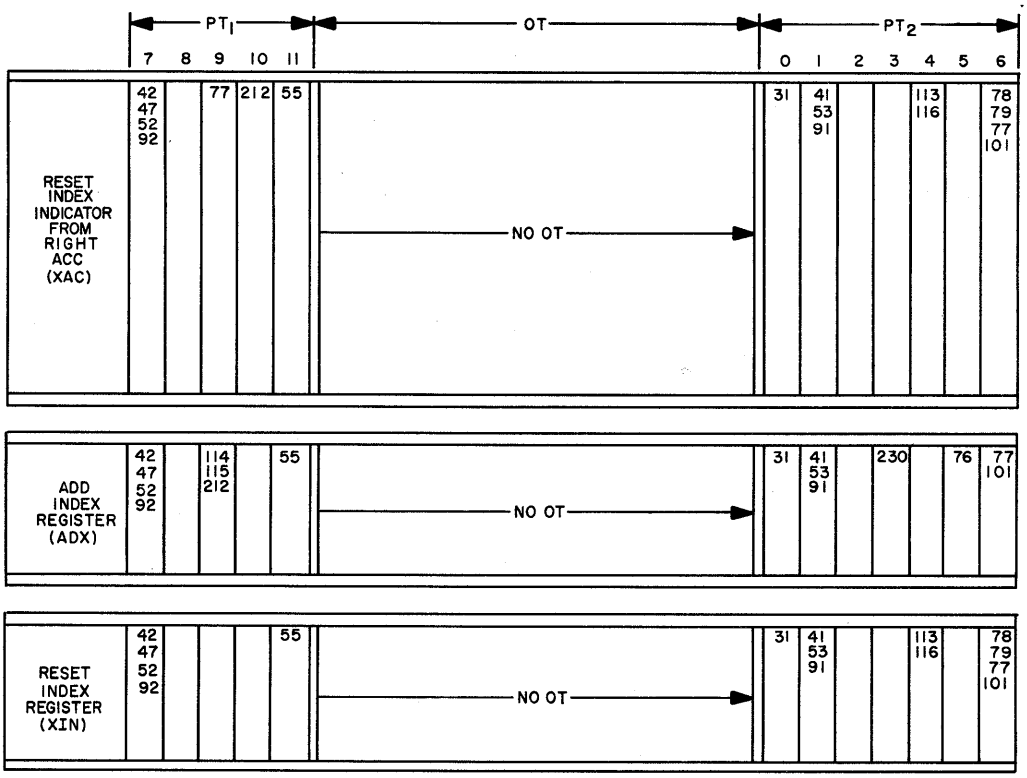


Figure 2–24. Reset Class, Command Sequence Chart

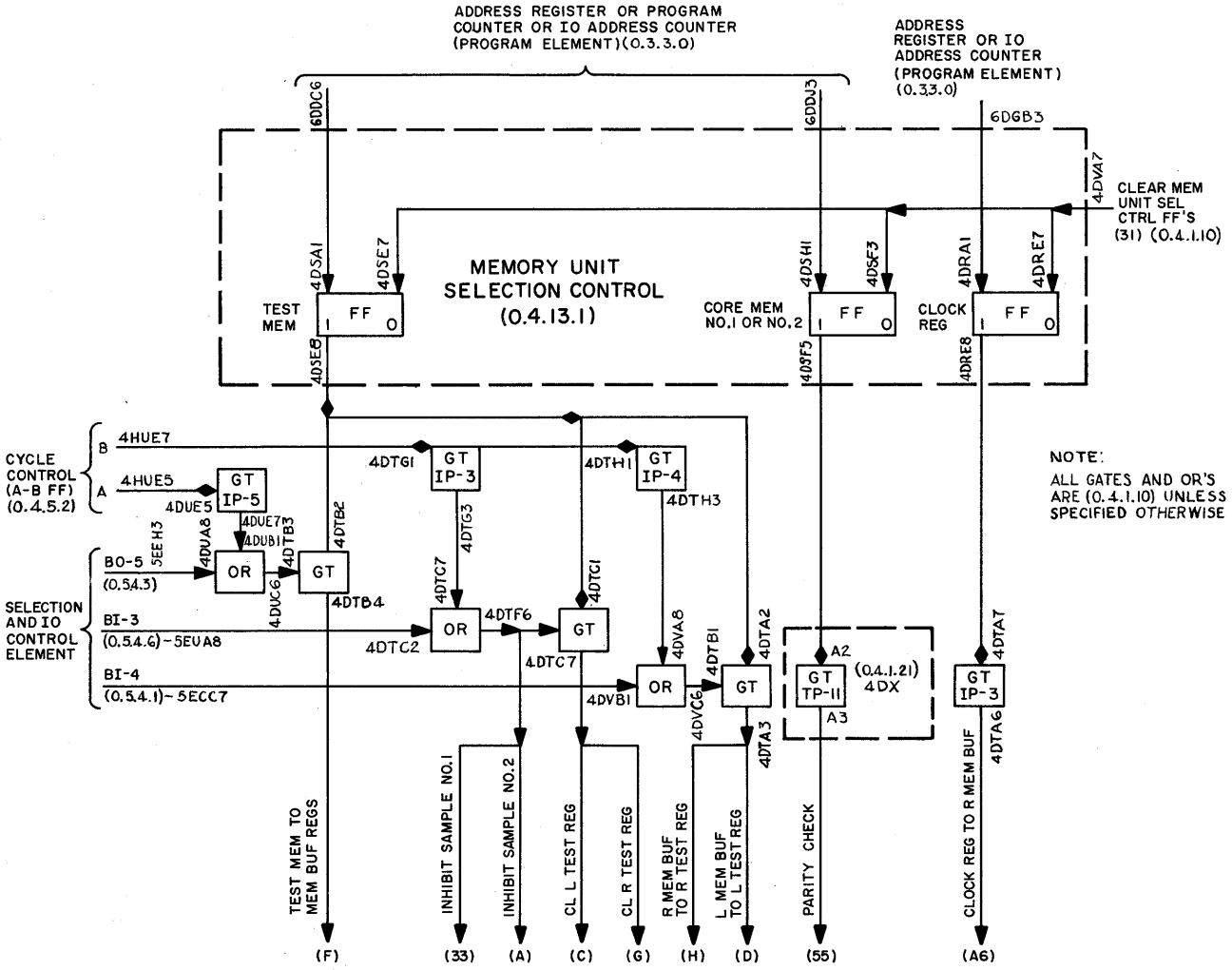


Figure 2-25. Memory Unit Selection Control, Logical Block Diagram

CHAPTER 3

PULSE GENERATION AND CONTROLS

3.1 BLOCK DIAGRAM ANALYSIS

As stated in the introduction, the instruction control element furnishes three classes of pulses to the Central Computer System: time pulses (TP), instruction pulses (IP), and 2-megacycle pulses. All of these pulses are 0.1 microsecond wide, have an amplitude between 20 and 40 volts, and occur at 0.5-microsecond intervals. The time pulses are formed from the 2-megacycle pulses by undergoing a sequencing process whereby they subdivide the basic machine cycle into fifteen 0.5-microsecond intervals. The instruction pulses, also formed from the 2-megacycle pulses, are produced in step with the time pulses, and in a similar manner. The instruction pulses are used to pulse the command generators in the instruction control element which are gated by d-c levels as described in Chapter 2 and become commands. The main function of the time pulses is to provide a timing reference for the Central Computer System. Certain time pulses are sent to the core memory element to initiate memory cycles; others are sent to the selection and IO control element, where they are used in generating break-in and break-out pulses. The 2-megacycle pulses are used only during pauses. The instruction pulses are suppressed during these times and their command generators are inactive. The command generators whose outputs are required during pauses are pulsed by the 2-megacycle pulses.

The interconnections of the units comprising the pulse generation and control equipment are illustrated in figure 2–26. These units are the crystal oscillator, the time pulse distributor control, the time pulse distributor, the step counter, and the divide time pulse distributor. With the exception of the units enclosed with dashed lines, all the units shown in the figure are included in the instruction control element. The oscillator runs continuously and supplies pulses to the time pulse distributor control. The 2-megacycle pulses applied to the time pulse distributor control are gated by signals from other units which reflect the status of the Central Computer System.

The selection and IO control element furnishes information concerning break operations; the command generators supply specific commands derived from the decoding process; the step counter provides flip-flop synchronizing signals and also information concerning its own current status. In accordance with the input data, the time pulse distributor control directs the operation of the time pulse distributor, senses for additional break requests in the selection and IO control element, and, if it has turned off the instruction pulses in the time pulse distributor, furnishes 2-megacycle pulses to the command generators. The time pulse distributor, under the direction of the time pulse distributor control, can produce time pulses and instruction pulses, time pulses alone, or no pulses. The time pulses are sent to the selection and IO control element for use in executing break cycles, and they are also sent as ungated commands to other elements in the Central Computer System. The instruction pulses, as described in Chapter 2, are sent to the command generators, where they become commands.

The divide time pulse distributor is an auxiliary unit to the time pulse distributor, replacing it during *Divide* instructions. This unit determines the duration of a divide cycle, transmits command pulses to the arithmetic element during this cycle, and sends a stepping pulse to the step counter at the end of every divide cycle. The step counter, which keeps track of the number of operations still to be performed in a pause, is set either by a command generator (74 or 75) or by the right memeory buffer register. It is cleared at the end of a pause by a signal from a command generator. The step counter has to furnish the following information to the instruction matrices: d-c signal when the number in the step counter is greater than 0 ($SC \neq 0$) and an accompanying signal when the number is greater than 6 ($SC > 6$). It must furnish a pulse to the time pulse distributor control at the following times: when the counter is going from 6 to 5 (only during shift class, *Multiply,* and *Twin and Multiply* instructions); when the counter is going from 2 to 1; and when the counter is going from 1 to 0 (if only one shift is required by the instruction).

The pulse generation and control equipment employs flip-flop synchronizing circuits to insure

against possible erratic operation of the Central Computer System. If the synchronizing circuits were not incorporated, certain key flip-flops might condition their gate tubes in such a way that only a part of a pulse applied to the gate tube would be transmitted. Reduced amplitude pulses are not reliable signals since they may not perform their assigned function. Only four of the flip-flops to be treated in this section are synchronized; these are located in the time pulse distributor control. They are the time pulse distributor control, the continue, the pause, and the 2-megacycle operate flip-flops. Figure 2–27 shows the synchronizing circuit of the 2-megacycle operate flip-flop and is typical of the other three.

The set-2-megacycle operate-sync flip-flop is set by command 138 occurring at OT–9 during the appropriate instruction. The pulse which is to be made reliable will eventually appear on the output line. Therefore, the gate tube conditioned by the 2-megacycle operate flip-flop is the one whose d-c level must be in synchronism with the applied

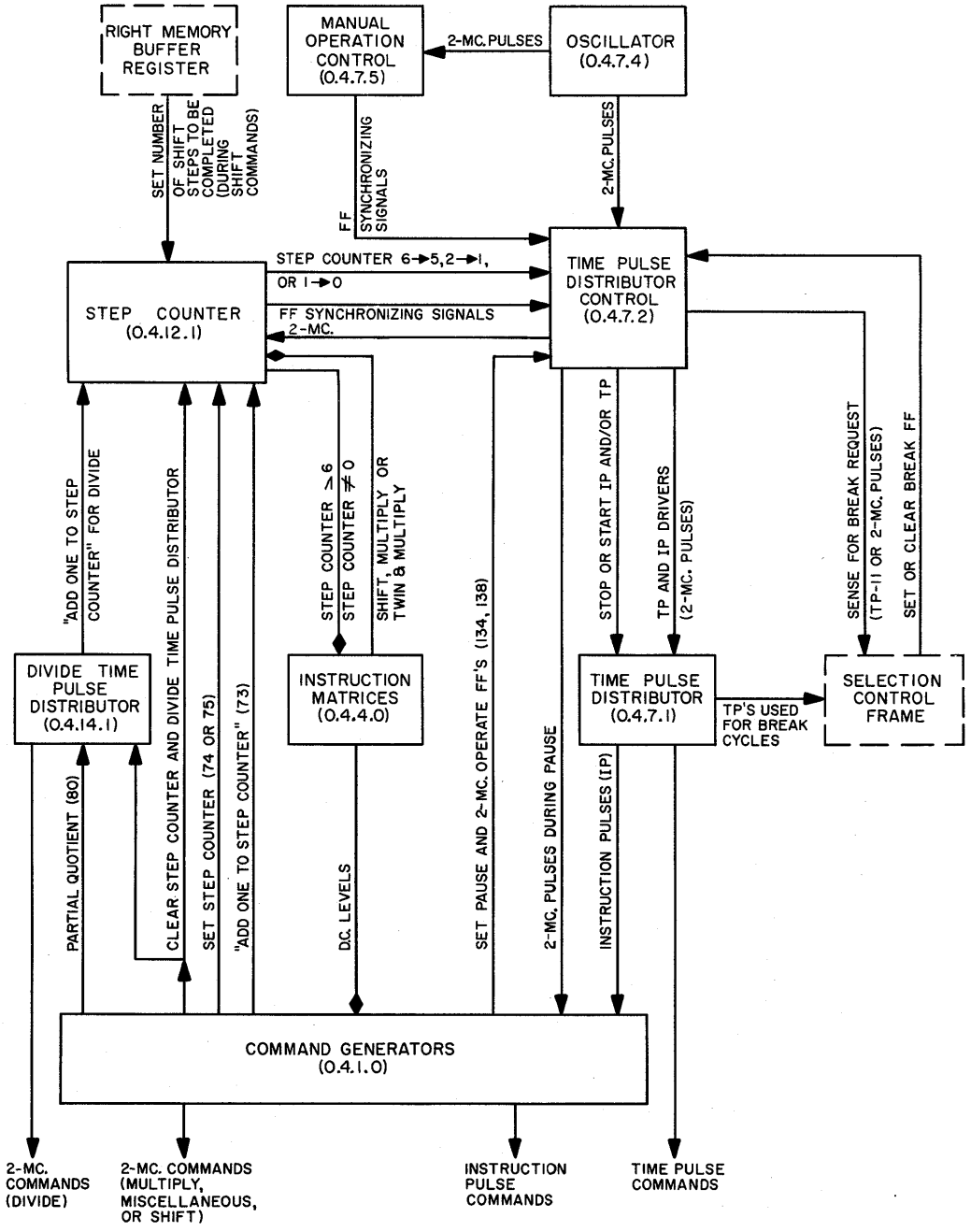


Figure 2–26. Pulse Generation and Control, Logical Block Diagram

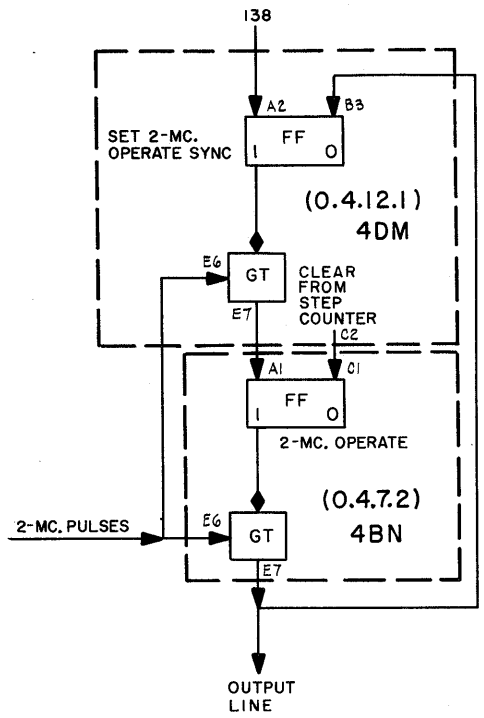


Figure 2–27. Synchronizing Circuit

2-megacycle pulses. Setting the synchronizing flip-flop conditions the upper, non-critical, gate tube. The first 2-megacycle pulse passes through the upper gate tube and sets the 2-megacycle operate flip-flop. However, the delay involved in setting this flip-flop does not permit the first pulse to pass through the critical gate tube and on to the output line. In the 0.5-microsecond interval before the appearance of the second 2-megacycle pulse, the 2-megacycle operate flip-flop has ample time to settle and fully condition its gate tube. Accordingly, the second pulse and all succeeding pulses are passed to the output line as full-amplitude pulses. The first pulse to appear in the output will, as a secondary function, clear the synchronizing flip-flop, thus disabling its circuit until such time as it is again required. It should be noted that a similar circuit can be used to clear, as well as set, a key flip-flop. Such a modification is used for the pause flip-flop. Each unit in the pulse generation equipment is treated in detail in the ensuing paragraphs.

3.2 CRYSTAL OSCILLATOR (0. 4. 7. 4)

The primary source of the pulses used in the Central Computer System of AN/FSQ–7 (XD–1) Combat Direction Central equipment is the crystal oscillator. The sine waves generated by the crystal-controlled oscillator in this unit are clipped and shaped into pulses of 0.1-microsecond duration occurring at 0.5-microsecond intervals. The pulses thus formed are fed to the time pulse distributor control.

3.3 TIME PULSE DISTRIBUTOR CONTROL

The time pulse distributor control (TPDC) directs the action of the time pulse distributor (TPD). The TPDC, shown in figure 2–28 contains eight flip-flops, only five of which are described here. The five described are the time pulse distributor control, pause, break, 2-megacycle operate, and continue flip-flops. The remaining three flip-flops are controlled by pushbuttons on the maintenance console and are not described in this publication.

The time pulse distributor control flip-flop acts as a switch which, by setting the d-c level of gate tube 1, controls the flow of 2-megacycle pulses to the command generators, to the IP driver line feeding the TPD, and the break request circuit in the selection and IO control element. The time pulse distributor control flip-flop is normally on during automatic operation of the Central Computer System, thereby conditioning gate tube 1. This gate tube is pulsed by a line from the oscillator and passes 2-megacycle pulses to gate tubes 2, 3, 4, and 5. These gate tubes are conditioned, respectively, by the pause, 2-megacycle operate, break, and a combination of the pause and break flip-flops. The 0 side of the time pulse distributor control flip-flop conditions gate tube 6, which merely clears the time pulse distributor control synchronizing flip-flop (located in the manual operations control).

The pause flip-flop is set (through a synchronizing circuit in the step counter) to the 1 side by command 134 *(pause)*, this command being generated at IP–10 by those instructions requiring a pause. In going from 0 to 1, the pause flip-flop turns off the TPD unless a break is in progress. It also enables gate tube 5 to furnish 2-megacycle pulses for sensing the break request circuit in the selection and IO control element. (This sensing is performed by PT–11 when the TPD is on.) At the end of the required pause, the step counter provides a pulse which clears the pause flip-flop, restarting the TPD.

The break flip-flop in the TPDC acts in conjunction with the break flip-flop in the selection and IO control element. The two flip-flops are set simultaneously when a break is to be performed. If the TPD is on, the sense pulse which sets the break flip-flops occurs at TP–11; if the TPD is off, the sensing is done at a 2-megacycle rate. The

break flip-flop in the selection and IO control element permits break-in or break-out cycles to be performed. The break flip-flop in the TPDC starts the TPD, which then generates the TP's used for the break cycle. At the same time, setting the break flip-flop effectively opens the IP driver line (by deconditioning gate tubes), thus suppressing IP's during break operations. This suppression is independent of the status of the pause flip-flop.

The 2-megacycle operate flip-flop, set to the 1 side (through a synchronizing circuit in the step counter) by command 138, acts as a switch for the 2-megacycle pulses which pulse the command generators during pauses. The switching is performed by gate tube 3; it is conditioned by the 1 side of

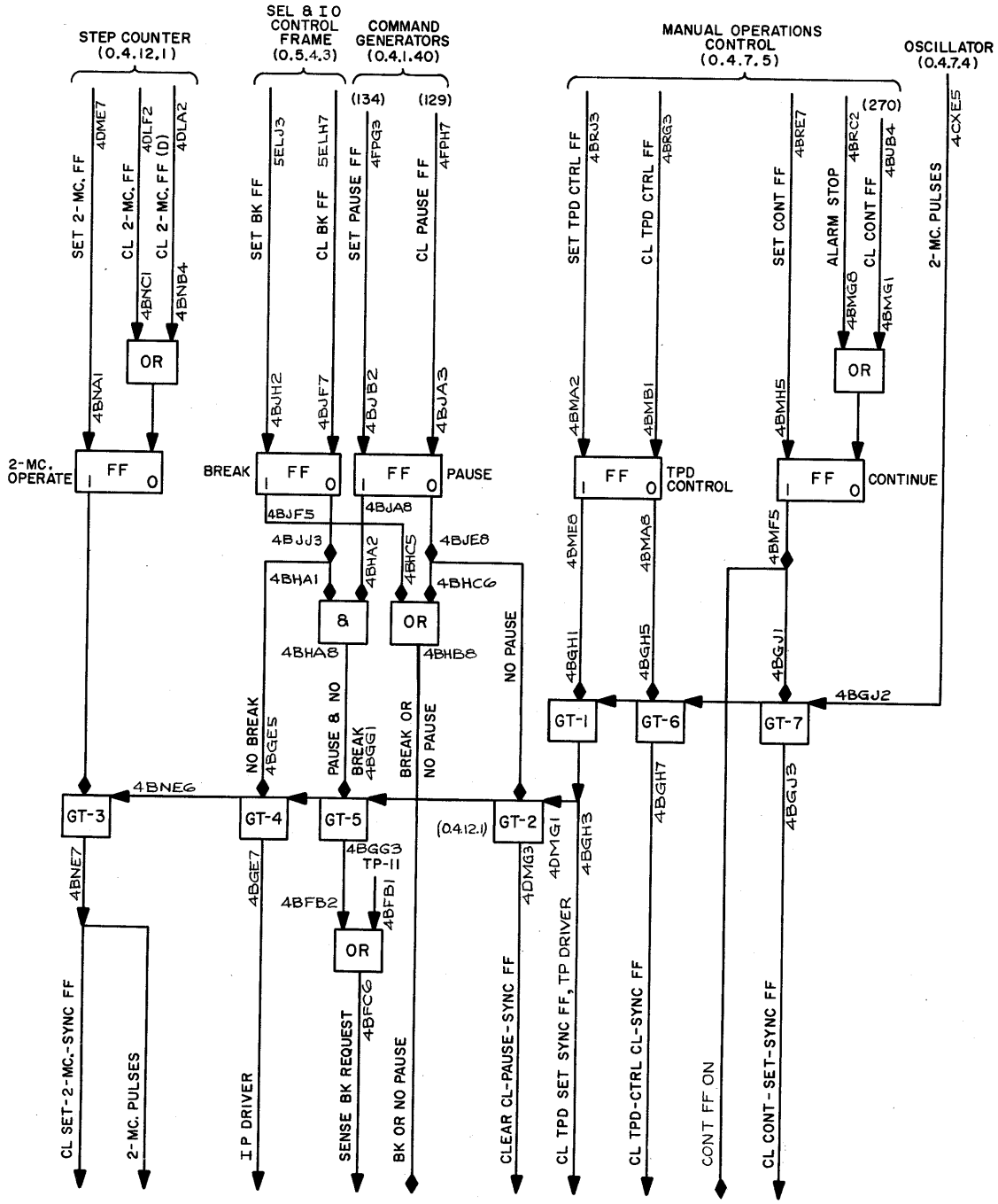


Figure 2–28. Time Pulse Distributor Control, Logical Block Diagram

the 2-megacycle operate flip-flop. The output of gate tube 3, in addition to furnishing the pulses to the command generators, clears the synchronizing flip-flop (located in the step counter) for the 2-megacycle operate flip-flop. The synchronizing circuit acts only for the first few pulses of a 2-megacycle operation. The flow of 2-megacycle pulses to the command generators is discontinued when the 2-megacycle operate flip-flop is cleared by a pulse from the step counter.

The continue flip-flop is set to the 1 side when the Central Computer System is started. It is cleared only during an alarm condition or during the execution of *Program Stop* instruction. The 1 side of this flip-flop conditions gate tube 7 and is one of the conditioning lines (in conjunction with the 1 line of flip-flop 7 in the TPD) for an AND circuit which determines whether the TPD is to continue generating TP's or is to stop at TP–6.

3.4 TIME PULSE DISTRIBUTOR

The time pulse distributor (TPD) converts free-running pulses from the oscillator into 0.5-microsecond instruction pulses and time pulses. The TP pulses are produced as consecutive chains of 15 sequenced pulses, only 12 of which are used. The IP pulses are produced as consecutive trains of 15 pulses, only 11 of which are used (IP–0 suppressed). The extra pulses are used in each case to elongate the time duration of the individual pulse trains (machine cycles) to conform with the 7.5-microsecond second memory cycle. The 2-megacycle pulses are fed directly to the TPD but their conversion into IP's and TP's is controlled by the time pulse distributor control (TPDC). This control dictates whether the TPD will produce TP and IP pulses simultaneously, TP pulses alone, or will be turned off.

A logical block diagram of the TPD appears in figure 2–29. The control element for the TPD is the AND circuit associated with flip-flop 0. When this AND circuit is up (break or no pause condition), gate tube 0 is conditioned and will pass a 2-megacycle pulse supplied by the TP driver line. The output of this gate tube clears flip-flop 0 and conditions gate tube 1 by conditioning flip-flop 1. This same output is sent to the Central Computer System as TP–0. The second 2-megacycle pulse from the TP driver line is now rejected by gate tube 0 but passed by gate tube 1. The output of gate tube 1 clears flip-flop 1, conditions gate tube 2, and is used in the Central Computer System as TP–1. This process continues without interruption until flip-flop 7 is set. An additional condition must be met at this point if the process is to continue; the continue flip-flop (in the TPDC) must be in the 1 side. If this condition is met, the process is not interrupted and continues until the 15 time pulses have been formed. The final pulse, TP–11A, in addition to clearing flip-flop 11A, resets flip-flop 0 and the entire process is repeated.

A second set of four gate tubes parallels gate tubes 1 through 11A used for TP generation. These tubes generate the instruction pulses and are pulsed by the IP driver line. Note that no gate tube 0 is used, thereby suppressing IP–0. Instruction pulses cannot be generated independently because their condition flip-flops are controlled by the TP generation process. However, the IP pulses can be suppressed independently of the TP pulses by de-energizing the IP driver line.

3.5 STEP COUNTER

The function of the step counter is to keep track of the number of shifts completed during an instruction and to transmit to the Central Computer System, during the computing process, signals which reflect the current status of the counter circuit. The step counter contains a six-bit counter circuit, two synchronizing flip-flops, and several AND and OR circuits which reflect the status of the counter circuit while it is performing its function. Figure 2–30 is a logical block diagram of the step counter. The 1 side of each flip-flop in the counter has an additional input that is not shown. All these inputs come from the right memory buffer register and convey meaningful information (number of shifts to be performed) only during shift instructions.

The counter circuit represents a six-bit number in the binary system. This number of bits is necessary to accommodate the maximum number of shifts that a shift instruction can call for. Three types of instructions (shift class, *Multiply*, and *Divide*) utilize the counter circuit. Whenever information is transferred from the right memory buffer register to the address register (in the program element), the information in bits 10 through 15 of the word transferred is simultaneously sent to the step counter. This information indicates the number of shifts required during a shift class operation. In the case of a *Multiply* instruction, 15 shifts are always required; in the case of a divide instruction, 17 shifts are always required. In each case, the prescribed number of shifts is set into the counter by a command which writes over the information set in by the right memory

buffer register and which is meaningless for these latter two instructions. Once the number has been set into the counting circuit, the step counter is reduced by 1 each time a shift is completed. These shifts occur only during 2-megacycle operation.

The *Multiply* instruction requires 15 individual multiplication operations, each operation being performed by a single 2-megacycle pulse. Accordingly, the step counter is set to 15 for this instruction. The step counter is reduced by 1 for each multiplication operation, and after 14 such operations the counter is stepped from 2 to 1. The step counter circuit is designed so that when it is stepped from 2 to 1, one additional 2-megacycle pulse is developed before the generation of 2-megacycle pulses is stopped. This last 2-megacycle pulse performs the final or 15th multiplication operation. For the *Multiply* instruction, therefore, the step counter is always set to 15.

The *Divide* instruction requires 16 individual division operations, but each division operation requires five 2-megacycle pulses. The step counter is reduced by 1 by the fourth pulse in each division operation. The 2-megacycle pulse which immediately precedes the stepping of the counter then merely performs the final step for the particular division operation in progress. (This differs from the case of multiplication where the additional pulse performed a complete multiplication operation.) If the step counter were set to 16 for a *Divide* instruction, the step counter would be going from 2 to 1 while the 15th divide operation was being performed. The additional pulse furnished before 2-megacycle operation was stopped by the step counter would then merely complete the final step for the 15th division operation, and the 16th divide operation would not be performed. Therefore, the step counter is set to 17 for the *Divide* instruction. The counter is then stepped from 2 to 1 while the 16th divide operation is in progress, and the additional 2-megacycle pulse completes the fifth and final step for this 16th operation.

For the shift class and *Multiply* instructions, each 2-megacycle pulse performs two functions; it shifts the operand by one bit position and simultaneously pulses the counter circuit in the step counter. For a *Divide* instruction, every fifth 2-megacycle pulse performs these two functions. (The application of every fifth pulse to the step counter is accomplished by the divide time pulse distributor.) In either case, the pulse sent to the step counter is applied to the gate tube associated with the lowest order bit of the counter circuit by the add 1 to step counter line.

To illustrate the operation of the counter circuit, assume that a *Divide* instruction is to be performed (17 shifts required). At OT–7, the counting circuit is set to 17 by the *set step counter to 17* command (74). This command sets 1's into the second and sixth bits and clears the remaining bits to 0 (010001). The next pulse reduces the number to 16 (010000). Now consider the second pulse. As before, the pulse is applied to gate tube 6 and also complements the flip-flop. However, the inherent delay in the flip-flop complementing process allows the gate tube to remain conditioned (since it originally contained a 0) long enough to pass the 2-megacycle pulse. This pulse is transmitted to bit 5, where it simultaneously inspects gate tube 5 and complements flip-flop 5. Since flip-flop 5 contained a 0, its gate tube transmits a pulse to bit 4 and, by the same process, the complementing pulse is passed through the counter until, at bit 2, the propagation of the pulse is stopped by gate tube 2. This gate tube is not conditioned since a 1 is held in bit 2, but note that flip-flop 2 is complemented. The counter continues the count in this manner until it reaches 0, at which time the pause flip-flop in the time pulse distributor control is cleared, and the next program cycle is initiated by the time pulse distributor.

The Central Computer System must know the current status of the counter at all times in order that it may synchronize the operations necessary to execute an instruction. The step counter furnishes the Central Computer System with signals that indicate the following: the number held in the counter is greater than 0 (SC $\neq 0$), greater than 6 (SC$>$6), less than 8 (SC$<$8), less than 4 (SC$<$4), and less than 2 (SC$<$2). The SC $\neq$ 0 condition is detected by an OR circuit whose inputs are the 1 lines from all the bits in the counter. The only condition for no output from this OR circuit is that each bit contain a 0. A combination of an AND and an OR circuit is used to determine if SC$>$6. If any of the three highest-order bits contains a 1, the number in the counting circuit is necessarily greater than 6. Therefore, the 1 lines of each of these bits are used as inputs to an OR circuit whose output states SC$>$6. Even when these three bits contain 0's, one possibility of a number greater than 6 (111) still exists. This condition is accounted for by combining the 1 lines of the three lowest-order bits in an AND circuit whose output is the fourth input to the SC$>$6 OR circuit. The 0 lines from the three highest-order

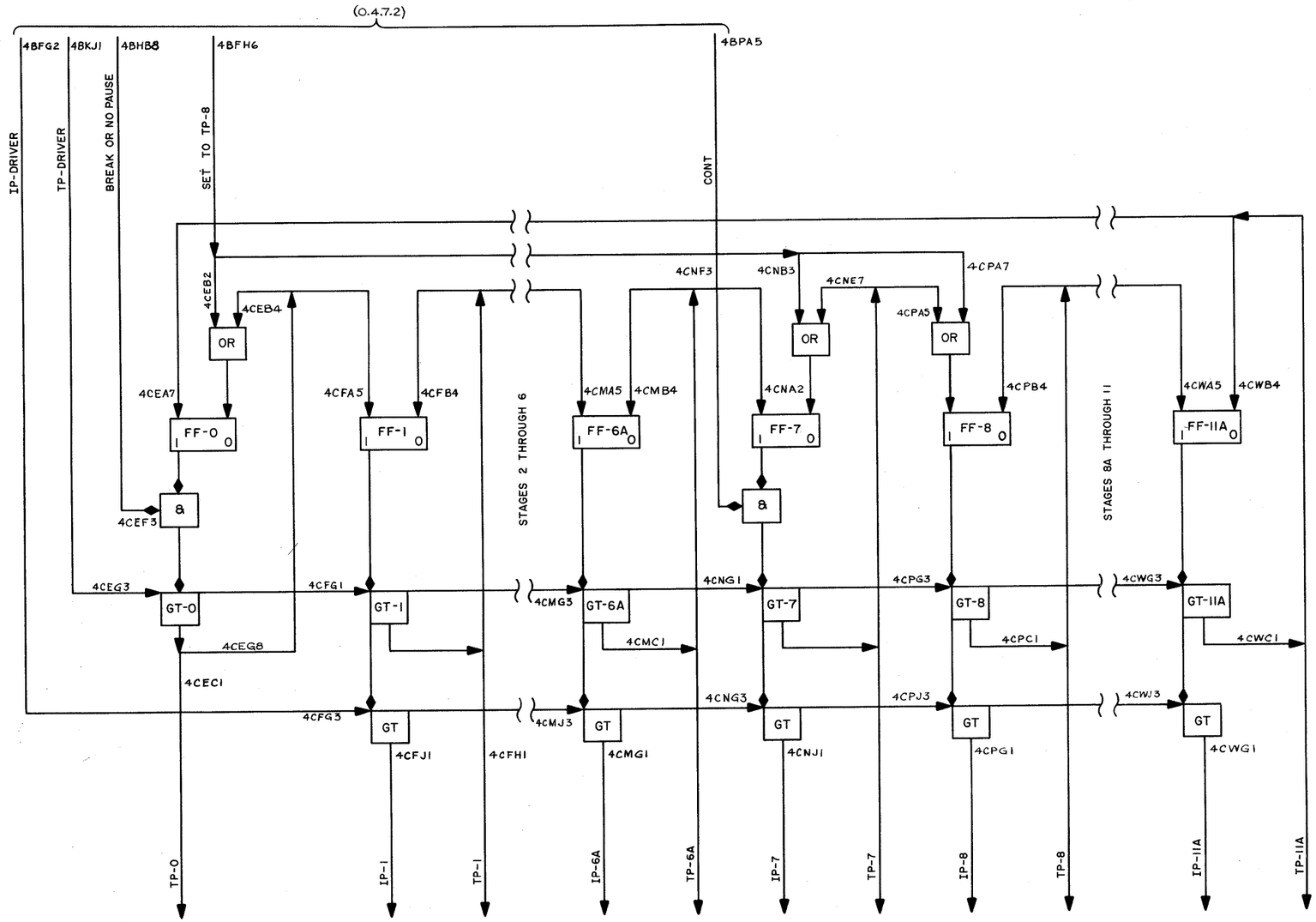


Figure 2–29. Time Pulse Distributor, Logical Block Diagram

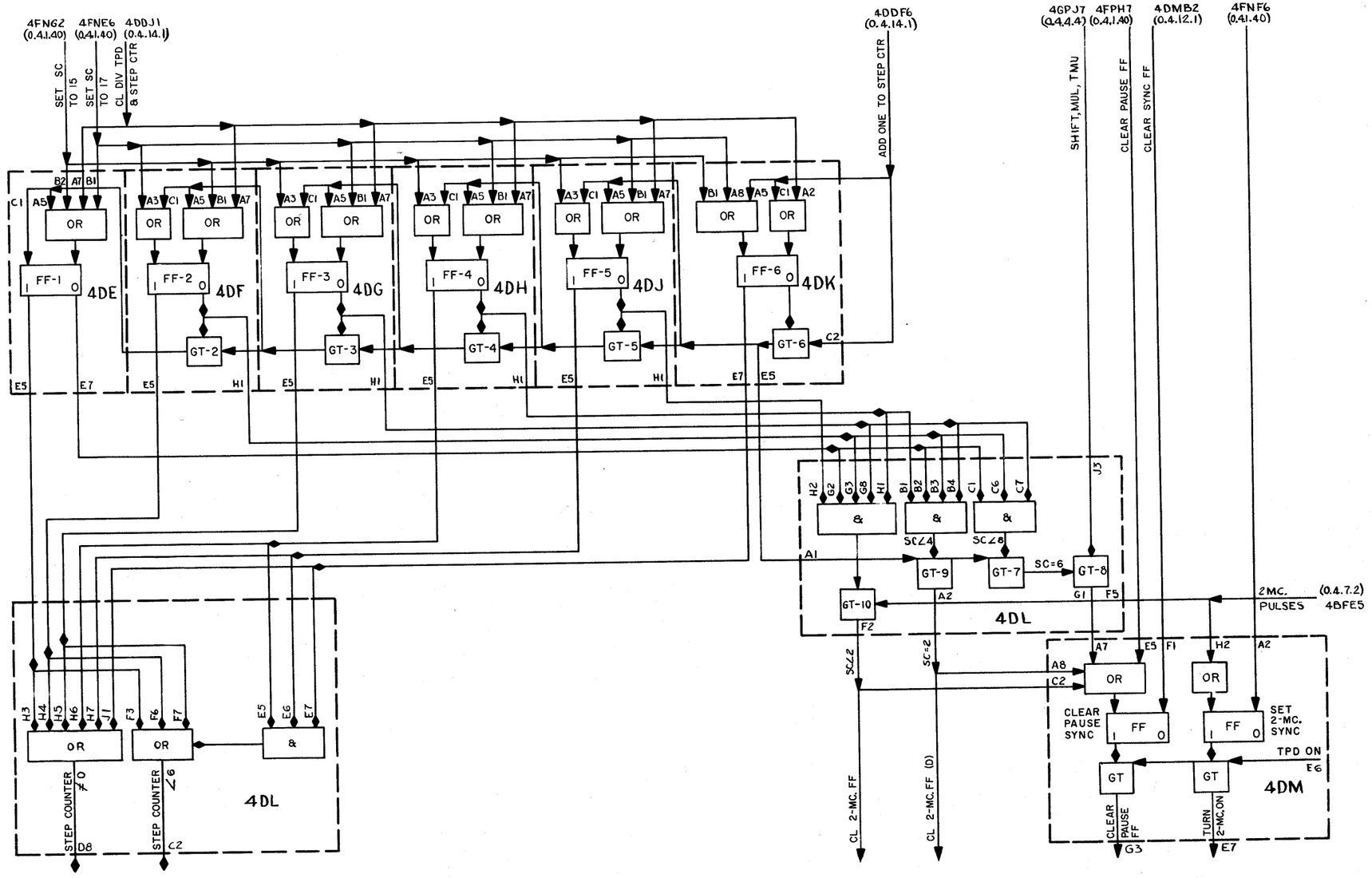


Figure 2–30. Step Counter, Logical Block Diagram

bits are used to discern the SC<8 status because a 1 in any of these bit positions will indicate a number greater than 8. Similarly, the SC<4 AND circuit utilizes the 0 lines of the four highest-order bits and the SC<2 AND circuit employs the 0 lines from the five highest-order bits. The outputs of the SC $\neq$ 0 and SC>6 AND circuit are used for conditioning command generators. The SC>6 line, in conjunction with the shift lines from the cycle control, is used to turn on command 134 for the duration of any shift instruction. The SC $\neq$ 0 line turns on command generator 138 during these instructions and also turns on both command generators 134 and 138 during a *Shift Left and Round* instruction (miscellaneous class). As soon as the counter is less than 8, the SC<8 AND circuit conditions gate tube 7. This gate tube is pulsed by the add 1 to step counter line every time an even number is contained in the counter. Therefore, when the counter reaches 6, a pulse is transmitted by gate tube 7 and inspects gate tube 8. Gate tube 8 is conditioned only if a shift class, *Multiply*, or *Twin and Multiply* instruction is in progress. In such cases, gate tube 8 passes the pulse from gate tube 7. This pulse is then used to set the clear-pause-sync flip-flop which in turn clears the pause flip-flop, restarting the time pulse distributor. This is done during the shift class, *Multiply*, and *Twin and Multiply* instructions to speed Central Computer System operation since instruction pulses will be resumed, initiating the next program cycle, while there are still five shift operations to be completed. All other instructions which utilize the step counter, however, must wait until the counter is reduced to 0 before initiating the next program cycle. These latter instructions set the clear-pause-sync flip-flop directly with the output line of the SC<4 AND circuit. Gate tube 9 is conditioned by this AND circuit and, like gate tube 7, waits until the next lower even number (2) appears in the counter before passing the pulse.

Because of inherent delays in the circuits involved, restarting of the time pulse distributor is initiated as the counter goes from 2 to 1 rather than 1 to 0. This same signal also clears the 2-megacycle flip-flop (in the time pulse distributor control) which terminates the pause (2-megacycle operation). In the special case where a shift class instruction requires only one shift, the SC<2 AND circuit is used to condition gate tube 10. Essentially, gate tube 10 is pulsed directly by the 2-megacycle oscillator and duplicates the function of gate tube 9. This is done because the inherent delay in the add 1 to step counter line which pulses gate tube 9 does not permit the pause flip-flop to restart the time pulse distributor quickly enough when only 1 shift is performed. By pulsing gate tube 10 with the 2-megacycle line, this delay is bypassed.

The step counter contains two flip-flops, the set-2-megacycle-sync and the clear-pause-sync flip-flops, which are used to synchronize the setting of the 2-megacycle operate flip-flop and the clearing of the pause flip-flop, respectively, with the 2-megacycle pulses from the oscillator. The two latter flip-flops condition gate tubes which are pulsed by free-running 2-megacycle pulses. The synchronizing flip-flop is cleared by the first 2-megacycle pulse allowed to pass through the gate tube conditioned by setting the 2-megacycle operate flip-flop. The clear-pause-sync flip-flop is set to 1 by the output of either gate tube 9 or gate tube 10 (depending on the number of shifts called for by the instruction). It is cleared by the first 2-megacycle pulse allowed to pass through the gate tube conditioned by clearing the pause flip-flop.

3.6 DIVIDE TIME PULSE DISTRIBUTOR

A *Divide* instruction requires 16 shifts during its execution. For all other instructions employing shifts, the step counter is reduced by 1 at a 2-megacycle rate. When dividing, however, stepping the counter at this rate is unsatisfactory because a number of suboperations must be completed between shifts. Therefore, the timing device used for division is the divide time pulse distributor (DTPD) which supplants the TPD and steps the counter at the proper rate.

The DTPD, shown in figure 2–31, operates in a manner similar to that of the TPD. Five flip-flops constitute the timing ring. The partial quotient line is pulsed at a 2-megacycle rate by command 80. The first 2-megacycle pulse passes through gate tube 0 (conditioned by flip-flop 0). This output pulse is set to the arithmetic element, clears flip-flop 0, and sets flip-flop 1. The second 2-megacycle pulse is passed by gate tube 1, conveying a signal to the arithmetic element, clearing its own flip-flop and setting flip-flop 2. The third 2-megacycle pulse appearing on the partial quotient line, while it clears flip-flop 2 and sets flip-flop 3, does not send any signal to the Central Computer System. The output of gate tube 3, through an OR circuit, energizes the same line (add 1 to step counter) which is used by command 73 during other instructions to reduce the step counter by 1. When command 73 is used, reducing

pulses for the step counter occur at a 2-megacycle rate. When the DTPD is used to reduce the step counter, however, five of these pulses must be applied to the timing ring before the DTPD again supplies a pulse to the step counter via the add 1 to step counter line. At the end of every *Divide* instruction, a pulse on the clear DTPD and step counter line clears every flip-flop in the DTPD except flip-flop 0, which is set to 1 in readiness for the next *Divide* instruction.

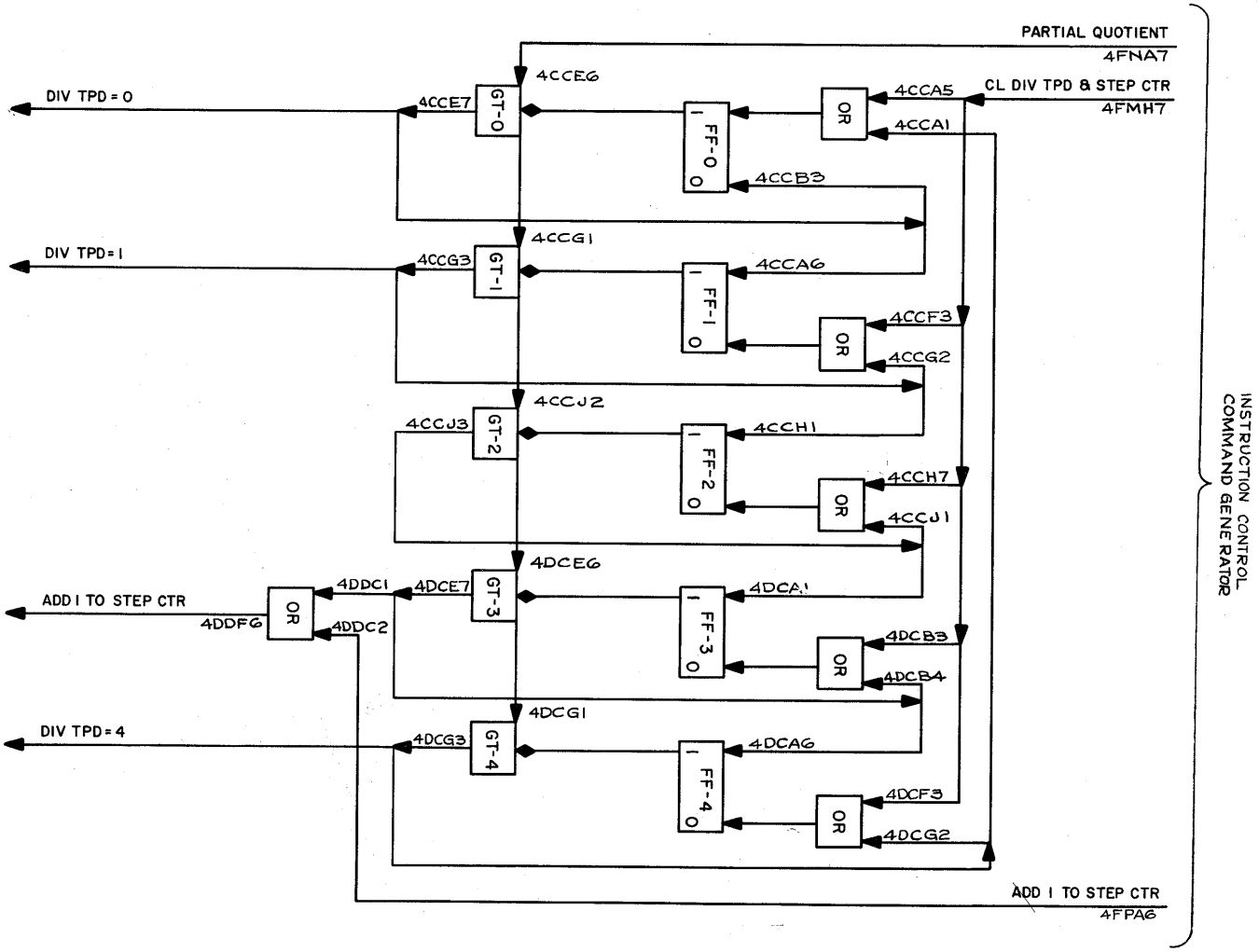


Figure 2–31. Divide Time Pulse Distributor, Logical Block Diagram

CHAPTER 4
COMMAND GENERATORS

The command generators are a group of gate tubes that combine the d-c levels from the instruction matrix and the instruction pulses from the time pulse distributor to produce the series of commands necessary to execute an instruction. The Central Computer System executes 153 different commands. Table 2–3 below shows the distribution of these commands.

The gate tubes that comprise the command generators are often followed by pulse amplifiers. The pulse amplifiers are necessary if the command goes to two destinations, as in the case for commands that go to both arithmetic units. Some command generators also generate two different commands at the same time. For example, the contents of a register may be transferred and the register cleared by the same time pulse. The transfer and the clear commands are amplified by different pulse amplifiers. If a command is generated by two different time pulses, two command generators are required and the outputs of the two command generators are combined in a pulsed OR circuit. The output of the pulsed OR is amplified by a pulse amplifier before it is sent to its destination.

The command generators are placed in the pluggable units according to the destination of the command pulses; all commands that apply to the accumulator registers are grouped together, commands that go to the index registers are grouped together, etc.

TABLE 2–3. DISTRIBUTION OF INSTRUCTION CONTROL ELEMENT COMMANDS

UNIT	COMMAND NUMBERS
Arithmetic Elements	
A Registers	11 21 22 23 25 26 211 225 226 228 229 230
Accumulator Registers	1 2 4 5 6 7 8 9 10 12 13 16 17 18 19 20 121 123 162 165 166 201 202 204 205 206 207 208 209 210 212 213 214 216 217 219
Adders	14 60 61 62 63 64 66 67 69 260 261 263 264 266 267
B Registers	81 82 83 84 85 87 88 89 281 282 283 284 285 287 288 289
Clock Register	A6
Memory Buffer Registers	D H 39 40 41 42 43 43 44 47 51 52 53 55
IO Registers	144 147
Test Memory Registers	31
Test Registers	C F G

TABLE 2-3. DISTRIBUTION OF INSTRUCTION CONTROL ELEMENT COMMANDS (cont'd)

UNIT	COMMAND NUMBERS
Instruction Control Element	
Cycle Control	131 132 161 163 167 170 294
Divide Time Pulse Distributor	80
Manual Operations Control	270
Memory Unit Selection Register	21
Operation Register	101
Step Counter	73 74 75 134 138
Program Element	
Address Register	71 72 76 77 78 79 151 152 154
Drum Control Register	146
IO Address Counter	148
IO Word Counter	149 150 157 158 290
Index Register No. 1	113 114 164
Index Register No. 2	115 116 174
Program Counter	91 92 93 94
Selection and IO Control Element	
Command Generators	180 182
Index Internal Register	103
Operate Gate Tube	104
PerSelBsn Matrix	140 156
Memory Element	
Command Generators	A B 32 33
Core Memory No. 1 Address Register	31
Core Memory No. 2 Address Register	31
Tapes	
Tape Unit	321
Drum Unit	
Read or Write Register	155 325

PART 3
ARITHMETIC ELEMENT

CHAPTER 1
INTRODUCTION

This Part presents the theory of operation for the left and right arithmetic elements of AN/FSQ–7 Combat Direction Central. Each element is contained in its own arithmetic unit. In keeping with the general practice, a distinction is made between an element and a unit, the element being the logical or functional entity and the unit being the physical entity. The components contained in the two units are listed in table 3–1.

TABLE 3–1. ARITHMETIC ELEMENT REGISTERS

LEFT ARITHMETIC UNIT	RIGHT ARITHMETIC UNIT
Left A register	Right A register
Left accumulator register	Right accumlator register
Left adders	Right adders
Left B register	Right B register
Left memory buffer register	Right memory buffer register
Left IO register	Right IO register
Left test register	Right tester register
	Clock register

Because of engineering considerations, circuits which logically belong to other elements appear in the arithmetic units. The circuits whose components fall into this category will not be considered here. These components are the IO registers, the test registers, and the clock register. An exception is made in the case of the memory buffer registers because these registers serve as connecting links between magnetic core memory and all the elements of the Central Computer System. In this respect, they are also part of the arithmetic element. The arithmetic elements are essentially identical in their physical and functional characteristics. Accordingly, only the left arithmetic element will be treated in the ensuing discussion except where, in the interests of completeness, the logical analysis warrants the inclusion of the right arithmetic element.

As its major function, the arithmetic element implements the four basic arithmetic operations; i.e., addition, subtraction, multiplication, and division. The operands for these calculations are supplied to this element from magnetic core memory by the memory buffer register. The basic operations are executed in accordance with sequenced command pulses furnished by the instruction control unit. See table 3–2 for an alphabetical listing of these commands. The results of these calculations can be held in the arithmetic element for immediate processing, or can be transferred to the memory buffer register for subsequent storage in magnetic core memory. A second function of this element is to accept selected information from magnetic core memory, modify it in a specified manner, and then return the altered information to magnetic core memory. The arithmetic element can also be used as a temporary storage device for information required by the program element for the performance of its functions.

Figure 3–1 is a simplified block digram of the left and right arithmetic elements. It depicts the transfer paths within each element, the paths which link the two elements, and the external connections to the program element and magnetic core memory. The signal paths from the instruction control element are not included, but they affect every register in the two elements. As is shown, only the right element is provided with transfer paths to the program element. These paths are associated with the right accumulator and right A registers, and are the means by which the right element can be utilized as a temporary storage device for program element information. This property will be considered in greater detail in the section concerned with the branch class instructions. The left and right arithmetic elements

TABLE 3–2. ARITHMETIC ELEMENT COMMANDS

UNIT	COMMAND NAME	COMMAND NUMBER
A Registers	*clear left A register*	21
	clear right A register	230
	complement left A register	26
	complement right A register	226
	correct left remainder	11
	correct right remainder	211
	left A register to left memory buffer	23
	left logical multiply	25
	make left A register positive	22
	make right A register positive	229
	right A register to right memory buffer	228
	right logical multiply	225
Accumulator Registers	*clear left accumulator*	10
	clear right accumulator	210
	clear left sign control	8
	clear right sign control	208
	complement left accumulator	19
	complement right accumulator	219
	left accumulator 1 bit to sign bit	4
	left accumulator bits 2 through 15 to 1 through 14	2
	left accumulator 15 bit to left B register sign bit	7
	left accumulator to left B register	1
	left accumulator to left memory buffer	17
	left accumulator sign bit to 15	6
	left accumulator sign bit to Left B register 15	5
	left accumulator sign bit to right accumulator 15	12
	left correct sign	9
	left round off sign	214
	make left accumulator and left B register positive	16
	make left accumulator positive	13
	make right accumulator and right B register positive	20
Accumulator Registers (cont'd)	*make right accumulator positive*	213
	right accumulator 1 to sign bit	204
	right accumulator bits 2 through 15 to 1 through 14	202
	right accumulator 15 to right B register sign bit	207
	right accumulator to address register	212
	right accumulator to right B register	201
	right accumulator to right memory buffer	121
	right accumulator sign bit to 15	206
	right accumulator sign bit to left accumulator 15	217
	right accumulator sign bit to right B register 15	205
	right correct sign	209
	ripple left accumulator right	18
	ripple right accumulator right	123
	right round off sign	216
	test the left accumulator	165
	test right accumulator s bit for one	166
	test right and left accumulator	162
Adders	*complement left divide connect flip-flop*	67
	complement right divide connect flip-flop	267
	left accumulator conditional shift left	60
	left carry one	62
	left carry zero	64
	left end carry	63
	make left A register and left accumulator signs unlike	61
	make right A register and right accumulator signs unlike	261
	right accumulator conditional shift left	260
	right carry one	69
	right carry zero	264

TABLE 3-2. ARITHMETIC ELEMENT COMMANDS (cont'd)

UNIT	COMMAND NAME	COMMAND NUMBER
Adders (cont'd)	*record left overflow*	66
	record right overflow	266
	right end carry	263
	right end carry after add one	14
B Registers	*clear left B register*	84
	clear right B register	284
	left B register bits 1 through 15 to S through 14	82
	left B register bit 9 through 14 to 1 through 15	85
	left B register to left A register	88
	left B register S to left accumulator 15	81
	left B register store to left B register S	89
	left partial product	83
	left round off	87
	right B register bits 1 through 15 to S through 14	282
	right B register bits S through 14 to 1 through 15	285
	right B register to right A register	288
	right B register S to right accumulator 15	281
B Registers (cont'd)	*right B register store to right B register S*	289
	right partial product	283
	right round off	287
Memory Buffer Registers	*clear left memory buffer*	41
	clear right memory buffer	53
	left memory buffer to left A register	43
	left memory buffer to left B register	39
	left memory buffer to left test register	D
	left memory buffer to operation register	42
	left memory buffer to right A register	44
	parity check	55
	parity count	47
	right memory buffer to address register	52
	right memory buffer to right A register	51
	right memory buffer to right B register	40
	right memory buffer to test register	H

are interconnected by two paths, the first path linking the two accumulator registers, and the second path linking the left memory buffer register to the right A register. The path connecting the two accumulator registers enables an exchange of information between registers during certain shift class instructions. The latter path is used for the twin-type instructions *(TAD, TSU, TMU, TDV)*, wherein the operand in the left memory buffer register is to be used in both the left and right elements. (No facilities exist for making a similar transfer from the right memory buffer register.)

The external and interconnecting paths having been considered, the internal connections for the left arithmetic element can now be investigated. This discussion will also apply to the internal connections of the right arithmetic element.

All memory information relating to this element enters or leaves through the memory buffer registers. From this register, the incoming information can be transferred directly by means of 16-bit parallel transfer paths to either the A register or the B register. Outgoing information can be placed directly in the memory buffer register from either the accumulator register or the A register. This memory buffer register contains the parity check circuit which monitors information transfers between itself and memory. An extra bit (parity bit) is required for this operation. This makes the memory buffer register 17 bits in length. All other registers in this element contain 16 bits (sign bit plus 15 magnitude bits). Refer to Chapter 3 of this Part for a detailed discussion of the parity check circuit.

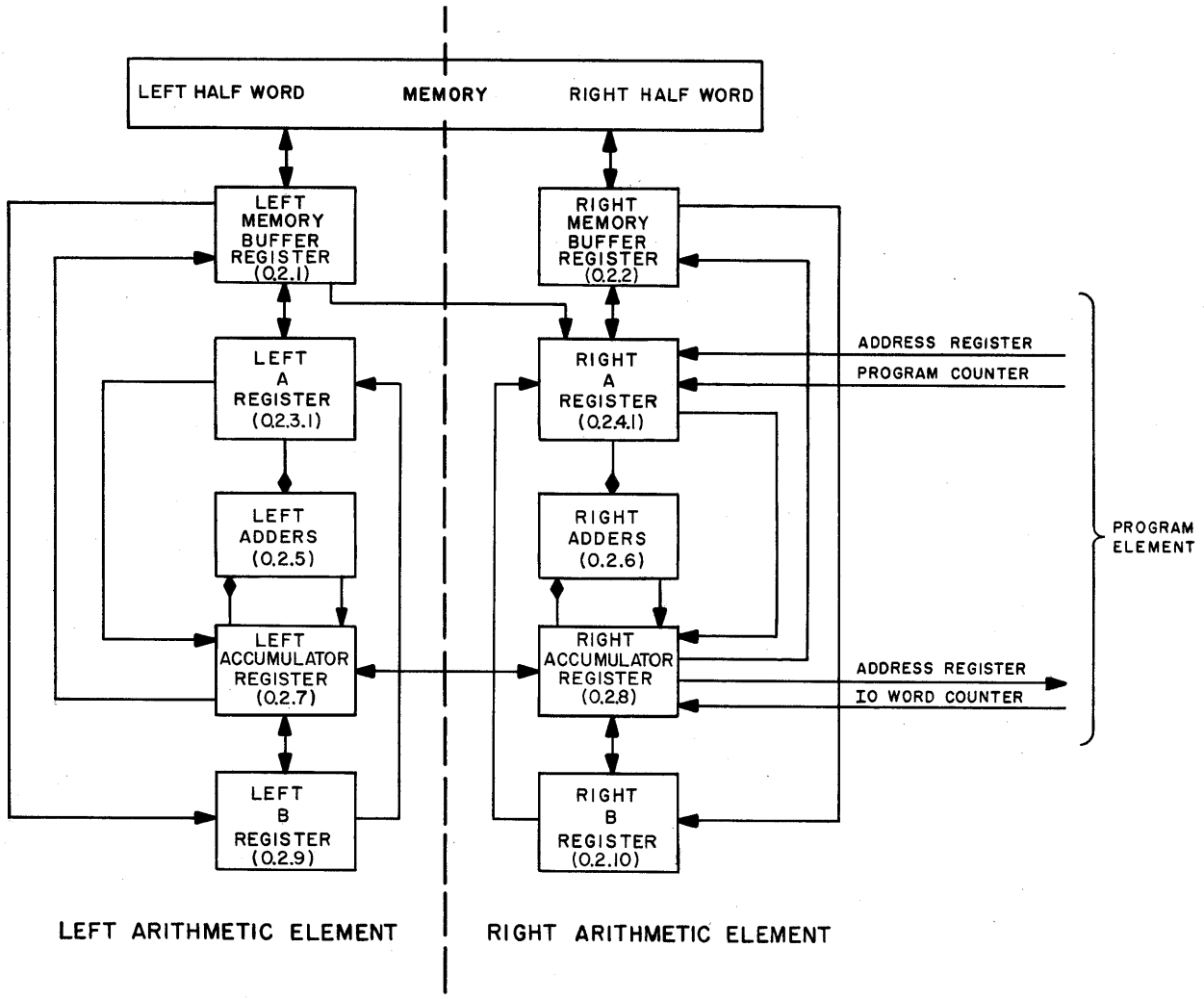


Figure 3–1. Arithmetic Elements, Simplified Block Diagram

In most cases, the information contained in the memory buffer register is transferred to the A register. The latter register holds the operands listed in table 3–3 during the four arithmetic operations.

TABLE 3–3. A REGISTER OPERANDS

ARITHMETIC OPERATION	OPERAND
Addition	Addend
Subtraction	Minuend
Multiplication	Multiplicand
Division	Divisor

During all of the arithmetic operations, the A register forms a functional unit with the adders and the accumulator register. The contents of the A register are reflected in the adder circuits by means of the d-c signal path indicated in figure 3–1. This path consists of 32 individual conditioning lines, two for each bit of the register. The signal path indicated between the A register and the accumulator register (logical multiply) is not used in the execution of the four basic arithmetic operations. However, it is used during the *Extract* and *Deposit* instructions to modify the information held in the accumulator register. The only remaining transfer path affecting the A register is the 16-bit parallel transfer line from the B register.

There are 16 identical adders in the arithmetic element, one adder associated with each pair of corresponding bits in the A register and accumulator register. Since every one of the basic arithmetic operations employs a variation of the addition process, the adders are always involved in combining operands held in the A register and

accumulator register. The answer derived by these adders always appears in the accumulator register but shifted one place to the right. This entails use of the B register sign bit to temporarily store the sum of the right-most bits of the two registers. This inherent shift right has been incorporated to speed the multiplication process:

The accumulator register holds the following information during the four basic arithmetic information during the four basic arithmetic operations. As indicated in table 3–4, the accumulator register provides the adders with one of the operands for each of the basic operations except multiplication. For all operations except division, it also holds the final answer. In the case of multiplication, the 16–bit words, one held in the A register and one in the B register, are multiplied to produce a 32-bit product. To accommodate a word of this size, the accumlator and B registers are formed into a single (32-bit) register. Under these conditions, the sign bit of the B register holds the 16th magnitude bit of the product. The sign of the product is given by the accumulator register sign bit. In the case of division, the dividend is held in this same long register. After the division has been completed, the quotient appears in the B register, the remainder appears in the accumulator register, and the sign of both appears in the accumulator register sign bit position. The sign bit of the B register in this instance contains the most significant bit of the quotient. Facilities are provided for shifting information between the accumulator and B registers so that any desired alignment of the information contained therein can be realized. By utilizing these facilities, the numbers in the two registers after division can be reversed, the quotient now appearing in the accumulator register and the remainder in the B register.

TABLE 3–4. INFORMATION CONTENT OF ACCUMULATOR REGISTER

ARITHMETIC OPERATION	INITIAL CONTENTS	FINAL CONTENTS
Addition	Augend	Sum
Subtraction	Subtrahend	Difference
Multiplication	Clear	Sign plus 15 most significant bits of product
Division	Sign and 15 most significant bits of dividend	Sign of quotient and magnitude of remainder

The B register can be combined with the accumulator register to form the long register described in the previous paragraph, or it can be used as a 16-bit register. This register does not participate in the addition and subtraction processes. During multiplication it initially contains the 16-bit multiplier. As the operation progresses, however, the multiplier is shifted out of the B register one bit at a time, and is eventually lost. As a digit position is vacated, it is filled with the least significant bits of the product. During division, that portion of the dividend originally held in the B register is progressively shifted into the accumulator register. As the digit positions are vacated they are filled with the bits forming the quotient. To facilitate certain instructions, such as *Add B Register to Accumulator Register (ADB)*, a parallel transfer path is provided between the B register and the A register. This path exists because no direct method is provided for adding the B register to the accumulator register. Therefore, the process is one of standard addition in which the operand supplied to the adders by the A register is furnished by the B register instead of by magnetic core memory.

CHAPTER 2

ARITHMETIC PROCESSES

2.1 GENERAL

This section contains a description of the properties of the binary system which are used by AN/FSQ–7 Combat Direction Central in performing the four basic arithmetic operations (add, subtract, multiply, divide). The development is concerned mainly with the binary system, but reference is made to the decimal system whenever it will clarify the point in question. The numbers used for the illustrative examples are all less than unity in order to conform with actual computer operation. These numbers will consist of a single sign bit followed by three magnitude bits. The discussion is nevertheless applicable to the actual case in the computer where individual words consist of a single sign bit followed by 15 magnitude bits. It is assumed that the reader is familiar with the basic rules of binary arithmetic.

Each of the basic arithmetic operations will be described separately with the aid of a logical block diagram. However, before such an analysis can be undertaken, two characteristics of the Central Computer System must be discussed: the inability of the computer to subtract directly, and an inherent shift right of the sum during every addition process. Since multiplication, which employs over and over addition, takes up approximately 10 percent of the total computer time, the machine is designed to speed this operation in preference to all others. The inherent shift right is the chief method employed to decrease the time required for multiplication. This shift must be compensated for in other instructions, but since correction time for the other instructions does not require an additional machine cycle, the result is a net gain in overall computer speed. The second characteristic, the inability of the Central Computer System to perform a subtraction directly, necessitates that some method be employed whereby subtraction will be performed by an addition process. This method is found in the complement systems described in the ensuing paragraphs.

2.2 COMPLEMENT SYSTEMS

AN/FSQ–7 Combat Direction Central employs two complement systems, commonly termed the 1's complement and the 2's complement systems. Each is used for those applications where it will result in the maximum computer speed. Accordingly, the 1's complement system is used for all internal computer operations except the divide operation. The divide operation does employ the 2's complement system, but is so devised that the final quotient, if in complement form at all, will be in the 1's complement form. As will be explained in this discussion, the subjects of end carry and overflow are closely allied to the complement systems.

2.2.1 1's Complement System

The 1's complement of any binary number is obtained by replacing every bit in the number by its opposite. If, by convention, the first digit of the number represents its sign, then complementing a number affords a means of reversing the sign affixed to that number. In the case of the Central Computer System, the sign convention adopted identifies positive numbers by a 0 in the first digit position (sign bit) and negative numbers by a 1 in the sign bit. Therefore, if the given binary number is 0.111 ($+7/8$), the 1's complement of this number is 1.000 ($-7/8$). Note that the magnitude bits of the complement form of the number cannot be used directly in converting from the binary system to the decimal system (or any other system). Both the positive and the complement form of the given number represent a decimal magnitude of $7/8$. The magnitude bits of the positive form of this number can be interpreted in the standard way to yield the correct result:

$$\begin{aligned} 0.111 &= (1 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) \\ &= (1 \times 1/2) + (1 \times 1/4) + (1 \times 1/8) \\ &= 7/8 \end{aligned}$$

If, however, the magnitude bits of the complement form are interpreted in the same way, the result is meaningless, being -0. Accordingly, any binary number that has a 1 in its sign bit (negative) must be complemented before a direct binary-decimal conversion can be performed. While this restores the proper magniture bits, it also prefixes an improper sign to the number. Some means must be devised for recording the proper sign and assigning it to this number once the conversion process has been completed.

2.2.2 2's Complement System

The 2's complement of a binary number is obtained by adding 1 to the 1's complement of that number. As an example, the 2's complement of

0.111 (+7/8) is obtained in the following manner:

0.111	Positive binary number
1.000	1's complement
1.001	2's complement

Therefore, the derivation of the 2's complement of a binary number involves one more step (the addition of 1) than is required for the 1's complement procedure. (The importance of this consideration will become evident when a discussion of the relative advantages of both complement systems is presented.) Except for this consideration, the statements made previously for the 1's complement system apply also to the 2's complement system.

2.3 COMPUTER CHARACTERISTICS

The complement systems find application in the Central Computer System in several different ways. Although positive numbers are read in and out of the computer directly, negative numbers are read into the computer as the 1's complement of their positive counterparts. Internally, the computer uses the 1's complement exclusively except, as indicated previously, during the divide process. The divide process uses the 2's complement because, by so doing, it eliminates the need for end carry operation (to be explained later). The numbers are manipulated within the computer itself according to the logic of the instruction. By utilizing either complementation process, positive numbers are converted to negative and negative numbers to positive, as required. A special problem arises when a negative number must be read out in the form of a magnitude and sign. In order to obtain the correct magnitude, the computer must complement the number to make it positive. While this permits the computer to make a direct conversion, it also prefixes the incorrect sign to the number. To avoid writing out this incorrect sign, the computer stores the negative sign of the number before complementing it. After the complementing and conversion processes are completed, the computer affixes the sign to the proper magnitude. Before evaluating these considerations, the subject of end carry must be examined.

2.3.1 End Carry

Since the computer contains only adder circuits, it must do subtraction by adding the complement of the number to be subtracted. However, the answer will not always be correct if the 1's complement system is used. The correctness of answer is ascertained by seeing if there is a carry out (1 or 0) from the sign bit of the sum. If the carry out is 0, the answer is correct as it stands; if the carry out is 1, the answer is incorrect. The answer can always be corrected by adding a 1 to the lowest order bit of the sum. In AN/FSQ-7 Combat Direction Central, the 1 carried out of the sign bit is used to effect this correction. The procedure is termed end carry. It is a property of the binary number system that a second end carry can never be generated by the first end carry. This process is illustrated in the following example where 1/8 is to be subtracted from 5/8:

	+5/8	0.101	
+	(−1/8)	1.110	Add 1's complement of +1/8
	1	0.011	Carry out of 1 indicates incorrect answer
		1	Add 1 (end carry)
	+1/2	0.100	Correct answer

By using the 2's complement of a number, the answer will automatically be correct whether or not there is a carry out of 1. For this reason, an end carry is never required when using the 2's complement. Using the previous example:

	+5/8	0.101	
+	(−1/8)	1.111	Add 2's complement of +1/8
	+1/2 1	0.100	Answer correct, carry out of 1 meaningless

The circuit used to perform end carry is illustrated in figure 3–2. The carry one line from the sign bit adder is connected to the 1 side of the carry storage flip-flop. When a carry out of 1 occurs, the flip-flop is set to the 1 side, thereby conditioning two gate tubes. Application of the carry 1 pulse to the adder for the lowest order bit is now delayed until gate tube 1 is pulsed. The reason for this delay is twofold: the sum of the original two numbers in the accumulator and A register appears in the accumulator register dis-

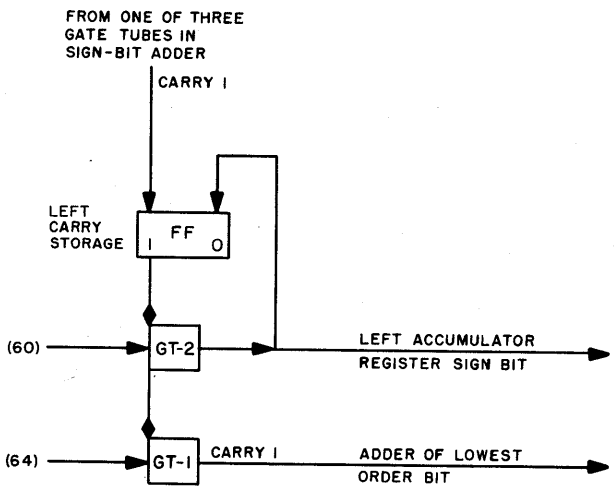


Figure 3–2. End Carry Circuit

placed one bit position to the right, and time must be allotted for shifting back the sum one bit position to the left before a 1 can be added to it; furthermore, the A register must be cleared concurrently so that its contents will not be added in a second time. Thus, gate tube 1 is pulsed by *end carry* command (64) only after the sum is shifted left and the A register is cleared. The new sum, developed by adding a 1 to the previous sum, also experiences a shift right which must be corrected. This is performed by gate tube 2 which is pulsed by the *accumulator conditional shift left* command (60). Command 60 is conditional because it must take effect only if an end carry has been called for and executed. Otherwise, the accumulator would be shifted left unncessarily and the most significant bit of the sum would be lost. Gate tube 2, by paralleling the status of gate tube 1, acts as a switch which permits the conditional shift left to take place only if gate tube 1 has allowed an end carry to take place. It will suppress the shift if gate tube 1, upon being pulsed by the *end carry* command (63), has suppressed the end carry pulse. In addition, the output of gate tube 2 clears the carry storage flip-flop in preparation for the next add process.

From the preceding discussion, it can be seen that both complement systems have their disadvantages. The possibility of end carry in the 1's complement system can be eliminated by using the 2's complement system but an additional step is then required for the complementing process itself. Thus, the 2's complement system is slower when read-out conversions to magnitude and sign are required, but it is faster during internal operations since no end carry control is used. For every mathematical process in AN/FSQ–7 Combat Direction Central except the divide process, the basic memory cycle is considerably longer than the operate time required to execute the instruction. The time necessary to perform an end carry is therefore available without adding to the total execution time of an instruction. In the case of a divide-type instruction, the basic memory cycle is shorter than the operate time required, making the use of the faster 2's complement mandatory. The divide process therefore uses the 2's complement and is so arranged that, at the end of the process, the quotient and remainder, if in complement form at all, are expressed in 1's complement form. The problem of a long read-out time when using the 2's complement system therefore never arises.

2.3.2 Overflow

An overflow condition occurs whenever two numbers, upon being added by the computer, result in a sum equal to or greater than unity, thus exceeding the capacity of the computer. Ordinarily, an overflow condition indicates that an error has been made in the program. However, an overflow may, at the discretion of the programmer, be deliberately introduced into the machine at certain times. In either case, the machine can be sensed for an overflow condition and the program modified accordingly.

When two numbers of opposite signs are added, and each of the numbers is less than unity in absolute value, then their sum must be less than unity in absolute value. Thus an overflow condition cannot occur in such situations but is limited to those cases where the signs of the addend and the augend are alike. Therefore, in every addition process, the signs of the addend and the augend are inspected. If they are unlike, no overflow can occur. If they are alike, an overflow can occur and a further inspection is made of the sum sign bit. If this sign bit is unlike the sign bits of the augend and addend, an overflow condition is indicated. In the case where the augend and addend are both positive (0), the bit 1 adder must send a carry 1 pulse to the sign bit adder for such a condition to exist. In the case where the augend and addend are both negative (1), the bit 1 adder must send a carry 0 pulse to the sign bit adder for an overflow condition to exist. In this latter case, however, the conditions leading to overflow also result in an end carry. It is possible that the addition of a 1 to the lowest order bit because of the end carry (see end carry discussion) will reverse the sum sign bit, making it now agree with the common sign of the augend and addend. In brief, a false overflow indication is possible when two negative numbers are being added, but, in this case, the false indication is always corrected by the end carry operation. Accordingly, in the overflow-detecting circuit, an auxiliary overflow flip-flop is required so that the overflow flip-flop itself is never set before an overflow indication has been verified.

A simplified diagram of the overflow circuit is shown in figure 3–3. The 0 lines from the sign bit flip-flops of the accumulator and A register condition AND circuit 1; the 1 lines condition AND circuit 2. The d-c level output of AND circuit 1 conditions gate tube 1, which is pulsed by the carry 1 line from the bit 1 adder. Gate tube 2, conditioned by AND circuit 2, is pulsed by the carry 0 line from the bit 1 adder. These gate tubes create the effect of three-way AND circuits. An output pulse from gate tube 1 indicates that the signs of the augend and addend are both positive

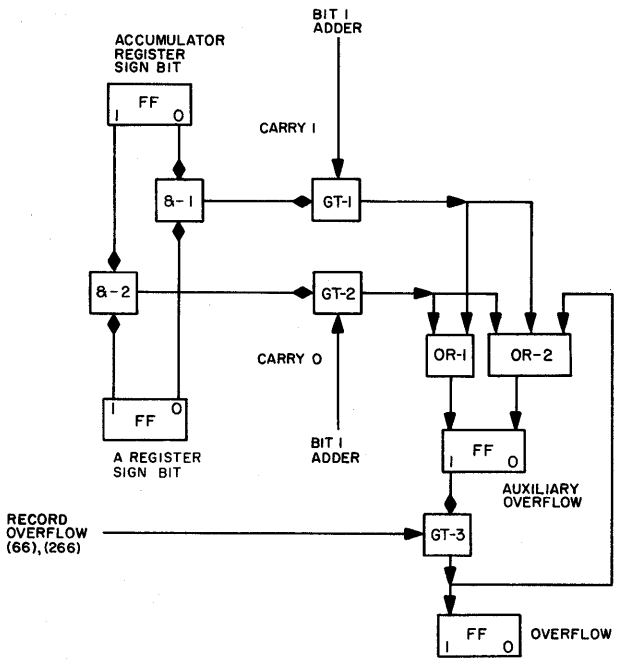


Figure 3–3. Overflow Circuit

and a 1 (carry 1 from bit 1 adder) has been added to their sum. An output pulse from gate tube 2 indicates that the signs of the augend and addend are both negative and a 0 (carry 0 from bit 1 adder) has been added. This combination of gate tubes and AND circuits thus covers all the possible conditions that might cause an overflow condition, which are listed in table 3–5.

The operation of the overflow circuit will be illustrated using the following examples:

(a)	−3/8		1.100	Augend
	−5/8		1.010	Addend
	−8/8	1	0.110	Uncorrected sum (carry out of 1)
			0.001	End carry
			0.111	Corrected sum
(b)	−3/8		1.100	Augend
	−4/8		1.011	Addend
	−7/8	1	0.111	Uncorrected sum (carry out of 1)
			0.001	End carry
			1.000	Corrected sum

In example (a), both addend and augend are negative and there is a carry of 0 from the bit 1 sum to the sign bit. Accordingly, gate tube 2, conditioned by AND circuit 2, complements the auxiliary overflow flip-flop by pulsing OR circuit 1 and OR circuit 2. This flip-flop is always in the clear status at the start of this operation, so that the complement pulse always sets it to the 1 side. The auxiliary overflow flip-flop conditions gate tube 3, but this gate tube does not transmit a set pulse to the overflow flip-flop until it is sensed by a command pulse (266) some time later. This delay is required to allow an end carry to be performed if there is a carryout of 1 from the sum sign bit. In example (a), there is an end carry and the signs of the new augend (incorrect sum) and the new addend (end carry) are alike (both positive) while there is a 0 carry from the bit 1 adder to the sign bit adder. Although AND circuit 1 and hence gate tube 1 are conditioned, the auxiliary overflow flip-flop is not complemented because the carry 1 line is not pulsed, leaving it in the 1 status. Accordingly, the overflow flip-flop is set when gate tube 3 is sensed by the *record overflow* command (66).

Example (b) illustrates a case where an overflow condition is at first indicated but does not really exist. This example is similar to example (a) except that the end carry operation causes the corrected sum sign to be opposite to that of the uncorrected sum. At this point, the following conditions exist: the new augend (uncorrected sum) and the new addend (end carry) are both positive (0) while there is a carry of 1 from the bit 1 adder

TABLE 3–5. CONDITIONS FOR OVERFLOW

SIGN OF AUGEND	SIGN OF ADDEND	CARRY FROM BIT 1 ADDER	SUM SIGN	END CARRY	CORRECT SUM SIGN	OVERFLOW
0	0	0	0	No	—	No
0	0	1	1	No	—	Yes
*1	1	0	0	Yes	1	No
1	1	0	0	Yes	0	Yes
1	1	1	1	Yes	1	No

**False overflow indication.*

to the sign bit adder. Thus, gate tube 1 complements the auxiliary overflow flip-flop a second time, clearing it to the 0 side. Consequently, gate tube 3 is deconditioned, and the overflow flip-flop is not set when the *record overflow* (66) command occurs.

2.4 SYNCHRONOUS ADD AND SHIFT

The fundamental process used for arithmetic operations by AN/FSQ-7 Combat Direction Central is the synchronous add and shift process. Every arithmetic operation (add, subtract, multiply, and divide) employs some modification of this system. One of the distinguishing features of this method is an inherent shift right of the sum bits while an add operation is in progress. This shift appreciably reduces the time required by the computer to perform a multiplication operation. The addition process, however, requires a shift to the left in order to nullify the inherent shift right. For all instructions other than multiply, the execution time required is not lengthened by this consideration because sufficient extra time is available during the normal instruction cycle to effect this correction. This will become evident when, later in the discussion, the arithmetic operations are treated in detail.

The basic elements employed in a synchronous add and shift process are functionally connected as shown in figure 3-4. As illustrated in the figure, two complete registers (accumulator and A registers) and the sign bit of a third (B register) are required for this process. For purposes of simplification, complete registers (16 bits in the actual computer) are depicted as three bit registers. The X bit is the prototype for all bits between the sign bit and the final bit of a register.

The augend is held in the accumulator and the addend is held in the A register. The result of an addition in the binary system is a sum bit (0 or 1) and a carry bit (0 or 1). The adders, one for each pair of corresponding bits of the accumulator and A register, act as three-way AND circuits, combining the information from these bits with the carry information from the previous bit. Essentially, all of the adders function identically and a typical adder is shown in figure 3-5. The two bits being added condition one of the four AND circuits shown on the figure. These four AND circuits are sufficient to include all the possible combinations of digits that might have to be added. Actually, only three distinct possibilities exist since the sum of a 0 and a 1 is the same as

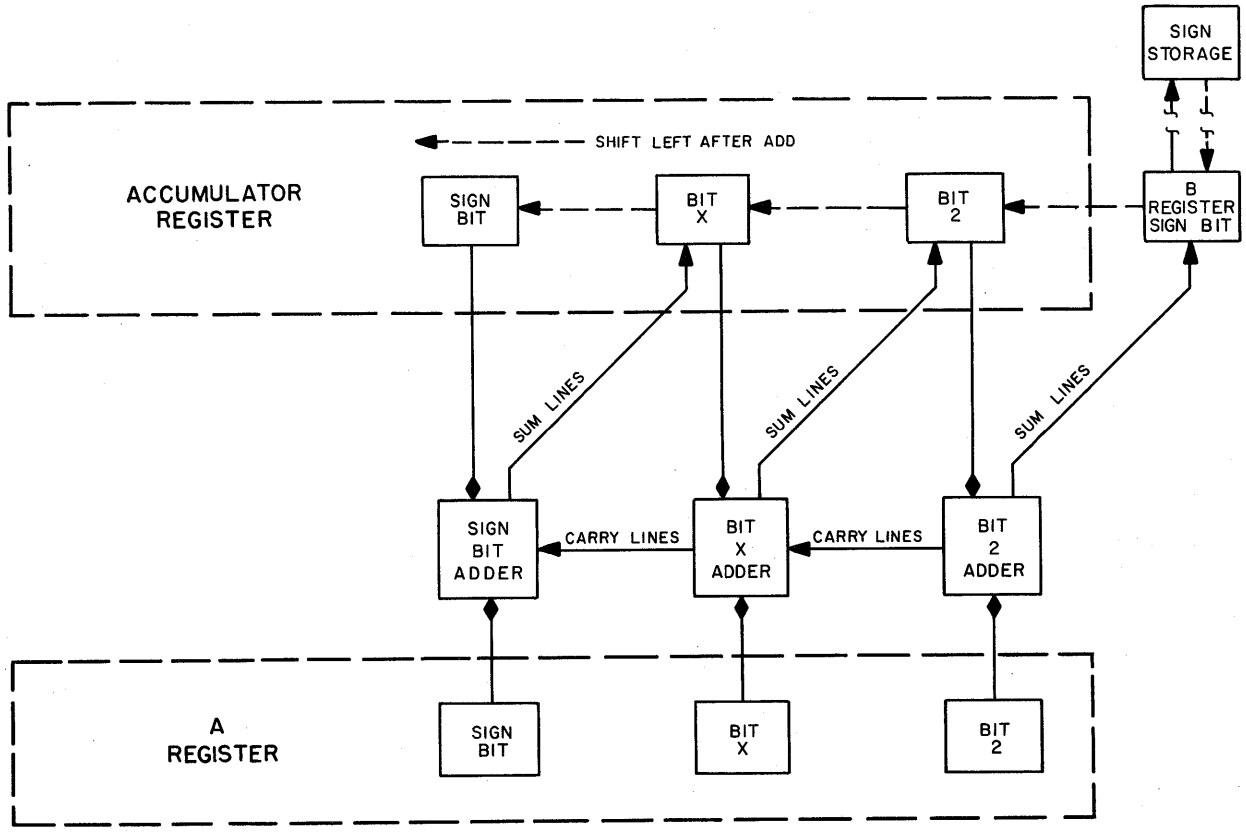


Figure 3-4. Inherent Shift Right

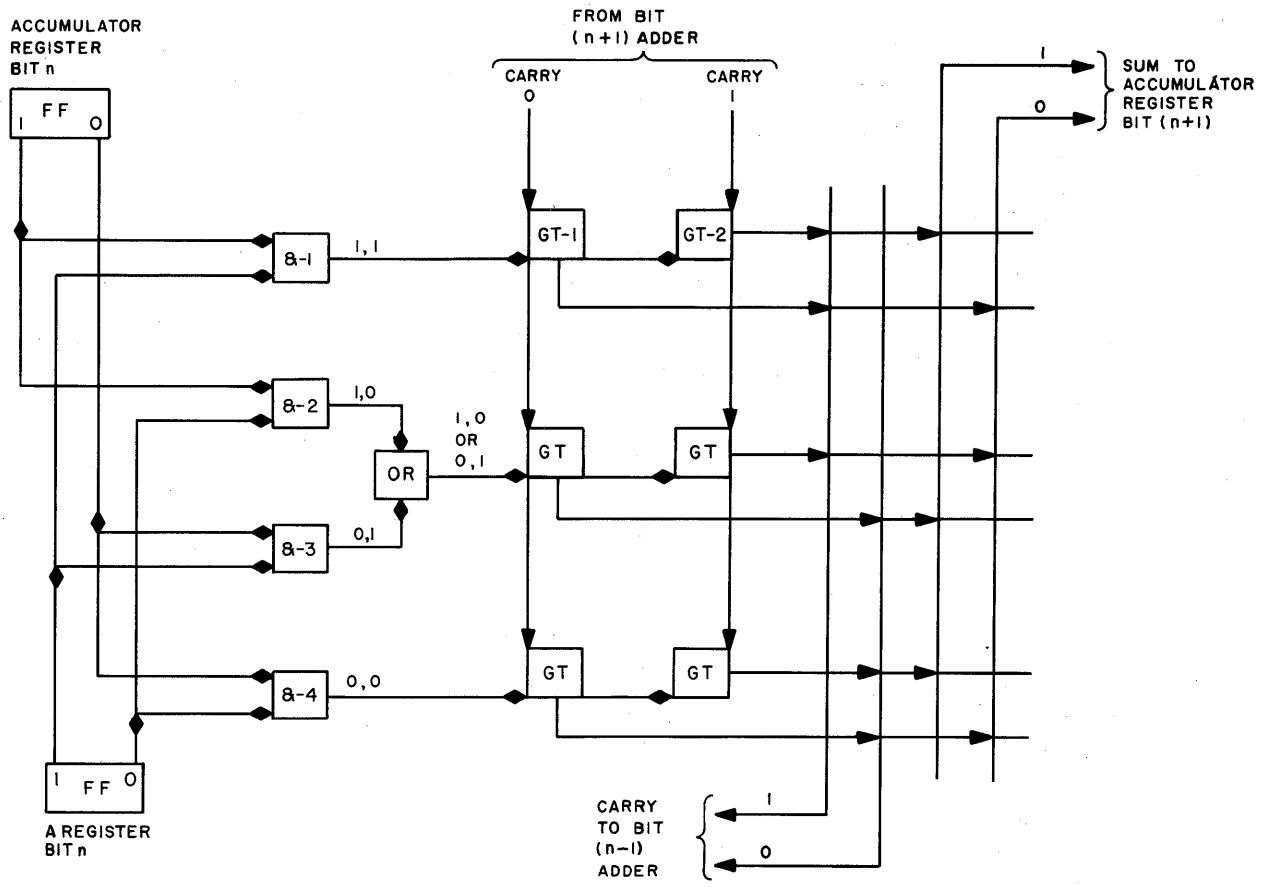


Figure 3–5. Adder Circuit

that of a 1 and a 0. The outputs of the AND circuits for these two combinations are combined in an OR circuit to produce a single output. Thus, three output lines are sufficient to indicate the possible combinations that can exist (1, 1; 1, 0 or 0, 1; 0, 0). A pair of gate tubes is associated with each of the three output lines but only one of the three pairs will be conditioned depending on the pair of digits contained in the register bits. One gate tube of each pair is connected to the carry 0 line, while the other gate tube is connected to the carry 1 line. These carry lines are the outputs of the adder circuit associated with the previous bit (next lowest order bit) and relay the carry resulting from addition in that bit.

The output of the gate tube that has been conditioned and pulsed is distributed as shown by the grid in figure 3–5. This grid is a functional representation of the actual circuits employed in the distribution and is not meant to represent electrical conductors. Two pairs of output lines form the vertical pattern of the grid. The six output lines from the gate tubes intersect the vertical pattern and complete the grid. An output can appear only at those intersections at which a terminating arrow is shown. Since only one gate tube is active at a given time, only two points in the grid are energized at one time. To illustrate the operation of the adder up to this point, assume that each of the two register bits contains a 1. Accordingly, AND circuit 1 conditions gate tubes 1 and 2. Assuming, further, that there is a carry of 1 from the previous adder (n + 1), gate tube 1, as shown by the grid, will provide a carry 1 pulse to the next adder (n — 1). It will also transmit a sum-is-1 pulse to the accumulator bit associated with the previous adder in accordance with the inherent shift right designed into the addition process.

The inherent shift right is accomplished by transferring the sum bit developed by each adder one position to the right when it appears in the accumulator. This shift necessitates the use of an extra flip-flop to accommodate the sum bit developed by that adder which operates on the lowest order bits of the accumulator and A registers. The extra flip-flop used for this storage is the sign bit flip-flop of the B register. To preserve the information originally held in the B register sign bit, the contents of this bit are temporarily stored in the sign storage flip-flop. This occurs before the

transfer of sum information from the bit 2 adder to the B register sign bit. This completes the add phase of the synchronous add and shift process. Since, in the add process, each bit of the sum has been shifted one bit position to the right, the bits must now all be shifted one bit position to the left in order that the correct answer appear in the accumulator. A broadside shift left process is used to perform this function. Once the sum is shifted to the left, the B register sign bit flip-flop no longer holds the final bit of the sum, and regains its original contents from the sign storage flip-flop.

2.5 ADDITION AND SUBTRACTION

The circuits required by the Central Computer System for adding or subtracting two numbers are shown in block form in figure 3–6. (Note that this figure is an elaboration of figure 3–4 used to illustrate the inherent shift right.) Four add-type instructions, *Add (ADD)*, *Clear and Add (CAD)*, *Twin and Add (TAD)*, and *Clear and Add Magnitude (CAM)* use the process to be described. This same process, with slight modification, is also used for the following subtract-type instructions: *Subtract (SUB)*, *Clear and Subtract (CSU)*, *Twin and Subtract (TSU)*, and *Difference Magnitudes (DIM)*. From a mathematical point of view, there is no difference between the subtraction of a positive number and the addition of the corresponding negative number. This offers a very simple method with which the Central Computer System can perform subtraction. When a subtraction is called for, the subtrahend (held in the A register) is complemented (made negative) and then an ordinary add operation is executed.

An ordinary add process is always initiated by pulsing the carry 0 line of the right-most adder (bit 2 adder in figure 3–6) with a command pulse generated in the instruction control element. The sum developed by this adder is placed in the sign bit flip-flop of the B register and the proper carry information is sent to the next adder. This process ripples through all 16 adders. When the two sign bits are added at the end of this process, however, the possibility of end carry and/or overflow must be considered. For this purpose, both an end carry circuit and an overflow circuit are connected to the sign bit adder. The end carry operates only if the addition (or subtraction) of the two numbers has resulted in a carry out of 1 from the sign bit adder. The function of the end carry circuit is to

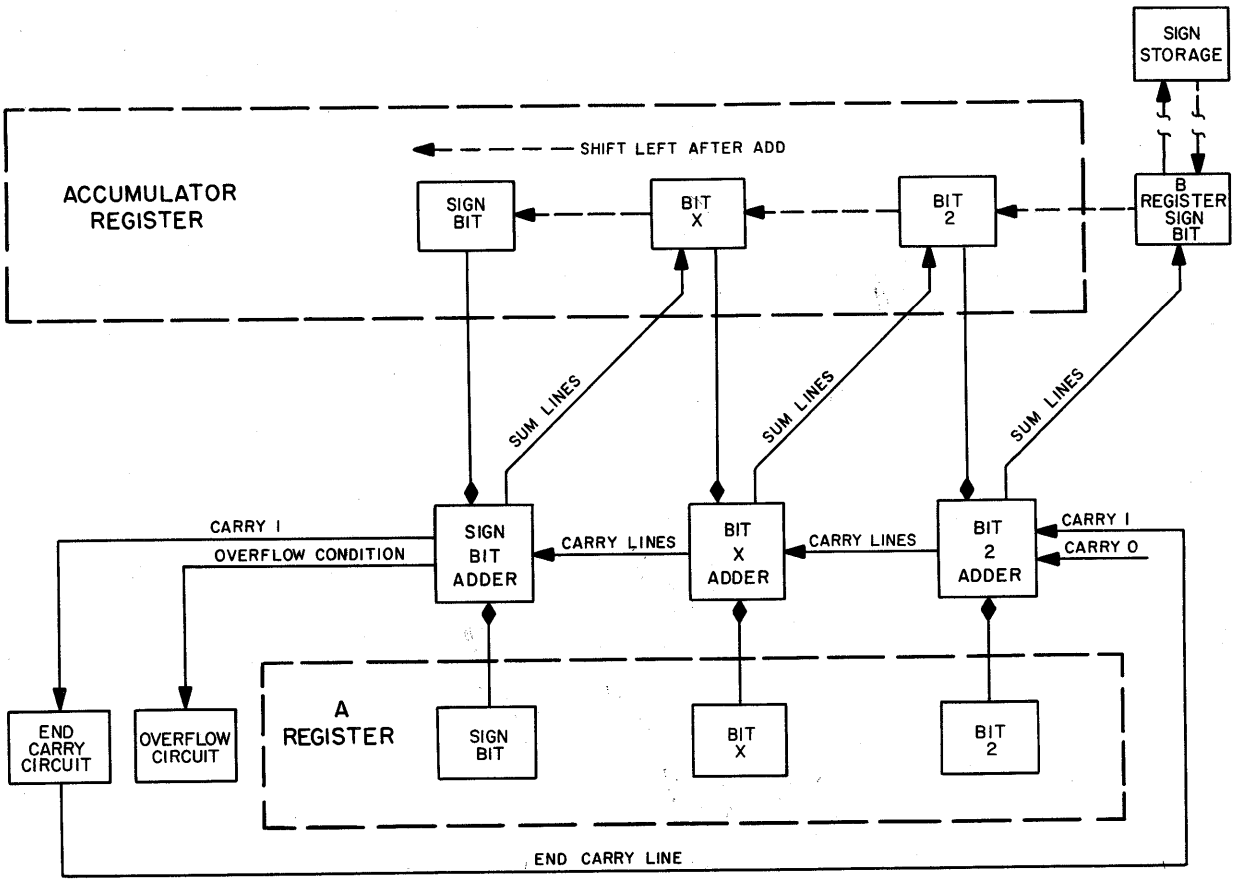


Figure 3–6. Synchronous Add and Shift Process

transmit a carry 1 pulse back to the lowest order bit adder after and only after the shift left (to correct the inherent shift right) has been accomplished. In effect, a new addition is performed in which a 1 is added to the sum previously obtained. The overflow circuit detects the presence of a sum equal to or greater than unity.

From the considerations thus far, a sequence of commands for the add process is arranged in table 3–6:

TABLE 3–6. COMMAND SEQUENCE FOR ADD PROCESS

COMMAND	FUNCTION
carry zero	To initiate the addition
shift left	To correctly align the sum in the accumulator after adding
clear A register	To prevent adding the number in the A register to the sum a second time if end carry is to be performed
end carry	To sense for a carry out of one and correct the sum accordingly
conditional shift left	To realign the sum in the accumulator if there has been an end carry. (This command is generated but not executed if there has been no end carry)

The sequence of commands given for the add process is suitable for subtraction operations except that the A register must be complemented before the *carry zero* command is given. Except for this modification, the process required to execute any of the four subtraction instructions is identical to that of the four addition instructions just described.

The variations among the four add-type instructions are specified below. Assuming that at the beginning of the instruction, the left and right A registers contain the numbers X_L and X_R, respectively, and the left and right accumulators contain the numbers Y_L and Y_R, the end results in the accumulators is given in table 3–7.

Under the same initial conditions, the variations among the four subtract-type instructions are given in table 3–8.

TABLE 3–7. ACCUMULATOR CONTENT DUE TO ADD PROCESS

INSTRUCTION	LEFT ACCUMULATOR	RIGHT ACCUMULATOR
Clear and Add	X_L	X_R
Add	X_L+Y_L	X_R+Y_R
Twin and Add	X_L+Y_L	X_L+Y_R
Clear and Add Magnitude	X_L	X_R

TABLE 3–8. ACCUMULATOR CONTENT DUE TO SUBTRACT PROCESS

INSTRUCTION	LEFT ACCUMULATOR	RIGHT ACCUMULATOR
Clear and Subtract	$-X_L$	$-X_R$
Subtract	Y_L-X_L	Y_R-X_R
Twin and Subtract	Y_L-X_L	Y_R-X_L
Difference Magnitudes	X_L-X_L	Y_R-X_R

2.6 MULTIPLICATION

The circuits required by the Central Computer System for multiplying are those shown in figure 3–6 with the exception that the entire B register is employed instead of the sign bit alone. The procedure to be outlined applies to the two multiply instructions, *Multiply (MUL)* and *Twin and Multiply (TMU)*.

When two binary numbers are multiplied, the product is formed by over and over addition of the multiplicand. As in every multiplication process, each partial product must be shifted relative to the previous partial product before being added to it. The problem of obtaining partial products in the binary system is comparatively simple because only two possibilities exist; if the multiplier bit is a 1, the partial product is identical to the multiplicand, and if the multiplier bit is a 0, the partial product is also 0. As an example, consider the product of 0.010 and 0.101:

```
  0.0 1 0      (8/32)
  0.1 0 1      (20/32)
  0 0 1 0
0 0 0 0
0 0 1 0
0 0 0 0
0.0 0 1 0 1 0  (5/32)
```

Note in the above sample that a shift is required for each partial product even if the mul-

tiplier bit is a 0. The inherent shift right speeds this process in the following way. The first partial product will appear in the combined accumulator-B register already shifted one bit position to the right. The next partial product can therefore be added immediately to the previous partial product because the two are already properly aligned.

The general rule for multiplication as used in AN/FSQ–7 Combat Direction Central is therefore as follows: if the multiplier bit is a 1, the multiplicand (held in the A register) is added into the accumulator but shifted one position to the right; if the multiplier bit is a 0, nothing more is added to the accumulator, but the accumulator is shifted one position to the right. If the right-most bit of the multiplier (bit 15 in the B register) is inspected first, the 15 *partial product* commands are generated, the net result is that the partial product associated with bit 15 will be shifted a total of 15 positions. The process is similar for other bits until finally on the 15th *partial product* command, the partial product associated with bit 1 of the multiplier is shifted one place to the right. As each bit of the multiplier is examined, it is of no further use and can be discarded. Therefore, the register holding the multiplier is shifted to the right each time its lowest order bit has been examined, thus vacating a bit at its left end and destroying the bit which has just been examined. The position vacated is then filled by the least significant bit of the accumulator (partial product) which is shifted one place to the right every time a *partial product* command is executed. The B register originally holds the multiplier and eventually the 15 least significant bits of the final product. The final product will consist of 30 bits plus a sign held in the combined accumulator-B register. The 15 least significant bits are found in positions S to 14 of the B register. The bit in position 15 of the B register (originally the sign of the multiplier) which will always be 0 for positive products and 1 for negative products, has no significance. The internal process just described for the computer has been restricted to positive numbers. This is accomplished by making the multiplicand and the multiplier positive and adjusting the final product in accordance with the original signs of the multiplicand and multiplier. A record of the original signs of the multiplier and multiplicand is kept by the sign control flip-flop. This flip-flop is complemented if the A register is changed from negative to positive and is complemented again if the accumulator is complemented from negative to positive. The multiplier is placed in the accumulator at the beginning of the multiply instructions. Hence, it is the accumulator which must be complemented (that is, made positive) before its contents are transferred to the B register. The accumulator is then cleared before any actual multiplication begins. At the end of a multiplication process, the sign flip-flop is examined. If it contains a 1 (either one of the two operands negative), a negative sign is affixed to the final product; if it contains a 0 (both or neither of the operands negative), a positive sign is affixed to the final product.

Since the multiplier is 15 bits in length, it is necessary to count the number of *partial product* commands and stop the process after 15 steps. This is accomplished by setting the step counter to 15 at the beginning of the operation and stopping operation when the step counter reaches 0. A sequence of commands can then be written for multiply as given in table 3–9.

TABLE 3–9. COMMAND SEQUENCE FOR MULTIPLY PROCESS

COMMAND	FUNCTION
make A register positive	To make multiplicand positive
make accumulator register positive	To make multiplier positive
clear B register	In the preparation for accepting multiplier
accumulator register to B register	To place multiplier in B register
partial product	To perform one of 15 partial product processes (step counter reduced by 1)
Repeat last operation 14 times (step counter = 0)	
correct sign	To affix proper sign to final product

The variations between the two instructions using the multiply process just described may be explained as follows: Assuming at the beginning of the instruction that the left and right A registers contain numbers X_L and X_R and the left and right accumulator registers contain the numbers Y_L and Y_R, the end results in the combined accumulator-B registers are given in table 3–10.

TABLE 3–10. ACCUMULATOR-B REGISTER CONTENT DUE TO MULTIPLY PROCESS

INSTRUCTION	L ACCUMULATOR-B REGISTER	R ACCUMULATOR-B REGISTER
Multiply (MUL)	$X_L \times Y_L$	$Y_R \times Y_R$
Twin and Multiply (TMU)	$X_L \times Y_L$	$X_L \times Y_R$

2.7 DIVISION

Division is distinct from the three other common arithmetic processes in that it contains an element of trial and error. The basic divide process in this computer consists of a series of attempts to subtract the divisor first from the dividend and, in subsequent steps, from the current remainder. If a subtraction results in a positive balance, a quotient bit of one is recorded. If the balance is negative, a quotient bit of zero is recorded. In this case, the dividend (or current remainder) must be restored to its original value in preparation for a second trial subtraction. This restoring process is time consuming and is effectively eliminated by a non-restoring process used in AN/FSQ–7 Combat Direction Central computer.

To illustrate the non-restoring divide process, consider the example where binary number Y is to be divided by binary number X. Both numbers are always made positive as a preparatory step to the actual division, and the final sign to be affixed to the quotient is stored in the sign control flip-flop. Since the Central Computer System cannot utilize numbers whose absolute values are equal to or greater than unity, the condition is imposed that divisor X be larger than dividend Y in order that the magnitude of the quotient be smaller than unity. Since Y must be smaller than X and both numbers are positive, the computer does not attempt to subtract X from Y because this subtraction will always result in a negative balance. Therefore, the computer automatically writes a 0 in the sign bit of the quotient at the start of every division process. As the next step, the dividend is doubled (shifted left one bit position) in an attempt to make it larger than the divisor. At this point, a trial subtraction is made. For this operation, the 2's complement of the divisor is added to the dividend. If this subtraction results in a positive remainder, then the magnitude of X was smaller than the magnitude of the doubled dividend (2Y) and a 1 is properly written as the first magnitude bit of the quotient. If this subtraction results in a negative remainder, then the magnitude of X was greater than the magnitude of 2Y and a 0 is written as the first magnitude bit of the quotient. The effect of subtracting the divisor must be nullified before the dividend can be redoubled in a second attempt to make it larger than the divisor. Both the restoring and non-restoring processes for accomplishing this are illustrated in table 3–11.

TABLE 3–11. TRIAL SUBTRACTION IN THE DIVISION PROCESS

RESTORING PROCESS		NON-RESTORING PROCESS
0.0 1 X/2 Y	(Assume X > 2Y)	0.0 1 X/2 Y
−X 2 Y−X	Current Remainder 1	−X 2 Y−X
+X 2 Y		
2 (2 Y)	(Assume 4Y > X)	2 (2 Y−X)
−X 4Y−X	Current Remainder 2	+X 4 Y−X

Note that by using the non-restoring process to nullify the first subtraction of the divisor from the dividend, one operation is saved. If the restoring process were used, the computer would always have to add the divisor to the current remainder if this remainder were negative. In this way, the original dividend (2Y) would again be available and a second trial subtraction could be made after redoubling. In the non-restoring process, however, the divisor is not added to a negative current remainder, but this remainder is itself doubled. By examining the sign of the current remainder and making the sign of the divisor its opposite for use in the second trial subtraction, the same result is achieved. If the example had been continued, the next step in both processes (current remainder 2) would be to subtract X from 4Y—X. Thus, the non-restoring process is time saving only in those cases where a current remainder is negative.

In the computer itself, the disposition of the numbers involved in division are indicated in table 3–12.

TABLE 3–12. ARITHMETIC REGISTER CONTENT DURING DIVISION PROCESS

QUANTITY	LOCATION
Divisor	A Register
Dividend	Combined Accumulator-B Register
Quotient	B Register
Remainder	Accumulator

The successive quotient bits are determined in their turn and sent to the B register 15 position. (This routing of the quotient bits is accomplished by means of the divide connect flip-flop.)

The next time the accumulator-B register is shifted to the left (thus doubling its value), the previously determined quotient bit is moved to the B register 14 position, thus making room for the next quotient bit in B register 15 position. This process continues until 16 quotient bits are obtained. These will be contained in B register bits S through 15. Consider the last quotient bit; if it is a 1, a successful subtraction is indicated and the accumulator contains the correct positive remainder. However, if the last quotient bit is a 0, the last trial subtraction resulted in a negative remainder which has been overdrawn by the amount of the divisor. This may be corrected by adding the positive form of the divisor to the remainder, thus nullifying the effect of the trial subtraction.

A sequence of commands appropriate for the division process can be written as indicated in table 3–13.

Note that the *partial quotient* command is repeated 16 times. However, at the beginning of a *Divide* instruction, the step counter is set to 17. The reason for this is given in 3.5 of Part 2. The end result is that the step counter actually contains a 1 and not a 0 as indicated when command 80 is discontinued. In a logical sense, however, the effect on the computer is the same as if the step counter had gone to zero in some other instruction involving this counter.

The variations between the two instructions using the divide process just described may be explained as follows. Assuming that at the beginning of the instruction the left and right A registers contain numbers X_L and X_R, respectively, and the left and right accumulator registers contain the numbers Y_L and Y_R, the end results in the combined accumulator-B registers is given in table 3–14.

TABLE 3–13. COMMAND SEQUENCE FOR DIVISION PROCESS

COMMAND	FUNCTION
complement divide-connect flip-flop	To connect output of sign bit adder to input of B register bit 15
make accumulator-B register positive	To make dividend positive
make A register positive	To make divisor positive
make accumulator-B register and A register signs unlike	To comply with non-restoring method of division
partial quotient	To perform shift left of dividend, to perform trial subtraction, and to determine quotient bit
Repeat last step 15 times (step counter = 0)	
make accumulator register and A register signs unlike	To prepare for correction of remainder
complement divide - connect flip-flop	To disconnect output of sign bit adder from input of B register bit 15
correct remainder	To nullify effect of last trial subtraction
conditional shift left	To correct inherent shift right
correct sign	To affix proper sign to final quotient

TABLE 3–14. DIFFERENCE BETWEEN *DIVIDE* AND *TWIN AND DIVIDE* INSTRUCTIONS

INSTRUCTION	LEFT ACCUMULATOR	LEFT B REGISTER	RIGHT ACCUMULATOR	RIGHT B REGISTER
Divide	Remainder	Y_L/X_L	Remainder	Y_R/X_R
Twin and Divide	Remainder	Y_L/X_L	Remainder	Y_R/X_L

CHAPTER 3

BASIC OPERATIONS

3.1 INTRODUCTION

AN/FSQ–7 Combat Direction Central executes an instruction by combining a series of basic operations in a predetermined order. Selection and sequencing of the proper basic operations for a particular instruction are controlled by commands generated in the instruction control element. There are six basic operations utilized in the arithmetic element: clear, complement, transfer, parity, shift, and sense.

3.2 CLEAR OPERATION

A clear operation will always switch the flip-flop on which it is operating to the 0 side. This is accomplished by pulsing the 0 side of the flip-flop. (See fig. 3–7.) If an entire register is cleared, a common line is used to place all the flip-flops comprising that register to the 0 side. Thus, the information in a flip-flop (or a register) is destroyed by a clear operation.

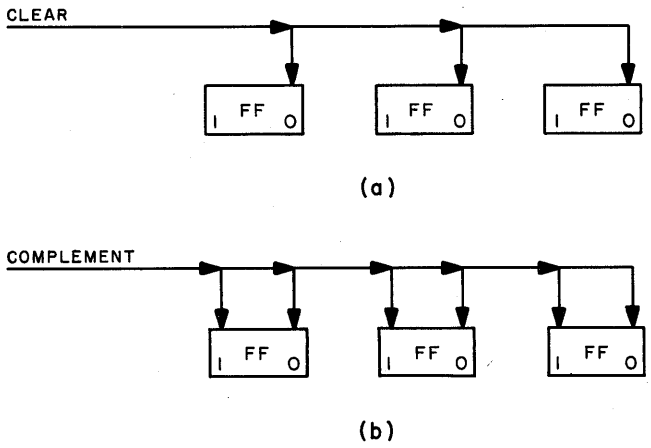


Figure 3–7. Clear and Complement Operations

3.3 COMPLEMENT OPERATION

A complement operation reverses the status of the flip-flop on which it is operating. To accomplish this reversal (from 0 to 1 or vice versa) both sides of the flip-flop are pulsed simultaneously. As in the clear operation, a common input line is used for a register, but in complementing, both sides of each flip-flop in the register are pulsed by this common line. (See fig. 3–7.) Justification for this procedure is found in the following explanation. The conducting half of the flip-flop is not affected by the positive pulse because the control grid is already well above cutoff. However, the grid of the non-conducting tube will be driven positive, forcing this tube to conduct and thereby triggering the flip-flop process.

3.4 TRANSFER OPERATION

A transfer operation involves duplicating the information contained in one register in a second register. The second register must always be cleared before a transfer is performed. The circuit used to effect a transfer operation is illustrated in figure 3–8. The information to be transferred is contained in the upper register (source); the problem is to transfer information to the lower register (destination) which has been previously cleared. A gate tube is associated with the 1 side of each flip-flop in the source register. If this flip-flop contains a 1, it will condition its gate tube. To effect the transfer, a pulse is applied to the transfer line which inspects all these gate tubes simultaneously. Only those gate tubes which have been conditioned by the source flip-flops will transmit a pulse to their corresponding flip-flops in the destination register, thereby changing their status from a 0 to a 1. Thus, each conditioned gate tube permits a 1 in the source register to be duplicated in the destination register. In the case of a 0 in a source register flip-flop, the destination flip-flop is not affected, having already been cleared to 0.

3.5 PARITY OPERATION

A parity operation insures that a transfer of information between two registers is accomplished without changing the information. This operation is performed in three steps. The first step (parity count) determines whether the number of 1's in a 32-bit word is odd or even. If the number of 1's is even, a 1 is set in the 33rd flip-flop (parity bit)

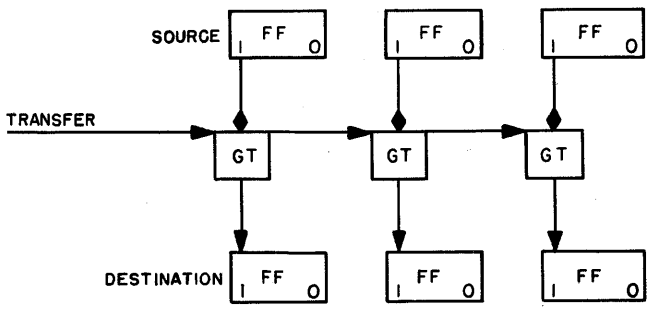


Figure 3–8. Transfer Operation

If the number is odd, the parity bit flip-flop is left at 0. This arrangement will always make the sum of 1's in the 33-bit register an odd number. Once this count has been performed, the 33-bit word is transferred to its memory destination. When, at a later time, this word is read back into the memory buffer register, the parity bit is also transferred. The second step is another parity count which again determines whether the sum of 1's in the 32-bit word is odd or even. This result is utilized by an end circuit consisting essentially of the parity bit and parity count flip-flops. The end circuit is sensed at step three by a parity check pulse and gives an alarm if the two parity counts are dissimilar. It should be noted that this parity operation will not detect errors if an even number of mistakes is made during transfers.

The Central Computer System contains only one register which employs a parity circuit. This is the memory buffer register which must be used every time information is to be transferred into or out of core memory. The circuits involved in a parity operation are illustrated in figure 3–9. It is assumed in this discussion that the first step (parity count of the memory buffer register) has been completed and the 33-bit word has been stored in magnetic core memory. Upon re-reading the word from magnetic core memory at some later time, the following status exists in the memory buffer register and its associated end circuit: the parity count flip-flop is at 0 (this flip-flop is always restored to 0 after a successful transfer); the status of the parity bit flip-flop corresponds to the parity bit for this word in magnetic core memory; and the memory buffer register contains the word itself. This configuration exists only if a correct transfer has been effected. Only four possibilities exist when executing a transfer. (Refer to table 3–15.)

TABLE 3–15. PARITY CHECK RELIABILITY

PARITY BIT	TRANSFER
1	Correct
1	Incorrect
0	Correct
0	Incorrect

For purposes of illustration, assume that the registers involved each contains four flip-flops, three for magnitude bits and one for parity bit. Further, assume that the number 101 has been stored in memory with a 1 in its memory parity bit flip-flop. In a successful transfer, 101 appears in the memory buffer register and the 1 in the memory parity bit flip-flop sets both the parity count and parity bit flip-flops in the buffer end circuit to 1. Now a parity count is initiated by applying a command pulse (47) to gate tubes 1 and 2 simultaneously. The information about the current sum of 1's will be propagated through the register by two lines, the odd line and the even line. Since the flip-flop associated with gate tubes 1 and 2 contains a 1, gate tube 2 is conditioned and the parity count pulse is passed to the odd line. This pulse now simultaneously inspects gate tubes 3 and 4 associated with the center flip-flop. In this example, the flip-flop contains a 0, and the number of 1's, including the contents of this flip-flop, is still odd. This fact is recognized by the circuit by conditioning gate tube 3 which, as shown in the figure, passes the pulse out to the odd line again. The final flip-flop contains a 1, conditioning gate tube 8. The sum of 1's is now even and gate tube 8 therefore passes the pulse to the even line. (Note that if the sum of 1's had been odd at this point, the pulse would not have been propagated any further.) The pulse on the even line is fed to gate tubes 9 and 10 associated with the parity count flip-flop in the end circuit. The purpose of the end circuit, now that the count has been completed, is to determine if an alarm should be given. In this case, since the transfer was correct, no alarm should be given.

The end circuit operates in the following manner. The parity count flip-flop, having been set originally to one by the transfer, conditions gate tubes 10 and 11. As shown in the figure, gate tube 11, under these conditions, would give an alarm as soon as the parity check pulse is applied. To prevent such an indication, gate tube 11 must be de-conditioned before the occurrence of the parity check pulse (55). This is accomplished by passing the even line pulse (resulting from the parity count) through gate tube 10 to complement the parity count flip-flop, thus de-conditioning gate tubes 10 and 11. If, on the other hand, an incorrect transfer involving an odd number of errors had been made, no pulse would appear on the even line feeding the end circuit (the count now resulting in the sum of 1's being odd). Accordingly, the parity count flip-flop would not be cleared, gate tube 11 would remain conditioned, and the parity check pulse would pass through this tube, indicating an alarm.

There are two cases that remain to be considered; those in which the parity bit contains a 0 because an odd number of 1's appeared in the word examined by the first parity count. Upon transferring the word from memory to the memory

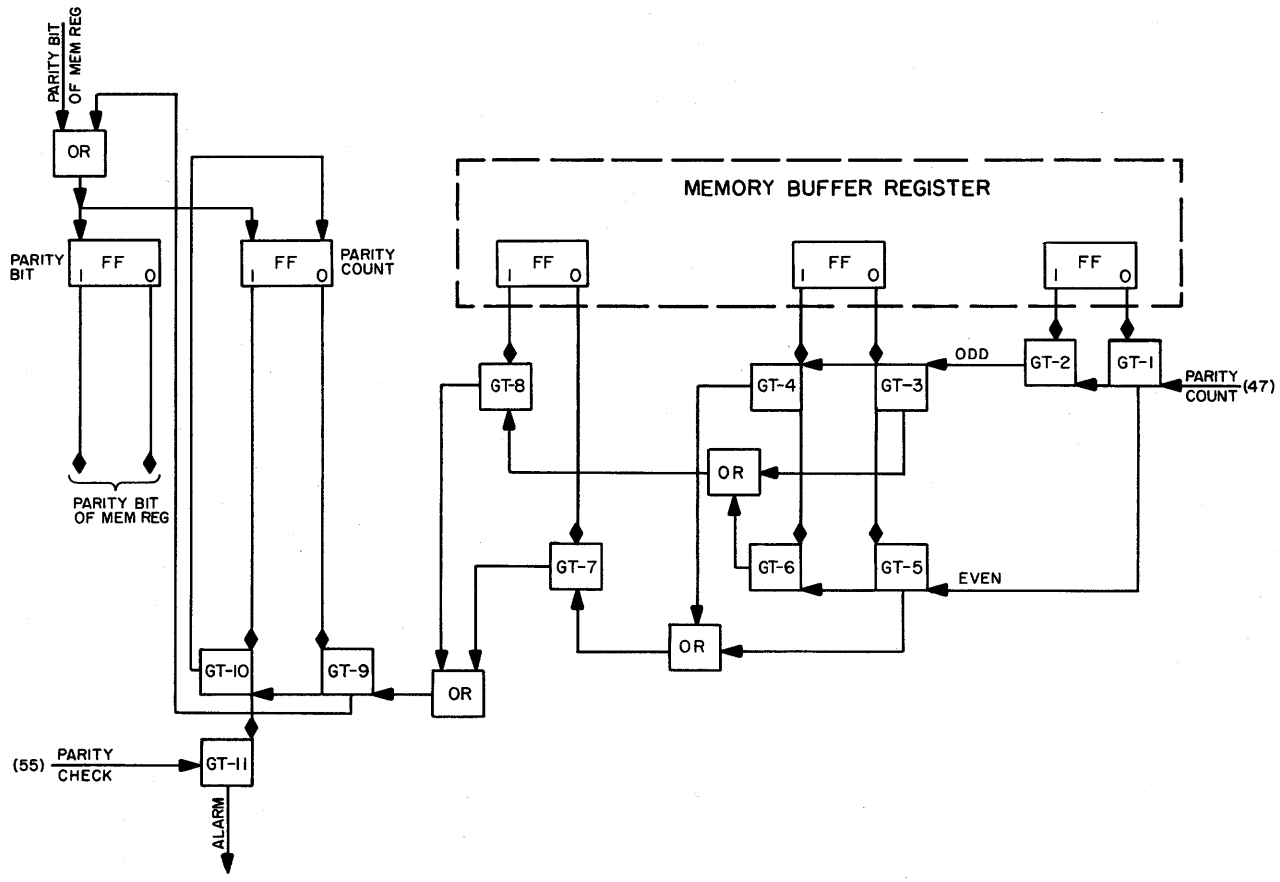


Figure 3–9. Parity Count Operation

buffer register, and assuming a correct transfer, the second parity count results in no pulse being applied to the end circuit (count odd). Since a 0 was transferred to the parity count flip-flops from magnetic core memory, gate tube 11 is not conditioned and no alarm is given when the parity check pulse appears. However, if an incorrect transfer had taken place (an odd number of errors only), an even count pulse would be applied to the end circuit. This pulse would pass through gate tube 9 and set both the parity bit and parity count flip-flops to 1. Gate tube 11, now conditioned by the 1 line of the parity count flip-flop, will yield an alarm signal when the parity check pulse appears.

3.6 SHIFT OPERATION

A shift operation displaces the information contained in a register one position to the left in a shift left operation, or one position to the right in a shift right operation. Shifting can be done in two ways, by a broadside method or a ripple method. In the former case, all the bits of a register are shifted simultaneously; in the latter case, the shift is propagated through the register and each bit is shifted in sequence. The two shifts, when used in their proper contexts, speed the operation of the computer. When only a shift operation is required, the broadside shift, inherently faster than the ripple shift, is used. However, certain operations are, of necessity, ripple operations in that the process proceeds through the register bit by bit. As each bit is operated upon, it becomes free to undergo subsequent operations. To follow a ripple operation in a register with a broadside shift, it is necessary to wait until the ripple operation has propagated through the whole register. To eliminate this time loss, a ripple operation is followed by a ripple shift which also propagates through the register bit by bit. In this way, part of the register is completing the original operation while that part of the register which has already been operated upon is being shifted.

The method used to accomplish a broadside shift is illustrated in figure 3–10, part A. A pair of gate tubes is associated with every flip-flop except the final bit. The figure illustrates a shift

right operation so that the outputs of the gate tubes are applied to the right adjacent flip-flop. Only one gate tube of each pair is conditioned in accordance with the status of the associated flip-flop. The shift right pulse is applied simultaneously to all the gate tubes so that all the information transfers occur at the same time. The slight delay inherent in the gate tubes insures that all the gate tubes are sensed before incoming information to a flip-flop can alter the original status of that flip-flop.

A ripple shift is illustrated in figure 3–10, part B. This operation is similar to the broadside operation with the exception that the initiating pulse is applied only to the next to last flip-flop (bit 2 in the figure). The bit 1 flip-flop is not sensed, and the information in the flip-flop remains unchanged until the transfer of information from bit 2 to bit 3 is completed. Thus, the information transfer propagates through the register, each bit in turn transferring its data to the bit at its right and providing the ripple pulse to the bit at its left.

A variation of the broadside shift is the cycle operation. In this operation, two registers are formed into a shifting ring so that the contents of each bit can be shifted around the ring without being lost.

3.7 SENSE OPERATION

A sense operation ascertains the status of a circuit or circuit element at a given time. The fundamental sensing circuit is illustrated in figure 3–11, part A.

The status of the flip-flop is examined only when a sense pulse is applied. In the figure shown, the appearance of a pulse on the sense information line indicates that the flip-flop contains a 1. The gate tube output is then utilized in those circuits whose action is contingent on the status of the flip-flop. A specific sensing configuration used in AN/FSQ–7 Combat Direction Central is shown in figure 3–11, part B. This circuit is used when the computer is required to make the sign bits of two registers unlike. If the signs are already unlike, the circuit produces no output. If the signs are alike, the gate tube is conditioned and a complement pulse appears on the output line when the make register signs unlike pulse is applied to the gate tube. This output pulse then complements one of the two registers.

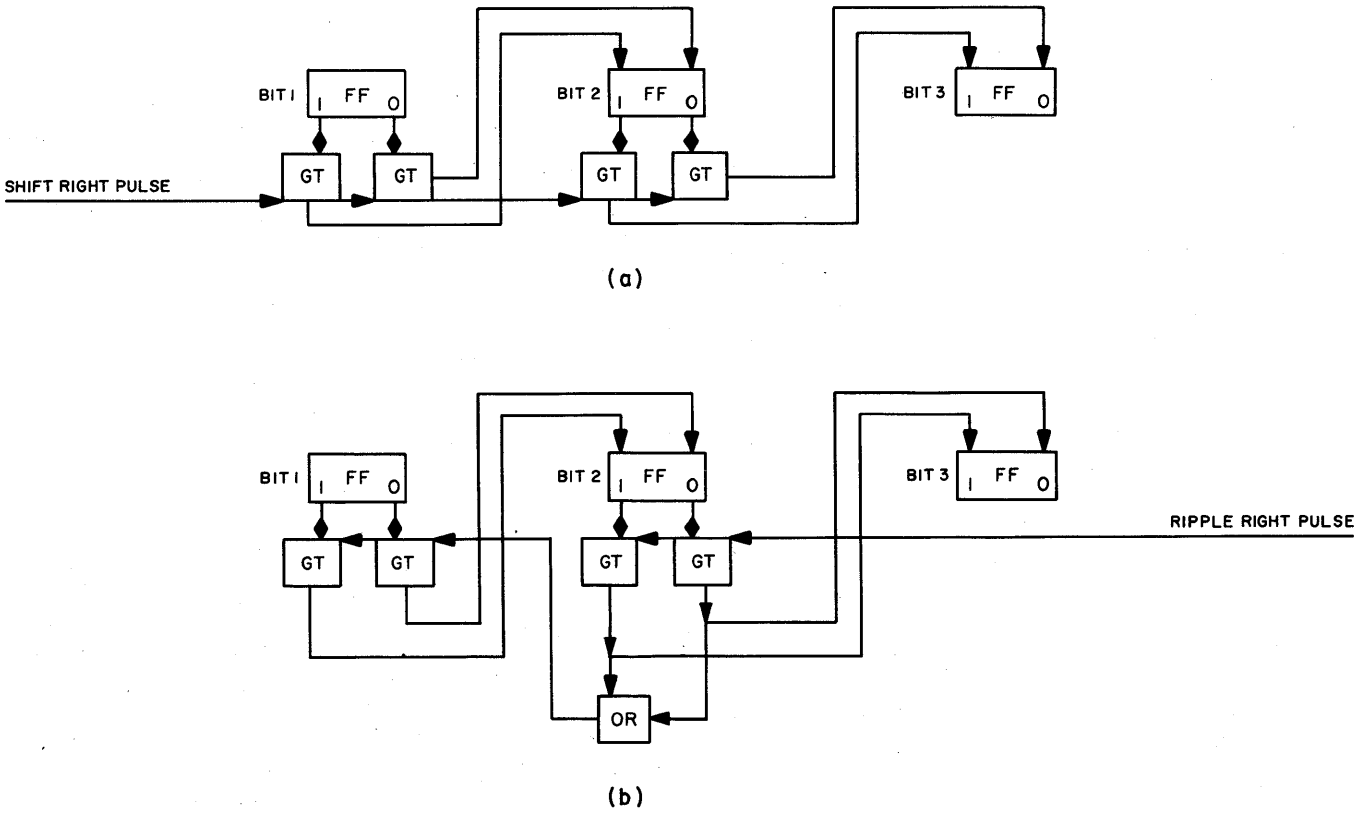


Figure 3–10. Broadside and Ripple Shift Operations

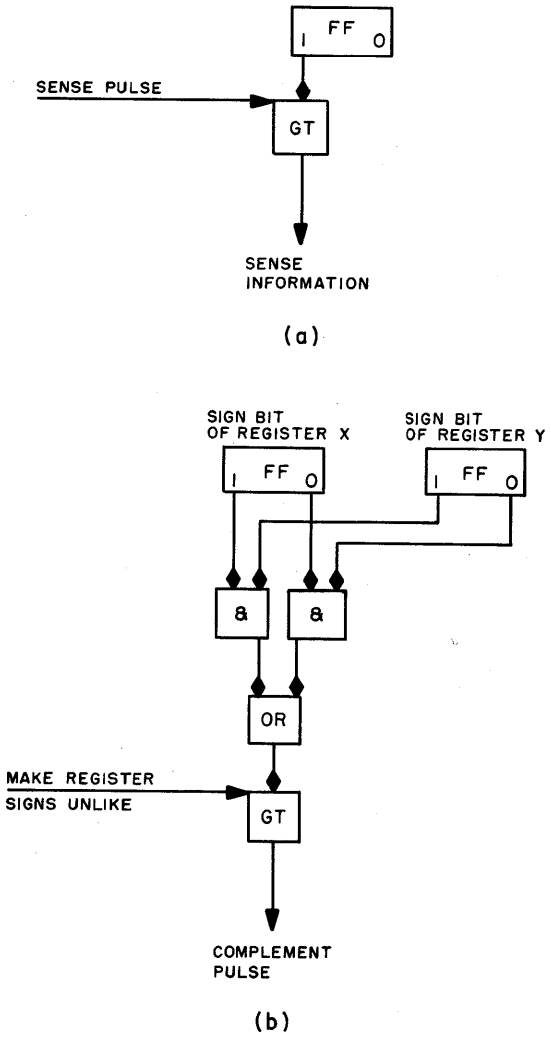


Figure 3-11. Sense Operation

CHAPTER 4

INSTRUCTION ANALYSIS

4.1 INTRODUCTION

This section contains a detailed examination of those instructions of the AN/FSQ–7 Combat Direction Central which utilize the circuits of the arithmetic elements for their execution. This criterion excludes from this discussion all instructions in the IO and reset classes, the *Branch and Index (BPX)* and *Sense (BSN)* instructions in the branch class, and the *Operate (PER)* and *Program Stop (HLT)* in the miscellaneous class. Logical block diagrams are used in analyzing those instructions which are considered. These diagrams are simplified to the extent that they only include the circuits in the arithmetic elements which are pertinent to the execution of the particular instruction under discussion. As was explained in Chapter 1 of this Part, only the left arithmetic element will be treated except in those instances where the two elements do not perform parallel operations. The memory buffer register is only shown on the logical block diagrams when its inclusion is required for clarity. In all other instances it is assumed that this register performs its normal function as an intermediate information transfer point between core memory and the arithmetic element.

4.2 MISCELLANEOUS CLASS

Of the six instructions comprising the miscellaneous class, only four utilize registers in the arithmetic element. These are the *Shift Left and Round (SLR), Clear and Subtract Word Counter (CSW), Extract (ETR),* and the *Load B registers (LDB)* instructions.

4.2.1 *Shift Left and Round (SLR)*

This instruction is used to modify a number which is held in the combined accumulator-B register. The modification is in accordance with the accumulator sign bit and the 17th bit of the word being rounded off. This 17th bit is held in the B register sign bit flip-flop. The following set of rules is observed by the instruction: if the accumulator sign bit is a 0 (number positive), a 1 is added to the number if the B register sign bit is a 1, but nothing will happen if this latter bit is a zero; if the accumulator sign bit is a 1 (number negative), a 1 will be subtracted from the number if the B register sign bit is a 0, but nothing will happen if this latter bit is a 1. Execution of this instruction will reduce a number consisting of 16 magnitude bits plus a sign, to one of 15 magnitude bits plus a sign. This reduction is necessary because all standard registers in the Central Computer System consists of a sign bit plus 15 magnitude bits.

The *SLR* instruction consists of two distinct operations, a shift left followed by a round operation. This instruction ordinarily follows either a *Divide* or a *Multiply* instruction at the end of a divide operation, the remainder appears in the accumulator register and the quotient in the B register. To transfer the quotient into the accumulator register, 15 left shifts are required and are specified by the address part of the instruction. Upon completion of the final shift, the most significant bit of the quotient appears as bit 1 of the accumulator register, the least significant bit as the B register sign bit, and the sign of the quotient, originally the accumulator sign bit, is unchanged. During this shifting process, the remainder is lost. The number is now rounded off by examining the B register sign bit and performing the appropriate operations. At the end of the *Multiply* instruction, the 31-bit product plus sign appears in the combined accumulator-B register. Since the most significant bits of the product already appear in the accumulator register, no shifts are required before rounding and the address part of the instruction specifies 0 shifts. The round operation is then performed in the same manner as explained previously in this paragraph.

The circuits in the arithmetic element which are used to execute this instruction are shown in figure 3–12 for the left arithmetic element. A parallel operation is simultaneously performed in the right arithmetic element. The arithmetic element does not begin its operations until the *Shift Left and Round* instruction has been decoded and 2-megacycle operation is under way. The first 2-megacycle operation shifts the magnitude bits of the number contained in the combined accumulator-B register one place to the left. The sign bit remains the same but is reproduced as bit 15 of the B register. Four commands (2, 5, 81, 82) occurring simultaneously effect this shift. As shown in the figure, command 2 pulses a string of gate tube pairs associated with bits 2 through 15 of the accumulator register. The outputs of each pair

of gate tubes transfers the information in its conditioning flip-flop to the next higher order flip-flop in the register as is explained in the discussion on broadside shift in 3.6 of this Part. Command 5 transfers, in a similar manner, the contents of the accumulator register sign bit flip-flop to the bit 15 flip-flop of the B register. Command 82 pulses bits 1 through 15 of the B register and accomplishes a broadside shift left in this register. The sign bit of the B register is transferred to bit 15 of the accumulator register by command 81. The number of shifts to be performed is contained in the step counter (located in the instruction control element) which is reduced by 1 each time a shift left is performed in the manner just described. When the step counter goes to 0 (specified number of shifts completed), 2-megacycle operation is discontinued and the instruction generates the sequence of commands (16, 21, 87, 60, 84, 214) for the round operation. Command 16 examines the sign of the accumulator register. If it is negative, the gate tube pulsed by this command complements the sign control flip-flop, the accumulator register, and the B register. The next command (21) clears the A register and is followed by command 87, *left round off,* which examines the B register sign bit. If this bit is a 0, the number in the accumulator register does not require rounding; if it is a 1, the number in the accumulator register must be increased by 1. Accordingly, command 87 is applied to a gate tube conditioned by the 1 side of the B register sign bit flip-flop. The output of this gate tube sets the carry storage flip-flop and pulses the carry 1 line of the bit 15 adder. Since the A register contains a 0 (previously cleared by command 21), pulsing of the carry 1 line adds a 1 to the number in the accumulator register. As in every add process, an inherent shift right results. This is corrected upon the application of command 60 to the gate tube previously conditioned (when the carry storage flip-flop was set by command 87). The correcting pulse clears the carry storage flip-flop, shifts the contents of bit 1 in the accumulator register into the sign bit, and, by means of two OR circuits, performs the same functions as commands 2 and 81; i.e., shifts bits 2 through 15 of the accumulator register one place to the left and transfers the contents of the B register sign bit flip-flop into the accumulator register 15-bit flip-flop. The B register is now cleared by command 84. As the final step in this instruction, the proper sign is affixed to the number in the accumulator register. If its sign had been changed (from negative to positive) by command 16 (issued at the beginning of the instruction), the sign control flip-flop will have been complemented to 1, conditioning the gate tube to which command 214 is applied. The resultant output clears the sign control flip-flop and complements the accumulator register, thus placing the number in its negative (complement) form.

4.2.2 *Clear and Substract Word Counter (CSW)*

This instruction is used when it is desired to place into the right accumulator register information concerning the number of words remaining to be processed during a *Read* or *Write* instruction. This information consists of the contents of the IO word counter (complement of the number of remaining words). This is done to enable these data to be manipulated arithmetically in accordance with subsequent instructions. Before the transfer, the right accumulator register is cleared by command 210. The actual transfer from the IO word counter affects only the 1 side of those flip-flops in the right accumulator register whose counterparts in the IO word counter contain 1's. The transfer can occur at either PT–1 or PT–', but not both. It will always occur at PT–1 unless the IO word counter is being stepped at this time.

4.2.3 *Extract (ETR)*

This instruction is used when it is necessary to alter only specific bits of a word in magnetic core memory, leaving the remaining bits of the word unchanged. The left half-word is sent to the left arithmetic element and the right half-word to the right arithmetic element. In both cases, the operations are identical and only the left half-word will be considered. An example to illustrate the use of this instruction is the case where the identification and current velocity of a target are to be continuously displayed. In this instance, the identification is constant but the velocity is variable. Assume that the first two bits of the half-word are the identification bits while the remaining bits specify the current velocity and that these data have been stored in a specified memory address for a particular target. Assume further that a word consisting of 1's in the first two leftmost bits and 0's in the remaining bits has been stored in a second memory address. (The function of this number will become apparent later on in the discussion.) The contents of this memory address are displayed (in converted form) on a cathode-ray tube face. When new data for this target are fed to the computing circuits, the new velocity is calculated by a preset program and appears in the accumulator register. However, the first two bits

COMMAND NUMBER	PULSE TIME
2,5,81,82	2 MC
16,21	PT-1
87	PT-2
60	PT-5
214	PT-6

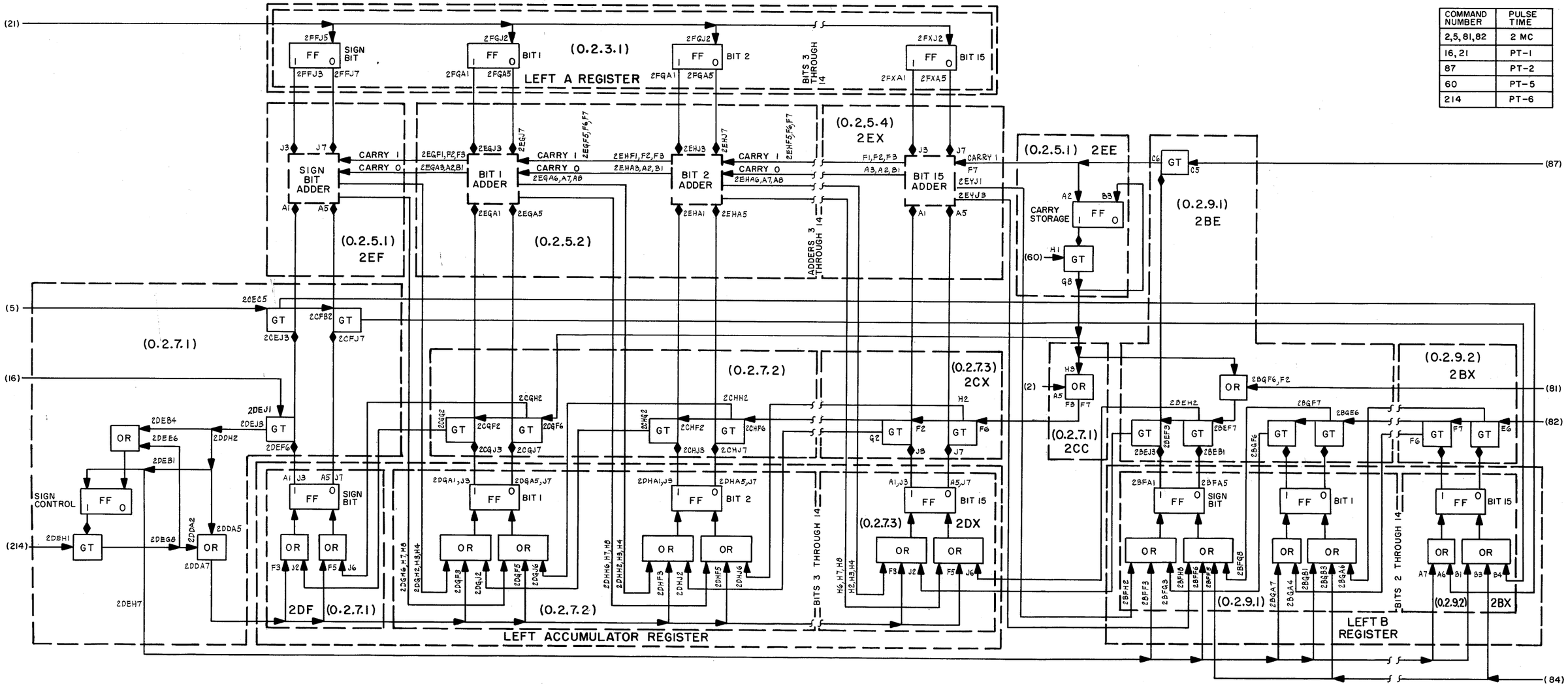
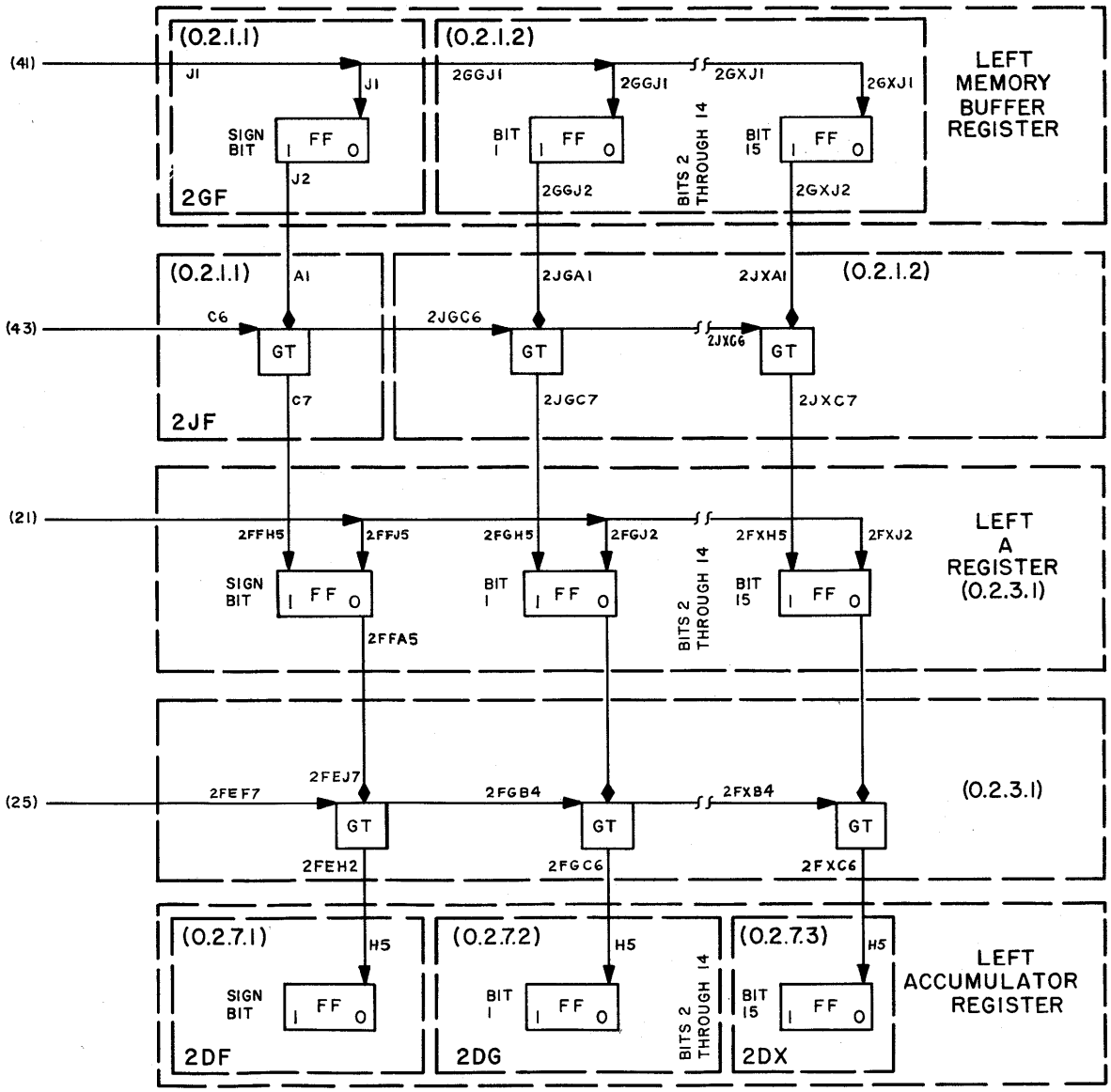


Figure 3–12. Shift Left and Round (SLR), Logical Block Diagram



COMMAND NUMBER	PULSE TIME
21,41	OT-1
43	OT-7
25	OT-9

Figure 3–13.* Extract (ETR), *Logical Block Diagram

in the accumulator register will not enter into the velocity calculation and are 0's. During the subsequent *Extract* instruction, these first two bits will be replaced by the target identification bits. These new velocity data must usually replace the previous velocity information in the memory address.

Before the *Extract* instruction is executed, the contents of the accumulator register are stored in a magnetic core memory location other than the two already used. A *Clear and Add* instruction is now performed during which the contents of the original memory address (identity and original velocity) are placed in the accumulator register. The *Extract* instruction is now performed. The circuits for this instruction are shown in figure 3–13. At the beginning of the operate time for this instruction, the memory buffer register and A register are cleared by commands 41 and 21, respectively. Following these clearing operations, the memory cycle coincident with this operate time transfers that number previously stored in magnetic core memory, which contains 1's in the

first two bits and 0's in the remaining bits, into the memory buffer register. Command 43 transfers this number into the A register. Command 25 is issued next and, as can be seen from the figure, affects the original information now held in the accumulator register in the following way: wherever the A register contains a 1, the corresponding accumulator register bit remains unchanged; wherever the A register contains a 0, the corresponding accumulator register bit becomes a 0. In effect, this multiplies the two registers together bit by bit. The reason for originally writing into magnetic core memory the number with 1's in the first two bits can now be justified, since the first two bits hold identification information which must remain unchanged at the end of this instruction. At this point, the accumulator register holds two identity bits followed by 0's in the remaining bits. This completes the *Extract* instruction insofar as the arithmetic element is concerned. This instruction has merely prepared a number in the accumulator register to which the latest velocity information previously stored in core memory can be added without losing the identification of the target.

4.2.4 *Load B Registers (LDB)*

This instruction transfers information from the memory buffer register into the B register. Before the transfer can be accomplished, the B register is cleared by command 84, after which the usual transfer operation is effected by command 39. (Refer to Chapter 3 of this Part.) This process is simultaneously duplicated in the right arithmetic element.

4.3 ADD CLASS

The nine instructions comprising the add classes all utilize registers in the arithmetic element. These are *Add* (*ADD*), *Subtract* (*SUB*), *Twin and Add* (*TAD*), *Twin and Subtract* (*TSU*), *Clear and Add* (*CAD*), *Clear and Subtract* (*CSU*), *Clear and Add Magnitude* (*CAM*), *Difference Magnitudes* (*DIM*), and *Add B Registers to Accumulator Registers* (*ADB*). There is a common procedure associated with all add class instructions. They are all initiated by command 64. This command performs two functions. It transmits a carry 0 pulse to the bit 15 adder and simultaneously pulses two gate tubes associated with the B register sign flip-flop. This latter operation causes the contents of the B register sign bit flip-flop to be transferred to the sign storage flip-flop, thus enabling the sum of the final (lowest order) bit in the accumulator and A registers to be stored temporarily in the B register sign flip-flop without losing the information originally contained in this flip-flop. The action of the adders has been described in detail in paragraph 2.4 of this Part. Fundamentally, they add three digits, sending the sum to the next lowest order flip-flop in the accumulator register and providing the appropriate carry pulse for the next highest order adder.

The process ripples through all adders, and if there is a carry out of 1 from the sign bit adder, the carry pulse sets the carry-storage flip-flop. This setting action conditions two gate tubes in preparation for the *end carry* and *conditional shift left* commands. Before these commands are used, however, the inherent shift right is corrected by commands 2, 4, and 81, thus aligning the sum in the accumulator register. Command 2 initiates a broadside shift which transfers the information in all flip-flops from 2 through 15 one place to the left. Command 4 shifts the contents of the bit 1 flip-flop in the accumulator register into the sign bit flop-flop of this same register. Command 81 transfers the B register sign bit into bit 15 of the accumulator register. These latter three commands are issued at the same pulse time, effectively transferring the entire sum one position to the left. Command 89 is coincident with the other three commands, and returns to the B register sign bit flip-flops its original contents. The A register is then cleared by command 21. After the inherent shift right has been corrected in this manner, an inspection of the carry storage flip-flop is made by command 63. If a 1 has been carried out of the sign bit adder, this command pulse is passed by a gate tube conditioned by the 1 side of the carry storage flip-flop, and carries a 1 into the bit 15 adder. Thus, the sum in the accumulator is increased by 1 and experiences a second inherent shift right. This shift is corrected by command 60, which senses a second gate tube conditioned by the 1 side of the carry storage flip-flop. The pulse provided by this gate tube performs the shift left functions previously performed by commands 2, 4, 81, and 89.

4.3.1 *Add (ADD), Subtract (SUB), Twin and Add (TAD), Twin and Subtract (TSU)*

Figure 3–14 shows the circuits in the left arithmetic element which are required for the execution of the *ADD, SUB, TAD,* and *TSU* instructions. Preparatory to executing the operations in the left arithmetic element, commands 24 and 41 clear the A register and the memory buffer register, respectively. The memory buffer register now receives from magnetic core memory one of the words which is to be used in the operation, the

COMMAND NUMBER	PULSE TIME
21,44	OT-1
43 OR 44	OT-7
26	OT-9
64	OT-10
2,4,81 89,21,41	PT-1
63	PT-2
60	PT-5

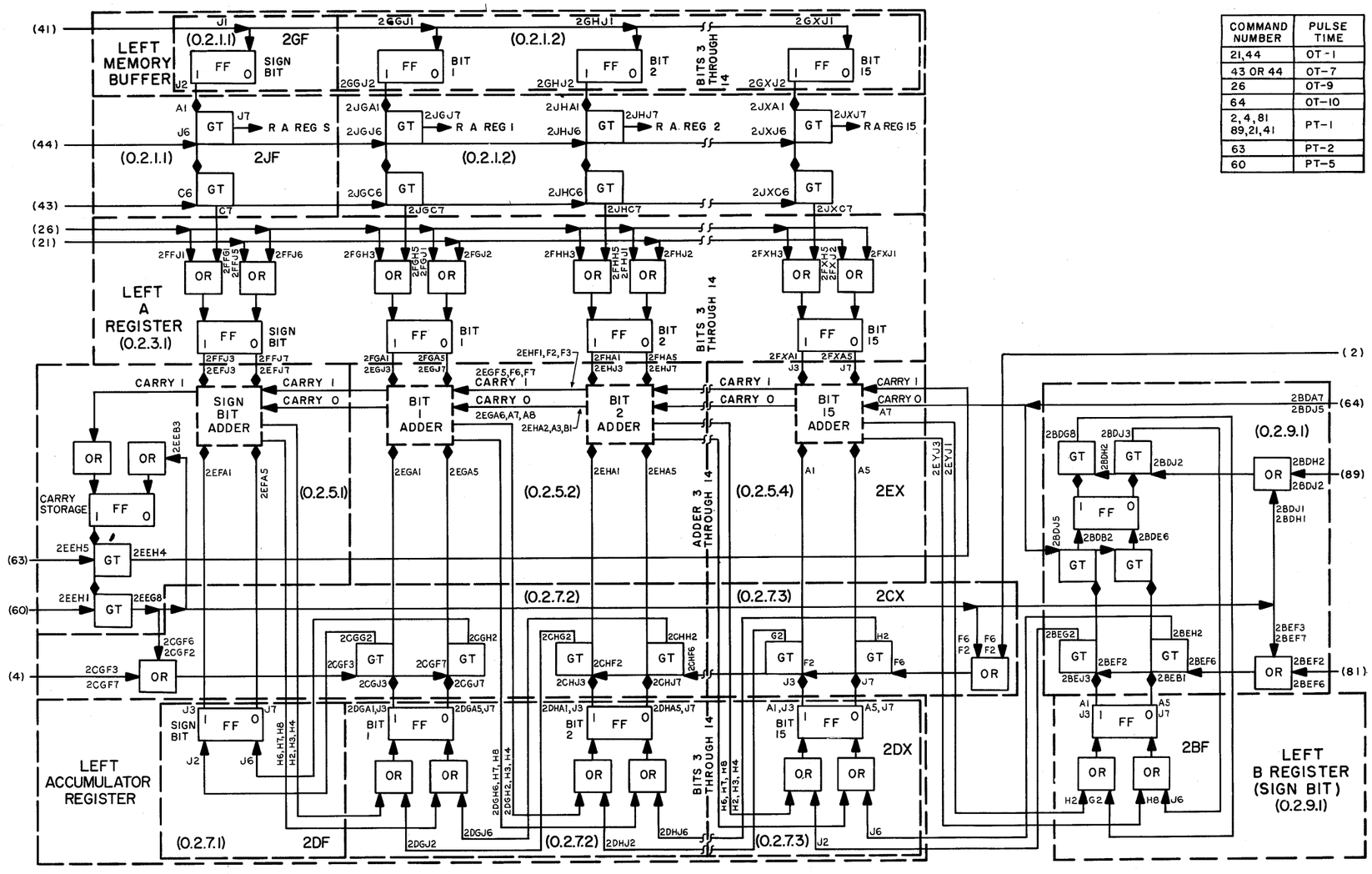


Figure 3-14. **Add (ADD), Subtract (SUB), Twin and Add (TAD), Twin and Subtract (TSU), *Logical Block Diagram***

other word being held in the accumulator register from the previous operation. The difference between the twin and non-twin instructions is in the manner of information transfer between the memory buffer registers and the A registers. In the non-twin instructions, the left memory buffer register supplies data to the left A register and the right memory buffer register supplies data to the right A register. As shown in the figure, this type of transfer is accomplished by command 43 for the left element. In the twin instructions, the left memory buffer register supplies the data to both the left and right A registers. This is accomplished by command 44. The right memory buffer register does not transfer data in these instructions. With the exception of the transfer by command 44, the operation of the left and right arithmetic elements are identical for the four instructions.

The difference between the add and subtract processes is that subtraction requires an extra command (26) which complements the contents of the A register just prior to the execution of the standard add process used in the arithmetic element.

With these considerations in mind, the following sequence can be established for the four add class instructions under discussion. Commands 21 and 41 are issued first and clear the A register and memory buffer register. For the non-twin instructions, command 43 which follows transfers the data in the left memory buffer register to the left A register. For the twin instructions, command 44 supplants command 43 and places the data in the left memory buffer in both left and right A registers. Command 26 will be issued next only if a subtract type instruction is in progress, and complements the contents of the A register. Following these preliminary steps, the add process itself is initiated by command 64, and continues identically for all four instructions. The inherent shift right is corrected by commands 2, 4, 81, and 89. At the same time, the memory buffer register is cleared in preparation for the next instruction and the A register is cleared (21) so that its original contents will not be added to the sum a second time in the event that an end carry is performed. The *end carry* command (63) is now issued, and is followed by command 60 which will correct the second inherent shift right if an end carry has been performed. As a final step, command 66 (not shown in the figure) tests for and records an overflow condition.

4.3.2 *Clear and Add (CAD), Clear and Subtract (CSU), Clear and Add Magnitude (CAM)*

Figure 3–15 shows the circuits in the left arithmetic element which are required for the execution of the *CAD, CSU,* and *CAM* instructions. Similar processes occur simultaneously in the right arithmetic element. The three instructions are similar in that they all involve the transfer of data from magnetic core memory to the accumulator register. The transfers utilize the standard addition process where the contents of the accumulator register are added to the contents of the A register. For each of these instructions, the accumulator registers are cleared to 0 before the actual add process is performed. Accordingly, the add process merely results in the transfer of data from memory, via the A register and adders, to the accumulator register.

In accordance with the specific instructions, the information from memory can be modified before it appears in the accumulator register. For the *Clear and Add* instruction, no modification is called for and the contents of the memory register specified by the address part of the instruction appear, finally, in the accumulator register. In the *Clear and Subtract* (*CSU*) instruction, the complement of the contents of the memory register specified by the address part of the instruction are placed in the accumulator register. In the *Clear and Add Magnitude* (*CAM*) instruction, the positive value of the contents of the specified memory address are placed in the accumulator.

Referring to figure 3–15, a sequence of commands for the *CAD, CSU,* and *CAM* instructions can now be established. Commands 21 and 41 clear the A register and memory buffer register, respectively. The accumulator register is cleared by command 10. During the accompanying memory cycle, the information in the specified memory address is transferred to the memory buffer register. The transfer of these data from the memory buffer register to the A register is accomplished by command 43. At this point, the computer distinguishes between the three clear-type instructions. Command 22 is issued next only when the *CAM* instruction is in progress. This command pulse senses a gate tube conditioned by the 1 side of the A register sign flip-flop. If this flip-flop is negative (contains a 1), the conditioned gate tube passes a pulse which complements the A register. If the sign flip-flop is positive (contains a 0), the A register is not affected. Thus, the contents of the A register are always positive before the add process is initiated. If the *CSU*

instruction is in progress, command 22 is supplanted by command 26. This latter command always complements the A register. Thus, the A register is made positive if it was originally negative, and negative if it was originally positive. Both commands 22 and 26 are not issued if the *CAD* instruction is in progress.

The next command to be issued is command 64, which pulses the carry 0 line of the bit 15 adder and initiates the add process. This is followed by commands 2, 4, 81, and 89, which correct for the inherent shift right occurring during every add process. The memory buffer and A registers are then cleared again by commands 41 and 21 in preparation for the next instruction. Commands 63 and 60 (used for end carry but not shown in the figure) are issued next, but serve no useful functions here since end carry can not occur for the three instructions under discussion.

4.3.3 *Difference Magnitudes (DIM)*

The function of this instruction is to calculate the difference between the absolute values of two quantities. The *DIM* instruction can be used in conjunction with the *Branch if Minus* (*BFM*) instruction to determine which of the two quantities is the larger in absolue value. The two quantities to be operated upon are held in the A and accumulator registers. The quantity in the A register is brought down from core memory during this instruction. The quantity in the accumulator register is the result of a previous arithmetic operation or is placed in that register previous to the *DIM* instruction.

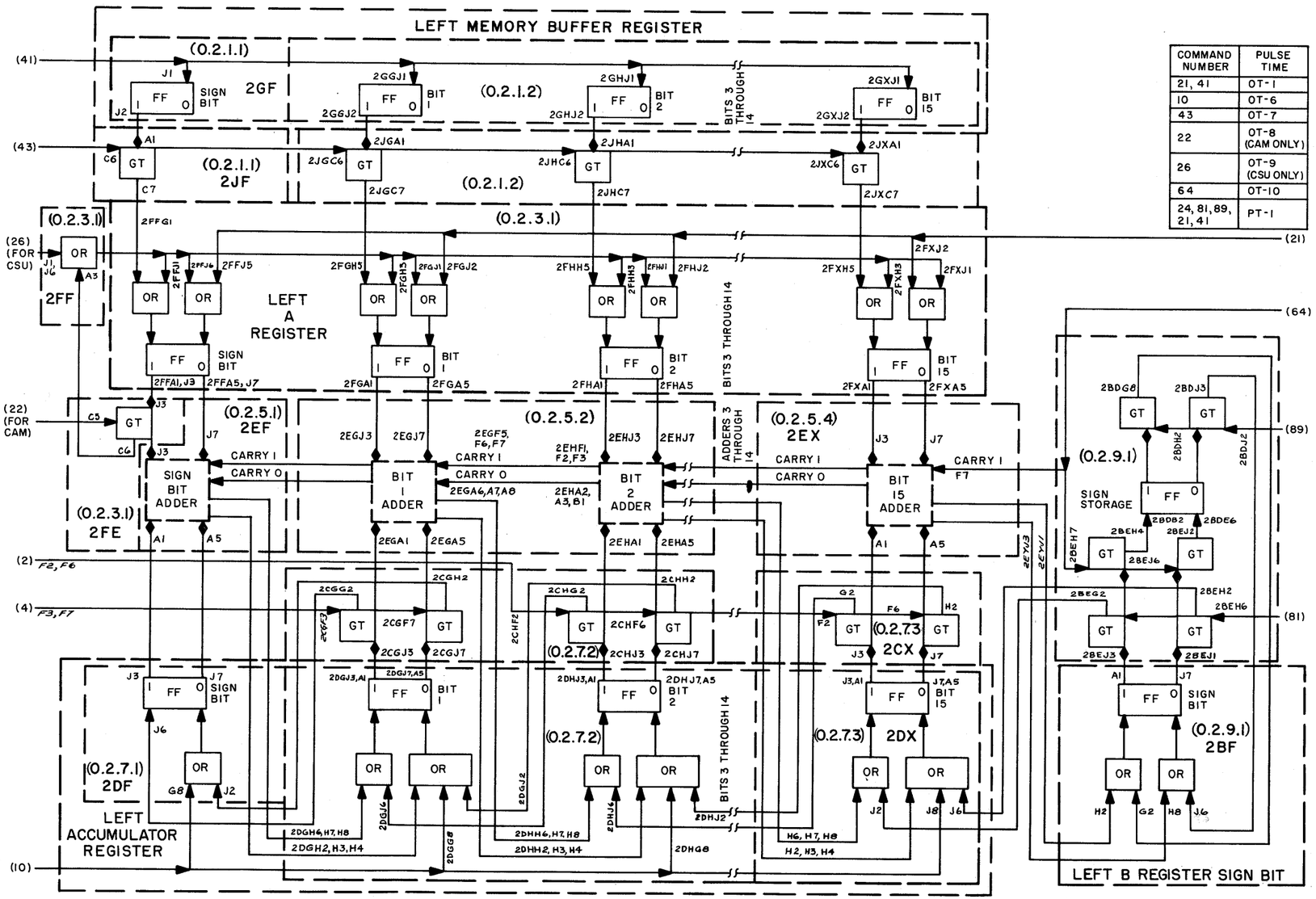
The data in the accumulator are copied into the B register during the instruction so that while the original accumulator register contents are replaced by the difference magnitude answer, these original contents still appear in the B register at the conclusion of the instruction. A parallel operation occurs simultaneously in the right arithmetic element.

Figure 3–16 shows the circuits in the left arithmetic element which are required for the execution of the *DIM* instruction. At the start of the operate time for this instruction, the memory buffer and A registers are cleared by commands 41 and 43, respectively. During the memory cycle, which is coincident with this operate time, information is transferred from memory into the A register. Command 84, which is issued next, clears the B register in preparation for command 1, which copies into that register the contents of the accumulator register. The contents of the A register are then made positive by command 22. This command pulse senses a gate tube conditioned by the 1 side of the A register sign flip-flop. If this flip-flop contains a 1 (negative), the gate tube output pulse complements the A register. If this flip-flop contains a 0 (positive), the register is not affected. The contents of the accumulator register are similarly made positive by command 13. Thus, absolute values now appear in the two registers. In order to obtain the difference between these two absolute values, the contents of the A register are complemented (made negative) by command 26. The add process is then initiated by command 64, which pulses the carry 0 line of the bit 15 adder. This is followed by commands 2, 4, 81, and 89, which correct for the inherent shift right occurring during every add process. Commands 41 and 21, occurring at this same pulse time, clear the memory buffer and A registers, respectively. The memory buffer register is cleared in preparation for the next instruction, while the A register is cleared to enable an end carry to be performed if it is required. The *end carry* command (63) senses a gate tube conditioned by the 1 side of the carry storage flip-flop and executes the end carry operation. Command 60, which follows, corrects the second inherent shift right, which occurs only if an end carry has been performed.

4.3.4 *Add B Registers to Accumulator Registers (ADB)*

The *ADB* instruction enables the contents of the B register to be added to the contents of the accumulator register without first storing the B register contents in core memory. This instruction differs from an ordinary add-type instruction in the method used for placing one of the operands in the A register. In the *ADB* instruction, one of the operands is originally held in the B register and is transferred directly to the A register. For the ordinary add-type instructions, one of the operands must be brought into the A register from core memory via the memory buffer register. Except for this difference in preparatory operations, the *ADB* instruction is identical to the ordinary add-type instructions discussed previously.

Figure 3–17 shows the circuits in the left arithmetic element which are required for the execution of this instruction. A parallel operation occurs simultaneously in the right arithmetic element. The B register supplies one of the operands normally furnished by the memory buffer register. The transfer of data from the memory buffer register to the A register is not inhibited. Accordingly, command 43, which effects



COMMAND NUMBER	PULSE TIME
21, 41	OT-1
10	OT-6
43	OT-7
22	OT-8 (CAM ONLY)
26	OT-9 (CSU ONLY)
64	OT-10
24, 81, 89, 21, 41	PT-1

Figure 3–15. **Clear and Add (CAD), Clear and Subtract (CSU), Clear and Add Magnitude (CAM), *Logical Block Diagram***

COMMAND NUMBER	PULSE TIME
21	OT-1
84	OT-6
1,22	OT-8
13,26	OT-9
64	OT-10
2,4,81 89,21	PT-1
63	PT-2
60	PT-5

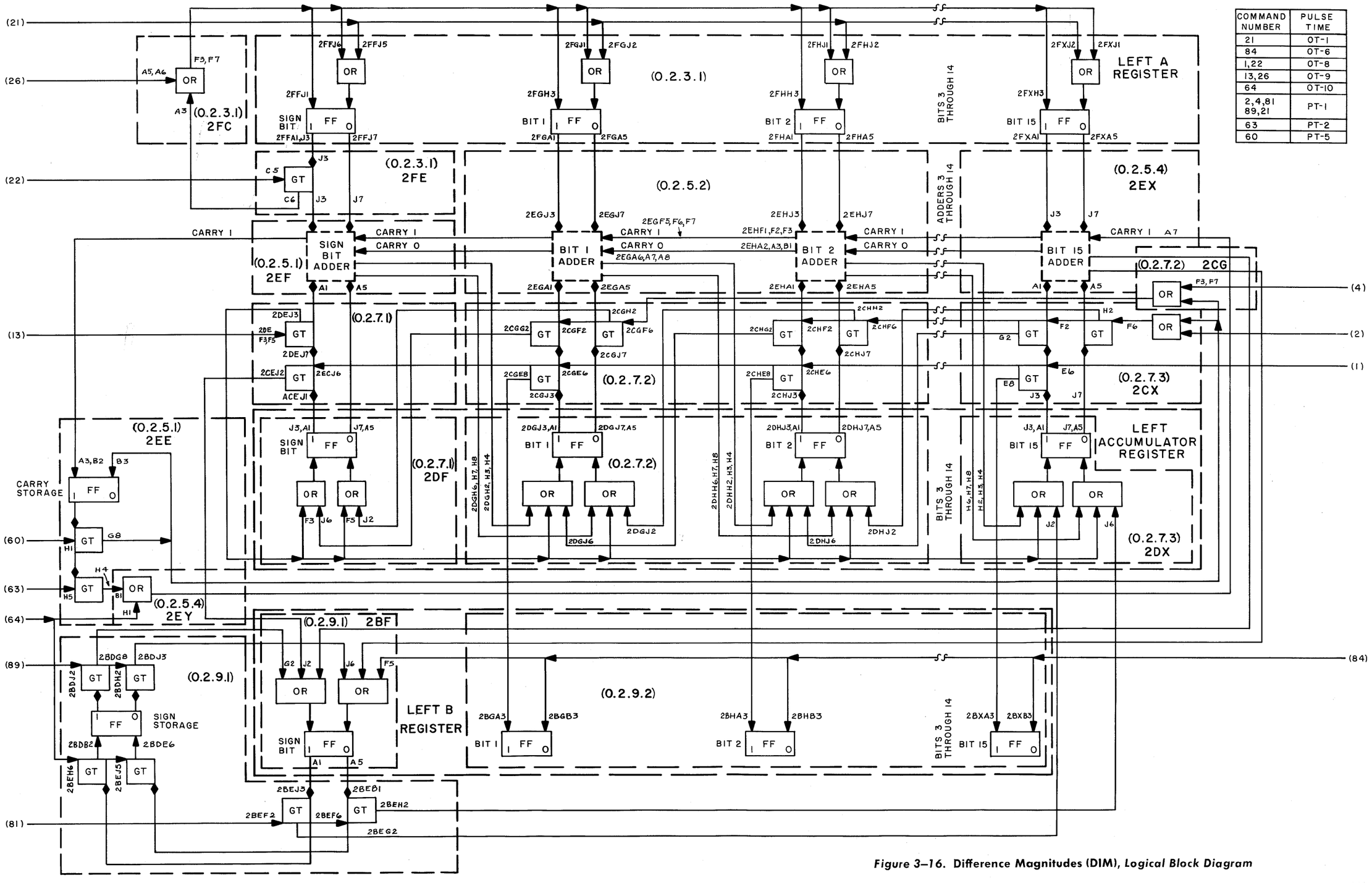
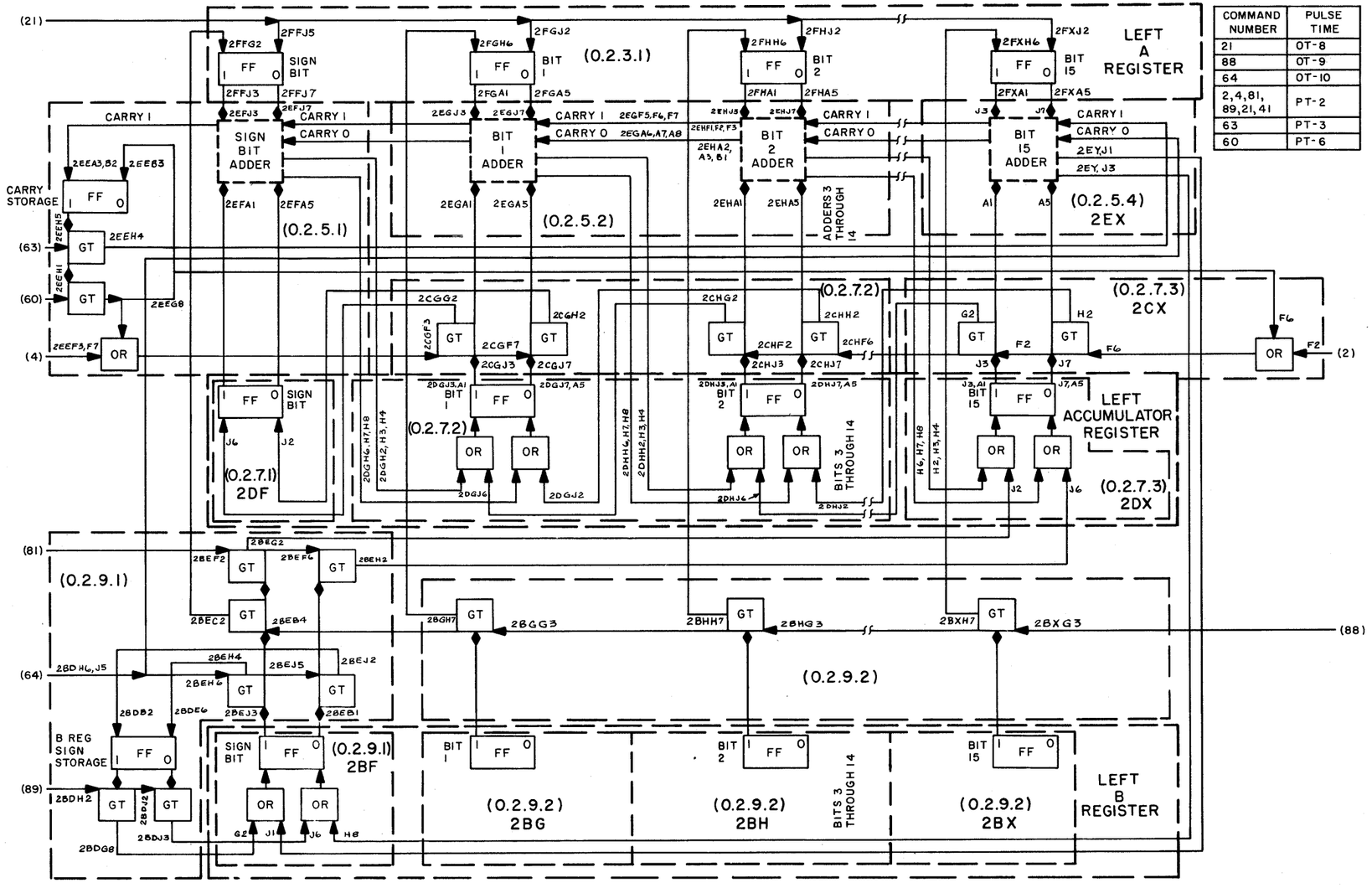


Figure 3-16. **Difference Magnitudes (DIM),** *Logical Block Diagram*



COMMAND NUMBER	PULSE TIME
21	0T-8
88	0T-9
64	0T-10
2,4,81, 89,21,41	PT-2
63	PT-3
60	PT-6

Figure 3–17. **Add B Registers to Accumulator Registers (ADB),** ***Logical Block Diagram***

this transfer, is followed by command 21, which clears the A register of this meaningless information. The contents of the B register are than transferred to the A register by command 88. Command 64, which is issued next, initiates the add process and is followed by commands 2, 4, 81, and 89, which correct the inherent shift right occurring for every add operation. The memory buffer register and the A register are cleared by commands 41 and 21, respectively. Command 63, which is issued next, executes an end carry operation if one is required. Command 60 corrects the second inherent shift right, which occurs only if an end carry has been performed.

4.4 MULTIPLY CLASS

4.4.1 *Multiply (MUL), Twin and Multiply (TMU)*

The only difference between these two instructions lies in the method of providing multiplicands to the two arithmetic elements. In the *Multiply* instruction, the multiplicand for the right arithmetic element is supplied by the right memory buffer register and the multiplicand for the left arithmetic element is supplied by the left memory buffer register. In the twin instruction, the left memory buffer register supplies the same multiplicand to both elements. The preparatory steps before actual multiplication begins are therefore slightly different, but the multiplication process itself is identical for both instructions.

The pertinent circuits used in performing both these instructions are shown in figure 3–18. The first preparatory step is to clear the A register and the memory buffer register by issuing commands 21 and 41, respectively. As the next step, the sign control flip-flop is cleared by command 8 in praparation for storing the eventual sign of the product. In the type of multiplication process used in this computer, the sign of the final product is determined before any multiplication takes place. This is accomplished by making both multiplicand and multiplier positive before beginning the operation. To do this, either the A register (holding the multiplicand) or the accumulator register (temporarily holding the multiplier), or both, are complemented if the operand it holds is negative. The final sign is stored by having the sign control flip-flop complemented each time either of these two registers is complemented. The sign control flip-flop is originally in its 0 or positive status. If it is complemented once (one of the operands negative), it is transferred to its 1 (negative) side. If the sign control flip-flop is complemented twice (both operands negative), the flip-flop is correctly returned to its positive indication. Accordingly, command 13 examines the accumulator register sign bit. (It is assumed that the number to be used subsequently as the multiplier in this operation has been placed in the accumulator register by a previous instruction.) If this sign bit is 0, the number in the register is already positive and no complement operation need be performed. The gate tube to which command 13 is applied, therefore, is conditioned by the 1 side of the sign bit flip-flop. If the number is negative, this gate tube is conditioned, and as shown in the figure, a pulse is propagated which complements the accumulator register and the sign control flip-flop. While this process is being carried out, command 43 is issued, transferring the multiplicand from the memory buffer register to the A register. In the case of the twin instruction, command 44 is also applied to the left memory buffer register at this time, transferring the contents of this register to the right A register. If the non-twin instruction is being executed, command 51 supplants command 44 and transfers the contents of the right memory buffer register to the right A register. As these transfers are being performed, the B register, which will hold the multiplier during the multiplication process, is cleared by command 84. The A register is now made positive by command 22, which operates on the A register in the same way that command 13 operated on the accumulator register. The sign of the final product is now held in the sign control flip-flop. Command 1 transfers the multiplier (now positive) to the B register from the accumulator register and the latter register is immediately cleared by command 10. Concurrent with these operations in the arithmetic element, the instruction control element sets the step counter to 15, turns off the time pulse distributor, and begins sending a series of 2-megacycle commands (command 83 iterated 15 times) to the arithmetic elements. The multiplication process is now initiated. Each partial product command examines the multiplier bit which is currently held in the 15th bit of the B register. As can be seen from the figure, the multiplicand is added to the contents of the accumulator register if the examined bit of the multiplier is a 1. The add process is carried out in the standard way for this computer; i.e., the carry 0 input of the bit 15 adder is pulsed and the process is rippled through the adders until, finally, the sum appears in the accumulator register shifted one place to the right. No end carry is involved because both numbers being added are positive. If the examined bit is a 0, the pulse,

when passed, performs only a right shift of the current partial product. In both cases, the pulse which ensues after examination of the 15th bit of the B register shifts the B register broadside one place. This brings the next multiplier bit up for examination by the next partial product command. This will result in the loss of one bit of the multiplier, but this bit is no longer of interest since its effect on the final product has already been taken into account. The step counter in the instruction control element counts down one for each partial product step, and having originally been set to 15, permits 15 of these operations (one for each magnitude bit of the multiplier) to take place before turning off the 2-megacycle operation and resuming the machine cycles. The product appears as a 31-bit word plus a positive sign in the combined accumulator-B register. It only remains to affix the proper sign to the product and this is done by command 9. This command complements the combined accumulator-B register if the sign control flip-flop is in the negative status.

4.4.2 *Divide (DVD), Twin and Divide (TDV)*

The same distinctions exist between these two instructions that exist between the *Multiply* and the *Twin and Multiply* instructions; i.e., during the twin instruction, the information to be held in the A register during the divide operation is furnished by the left memory buffer register to both arithmetic elements. During the non-twin instruction, each element receives the information for its A register from its own memory buffer register. Otherwise, the two elements operate on the data in an identical manner. This discussion will assume that a *Divide* instruction is to be executed.

Prior to this instruction, the dividend, being the result of a previous operation, appears in the combined accumulator-B register. The dividend therefore consists of a sign bit followed by 31 magnitude bits. In preparation for placing the divisor into the A register, the A register and the memory buffer register are cleared by commands 21 and 41, respectively. The circuit configurations for accomplishing this and the other steps in the *Divide* instruction are shown in figure 3–19. The divisor consists of a sign bit followed by 15 magnitude bits. This divisor is transferred from memory into the memory buffer register by the memory cycle operations and, upon issuance of command 43, is copied into the A register. (Command 44 would also be issued at this time if the *Twin and Divide* instruction were being executed.) As was explained in the section on arithmetic processes, the carry-out pulses from the sign bit adder, after a trial subtraction is performed, are used to place the quotient bit associated with that subtraction into the bit 15 flip-flop of the B register. Accordingly, the carry outputs of the sign bit adder must be directly connected to the inputs of the bit 15 flip-flop in the B register. The Central Computer System accomplishes this switching action by means of command 67. As shown in the figure, command 67 complements the divide connect flip-flop and clears the carry storage flip-flop. The divide connect flip-flop, by means of the two gate tubes conditioned by its 1 side, provides the direct connection between the carry outputs of the sign bit adder and the inputs of the B register 15 flip-flop. The next preliminary step is to make the dividend positive. This is accomplished by applying command 16 to the gate tube, which is conditioned by the 1 (negative) side of the accumulator register sign bit flip-flop. If the dividend is negative, the output of this gate tube will complement the combined accumulator-B register and the sign control flip-flop. The sign control flip-flop, always in the 0 status at the beginning of an instruction, will be set to 1 by this operation, indicating that one of the operands is negative at the start of the *Divide* instruction. The divisor, now in the A register, is also made positive by issuing command 22. If the divisor is negative, the gate tube to which command 22 is applied will be conditioned and a pulse will be propagated which will complement the A register. This same pulse complements the sign control flip-flop. If the dividend had also been negative, the sign control flip-flop would be returned to its 0 (positive) status, indicating that both operands were origially negative. If the dividend had originally been positve, the sign control flip-flop would be set to 1 (negative) indicating, as before, that the final sign of the quotient must be negative because only one of the operands was negative.

Sequential operations in the instruction control element have concurrently set the step counter located in that frame to 17, have turned on 2-megacycle operation, and have set the pause flip-flop, thus stopping the generation of time pulses and instruction pulses. The 2-megacycle pulses supplied during the ensuing pause do not directly pulse the registers and flip-flops in the arithmetic element. They are used instead to activate the divide time pulse distributor (located in the instruction control element.) This unit transmits a continuous sequence of timed pulses functionally separated into distinct groups. The five pulses in each group are termed DVTP-0, DVTP-1, DVTP 2, DVTP-3, and

COMMAND NUMBER	PULSE TIME
21	OT-1
8	OT-6
13, 84	OT-7
10, 22	OT-8
83	2 MC
9	PT-6

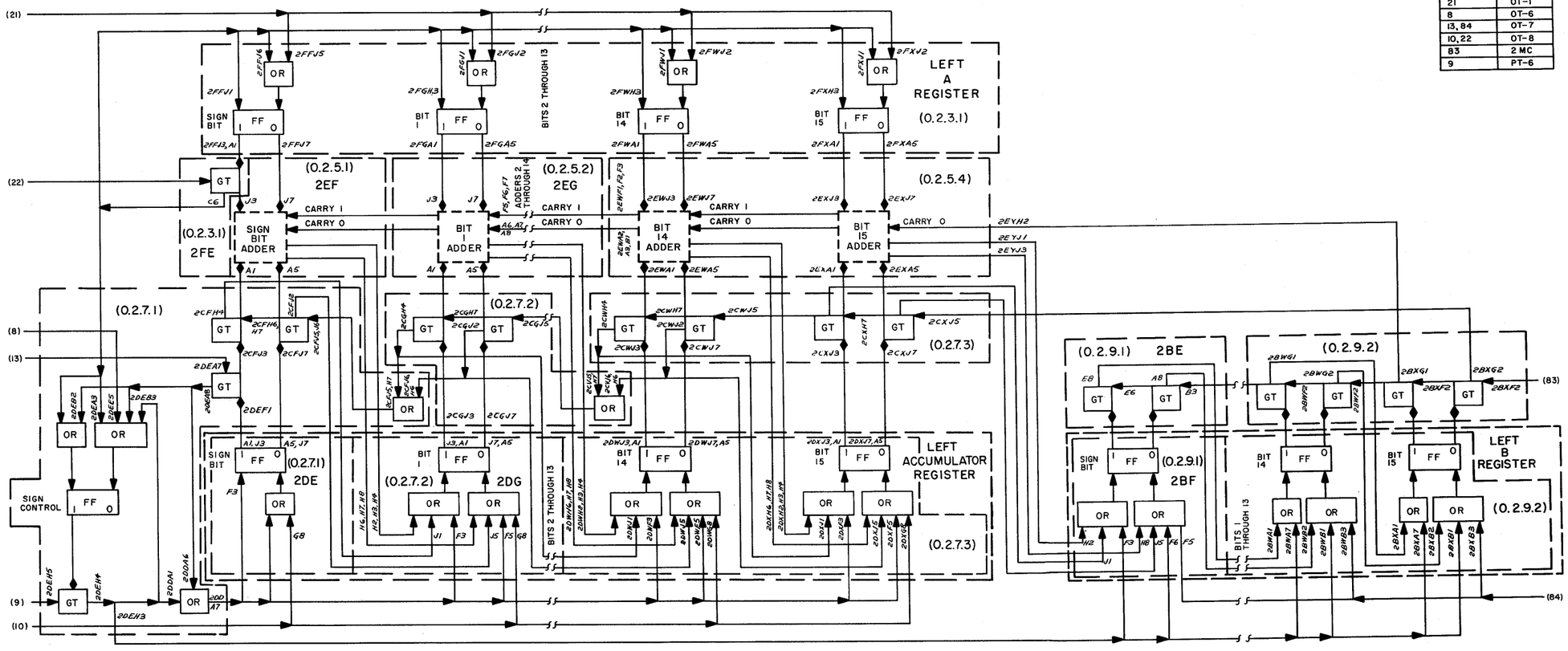
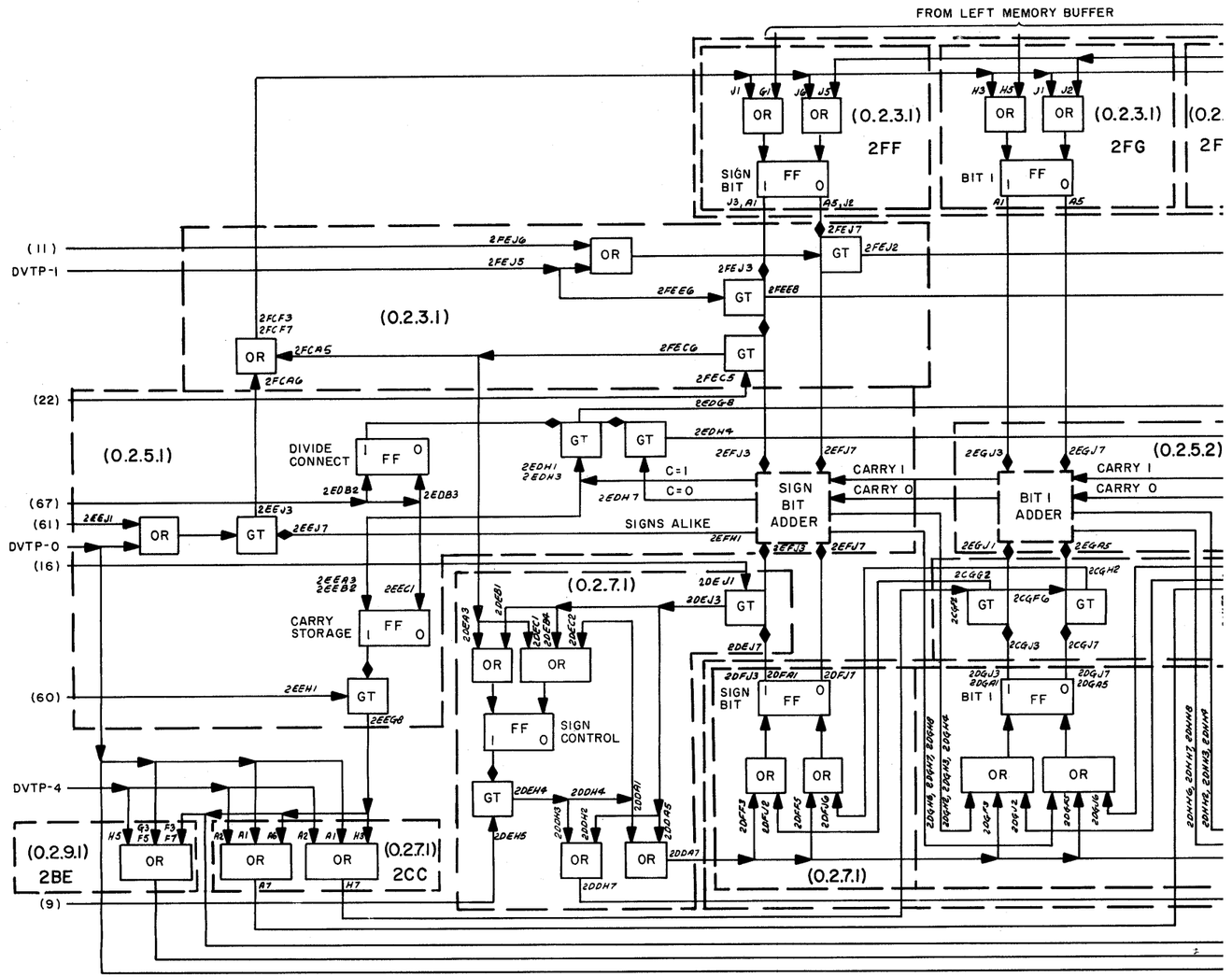


Figure 3-18. **Multiply (MUL), Twin and Multiply (TMU), *Logical Block Diagram***


FROM LEFT MEMORY BUFFER
(0.2.3.1)
2FF
2FG
SIGN BIT
BIT 1
FF
OR
GT
(11)
DVTP-1
(22)
(0.2.5.1)
(0.2.5.2)
DIVIDE CONNECT
CARRY STORAGE
SIGN CONTROL
SIGN BIT ADDER
BIT 1 ADDER
CARRY 1
CARRY 0
C=1
C=0
SIGNS ALIKE
(0.2.7.1)
(67)
(61)
DVTP-0
(16)
(60)
DVTP-4
(9)
(0.2.9.1)
2BE
2CC

COMMAND NUMBER	PULSE TIME
21, 67	OT-1
16	OT-7
22	OT-8
80 (DVTP-0, DVTP-1, DVTP-4)	2 MC
61, 67	PT_2-1
11	PT_2-2
60	PT_2-5
9	PT_2-6

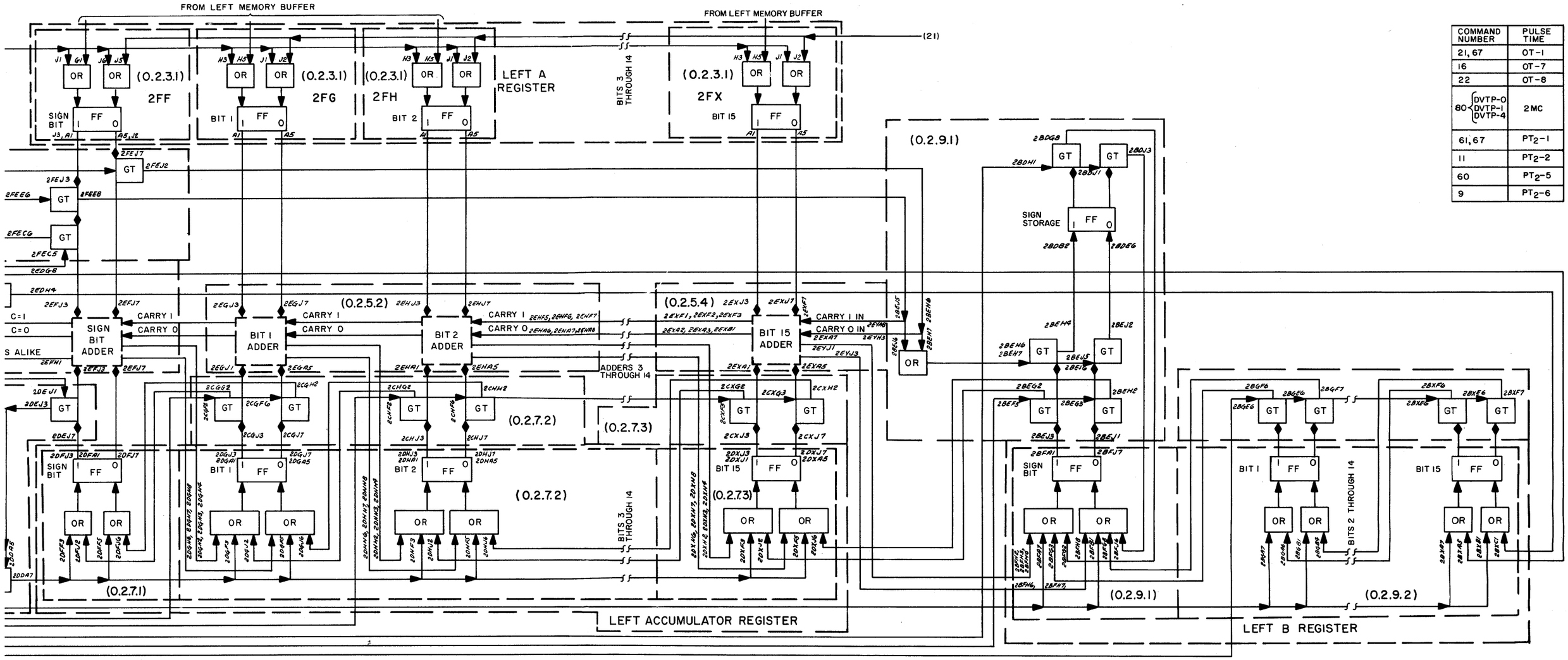


Figure 3-19. **Divide (DVD), Twin and Divide (TDV),** ***Logical Block Diagram***

DVTP-4. DVTP-0, DVTP-1, and DVTP-4 are sent to both arithmetic elements. DVTP-3 is sent to the step counter. DVTP-2 is not issued as a command pulse but does step the divide time pulse distributor. Command 80 is issued 80 times during the divide operation and, each time it is issued, it steps the divide time pulse distributor, thus generating the sequence of divide command pulses just defined.

DVTP-0 performs two functions: It shifts the combined accumulator-B register one place to the left, effectively doubling the dividend, and makes the signs of the divisor and dividend unlike. (Refer to 2.7 of this Part for justification of these and the suceeding steps.) The gate tube to which this first pulse of the basic divide cycle is applied is only conditioned if the two operands are alike in sign. (The circuit for conditioning this gate tube is contained in the sign bit adder and is not shown on figure 3–19. If the gate tube is conditioned, DVTP-0 propagates a pulse which complements the A register, reversing its sign indication. DVTP-1 is issued next and examines the sign bit flip-flop of the A register. If it is negative, the gate tube associated with its 1 side will be conditioned and the pulse generated by this tube activates the carry 1 line into the adder circuits. The end result will be that the 2's complement of the divisor will be added to the dividend since the A register holds, in this case, the 1's complement of the divisor. (It will be remembered that the 1's complement is converted to the 2's complement simply by adding a 1 to it.) If the sign bit flip-flop of the A register contains a 0, DVTP-1 will pass through the gate tube associated with the 0 side of the sign bit flip-flop, and pulse the carry 0 line of the adder circuits. This will add the positive form of the divisor to the dividend held in the accumulator-B register. DVTP-1 performs the same function for the right arithmetic element. DVTP-3 is not used operationally in the arithmetic element but is used to reduce the step counter indication by one, recording the fact that one divide cycle has been completed. Note that the time delay due to the unused DVTP-2 pulse allows needed time to propagate the add command (*carry* 0 or *carry* 1) through the adder. DVTP-4 now pulses those circuits which will correct the inherent shift right caused by the addition process initiated by DVTP-1. The divide cycle is now completed and is iterated 16 times, starting each time with DVTP-0. After the 16th cycle, the step counter will have been reduced to 1, indicating that all bits of the divisor have been taken into account (Refer to 3.5 of this Part.) The quotient will appear in the B register and the remainder in the accumulator register. The digit in the accumulator sign bit flip-flop determines the sign of both quantities.

The time pulse distributor was restarted as the step counter went from 2 to 1 and instruction pulse commands are now issued in the sequence required to complete the *Divide* instruction. The first command to affect the arithmetic element is command 67, which complements the divide connect flip-flop, returning it to its 0 status. This effectively disconnects the B register bit 15 flip-flop inputs from the sign bit adder carry outputs. The carry storage flip-flop is also cleared by this command. As explained in the section on arithmetic processes, provision must be made for the case where the final trial subtraction has resulted in a negative final current remainder because the divisor was greater in magnitude than the previous current remainder. Concurrently with command 67, therefore, command 61 is issued, which makes the sign of the divisor unlike that of the remainder held in the accumulator register at this point. The actual correction is brought about by command 11, which, as can be seen from the figure, examines the 0 side of the A register sign bit flip-flop. If this flip-flop contains a 0, the A register is added to the accumulator register by pulsing the carry 0 line of the adders as with command DVTP-1. The addition cancels the effect of the last trial subtraction. If the sign of the final current remainder as indicated by the accumulator sign bit had been positive, command 11 would not have altered the final remainder. Command 60, which is issued next, is used to correct the inherent shift right resulting from the correction of the final remainder. If no correction was necessary, the carry storage flip-flop will not be in the 1 status and the gate tube to which command 60 is applied will not be conditioned. Immediately after this *conditional shift left* command has been issued, command 9 is generated and sent to the arithmetic element. This command corrects the sign of the final answer in accordance with the contents of the sign control flip-flop. If this flip-flop contains a 1, the answer is negative; if it contains a 0, the answer is positive. Since the divide process is made to operate on two positive numbers, the sign which will result from the process itself will always be positive. Therefore, the *correct sign* command need only change the indication of the accumulator sign bit flip-flop if the final answer is negative (one of the original operands negative). Command 9 only looks at the 1 side of the sign control flip-flop. If the gate tube

associated with this side is conditioned, a pulse will be propagated which will complement the combined accumulator-B register.

To illustrate the previous discussion, assume that all the registers are 3-bit registers and that 6/16 (0.01100) is to be divided by 1/2 (0.10). In the ensuing problem, an X in a bit position will indicate that this bit position has been temporarily vacated by meaningful information, and underlined bits will indicate that they are magnitude bits of the quotient. The small boxes, one next to the accumulator sign bit and the other above the B register sign bit, represent the carry-out pulse resulting from a trial subtraction and the storage information from the B register sign during an addition process, respectively. These boxes will only be shown containing information when that information is pertinent to the development of the problem. The numbers will be arranged and operated upon as indicated in table 3–16.

TABLE 3–16. OPERATIONAL SEQUENCE FOR THE DIVIDE PROCESS

A REGISTER	COMBINED ACCUMULATOR-B REGISTER	OPERATION	COMMAND NUMBER	REMARKS
(Divisor)0.1 0	(Dividend) 0.0 1 1 0 0			Initial configuration of bits
1.0 1	0.1 1 0 0 X	Entire dividend shifted left and A register complemented to make divisor sign unlike dividend sign	DVTP-0	Dividend now doubled, and its sign bit replaced by contents of 1 bit flip-flop
	1.1 0 0 1 X.0 0 1 0 X	Two's complement of divisor added to dividend by pulsing carry 1 line of adders	DVTP-1	The addition process results in inherent shift right of accumuator register contents and also results in a carry-out of 1 from the sign bit adder
	0 X.0 0 1 0 1	Carry-out pulse of 1 from sign bit adder placed in least significant bit of B register	Automatic	Divide connect flip-flop is in the 1 status as set at beginning of divide instruction
	(Current Remainder) 0.0 1 0 0 1	Contents of accumulator register and B register sign bit shifted left and B register sign storage returned to B register sign	DVTP-4	Inherent shift right resulting from DVTP-1 now corrected
1.0 1	0.1 0 0 1 X	Contents of combined accumulator-B register shifted left. A register not complemented	DVTP-0	Correct remainder doubled. Signs of divisor and current remainder already unlike
	1.1 0 0 1 X.0 0 0 1 X	Two's complement of divisor added to current remainder by pulsing carry 1 line of adders	DVTP-1	An inherent shift right occurs again and the sign bit adder has a carry-out of 1
	0 X.0 0 0 1 1	Carry-out pulse of 1 from sign-bit adder placed in least significant bit of B register	Automatic	The second quotient bit now determined

TABLE 3-16. OPERATIONAL SEQUENCE FOR THE DIVIDE PROCESS (cont'd)

A REGISTER	COMBINED ACCUMULATOR-B REGISTER	OPERATION	COMMAND NUMBER	REMARKS
	0.0 0 0 1 1	Contents of accumulator register and B register sign bit shifted left and B register sign storage returned to B register sign	DVTP-4	Inherent shift right corrected
	0.0 0 1 1 X	Contents of combined accumulator-B register shifted left. A register not complemented	DVTP-0	Current remainder doubled. Signs of current remainder and A register already unlike
	1.1 0 1 0 X.1 1 0 1 X	Two's complement of divisor added to current remainder by pulsing carry 1 line of adders	DVTP-1	An inherent shift right occurs. The sign bit adder has a carry-out of 0
	1 X.1 1 0 1 0	Carry-out of 0 from sign bit adder placed in least significant bit of B register	Automatic	The final bit of the quotient has been determined
	1.1 0 1 1 0	Correctional shift	DVTP-4	Inherent shift now corrected. Accumulator sign bit contains a 1 indicating that remainder must be corrected
0.1 0		Accumulator and A register signs made unlike	61	In preparation for correcting remainder
		Divide connect flip-flop complemented to 0	67	The carry output of the sign bit adder is no longer connected to the least significant bit of the B register. Carry storage flip-flop cleared
	0.1 0 1 1 0.0 0 0 1 0	Positive form of divisor added to final remainder now located in accumulator register	11	Previous subtraction of divisor nullified. An inherent shift right occurs. The carry-out of one sets the carry storage flip-flop
	0.0 0 1 1 0	Correctional shift left performed (conditional on an add process having been promulgated by command 11)	60	The inherent shift right is corrected and the magnitude bits of the quotient now completely fill the B register. The sign of both quotient and remainder is in the accumulator sign bit flip-flop

TABLE 3–16. OPERATIONAL SEQUENCE FOR THE DIVIDE PROCESS (cont'd)

A REGISTER	COMBINED ACCUMULATOR-B REGISTER	OPERATION	COMMAND NUMBER	REMARKS
	0.0 0 1 1 0	The sign control flip-flop is examined	9	Since the sign of the quotient as indicated by the sign control flip-flop already agrees with the sign in the accumulator register sign B bit, command 9 has no effect

4.5 SHIFT CLASS

The eight instructions comprising the shift class utilize only the accumulator register and B register in the arithmetic element. Provision is made in these registers for shifting information between adjacent bits either to the left or to the right. The left and right shifts in the B register, and the left shift in the accumulator register, are broadside shifts (refer to 3.5 of this Part); the right shift in the accumulator register, however, employs a ripple shift process. The instructions in the shift class can be separated into three distinct categories and will be treated accordingly.

4.5.1 *Cycles Left (DCL), Cycle Accumulators Left (FCL)*

The *DCL* and *FCL* instructions, which comprise the first category, arrange the registers into 32-bit shifting rings with connections established so that the contents of each digit position may be shifted to the adjacent digit position. The 32-bit ring is formed by the left and right accumulator registers for the *FCL* instruction and by an accumulator-B register combination for the *DCL* instruction. These instructions are termed cycle instructions, and their major characteristic is that information in all the bits, including the sign bit, can be shifted continuously around the ring without losing any data.

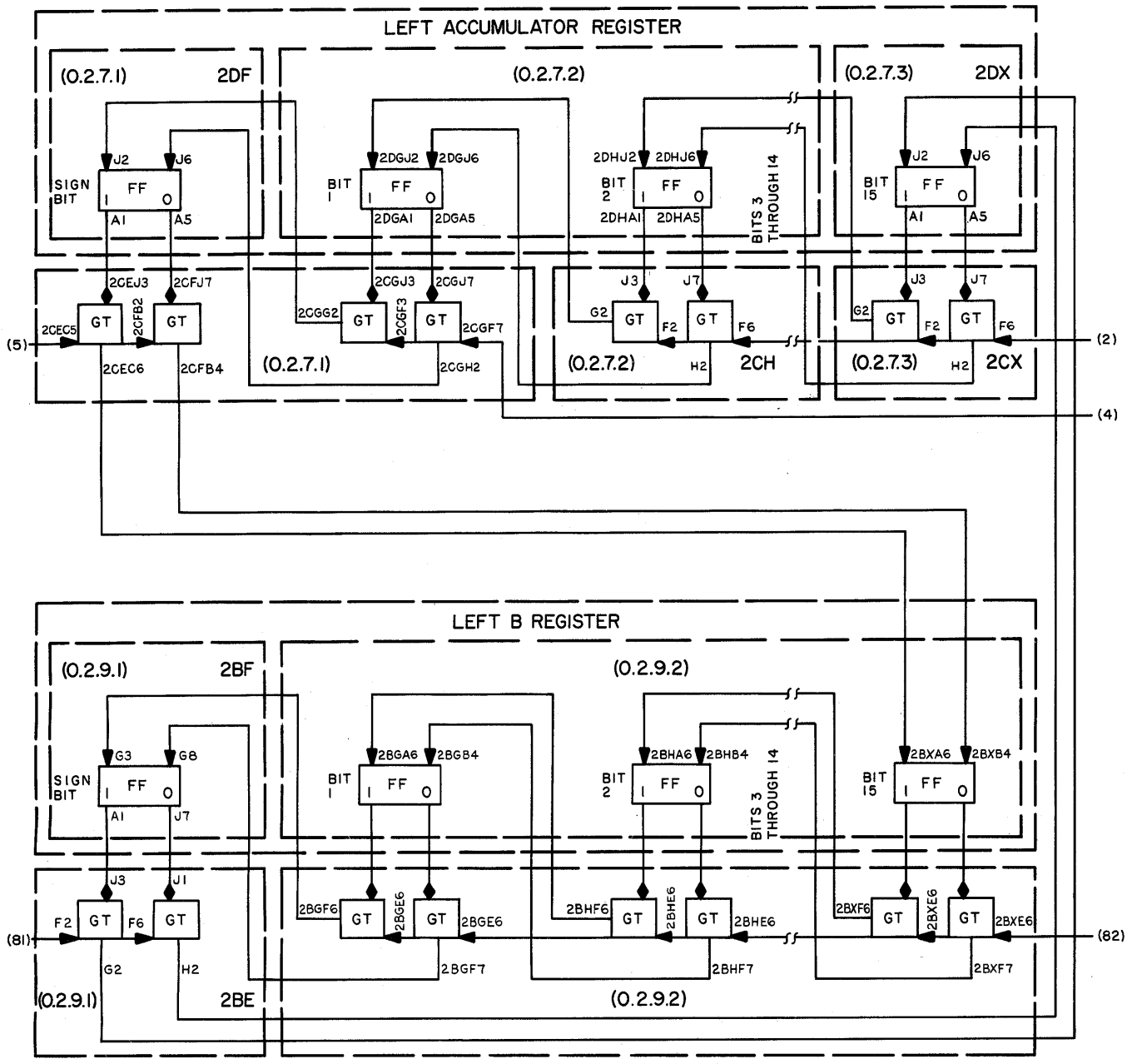
The circuits in the left arithmetic element used to execute the *DCL* instruction appear in figure 3–20. A parallel operation occurs simultaneously in the right element. The shifts are initiated by commands 2, 4, 5, 81, and 82. All of these commands are issued simultaneously during a single 2-megacycle operation. If, for example, 14 shifts are required, then fourteen 2-megacycle operations are utilized. Command 2 shifts accumulator register bits 2 through 15 one digit position to the left; command 4 shifts the first bit of the accumulator register into the sign bit; command 5 shifts the accumulator register sign bit into the 15th bit of the B register; command 82 shifts B register bits 1 through 15 one digit position to the left; command 81 shifts the B register sign bit into the 15th bit of the accumulator register.

The circuits in the arithmetic elements used to execute the *FCL* instruction appear in figure 3–21. Note that the accumulator registers in both the left and right arithmetic elements are interconnected for this instruction. The shifts are initiated by commands 2, 4, 12, 202, 204, and 217. As in the case of the *DCL* instruction, these six commands are issued simultaneously during a single 2-megacycle operation. Command 2 shifts left accumulator register bits 2 through 15 one digit position to the left; command 4 shifts the first bit of the accumulator register into the sign bit; and command 12 shifts the left accumulator register sign bit into the 15th bit of the right accumulator register. Similar shifts in the right accumulator register are performed by commands 202, 204, and 217.

4.5.2 *Shift Left (DSL), Left Element Shift Right (LSR), Right Element Shift Right (RSR)*

The *DSL, LSR,* and *RSR* instructions, which comprise the second category, form the registers into 31-bit shifting registers. In the *DSL* instruction, the information in the most significant digit position (bit 1) is lost with every shift. In the *LSR* and *RSR* instructions, the information in the least significant digit position (bit 15 of the B register) is lost with every shift. The sign bit of the 31-bit register is not changed by the execution of any instruction in this category.

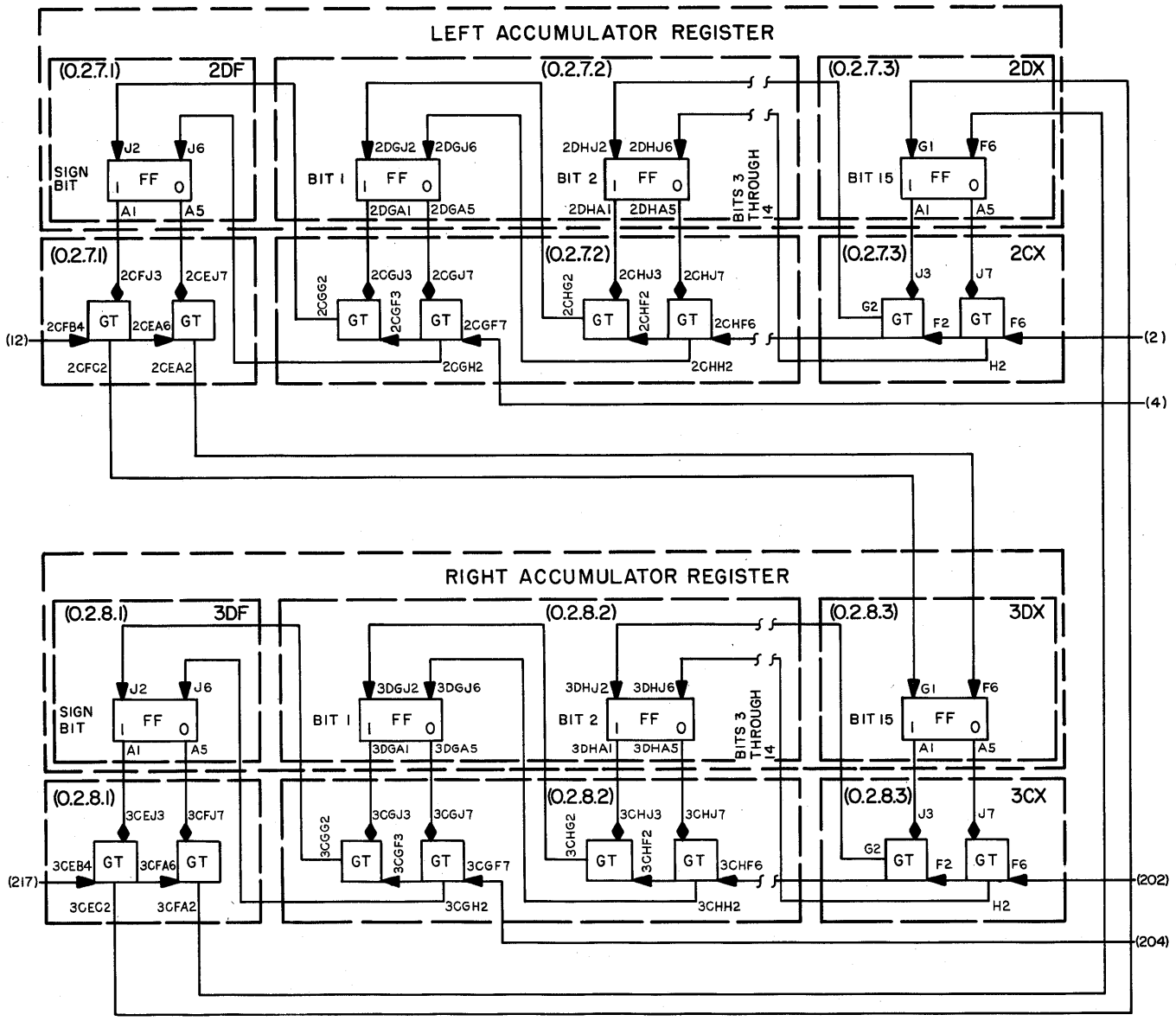
The circuits in the left arithmetic element used to execute the *DSL* instruction are shown in figure 3–22. A similar operation occurs simultaneously in the right element. The shifts are initiated by commands 2, 5, 82, and 81. These four commands are issued simultaneously during every 2-megacycle operation. Command 2 shifts accumula-



COMMAND NUMBER	PULSE TIME
2, 4, 5, 81, 82	2 MC

Figure 3–20. Cycle Left (DCL), Logical Block Diagram

tor register bits 2 through 15 one digit position to the left; command 5 shifts the accumulator register sign bit into the 15th bit of the B register; command 82 shifts B register bits 2 through 15 one digit position to the left; and command 81 shifts the B register sign bit into the 15th bit of the accumulator register. The information copied into the B register 15-bit from the accumulator register sign bit by command 5 is successively displaced one place to the left with each shift. If, for example, the sign bit of the accumulator register were a 1, and 16 shifts were called for, then each

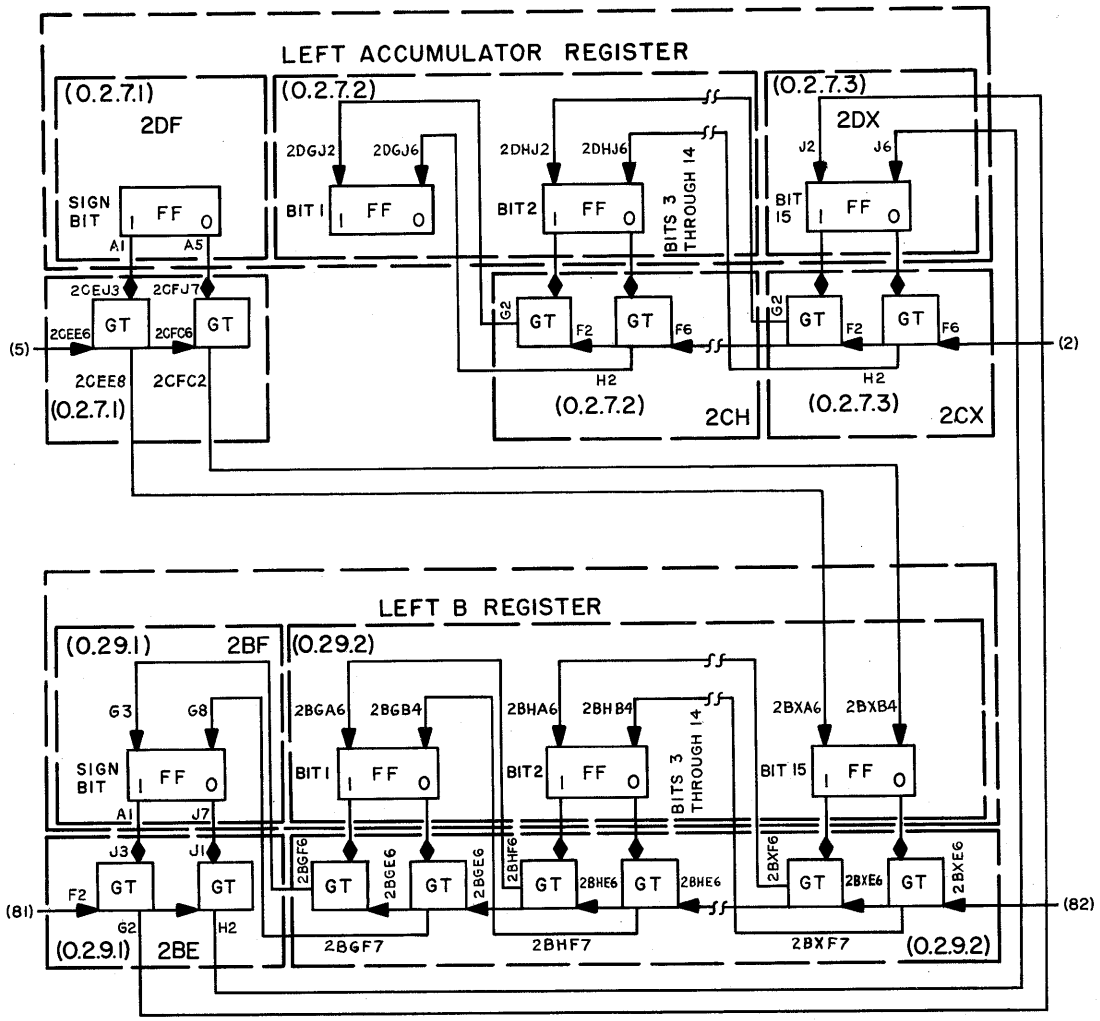


COMMAND NUMBER	PULSE TIME
2,4,12,202, 204,217	2MC

Figure 3–21. Cycle Accumulator Left (FCL), Logical Block Diagram

of the 16 bits of the B register would contain a 1 at the end of the instruction.

The circuits in the left arithmetic element used to execute the *LSR* instruction appear in figure 3–23. This instruction does not affect the right arithmetic element. The shifts are initiated by commands 85, 7, and 18, these three commands occurring during each 2-mc operation. Command 85 initiates a broadside shift in the B register, shifting the contents of the register (including the sign) one digit position to the right. The left accumulator register 15-bit is transferred to the sign bit of the left B register by command 7. The shift right for the remaining bits of the accumulator register is accomplished by means of a ripple shift process initiated by command 18. This command is applied to a pair of gate tubes conditioned by the bit 14 flip-flop in the accumulator register. If bit 14 is a 1, the gate tube conditioned by the 1 side passes a pulse which performs two functions: it sets the flip-flop of bit 15 to 1, and it passes through the OR circuit associated with bit 14, thus providing the shifting pulse for bit 13. The OR circuit enables a shifting pulse to be applied to



COMMAND NUMBER	PULSE TIME
2,5,81,82	2 MC

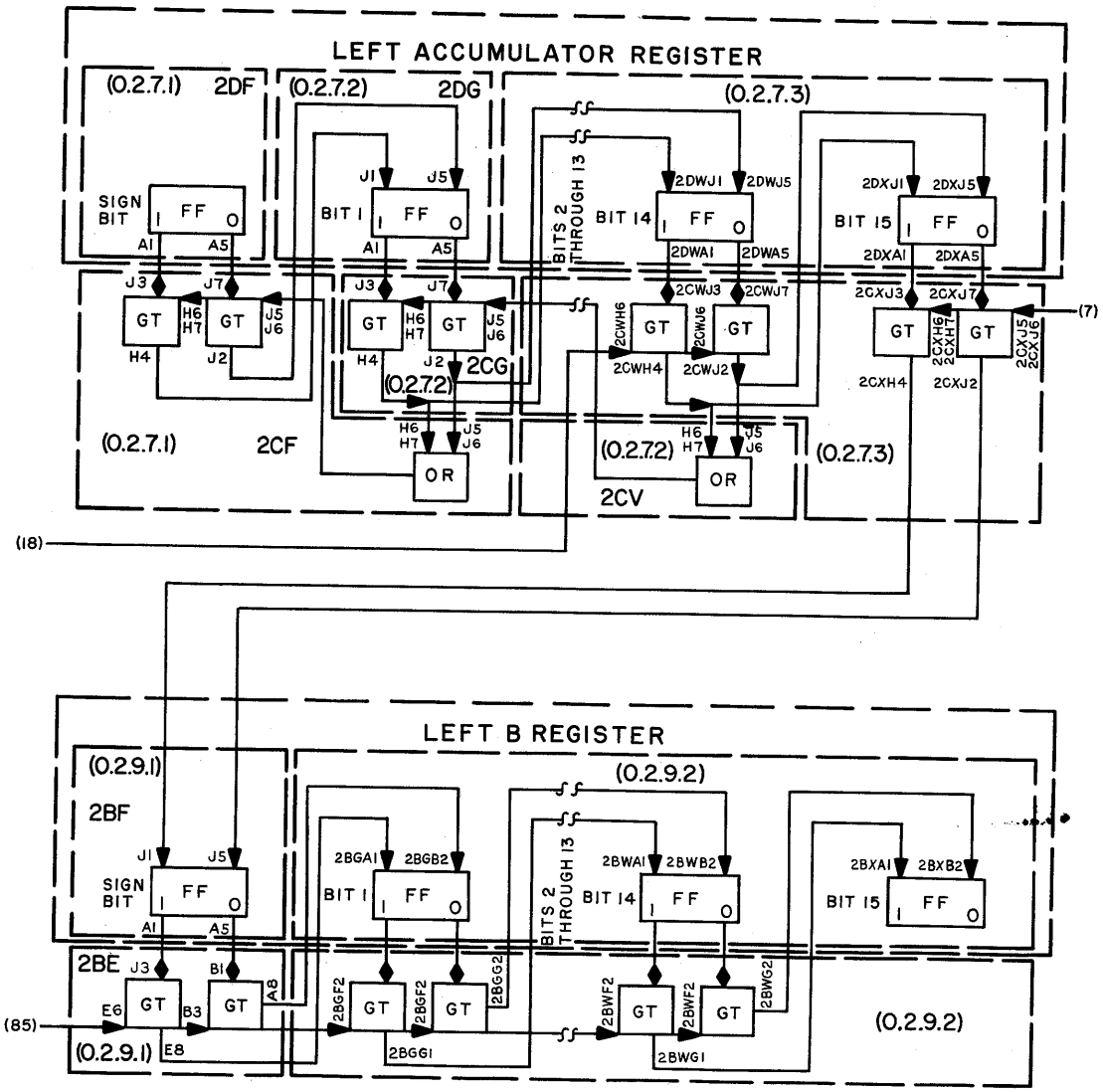
Figure 3–22. Shift Left (DSL), Logical Block Diagram

the next highest order bit, whether the adjacent flip-flop (previously operated upon) is in the 1 or 0 status. In the manner just described, the shift ripples through the entire accumulator register bit by bit. The sign flip-flop is unaffected by the shift, but the content of this flip-flop is copied into the 1 bit during each shift. If the sign bit were a 1, and 15 shifts were required, then each flip-flop of the left accumulator register would contain a 1 at the end of the instruction.

The *RSR* instruction is similar to the *LSR* instruction except that the shift right is executed in the right accumulator-right B register combination. The shift in the right B register is a broadside shift initiated by commands 285 and 207. These commands perform the corresponding functions in the *RSR* instruction that were performed by commands 85 and 7 in the *LSR* instruction. The ripple shift for all but the 15th bit in the right accumulator register is initiated by command 123, corresponding to command 18 in the *LSR* instruction.

4.5.3 *Shift Accumulators Left (ASL), Shift Accumulators Right (ASR)*

The *ASL* and *ASR* instructions, which comprise the third and final category of instructions in the shift class, arrange the accumulator registers into 16-bit shifting registers. The sign bits of the accumulator registers are not changed by the execution of either instruction.



COMMAND NUMBER	PULSE TIME
85,7,18 (LSR)	2 MC

Figure 3–23. **Left Element Shift Right (SLR),** ***Logical Block Diagram***

The circuits in the left arithmetic element used to execute the *ASL* instruction are shown in figure 3–24. A parallel operation occurs simultaneously in the right element. The contents of the most significant digit position (bit 1) is lost with every shift during this instruction. Though the content of the sign bit flip-flop is not affected, this information is copied into the 15th bit during each shift. The information would fill all bits but the sign bit. If 15 or more shifts are called for, the shifts are initiated by commands 2 and 6, both of which occur during each 2-megacycle operation. Command 6 copies the sign bit into the 15th bit. The transfer of information of bits 2 through 15 one digit position to the left is accomplished by a broadside shift initiated by command 2.

The circuits in the left arithmetic element used to execute the *ASR* instruction appear in figure 3–25. A similar operation occurs simultaneously in the right element. As is the case for all right shifts in the accumulator registers, a ripple shift process is employed and is initiated in the accumulator register by command 18. The content of the sign bit flip-flop is copied into the most significant digit position (bit 1) for each shift, and the contents of the least significant digit position (bit 15) is lost with each shift.

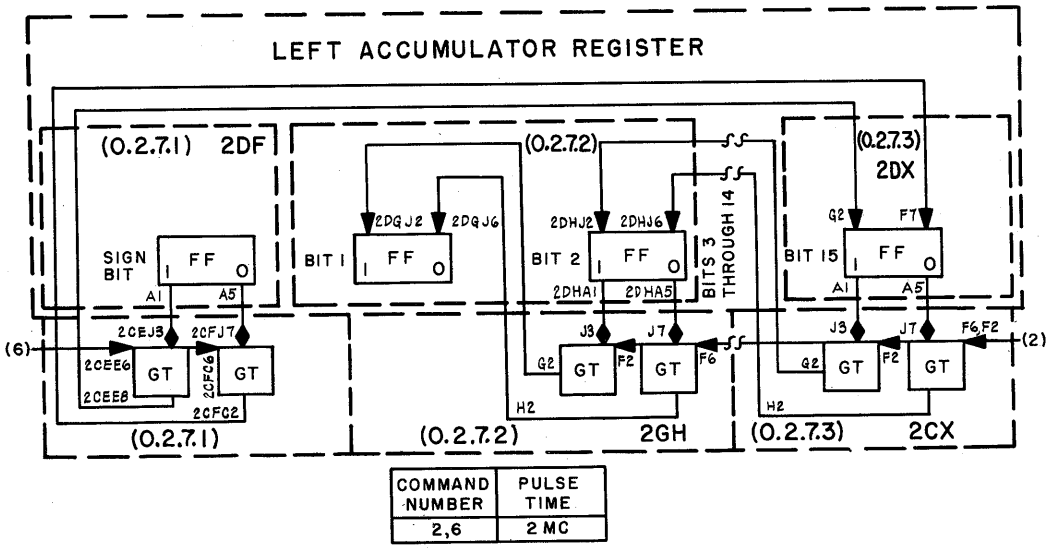


Figure 3–24. **Shift Accumulators Left (ASL), *Logical Block Diagram***

4.6 STORE CLASS

Each of the seven instructions comprising the store class utilizes registers in the arithmetic elements. These are *Store (FST), Left Store (LST), Right Store (RST), Store Address (STA), Exchange (ECH), Add One (AOR),* and *Deposit (DEP).* The store class instructions effect the transfer of information from the Central Computer System to magnetic core memory. In some cases data resulting from internal computer operations are stored; in other cases data brought out of magnetic core memory are modified in a specific way and then stored. The discussion will be carried only as far as the memory buffer registers. The subsequent transfer of data from these buffers to magnetic core memory is treated in the section on magnetic core memory.

4.6.1 *Store (FST), Left Store (LST), Right Store (RST), Store Address (STA)*

The *FST, LST, RST,* and *STA* instructions are similar in that they all are concerned with the transfer of data resulting from computer operations into the memory buffer registers. For each of these instructions, the actual data transfer is preceded by clearing operations on the memory buffer registers. The left memory buffer is cleared by command 41, while the right memory buffer register is cleared by command 53. At the end of each of these instructions, the data in the memory buffer registers are transferred to magnetic core memory, again clearing the memory buffer registers in preparation for the next instruction.

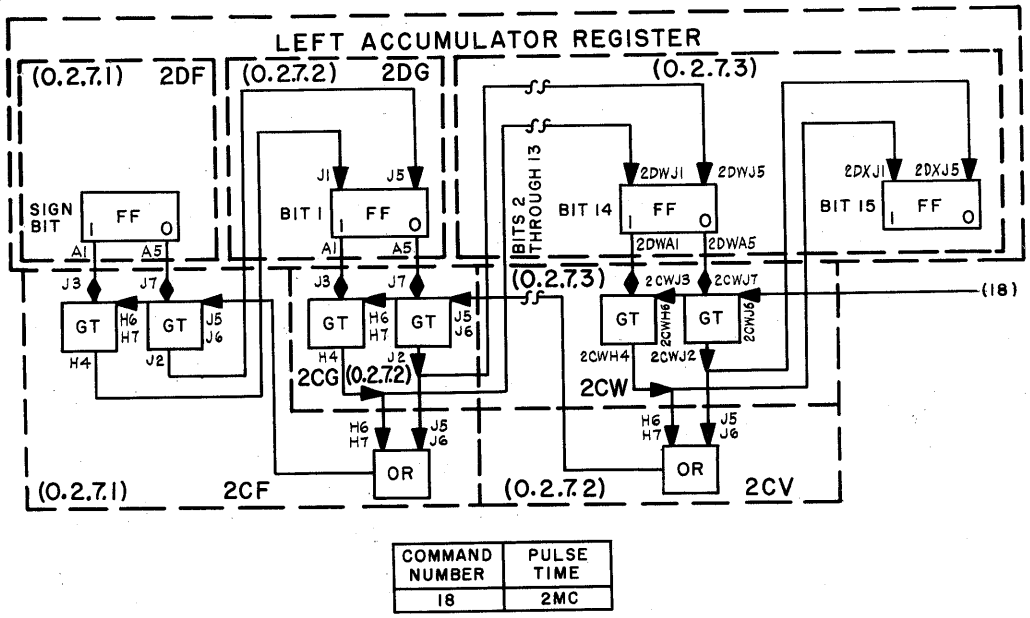
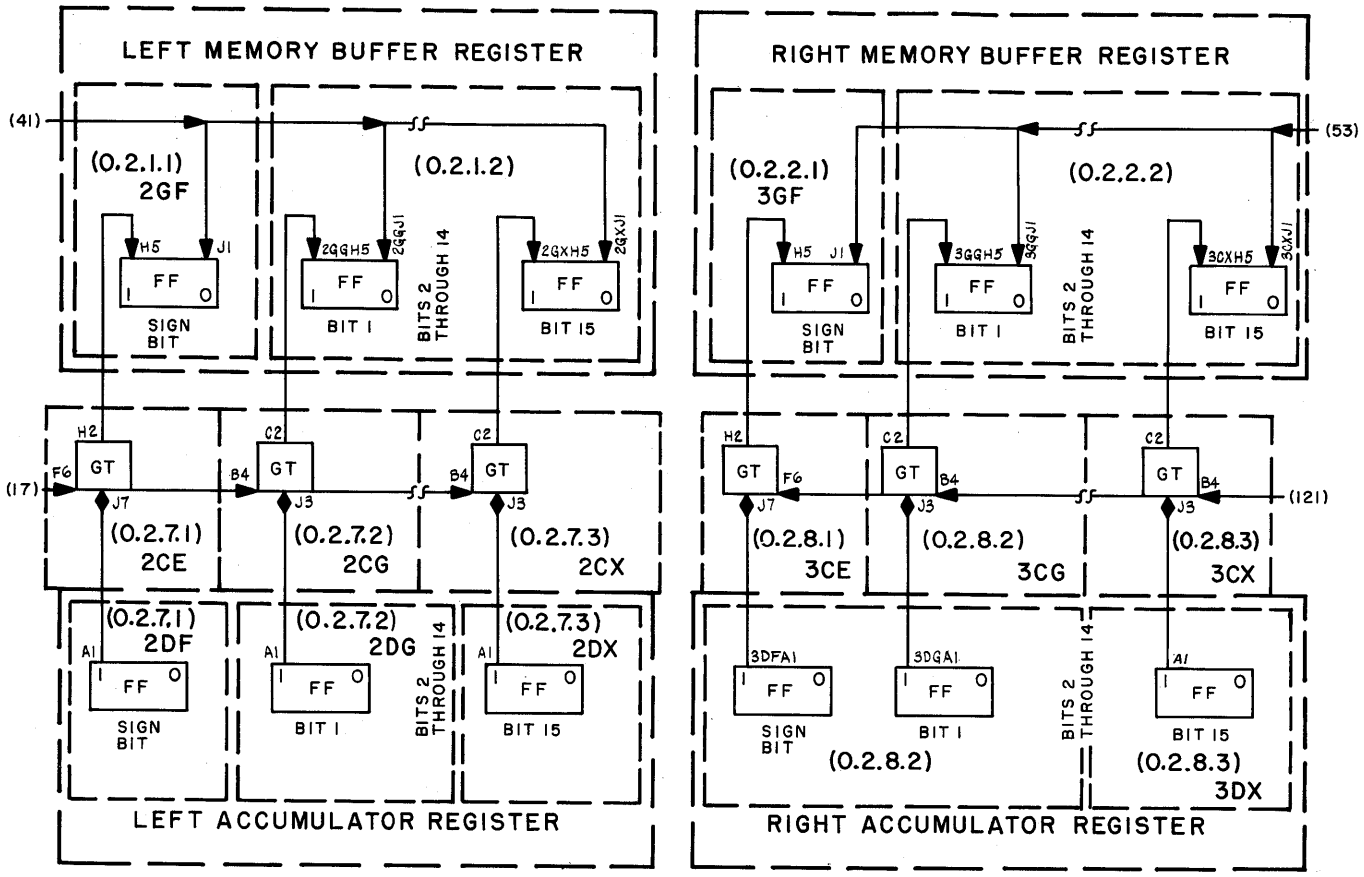


Figure 3–25. **Shift Accumulators Right (ASR), *Logical Block Diagram***



COMMAND NUMBER	PULSE TIME
41,53	OTB-1
17,121	OTB-2
41,53	PT2-1

Figure 3–26. Store (FST), Logical Block Diagram

The circuits in the arithmetic element used to execute the *FST* instruction are shown in figure 3–26. This instruction transfers the contents of the left and right accumulator registers into the left and right memory buffer registers, respectively. Following the clearing of both memory buffer registers by commands 41 and 53, the contents of the left accumulator register are transferred to the left memory buffer register by command 17. The transfer of data from the right accumulator register to the right memory buffer register is initiated by command 121.

Figure 3–27 illustrates the circuits in the arithmetic element used to execute the *LST* instruction. This instruction replaces the contents of the left memory buffer register with the contents of the left accumulator register. The contents of the right memory buffer register are not affected by this instruction. Following the clearing of both memory buffer registers by commands 41 and 53, new information from core memory is placed in these registers during the read portion of the subsequent memory cycle. Command 230, which is issued next, clears the right A register and is followed by command 51, which transfers the data in the right memory buffer registers into the right A register. The right A register now acts as a temporary storage location for the data in the right memory buffer register. Both memory buffer registers are again cleared by commands 41 and 53. Commands 17 and 228 are then issued and perform the following functions: command 17 transfers the data in the left accumulator register into the left memory buffer register; command 228 transfers the data temporarily stored in the right A register back into the right memory buffer register.

COMMAND NUMBER	PULSE TIME
41, 53	OTA-1
230	OTA-6
51	OTA-7
41, 53	OTB-1
17, 228	OTB-2
41, 53	PT_2-1

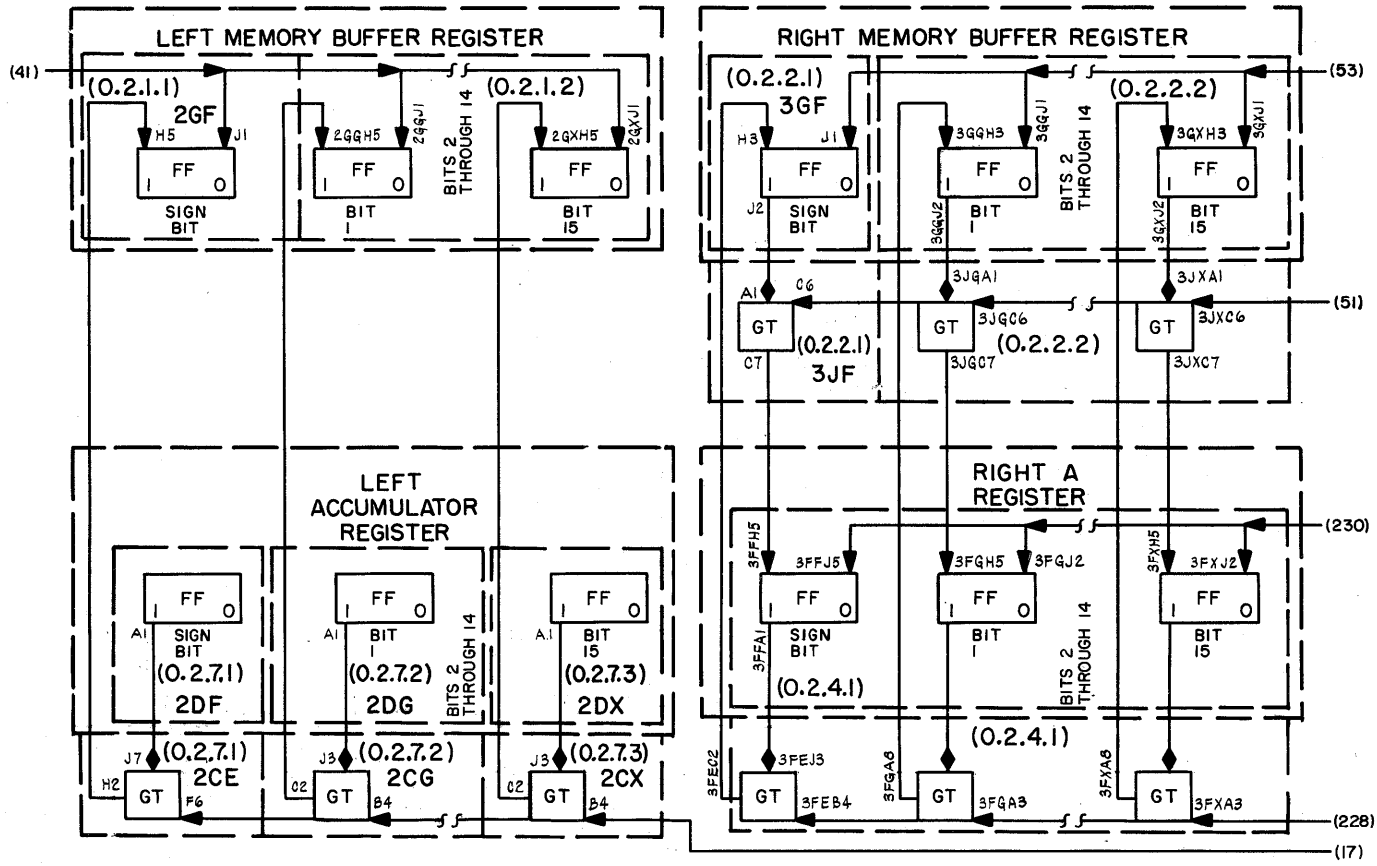


Figure 3–27. **Left Store (LST),** ***Logical Block Diagram***

The *RST* instruction, figure 3–28, is executed in a manner similar to the *LST* instruction. In this case, however, the contents of the right accumulator register are transferred to the right memory buffer register while the original contents of the left memory buffer register are preserved. The left memory buffer register contents are temporarily stored in the left A register by command 43, the left A register being previously cleared by command 21. Both memory buffer registers are then cleared by commands 41 and 53, after which commands 121 and 23 are issued. Command 121 transfers the data in the right accumulator register into the right memory buffer register; command 23 transfers the temporarily stored content of the left A register into the left memory buffer registers.

The *STA* instruction replaces the contents of the right memory buffer register with the contents of the right A register. The contents of the left memory buffer register are not changed by this instruction. The circuits in the arithmetic element used to execute this instruction appear in figure 3–29. The contents of the left memory buffer register are temporarily stored in the left A register and subsequently returned to the buffer register exactly as in the *RST* instruction previously described. The transfer of information from the right A register to the right memory buffer register is initiated by command 228.

COMMAND NUMBER	PULSE TIME
41, 53	OTA-1
21	OTA-6
43	OTA-7
41, 53	OTB-1
23, 121	OTB-2
41, 53	PT_2-1

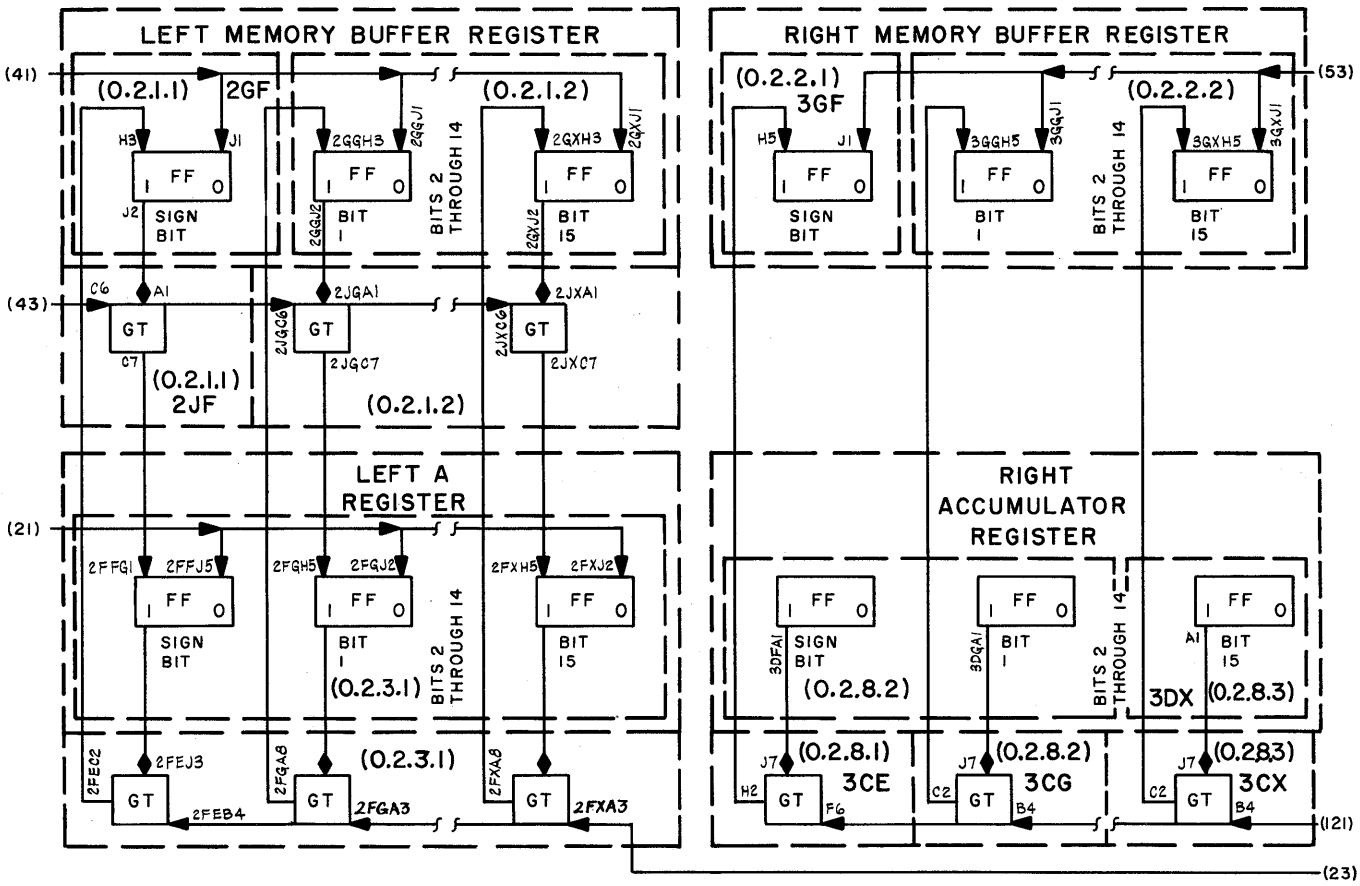


Figure 3–28. **Right Store (RST), *Logical Block Diagram***

4.6.2 *Exchange (ECH)*

The *ECH* instruction causes an interchange of information between the accumulator registers and the memory buffer registers. The circuits in the left arithmetic element used for the execution of this instruction are shown in figure 3–30. A parallel operation occurs simultaneously in the right element. The first step in preparing the left arithmetic element for this instruction is to clear the memory buffer and A registers by commands 41 and 21, respectively. Prior to issuing the next command, the memory buffer register, during the read portion of the accompanying memory cycle, has been loaded with the contents of the specified memory address. Command 43 then transfers this data to the A register. The memory buffer register is then cleared and thus made ready to accept the contents of the accumulator register, this transfer being intiated by command 17. At this point, the information to be transferred into magnetic core memory (the original contents of the accumulator register) is properly positioned for this transfer. The transfer occurs during the write portion of this same memory cycle. The original contents of the specified memory address, which is now in the A register, must be transferred into the accumulator register. Since the Central Computer System does not provide a direct transfer path between

COMMAND NUMBER	PULSE TIME
41, 53	OTA-1
21	OTA-6
43	OTA-7
41, 53	OTB-1
23, 228	OTB-2
41, 53	PT_2-1

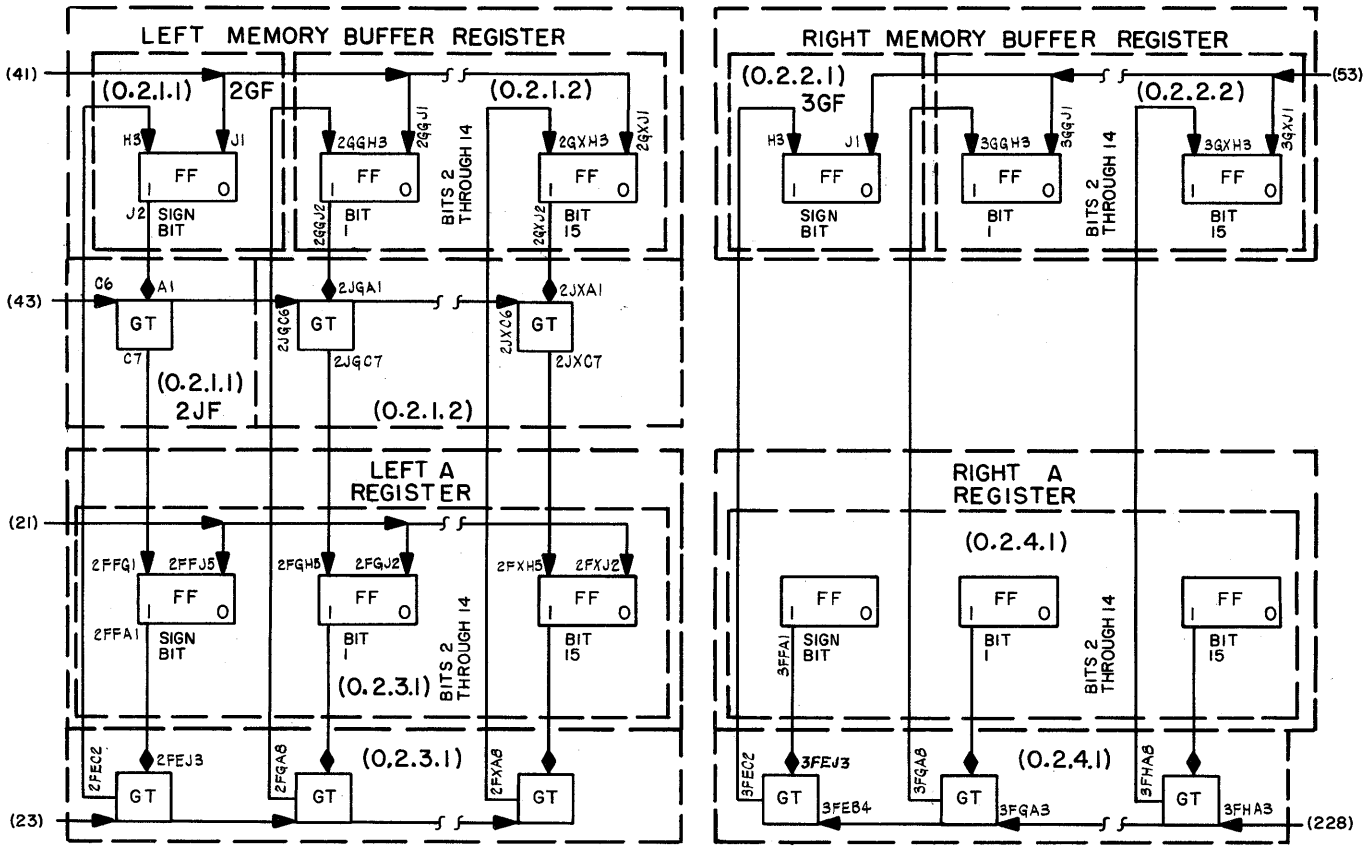


Figure 3–29. Store Address (STA), Logical Block Diagram

the A register and the accumulator register, this transfer is effected by an add process. Accordingly, the accumulator is cleared by command 10, and command 64 initiates the add process by pulsing the carry 0 line of the adder circuits. The inherent shift right, occurring in every add process, must be corrected before the contents of the original memory address appear correctly positioned in the accumulator register. The correction is accomplished by commands 2, 4, 81, and 89. As a final step, the memory buffer register, having already delivered its contents to magnetic core memory, is now cleared by command 41 in preparation for the next instruction.

4.6.3 *Add one (AOR)*

The *AOR* instruction adds 1 to the right half-word in magnetic core memory. The left half-word is not affected by this instruction. These latter data are temporarily stored in the left A register and returned to the left memory buffer register at the end of the instruction in the same manner as the *RST* instruction previously discussed. The circuits in the right arithmetic element used to execute the *AOR* instruction appear in figure 3–31. The right memory buffer register is prepared for this instruction by command 53, which clears that register. The right half-word in magnetic core memory is then placed in the right memory buffer

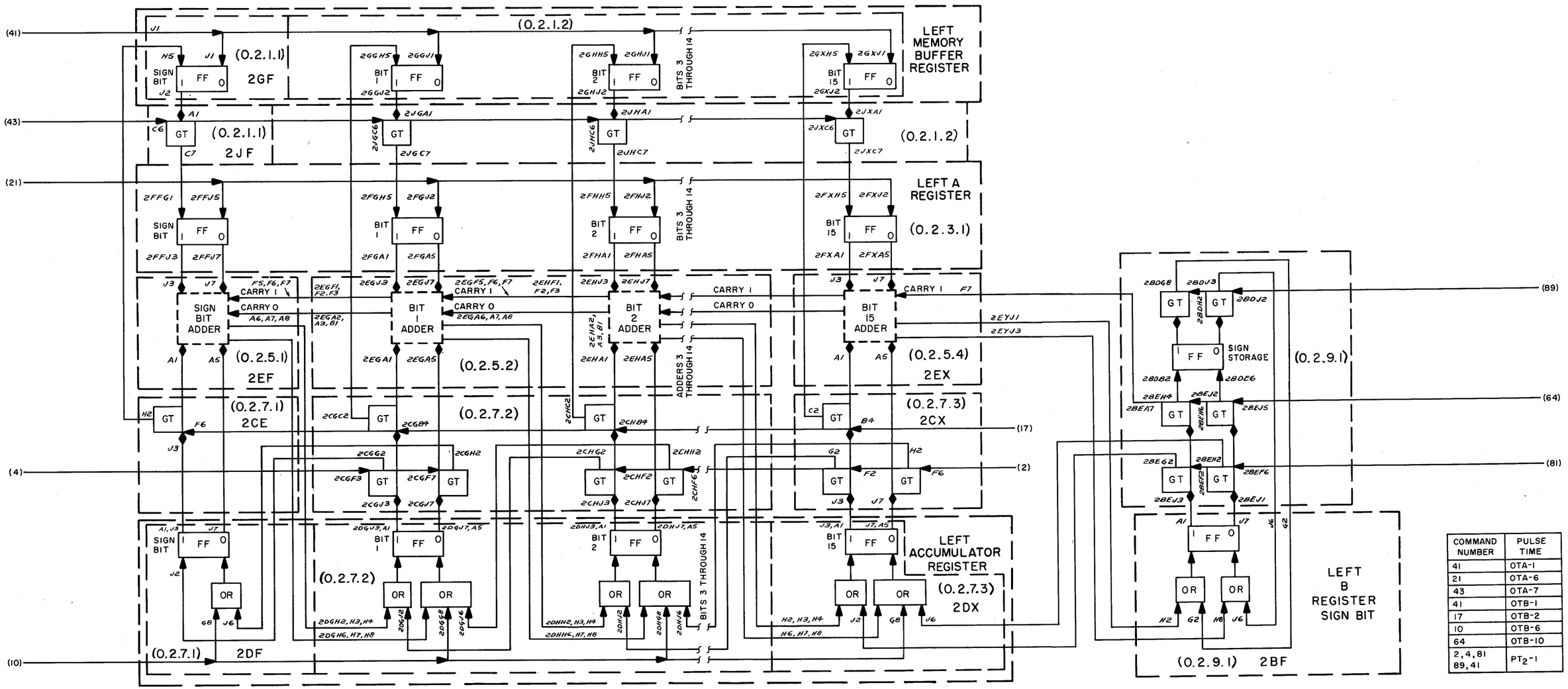
register during the read portion of the subsequent memory cycle. Commands 230 and 210 are issued next, and clear the right A and right accumulator registers, respectively. The information in the right memory buffer register is then transferred to the right A register by command 51, and the computer is now prepared to add 1 to the original right half-word. Command 69 initiates the add process by pulsing the carry 1 line of the bit 15 adder. The sum appears in the accumulator register displaced one digit position to the right. This inherent shift right, occurring during every add process, is corrected by commands 202, 204, 281, and 289. Command 14, issued next, pulses the gate tube conditioned by the 1 side of the carry storage flip-flop, and sets the bit 15 flip-flop in the accumulator register if a 1 has been carried out of the sign bit adder. The only time this can occur for the *AOR* instruction is when the half-word brought into the A register from memory is negative 0; i.e., all of the bits are 1. In this special case, adding 1 to the A register contents (the operation initiated by command 69) results in positive zero appearing in the accumulator register, and in the carry storage flip-flop being set to 1. The gate tube to which command 14 is applied is thereby conditioned, pulsing the 1 side of the accumulator bit 15 flit-flop. This results in the number 1 being held by the accumulator register. Command 53, which is issued simultaneously with command 14, clears the right memory buffer register, preparing it to receive the contents of the accumulator bit register. This transfer is initiated by command 121. The transfer of information from the memory buffer register to magnetic core memory occurs during the write portion of the accompanying memory cycle, after which the buffer register is again cleared by command 53 in preparation for the next instruction.

4.6.4 *Deposit (DEP)*

The function of the *DEP* instruction is to replace the specified bits of a word in core memory with corresponding bits contained in the accumulator register. The digit positions involved in the execution of this instruction are specified by placing 1's in the corresponding bit positions of the B register. The B register is best prepared for this operation by the *Load B Registers* instruction.

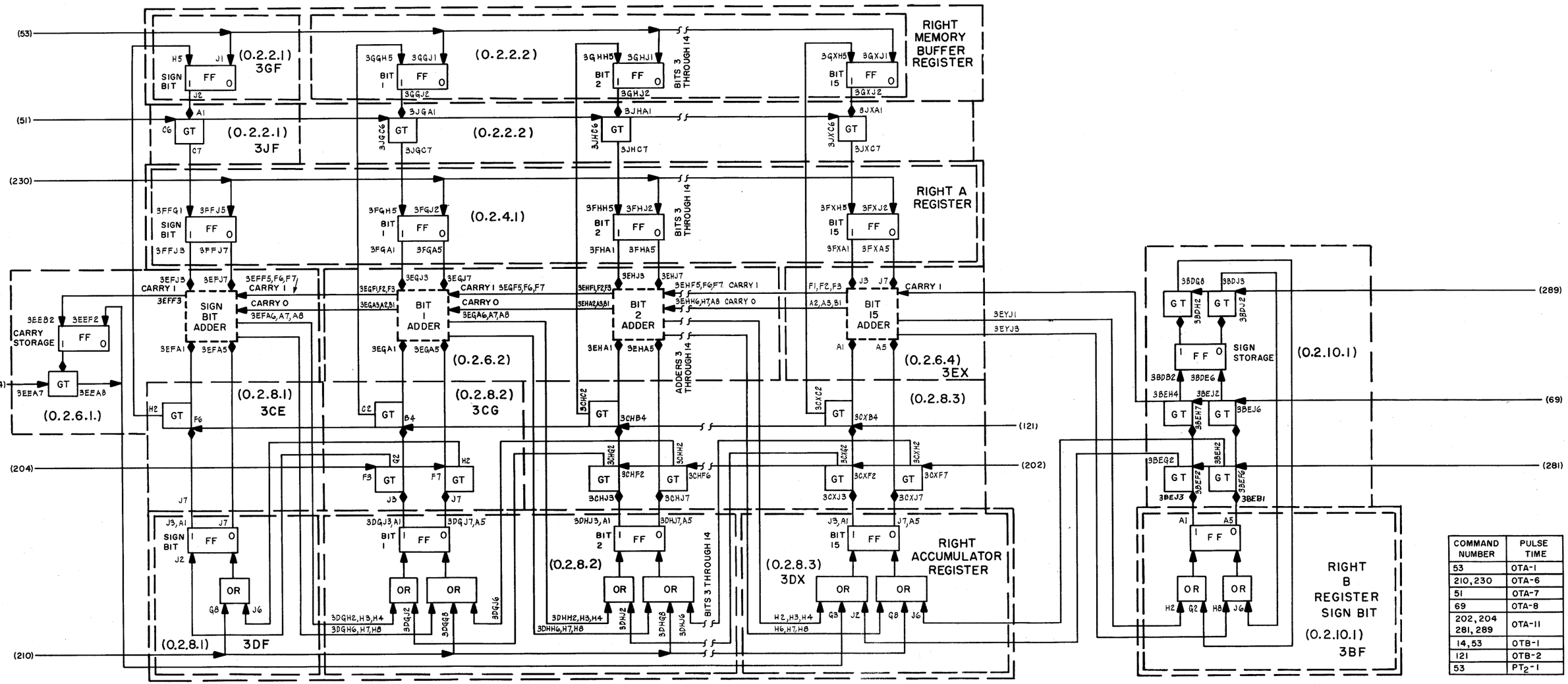
The circuits in the left arithmetic element used to execute the *DEP* instruction are shown in figure 3–32. A parallel operation occurs simultaneously in the right element. Commands 21 and 41, which are issued at the start of the instruction, clear the A register and memory buffer register, respecively. The accumulator register is then complemented by command 19. Since the bits to be changed are specified by those digit positions in the B register which contain 1's, the 1's in this register are copied into the A register by command 88. Those bits in the B register which are 0's already appear as 0's in the corresponding A register digit positions, the A register having been previously cleared. Command 25, issued next, pulses a set of gate tubes conditioned by the 0 sides of the A register flip-flops. This command is the *logical multiply* command. All the accumulator register digit positions whose corresponding A register digit positions contain 0's are cleared to 0, and all the accumulator register digit positions whose corresponding A register digit positions contain 1's are left unchanged. At this point, the accumulator register contains the complement of the required bit in each position which this instruction must alter. All other accumulator register bits are 0.

The information to be modified, now in the memory buffer register, is transferred to the A register by command 43. At the same time, the accumulator is recomplemented by command 19. The transfer of data from the memory buffer register to the A register (command 43) has certain significant aspects in that the A register, not having been cleared, still contains the information received as a consequence of command 88. Because only gate tubes conditioned by the 1 side of the flip-flops in the memory buffer register are instrumental in the transfer, there is a possibility that a 1 can exist in the A register after the transfer, even though a 0 was contained in the corresponding memory buffer register flip-flop. In this respect, this operation differs from the usual memory buffer register-A register transfer in which the A register is cleared prior to the transfer, resulting in the contents of the memory buffer register being exactly duplicated in the A register. Upon completion of command 43, therefore, the information in the A register consists of the data originally held in the specified memory address except for those digit positions which are to be changed by this instruction. Each of these bits is a 1 regardless of the original bit contained in the memory address specified. Command 25 is reissued and the A register and accumulator register are again logically multiplied. In this operation, the accumulator register contains a 1 in each of the digit positions corresponding to those in the original memory address which were to remain unchanged; the other bits of this register are the actual bits (0 or 1) which are to supplant the



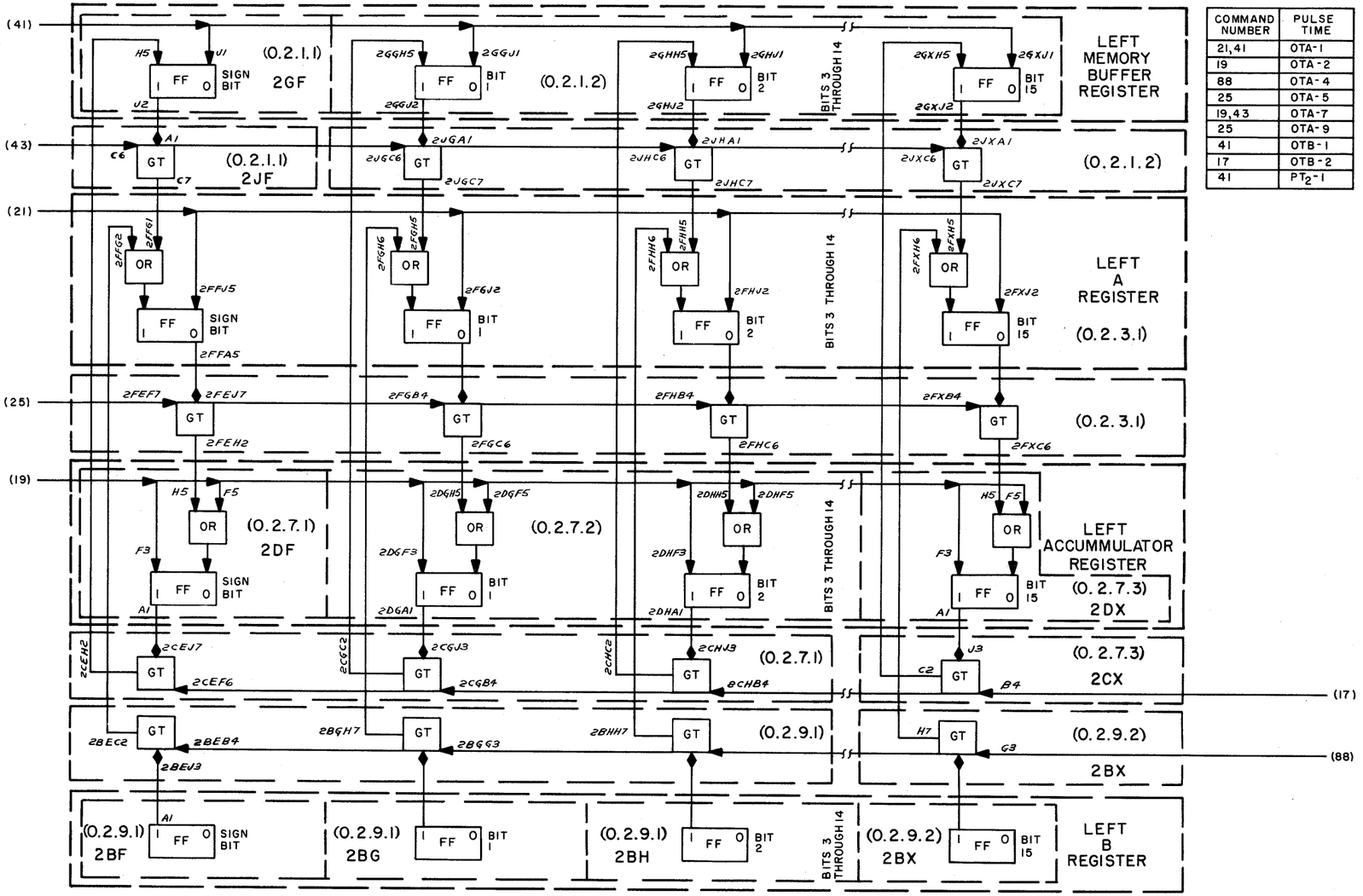
COMMAND NUMBER	PULSE TIME
41	OTA-1
21	OTA-6
43	OTA-7
41	OTB-1
17	OTB-2
10	OTB-6
64	OTB-10
2,4,81 89,41	PT_{2}-1

Figure 3-30. **Exchange (ECH),** *Logical Block Diagram*



COMMAND NUMBER	PULSE TIME
53	OTA-1
210, 230	OTA-6
51	OTA-7
69	OTA-8
202, 204 281, 289	OTA-11
14, 53	OTB-1
121	OTB-2
53	PT_2-1

Figure 3–31. **Right Add One (AOR),** ***Logical Block Diagram***



COMMAND NUMBER	PULSE TIME
21,41	OTA-1
19	OTA-2
88	OTA-4
25	OTA-5
19,43	OTA-7
25	OTA-9
41	OTB-1
17	OTB-2
41	PT_2-1

Figure 3–32. **Deposit (DEP),** ***Logical Block Diagram***

original bits of the memory address. The A register, on the other hand, holds the contents of the original memory address except in those digit positions, which, as originally specified by the B register, were to be altered. These latter digit positions contain 1's, regardless of their original contents in core memory before this instruction. Logically multiplying the two registers, therefore, results in the appearance of the desired information in the accumulator register. This information is placed in the memory buffer register by command 17, after this register is cleared by command 41.

To illustrate this instruction, assume that the memory address contains 1 0 1 1 0 1, the accumulator register contains 1 1 0 0 1 1, and the B register contains 0 1 0 1 0 0. The 1's in the second and fourth digit positions of the B register indicate that the second and fourth bits of the accumulator are to replace the contents of the corresponding digit positions in the specified memory address. These two digit positions are underlined in the illustration. The steps are outlined in the order in which they are performed. (Refer to table 3–17.)
In step 4, of table 3–17, the logical multiply operation, in effect, multiplies the numbers in the accumulator and A registers together bit by bit as shown below:

0 1 0 1 0 0 A Register

0 0 1 1 0 0 Accumulator (original)

0 0 0 1 0 0 Accumulator (final)

In step 5, the number resulting in the A register is a consequence of transferring the memory buffer register contents into the A register without first clearing the A register. As shown in figure 3–32, this can be interpreted in the following way: whenever there is a 1 in the memory buffer register, a 1 will appear in the corresponding digit position in the A register; wherever there is a 0 in the memory buffer register, the corresponding digit position in the A register will remain unchanged. Accordingly:

1 0 1 1 0 1 Memory Buffer Register

0 1 0 1 0 0 A Register (original)

1 1 1 1 0 1 A Register (final)

After completion of step 8, the memory buffer register contains the modified memory address contents. During the write portion of the subsequent memory cycle, this information is transferred into the specified memory address. Once this is accomplished, the memory buffer register is again cleared by command 41 in preparation for the next instruction.

4.7 BRANCH CLASS

The instructions comprising the branch class are used to modify the program sequence. In this respect, they should more properly be treated in the program element discussion. However, the arithmetic element is involved in determining if the conditions to branch have been met for certain of these instructions; viz., *Branch on Zero (BFZ)*, *Branch on Minus (BFM)*, *Branch on Right Minus (BRM)*, and *Branch on Left Minus (BLM)*. The *Branch and Index (BPX)* instruction also utilizes the A registers but only for storage of the initial address during cycling processes. The *Sense (BSN)* instruction does not affect the arithmetic element although the memory buffer registers in the arithmetic elements are, as is the case with every instruction, used for information transfer to and from magnetic core memory.

TABLE 3–17. OPERATIONAL SEQUENCE FOR *DEPOSIT* INSTRUCTION

STEP	COMMAND NUMBER	MEMORY ADDRESS	MEMORY BUFFER REGISTER	A REGISTER	ACCUMULATOR REGISTER	B REGISTER
1	21, 41	101101	000000	000000	110011	010100
2	19		Same	Same	001100	
3	88		101101	010100	Same	
4	25		Same	Same	000100	
5	19, 43		Same	111101	111011	
6	25		Same		111001	
7	41		000000			
8	17		111001			

4.7.1 *Branch on Zero (BFZ)*

The *BFZ* instruction allows the program to branch to the address specified by this instruction if and only if the two accumulator registers contain in any combination positive and/or negative 0's. The instruction execution is identical in both elements. The method used for sensing the contents of the accumulator register for this instruction restores the original accumulator register contents after the sensing is performed. This requirement is imposed so that the original accumulator register data will be available for the next programmed instruction if the register initially held a number other than negative or positive 0 (conditions for branch not met). The method adopted tests the accumulator register sign bit for 1 after a sequence of complement and add 1 operations. This sequence is devised so that the sign bit of the accumulator register will be 1 if and only if the original number held in this register was positive or negative 0.

The circuits in the left arithmetic element used during the *BFZ* instruction appear in figure 3–33. As the first step, the accumulator register is complemented by command 13. As can be seen in the figure, this command is conditional on a 1 being in the sign bit flip-flop of the accumulator register. This operation insures that a positive number exists in the accumulator at this time. If this command results in the accumulator register being complemented, the sign control flip-flop is set to 1. Command 19 is issued next and complements the accumulator register unconditionally, making its contents negative regardless of the sign of the original number. A 1 is now added to the contents of the register by command 69 which pulses the carry 1 line of the bit 15 adder. (Previously, the A register had been cleared by command 21, and contributes 0 during the standard add process.) The next set of commands (2, 4, 81, 89) returns the sum to its proper alignment in the accumulator register by correcting the inherent shift right which always occurs during an add process. Assuming that the accumulator register content had been 0 (positive or negative) originally, the accumulator register flip-flops would now each contain a 0. Command 19 is reissued, once again complementing the register. It is at this point that a 1 in the sign bit indicates that a 0 was originally held in the accumulator register. Accordingly, command 162 examines the gate tube conditioned by the 1 side of the sign flip-flop. If this gate tube is up, its output examines a second gate tube conditioned by the 1 side of the sign flip-flop in the right accumulator register. As a result, an output from this latter gate tube indicates that 0's were contained in both the left and right accumulator registers. Consequently, the output of the second gate tube is used to set the branch flip-flop (located in the instructions control element), the conditions for branch having been met. Simultaneously with command 162, command 69 again pulses the carry 1 line of the bit 15 adder. This second addition nullifies the effect of the first add 1 operation, the signs of the number being opposite in the two instances. A correctional shift left is again accomplished by commands 2, 4, 81, and 89. The operations thus far, besides testing for full 0, have restored the original magnitude information (in either true or complement form) in the accumulator register. To completely restore the original information in the accumulator register, it remains to affix the proper sign. This is done by command 214, which examines the gate tube conditioned by the 1 side of the sign control flip-flop. If the gate is conditioned, its output complements the accumulator register and clears the sign control flip-flop.

4.7.2 *Branch on Minus (BFM), Branch on Left Minus (BLM), Branch on Right Minus (BRM)*

The three other instructions in this class which make use of the arithmetic elements are only concerned with the sign bits of the accumulator registers. The circuits in the arithmetic elements used to execute the three instructions are illustrated in figure 3–34. In the case of the *BFM* instruction, command 162 is applied to two gate tubes in series, each one being conditioned by 1 (negative) side of the sign flip-flop in the left and right accumulator registers. If these signs are both negative, the resultant pulse is used to set the branch flip-flop in the instruction control element. This setting action will condition command generators 93 and 94. In the *BRM* instruction, command 162 is replaced by command 166, which only examines the sign bit of the right accumulator register. In the *BLM* instruction, command 165 is used to examine the sign bit of the left accumulator register. The action in all of these instructions is the same. If the condition is met, command generators 93 and 94 are conditioned. At the proper pulse time, these commands will transfer the contents of the program counter to the $\overline{A}$ register and then clear the program counter in preparation for the branch operation. Command 154 sends the contents of the address register to

COMMAND NUMBER	PULSE TIME
21	PT_1-9
13	OT-1
19	OT-2, OT-7
62	OT-3, OT-8
2,4,81,89	OT-6, OT-11
162	OT-8
214	PT_2-6

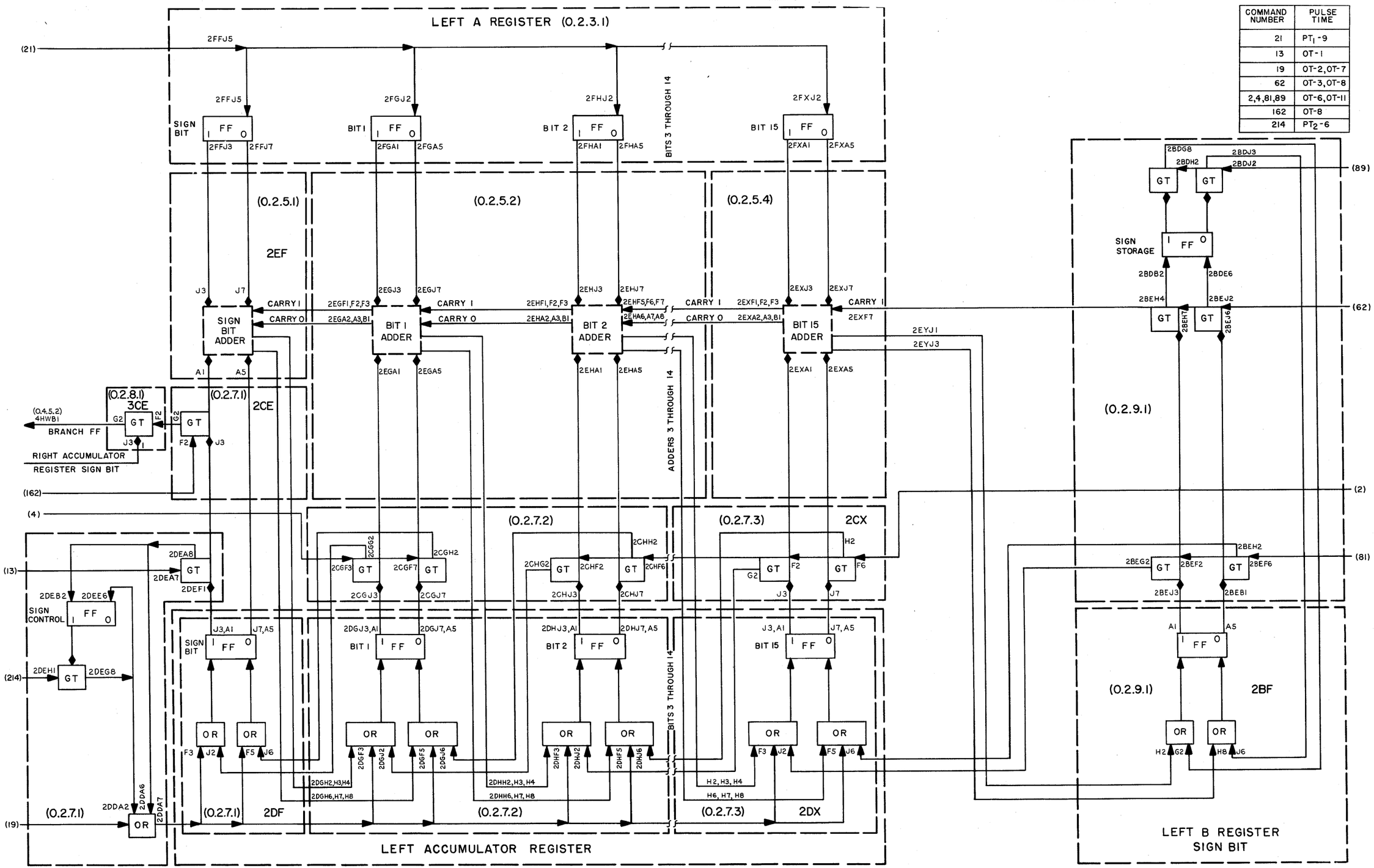
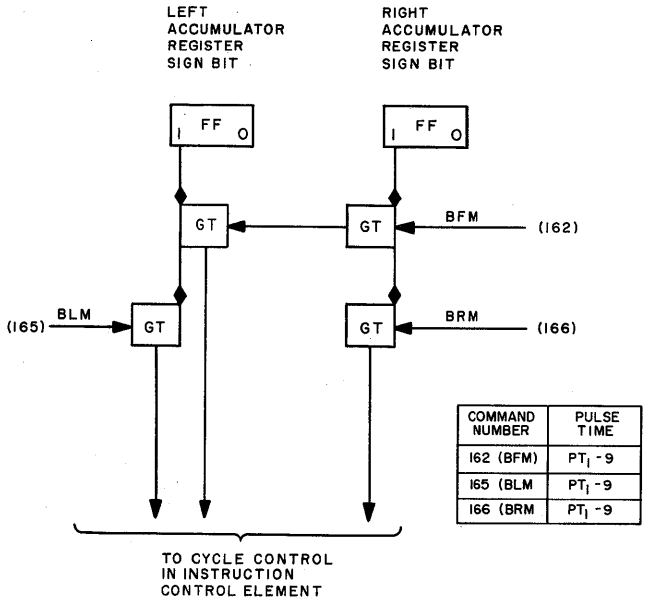


Figure 3–33. **Branch on Zero (BFZ),** ***Logical Block Diagram***

the program counter. Command 163 clears the branch flip-flop at the end of this instruction so that succeeding instructions do not generate spurious branch commands.



COMMAND NUMBER	PULSE TIME
162 (BFM)	PT_i -9
165 (BLM	PT_i -9
166 (BRM	PT_i -9

Figure 3–34.* Branch on Minus (BFM), Branch on Right Minus (BRM), Branch on Left Minus (BLM), *Logical Block Diagram

PART 4
PROGRAM ELEMENT

CHAPTER 1
INTRODUCTION

SECTION 1
GENERAL

The program element, acting in conjunction with the instruction control and selection and IO control elements, carries out the instruction sequence scheduled by the programmer. The program element registers perform the following functions in carrying out the scheduling and sequencing operations:

A. Select the memory unit and the memory address in that unit which is specified by a program instruction.

B. Keep track of the program instruction sequence so that the program is executed properly.

C. Keep track of IO operations.

D. Modify the address portion of the instruction word by the process of indexing. This process changes the address in the program element without affecting or changing it in core memory.

Program element operations are synchronized with the operations of both the instruction control element, which generates the commands that execute the program instructions, and the selection and IO control element, which generates the commands that execute the actual word transfers into or out of the Central Computer System. The control exercised by the program element is therefore logically divided into two categories:

a. Control of instructions during internal Central Computer System operations

b. Control of instructions during IO operations

An internal Central Computer System operation is defined as one in which certain instructions and data contained in core memory are used wholly within the Central Computer System. When memory is used in the transfer of words to or from external devices, it is referred to as an IO operation. Although by definition the IO class of instructions falls into the category of internal operations, this class is included in IO operations, since these instructions prepare the Central Computer System for word transfers.

During internal operations, the program element registers specify an address in core memory from which or into which instructions and operands (numerical data) are transferred during a machine cycle. The addresses required by the Central Computer System are contained in the right half of the instruction word, which is transferred to the program element from the right memory buffer register. The right half-word of the instruction, consisting of 16 bits, specifies a memory unit by bits R1 to R3 and a memory address by bits R4 to R15. The program element registers decode bits R1 to R3, thereby selecting a memory unit as well as specifying a memory address in this unit for use in Central Computer System operations.

An instruction is furnished to the Central Computer System during a program time (PT) cycle; the location of this instruction was specified during the program time of the previous instruction. If the instruction requires an operand, the program element registers supply the operand address during the subsequent operate time (OT) cycle. If at any time it is desired to store data in core memory, one of the program element registers furnishes the memory address in which these data are to be stored during an operate time B

cycle. For internal operations, the program element registers keep track of the program sequence and also perform the instruction modification (indexing), as specified by the operation part of the word.

The IO class of instructions, which prepares the Central Computer System for external or IO operations, is also indexable; however, the word transfer process itself is not indexable. During IO operations, various registers in the program element are loaded when instructions in the IO class are executed. One of these loaded registers specifies the initial core memory address from which or into which words are to be transferred. A second register specifies the number of words to be transferred, and a third register designates the address or identity code used for comparison with words transferred between core memory and the Drum System. For IO operations, the program element registers keep track of the number of words transferred and specify addresses in core memory for storage of these words. A more detailed discussion of these program element functions is given in Section 2.

1.1 PRELIMINARY DEFINITIONS

The following briefly reviews certain Central Computer System timing functions and describes the composition of the instruction word. A general understanding of these will greatly facilitate an understanding of the program element. For a detailed discussion, reference should be made to Part 1 and Part 2.

1.2 TIME AND INSTRUCTION PULSES

Central Computer System operations are timed or synchronized by time pulses (TP's) and instruction pulses (IP's). These pulses are generated by the time pulse distributor of the instruction control element, and are distributed as required throughout the Central Computer System. The relative occurrence times of these pulses with respect to other Central Computer System events are shown by the vertical lines on figure 4–1. Thus, it is seen that there are 15 time pulses in a machine cycle. (Refer to 1.4 of this Section.)

Instruction pulses are numbered in the following manner: IP–1, IP–2, IP–3, IP–4, IP–5, IP–6, IP–6A, IP–7, IP–8, IP–8A, IP–9, IP–10, IP–11, and IP–11A. A similar method of numbering is utilized for the time pulses (TP's). For example, a machine cycle, shown in figure 4–1, begins with instruction pulse (IP–1), and ends with instruction pulse 11A (IP–11A), at which time the cycle is re-sequenced from 1 through 11A for the next machine cycle. Although the TP and IP pulses are generated simultaneously, TP pulses are not used to execute instruction, but are sent instead to the selection and IO control element, where they are used to sequence and synchronize IO operations.

1.3 MEMORY CYCLE

The minimum time between successive words read from or into the memory element is called a memory cycle. This time interval is 7.5 microseconds. One memory cycle is required for every

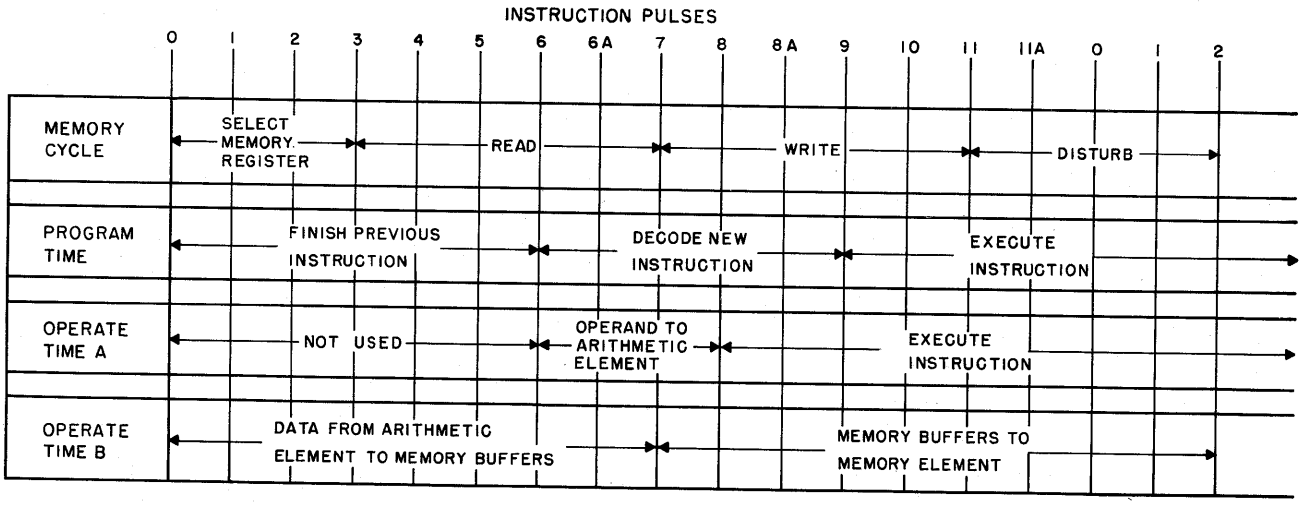


Figure 4–1. Memory and Machine Cycles

word read from or written into core memory. Details of a memory cycle are shown in figure 4–1. The time pulses (shown as occurring at 0.5-microsecond intervals along the horizontal axis of figure 4–1) establish approximate relationships between events in the memory cycle and in the machine cycles. (Refer to 1.4 of this Section.) However, except for those pulses which co-ordinate and synchronize the core memory with the rest of the Central Computer System, the internal memory timing is independent of the time pulses.

The memory cycle time will be reduced to 6.0 microseconds if operating tests prove it to be feasible. The machine cycle will then also be reduced by deleting the timing pulses marked "A" in figure 4–1.

1.4 MACHINE CYCLE

The machine cycle comprises an interval of time (7.5 microseconds) equal to the memory cycle, and its events, shown in figure 4–1, are controlled by the time or instruction pulses described above. Although the machine cycle is of the same duration as the memory cycle, the two should be distinguished: first, in certain program instructions, information from core memory is not used (for example, *Branch on Zero (BFZ); Sense (BSN); Select (SEL); Select Drum (SDR); Operate (PER);* and *Program Stop (HLT);* secondly, this equality of the machine and memory cycles is not generally true of computers.

1.5 INSTRUCTION CYCLES

Operation of the Central Computer System may also be discussed in terms of instruction cycles. An instruction cycle is defined as that time required for the Central Computer System to execute one complete instruction, and is usually composed of from one to three machine (or memory) cycles.

Of the 48 instructions used in AN/FSQ–7 (XD–1, –2) Combat Direction Centrals, 11 involve simple operations such as setting up control circuits or transferring words between two registers. Because of this simplicity of operation, these instructions are completed in one memory cycle (7.5 microseconds) or less, and are called one-memory-cycle instructions. (See fig. 4–2.)

Eighteen of the 48 instructions require two memory cycles for completion; these instructions are called two-memory-cycle instructions. Before such instructions can be completed an operand must be obtained from or stored in the memory element; a second memory cycle is therefore provided, during which core memory is referred to, the operand obtained or stored, and the instruction completed.

Six instructions require three memory cycles for completion; the instruction word is obtained and decoded during the first memory cycle; an operand is obtained and the required operation is performed during the second memory cycle; and the result of the operation is stored in the memory element during the third memory cycle.

The memory or machine cycles which compose an instruction cycle have been assigned distinctive names for easy reference. The names and characteristics of these are listed in table 4–1.

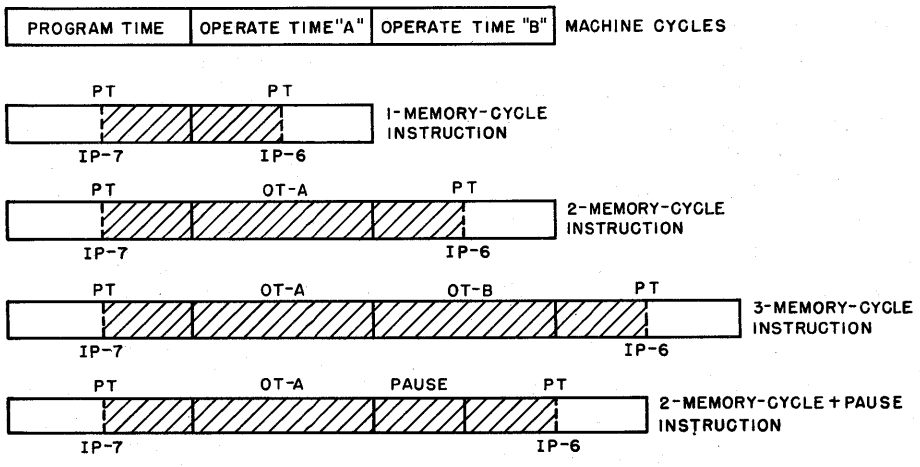


Figure 4–2. Machine and Instruction Cycles

TABLE 4–1. NAMES AND CHARACTERISTICS OF MACHINE OR MEMORY CYCLES

MEMORY OR MACHINE CYCLE	NAME	CHARACTERISTIC
First	Program Time (PT)	Decodes instruction and initiates execution
Second	Operate Time A (OTA)	Obtains operand and performs operation. If no operate time B follows, this cycle is called simply operate time
Third	Operate Time B (OTB)	Stores result of operation in core memory

Although a machine cycle begins with IP–1, an instruction cycle starts with IP–7 of a program time cycle, which is denoted by PT–7. It is at this time that the instruction word is transferred from the memory buffer register to other Central Computer System elements. Figure 4–2 shows the basic machine and instruction cycles. In these charts, the instruction cycles are shown as cross-hatched areas on the sequence of machine cycles. Thus, in a one-memory-cycle instruction, the decoding process starts at PT–7, and the instruction is completed by PT–6 of the subsequent machine cycle; the decoding of the next instruction is initiated at this time. Similarly, a two-memory-cycle instruction starts at PT–7, continues through the subsequent OTA cycle (in some cases, this will be an OTB cycle), and is completed by PT–6 of the next PT cycle, when the execution of the next instruction begins.

The times shown on figures 4–1 and 4–2 should be considered only as approximations, since these times vary from instruction to instruction. As has been noted, the time from IP–1 to IP–6 of a program time cycle is used to complete an instruction previously begun; this time interval is also utilized to bring the new instruction out of the memory element. The new instruction is placed in the memory buffer registers at or before PT–6, and is transferred to the operation, address, and index interval registers at PT–7. Thus an old instruction is completed at the same time that a new instruction is obtained from the memory element.

1.6 PAUSE

An exception to the usual sequence of machine cycles occurs in the case of 13 instructions associated with AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. In these instructions, additional time is required for the performance of a series of repetitious operations. Such operations are multiplication, which requires repeated addition; division, which requires repeated subtraction; or shifting, which may require any number of shifts. This additional time is supplied by stopping the generation of IP pulses in the instruction control element, thus also stopping the operation of instruction and machine cycle sequencing. This stoppage is known as a "pause", since the Central Computer System pauses in its usual sequential operation long enough to complete the repetitious operations. This is typical of the multiplication and division operations. An example of an instruction cycle utilizing a pause is depicted in the last diagram of figure 4–2. In the illustration, the decoding of the instruction begins at PT–7; the operand (in this case either the multiplicand or divisor) is obtained during the subsequent OTA cycle. At the end of the OTA cycle, the generation of instruction pulses is stopped, and the Central Computer System goes into a pause. The repetitious additions or subtractions composing a multiplication or division are executed during this period. The pause condition then ends and the program time cycle begins, completing the instruction by PT–6.

1.7 COMPOSITION OF INSTRUCTION WORD

An instruction word is read from the memory element during each program time (PT) cycle. This word contains coded information which specifies the particular operations the Central Computer System is to perform during the ensuing instruction cycle. As stated previously, this cycle nominally begins at PT–7, when decoding of the instruction word is just beginning. The decoding of the instruction word takes place principally in the instruction control element, and will not be considered here.

The 32 bits of the instruction word are given names to facilitate reference or discussion. (See fig. 4–3.) The 32-bit word is divided into two 16-bit half-words termed, respectively, the left and right half-words. Each half-word consists of a sign bit and 15 numerical bits which are designated LS, L1, L2, . . . L15 for the left-half word, and RS, R1, R2, . . . R15 for the right half-word.

The left half-word is sometimes also called the "operation part" or "instruction part" of the word and the right half-word the "address part."

1.7.1 Left Half-Word

Bits L1 through L3 are termed the index indicator, in that they are used to specify which one of

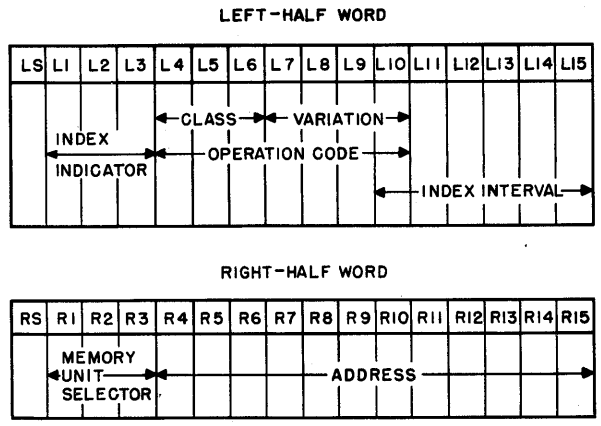


Figure 4–3. Composition of Instruction Word

two index registers or the right accumulator register is used in indexing. Indexing provides a means of altering or cycling the Central Computer System program for repetitive programming operations.

Bits L4 through L10 are termed the "operation code." These bits determine which of the 48 instructions associated with the Central Computer System are to be performed. Bits L4 through L6 specify one of eight classes of instruction, and L7 through L10 specify one of the variations of the basic instruction class. Four bits are required, since as many as nine variations (for the add class) are possible.

Bits L10 through L15 are termed the "index interval"; they are used to provide additional information required by particular instructions. The representation of the index interval varies with the instruction specified by bits L4 through L10.

Bit L10 is utilized in both the operation code and the index interval. However, this causes no ambiguity for whenever the index interval is used by an instruction, bit L10 is not needed to identify the instruction involved. Thus, when the whole index interval is utilized, the appropriate instruction is completely specified by only six bits of the operation code, instead of the usual seven.

1.7.2 Right Half-Word

The right half of the instruction word denotes the location in the memory element, or auxiliary storage elements, from which additional pertinent data may be obtained or stored. This additional data usually takes the form of an "operand" required in the execution of a mathematical operation. In addition, the right half-word is used to load some of the registers within the program element.

Bits R1 through R3 of the right half-word are called the "memory unit selector." These bits specify memory units in which the desired word may be found. Bits R4 through R15 specify the exact location of the stored word. The memory units available are: core memory No. 1, core memory No. 2, test memory, and the clock register. Since test memory has only 16 memory locations, four bits, R12 through R15, are sufficient to specify the location of a stored word. When bits R2 through R3 specify the clock register, the address bits are meaningless, since the clock register possesses only one memory location.

The preceding discussion (except for that of the clock register) applies also to the storage of words or results of Central Computer System calculations in the address specified in the right half-word. The address part of the word can specify only a definite memory location; whether a word is obtained from or stored in a memory location depends on the instruction contained in bits L4 through L10 of the operation part of the word.

The address part of the word is meaningless in certain instructions, since no reference to a memory unit is required. An example of this is the *Program Stop (HLT)* instruction, which causes the stoppage of the Central Computer System. In this case, it is apparent that no new information is necessary and that there is no information to be stored; the content of the address register is therefore both extraneous and meaningless.

SECTION 2
BLOCK DIAGRAM ANALYSIS

An overall block diagram of the program element is shown in figure 4–4. Simplified diagrams illustrating principal modes of program element operation are shown in figures 4–5 through 4–7. Reference to these figures should be made throughout the reading of this entire paragraph.

The program element contains nine registers. These registers may be divided into three groups: those which take part in internal Central Computer System operations, those which take part in IO operations, and those which are used in indexing operations.

2.1 ADDRESS REGISTER

The address register holds the right half of the instruction word received from core memory during a program cycle. As specified by the operation part of the instruction word, the contents of the address register are utilized in other registers of the Central Computer System. After this use, the address register may be cleared (at PT–6), or its contents may be retained and modified (indexed) by the contents of the index registers (No. 1 or No. 2), or the right accumulator register. This modification is an addition process, accomplished by means of the index adder circuits, which are an integral part of the address register. The sum is then held in the address register for further operations. The original contents of the address register are lost by this addition, but may be found in the original address in core memory from which the instruction was taken. The contents of the index register remain unchanged unless further operations are specified which change those contents. Note that not all instructions may be modified or indexed in this manner.

2.2 INDEX REGISTERS

Two index registers are provided in the program element. In addition, the right accumulator register in the arithmetic element may be used as an index register. It is the sole function of the index registers to index the address portion of the instruction word when this type of operation is specified by the left half of the instruction word. Index registers Nos. 1 and 2 are used in performing cyclic loops of instructions. The right accumulator register can be used as an index register when table look-up operations are performed. (Refer to Appendix A of this Part.) The right accumulator register can also be used to reset either index register No. 1 or No. 2 for a subsequent cyclic loop operation. Index registers Nos. 1 and 2 are loaded by instructions in the reset class. The right accumulator register is loaded by these instructions whose operations result in an answer which is held in this register.

2.3 PROGRAM COUNTER

The program counter is used to specify the memory address of the next instruction in the program. As soon as this instruction is transferred to the instruction register, a 1 is added to the contents of the program counter. This is the new address from which the next instruction will be taken.

The contents of the program counter may be changed by instructions in the branch class, which may be either conditional or unconditional. A conditional branch instruction depends on one of the following three factors: the status of either index register No. 1 or No. 2 (i.e., whether their contents are positive or negative), the status of certain registers in the arithmetic element, or the status of certain IO equipment external to the Central Computer System. An unconditional branch instruction has no qualifying factors.

When the condition is met for a conditional branch, or an unconditional branch occurs, the program branches to the address contained in the right half of the branch instruction word. This address is transferred from the address register to the program counter, which will then contain the new memory address from which the next instruction is to be taken. Succeeding instructions will thereafter be selected from sequential core memory locations starting with this new address.

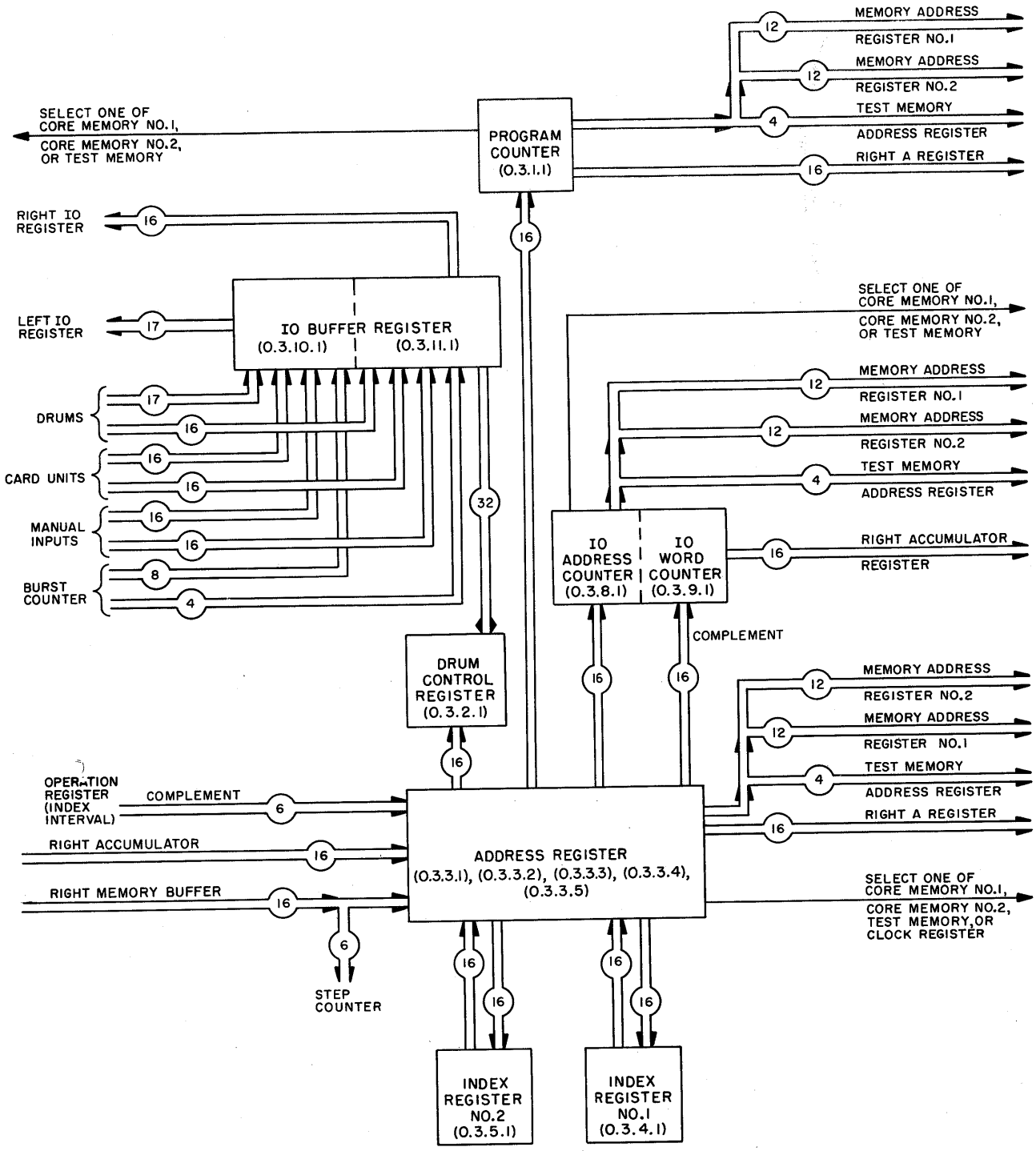


Figure 4–4. Program Element, Block Diagram

2.4 IO ADDRESS COUNTER

The IO address counter is used during IO operations to designate the core memory addresses from which or into which words are to be transferred during break cycles. The starting core memory address which is involved in word transfer is loaded into the IO address counter from the address register with the *Load IO Address Counter (LDC)* instruction. Transfer can occur in either direction between core memory and external storage equipment. As each word is transferred, a 1 is added to the contents of the IO address counter, thus insuring that the next sequential core memory address will be specified

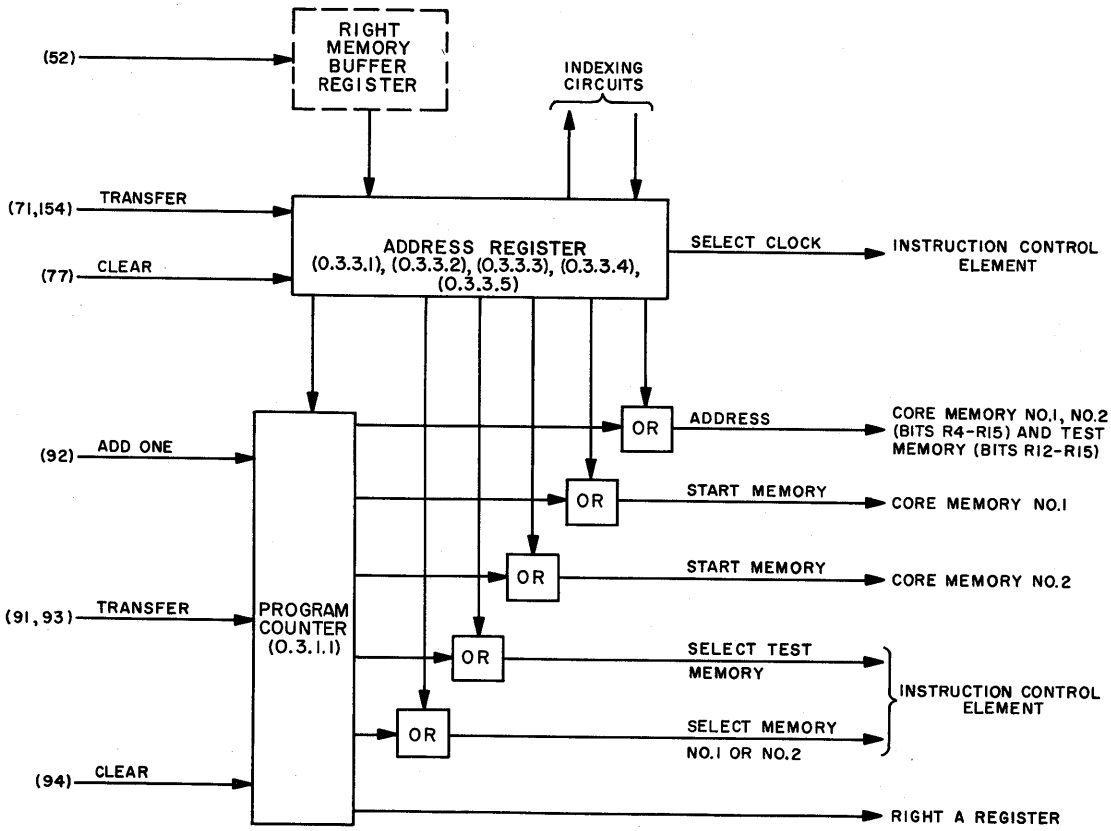


Figure 4–5. Program Element Internal Operations Control, Block Diagram

for the subsequent word transfer. The IO address counter and IO word counter are always used jointly during IO operations.

2.5 IO WORD COUNTER

The IO word counter is used during IO operations to count the number of words being transferred. It is loaded with the complement of the number of words to be transferred, as indicated by the *Read (RDS)* or *Write (WRT)* instructions. During break cycles, the IO word counter is stepped by 1 for each word transferred. Since the information held in the IO word counter is in complement form, successive additions of 1 to the complement reduces this number by 1 each time the register is stepped. Thus, the IO word counter is eventually reduced to negative 0 (all bits of the counter are 1's). The next stepping therefore develops an end carry pulse which clears the IO interlock, thus ending the word transfer and making the Central Computer System available for any subsequent operations.

2.6 DRUM CONTROL REGISTER

The drum control register provides circuits which compare a desired drum address or identification code with the drum address currently under the read-write heads. The desired address or identity code is specified by the *Select Drum (SDR)* instruction. The drum control register incorporates comparison circuits which are connected to the right IO buffer register.

The Drum System sends a compare pulse with each word placed in the IO buffer register to initiate comparison. If the address or identity code in the drum control register does not compare with the one in the IO buffer register, a no compare pulse is returned to the Drum System, and the word is not transferred. If they do compare, the Drum System does not receive a no compare pulse and proceeds on the assumption that a comparison has been made. Information transfer is then free to occur between the Central Computer System and the Drum System.

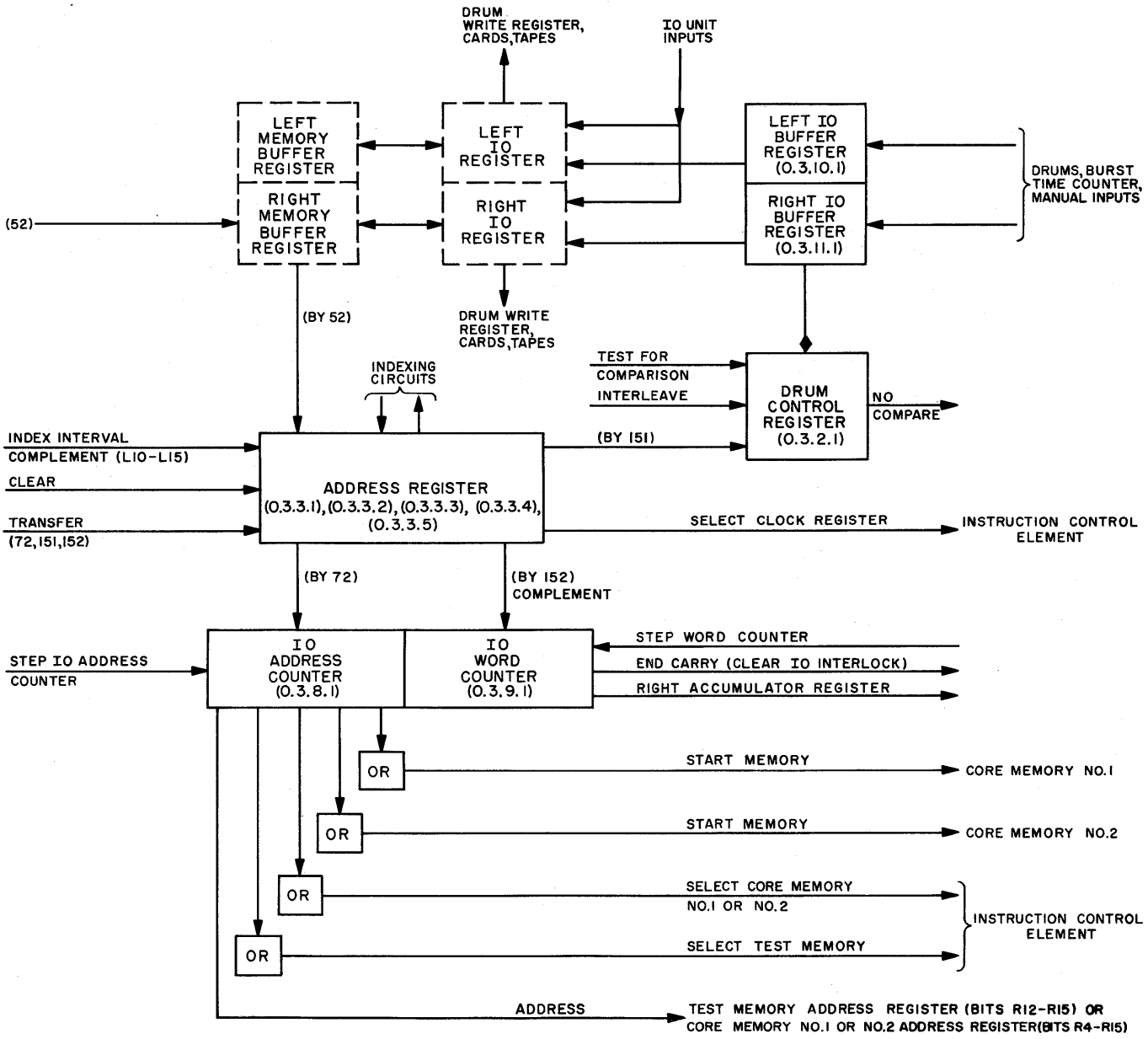


Figure 4-6. Program Element IO Operations Control, Block Diagram

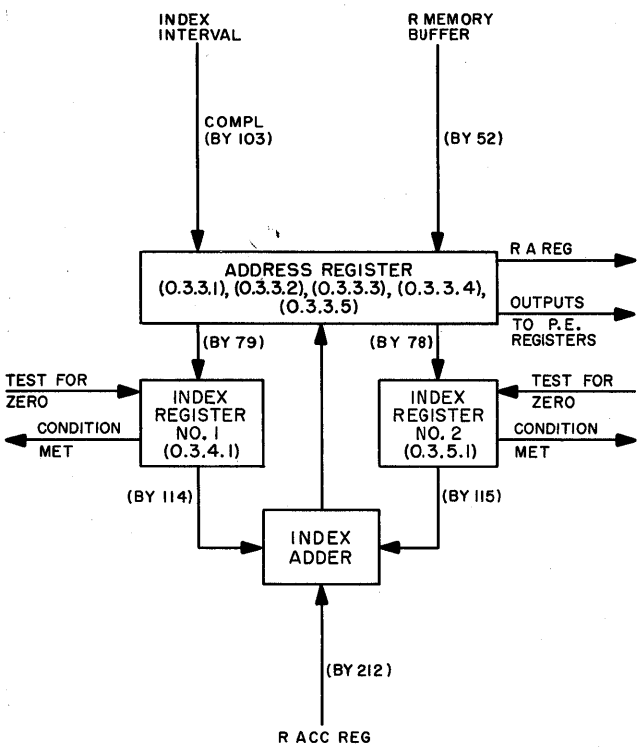


Figure 4–7. Instruction Indexing, Block Diagram

2.7 IO BUFFER REGISTER

The IO buffer register performs two main functions: adapts the high speed of the Central Computer System to the relatively low speeds of the various external systems by acting as a temporary storage device for information being transferred between the Central Computer System and a particular external system; and, secondly, in conjunction with the drum control register comparison circuits, to determine whether a computer-drum information transfer is to take place.

During break operations, the IO buffer register transfers words from drums, cards, manual inputs, and the burst time counter into core memory, via the IO register. When a drum is selected for IO operations, the drum address or identity code currently under the drum read-write heads is placed in the right IO buffer register. These addresses are compared with the desired address or identity code which has been placed in the drum control register by the *Select Drum (SDR)* instruction. If the comparison is successful, transfer of words occurs between Central Computer System and the Drum System.

SECTION 3
CONTROL DURING INTERNAL OPERATIONS

Information flow within the program element depends on the type of instruction being executed. For internal operations, the following three broad categories of instructions determine information flow in the program element:

a. Those in which the address portion specifies an operand. These instructions include the add class (except for the *Add B register (ADB)* instruction); the multiply class; the store class; and the *Load B Register (LDB)* and *Extract (ETR)* instructions of the miscellaneous class.

b. Those which cause the program sequence to be modified. These instructions are all in the branch class.

c. Those upon which the program element registers perform no operation other than that of sequencing in accordance with the programmer's schedule. These instructions include the shift class, and the *Shift Left and Round (SLR), Program Stop (HLT), Clear and Subtract Word Counter (CSW),* and *Operate (PER)* instructions of the miscellaneous class.

Figure 4–4 shows those program element registers which are involved in internal Central Computer System operations. To illustrate the influence of these three categories of instructions on information flow, one instruction from each category is broken down into the command sequence which executes the specific instruction. Those commands which do not directly affect information flow in the program element are described by command number only. (For a complete description of these instructions, refer to the command sequence charts and the list of instruction control element commands in Part 2.)

3.1 *ADD* INSTRUCTIONS

The first category of instructions which influence information flow in the program element is illustrated by the *Add* instruction. As shown in figure 4–5, command 52 transfers the right half-word from the right memory buffer register to the address register. At the same time, the program counter contents are increased by 1, and now contains the address of the next instruction. The memory address and the test memory address registers are then cleared so that the address contained in the address register may be transferred to the memory address registers just cleared by command 71. Command 71, in addition, sends a start memory pulse to the memory unit selected by bits R1 to R3 of the right half-word. The memory address must be in the memory address register prior to the read portion of the memory cycle. This insures that the contents of the specified memory address will be read out of core memory into the memory buffer registers before a new instruction cycle occurs, or before the operand, transferred from core memory during an OT cycle, is used for arithmetic operations. The left and right memory buffer registers are now cleared so that the operand from core memory may be transferred into these registers. Arithmetic operations are performed on this operand for the remainder of the OT cycle and the subsequent PT cycle.

At the start of the next PT cycle, the memory address and test memory address registers are cleared in preparation for receipt of the next instruction address contained in the program counter. This instruction address may be transferred to any one of the three memory address registers. (See fig. 4–5.) (Simultaneously, the left and right memory buffer registers are cleared in anticipation of receiving the new instruction.) The address register is then cleared so that the new instruction may be transferred to these registers from the memory buffer registers when the next instruction cycle starts.

3.2 *BRANCH* INSTRUCTIONS

The second category of instructions which influence information flow in the program element is illustrated by the *Branch on Minus (BFM)* instructions. In general, the branch class of instructions modifies the program sequence if certain test conditions are met.

Command 52 transfers the right half of the instruction word from the right memory buffer register to the address register. At the same

time, a 1 is added to the program counter so that it contains the address of the next instruction. The left and right accumulator registers are then tested. If the test condition is met, the program counter contents are transferred to the right A register, where they are temporarily stored. The program counter is also cleared to receive the branch address from the address register. The meeting of the test condition has caused information flow between the address register and the program counter. In addition, another information flow line is established with the transfer of information between the program counter and the right A register.

Prior to the start of the read portion of the memory cycle, the new address (branch address) is transferred from the program counter to the memory address register. The program sequence continues from this new address in the program counter, unless another branch class instruction occurs. The next instruction address will be the branch address plus 1, etc.

3.3 SEQUENCING INSTRUCTIONS

The third category of instructions influencing information flow in the program element is typified by the shift class and *Program Stop (HLT)* instructions. In these instructions, no reference is required to a memory or storage unit and the address portion of the instruction word is consequently meaningless. Even though the address portion of the word may be transferred to the program element and may or may not be transferred to the registers within this element for instructions in this category, this address is consequently cleared and lost. The address, therefore, serves no function in the program element. The sole function of the program element for this category is to sequence these instructions so that the program schedule is carried out.

SECTION 4
CONTROL DURING IO OPERATIONS

IO operations involve the transfer of blocks of words between consecutive core memory locations (addresses) and an IO device. Conditions for initiating the block transfer of words are prepared by the IO class of instructions in the program element, although the actual word transfer is controlled by the selection and IO control element.

Three instructions are programmed to prepare the Central Computer System for word transfers: *Load IO Address Counter (LDC)*, *Select (SEL)* or *Select Drums (SDR)*, and *Read (RDS)* or *Write (WRT)*. The sequence in which the first two instructions are programmed is immaterial, so long as both precede either the *RDS* or *WRT* instruction. During the execution of these instructions, the IO address counter, drum control register, and IO word counter, shown in figure 4–6, are loaded by a specific instruction address contained in the address register.

The execution of the *LDC* instruction transfers this instruction address from the right memory buffer register to the address register, and from there to the IO address counter. The address in the IO address counter designates the first of a sequential series of addresses in core memory which are to be used in subsequent input or output transfer operations.

The execution of the *SDR* instruction transfers this instruction address from the right memory buffer register to the drum control register, via the address register. The address in the drum control register designates a desired address or identity code against which the drum address or words in the IO buffer register are compared; this comparison is initiated by a *compare* command received from the Drum System. If the address or identity code does not compare, a no compare pulse is sent back to the Drum System to prevent transfer of words. If a successful comparison is made, words are transferred between the drums and core memory, starting at the address specified by the *LDC* instruction.

When the *RDS* or *WRT* instruction is executed, the complement of the word, rather than the word itself, is transferred to the IO word counter from the address register. The number contained in the address portion of the *RDS* or *WRT* instruction specifies the number of words to be transferred during IO operations. At the same time, this instruction turns on the IO interlock, allowing the transfer of data to begin when the first word is available from the IO device. This interlock prevents the Central Computer System from selecting more than one IO unit at a time, or from trying to execute another IO instruction before the selected IO unit has disconnected. Each time a word is transferred, both the IO address counter and the IO word counter are stepped; i.e., a 1 is added to the contents of these registers.

The IO process continues until the IO word counter steps to 0, generating an end carry (disconnect signal) which clears the IO interlock. The IO interlock delays all succeeding IO operations until the previous transfers are completed. When the Central Computer System is in the IO interlock on status, instructions of all classes except the IO class may be executed between actual word transfers. The IO units are slow in operation compared to the high-speed Central Computer System. Transfer of words between the two takes place in short, rapid bursts. Between words, relatively long time intervals occur. The Central Computer System is designed to utilize available time between word transfer to process program instructions or perform arithmetic operations.

The *Select (SEL)* instruction, used for all IO devices other than drums, has a meaningless address. The IO device to be used in IO operations is designated instead by index interval (Ix Int) bits L10 to L15. These bits are decoded by the index interval matrix of the operation register in the selection and IO control element.

During the execution of IO instructions, the address register temporarily stores the instruction address, either in its original or in its modified (indexed) form. The address is then transferred to the register designated by the instruction. During this time, either the program counter or the IO address counter selects either core memory No. 1 or No. 2, and also selects the memory address. As in internal operations, the addresses from either register must be transferred to one of the three memory address registers prior to the read portion of the memory cycle, so that the contents of the address may be transferred from core memory to the memory buffer register before a new instruction cycle starts.

SECTION 5
INSTRUCTION INDEXING

The index adder, an integral part of the address register, in conjunction with the index registers (index registers No. 1 or No. 2, or the right accumulator register), shown in figure 4–7, provide instruction indexing. Instruction indexing may be used to perform any of the following:

a. Cyclic loops or sub-programming, when data or instructions are repeated any number of times

b. Table look-up procedures, when table of sines, cosines, logarithms, etc. must be referred to

c. Adding a constant to data, when the constant is the same for a number of specific pieces of data

d. Inspecting the contents of any one of the index registers

These functions will be discussed in detail in Section 4, Chapter 3 of this Part.

Indexing consists of address modification by the addition of the contents of an index register to the contents of the address register by commands 114, 115, or 212. The resultant sum is held in the address register for further operations. Bits L1 to L3 of the operation part of the word specify which one of the three index registers is to take part in instruction indexing. The class and variation bits, L4 to L10, specify the operation. If an index interval is to be used when cyclic loops are performed, bits L10 to L15 specify the index interval, which is transferred to the address register in complement form.

Because of the fact that the index interval is sent to the address register in complement form, bits LS through L9 have to be set to 1's to obtain a true 16-bit complement; this is accomplished by having the same command that transfers the index interval to the address register also set bits LS through L9 to 1's.

The contents of the address register are transferred to index registers No. 1 or No. 2, respectively, by commands 79 or 78, during the execution of the *Branch and Index (BPX)*, *Reset Index Indicator (XIN)*, and *Reset Index Indicator from the Right Accumulator (XAC)* instructions. The *Add Index Register (ADX)* instruction transfers the sum of an index register and the address register to the right A register. This instruction is usually followed by the *Store Address (STA)* instruction, which then transfers this sum from the right A register to core memory, for use in reference or in future operations.

CHAPTER 2
SPECIAL-PURPOSE CIRCUITS

2.1 GENERAL

This Chapter describes special-purpose circuits used in the program element. These are the additive counter, the index adder, the interleave circuit, and the comparison circuit.

This discussion does not attempt to relate the operation of each circuit with that of other circuits of the Central Computer System. (Refer to Chapter 3 of this Part.) In addition, the diagrams show only those parts of the program element registers which directly affect the circuit under discussion. Note that on these diagrams, complement lines are shown entering the center of a flip-flop rather than as pulsing both sides. This symbol is adopted for simplicity, although the flip-flops are actually complemented by applying a single pulse to both grids simultaneously.

2.2 ADDITIVE COUNTER

The program counter, the IO address counter, and the IO word counter employ circuits which count in increments of 1. These circuits are termed additive counters. The basic configurations for the three counting circuits are very similar, differing only in the bit positions affected. The circuit used in the program counter is shown in figure 4–8, part A. Although a total of 8192 core memory registers are available in the two core memory units, each memory unit has its registers numbered consecutively from 0 through 4096. Therefore, 4096 (2^{12}) is the largest number which the program counter is required to hold. Accordingly, only 13 flip-flops (bits 3 through 15) are used in the counting circuit.

The counting process for the program counter is initiated by command 92, generated by the instruction control element at every PT–7. If the number held in the program counter is even, the bit 15 flip-flop must contain a 0. Increasing an even number by 1 in the binary system merely entails changing the final bit (bit 15 in this case) to 1. The remaining bits of the number are not affected. If the number held in the program counter is odd, the 15th bit must be a 1.

Increasing an odd number by 1 entails changing the final 1 to a 0 and also the generation of a carry pulse. Accordingly, command 92 always reverses the status of the bit 15 flip-flop. The same pulse is simultaneously applied to a carry gate tube, conditioned by the 1 side of this flip-flop. With an even number in the counter, the flip-flop is complemented to 1, but the gate tube, deconditioned at the instant it is pulsed by command 92, does not yield a carry pulse and the remaining bits therefore remain unchanged.

With an odd number in the counter, the flip-flop is complemented to 0, but the gate tube, inspected before the flip-flop status can be reversed, now propagates a carry pulse to bit 14. Bit 14, as well as the remaining bits, behaves in exactly the same manner as bit 15. Consequently, the carry pulse originating in bit 15 will propagate through the counter, complementing each flip-flop in turn until it encounters a flip-flop containing a 0. This 0 is complemented to 1, but the carry pulse is suppressed, and all higher-order bits are left unchanged.

The adder for the IO address counter is illustrated in figure 4–8, part B. This circuit is identical in operation to that of the program counter previously discussed, with the exception that 14 bits are used and the stepping pulse is furnished by the selection and IO control element. The additional bit is required so that the IO address counter can specify test memory as an address, this address being $20{,}000_8$; 14 bits are necessary to represent this number in binary form.

The circuit illustrated in figure 4–8, part C, is also an additive counter, identical in operation to the other two, and is used in the IO word counter. In this case, the complete 16-bit register is employed, and the stepping pulse is command 290, furnished by the instruction control element. The full register is used so that the highest number possible with the basic 16-bit register may be specified.

A significant difference between the IO word counter and the other counters is that the carry

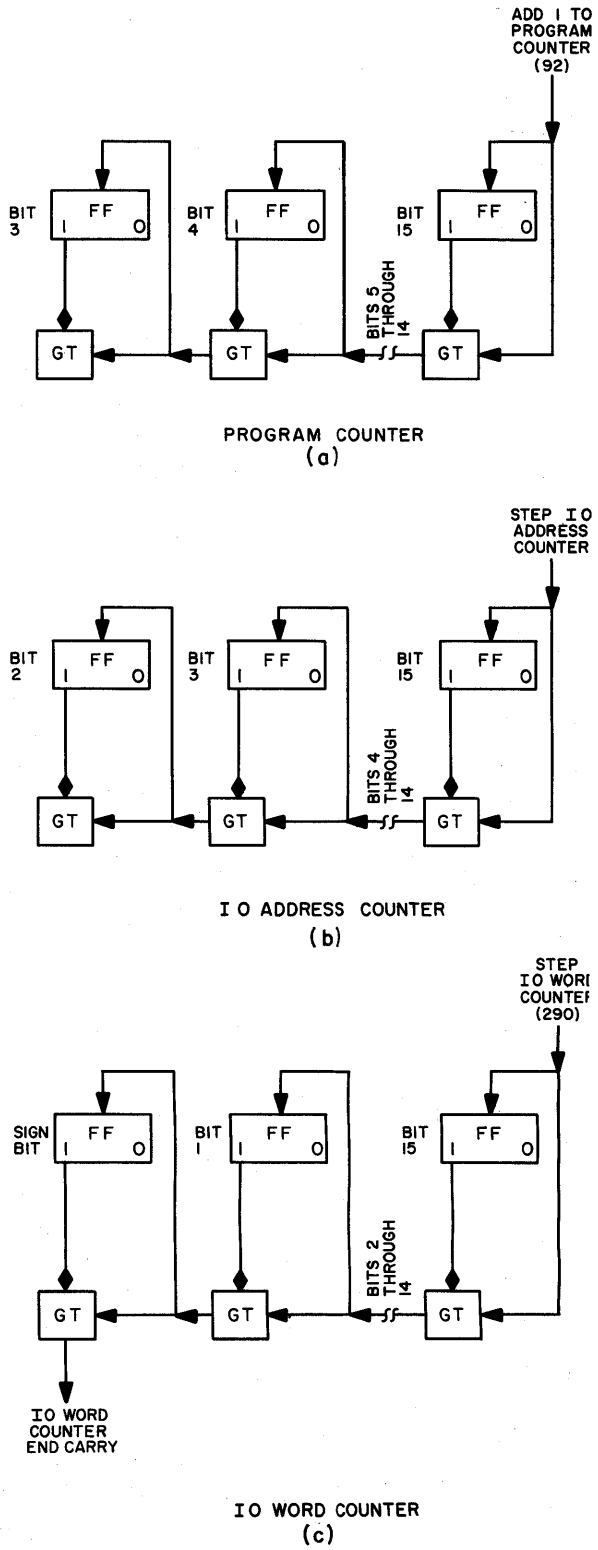


Figure 4–8. Additive Counters, Logical Diagram

pulse generated by the last bit of the register (sign bit) is used to control the IO interlock. An end carry pulse can be generated only if the IO word counter contains negative 0 at the time that command 290 is applied to it. Negative 0 indicates that for a particular IO operation, no words remain to be processed and, properly, the IO interlock is turned off.

2.3 INDEX ADDER

The contents of any one of the three index registers (No. 1, No. 2, or the right accumulator register) are used to modify the information in the address register during indexing operations. The index adder implements this modification process by employing parallel addition; i.e., all 16-bit positions are added simultaneously and independently, with provision made to take resulting carry pulses into account at some time after the initial addition. The index adder is slower than the arithmetic adder but, with a minimum of equipment, still develops the speed required to synchronize the indexing operation with the other operations of the Central Computer System.

The circuits of the index adder are shown in figure 4–9. Sixteen identical circuits (one for each bit position) and an end carry circuit comprise the index adder. Each flip-flop of the address register receives information from the corresponding flip-flop in the selected index register by means of a complement line. This complement line is the output of a gate tube conditioned by the 1 side of the index register flip-flop and pulsed by the appropriate indexing command: command 114 for index register No. 1; command 115 for index register No. 2; and command 212 for the right accumulator register when that register is used as index register No. 3. Consequently, a 1 in any index register flip-flop will cause the corresponding address register flip-flop to reverse its status when an indexing operation is initiated. A 0 in any index register flip-flop will not affect the status of its associated address flip-flop. Thus, column by column, the index register contents modify the address register contents in accordance with the rules of binary addition.

If the address and index register flip-flops for any given bit position each contain a 1, their sum will be a 0 and a carry pulse will result when the two bits are added. This carry pulse must be taken into account when adding the next highest order bit position. In the parallel add process employed here, addition is performed simultaneously in all 16-bit positions; hence, any resulting carry pulse must be delayed to provide sufficient settling time for those address register flip-flops which have been complemented during the initial phase of the addition process. This is accomplished in the following manner. The pulse from the index register

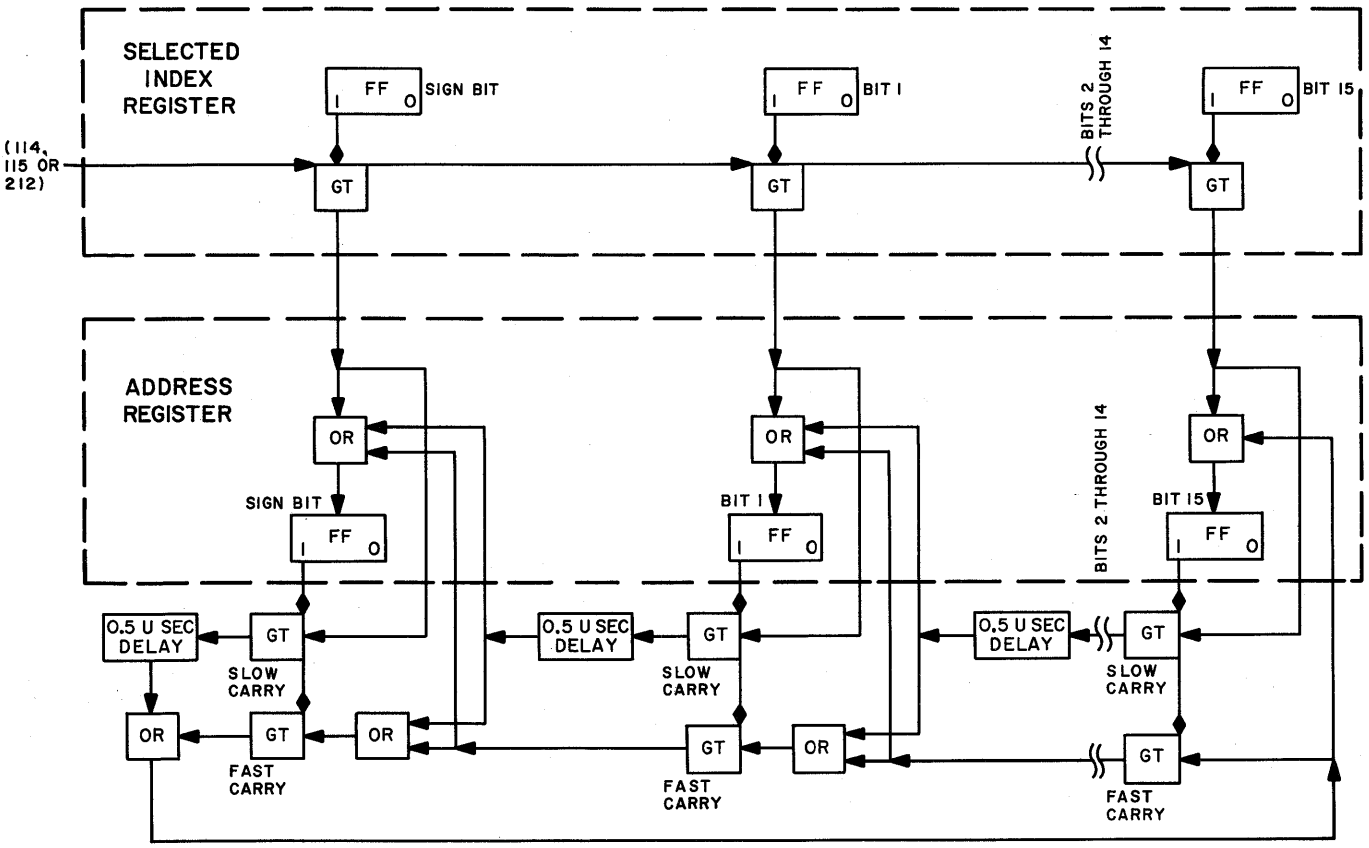


Figure 4–9. Index Adder, Logical Diagram

which complements the address register flip-flop also examines a gate tube (slow carry gate tube) conditioned by the 1 side of the address register flip-flop in question. This examination is completed before the status of the address register flip-flop is altered. If this flip-flop contains a 1 (1 and 1 being added), the slow carry gate tube, being conditioned, will transmit the index register pulse to the delay line as a carry pulse. The delay line retards the carry pulse for 0.5 microsecond, which is sufficient settling time for the address register flip-flops.

The process so far described is simultaneously executed in all 16-digit positions, and is termed the "partial add process." At the completion of the partial add process, the situation existing in the address register is such that each address register flip-flop which originally held a 0 will duplicate the contents of the corresponding index register flip-flop; each address register flip-flop which originally contained a 1 will contain the reverse of the contents of the corresponding index register flip-flop. In the case where two 1's are added, a carry pulse is stored in the delay line associated with that bit position. After an elapsed time of 0.5 microsecond, each delay line which had stored a carry pulse transmits that pulse to its left adjacent bit position and performs two functions: complements the flip-flop for this left adjacent bit position, and inspects a gate tube (fast carry gate tube) conditioned by the 1 side of this flip-flop.

As an illustration of the propagation of carry pulses through the address register, assume that the bit 1 flip-flop contains a 1, and that a carry pulse has been transmitted from bit 2; this carry pulse complements the bit 1 flip-flop and simultaneously inspects the fast carry gate tube conditioned by the 1 side of this flip-flop. Because of the inherent delay in changing the flip-flop status, the carry pulse is transmitted to the corresponding fast carry gate tube conditioned by the 1 side of the sign bit flip-flop. The fast carry pulse could not have been transmitted to the sign bit position unless the bit 1 flip-flop contained a 1. In such a case there is no possibility of a slow carry pulse having been generated during the partial add process, because in the generation of a slow carry pulse, the index register complement pulse inspects the slow carry gate tube. The output of this gate tube is applied to the delay line

and becomes the slow carry pulse. In order for this gate tube to yield an output, the bit 1 flip-flop must contain a 1 when inspected. However, the same pulse which inspects the slow carry gate tube will also complement the flip-flop. This operation changes the flip-flop status from 1 to 0, but only after the slow carry pulse has been generated. Therefore, the bit 1 flip-flop must, after being complemented, contain a 0 if a slow carry pulse has been developed. However, the bit 1 flip-flop was originally assumed to contain a 1. It has been demonstrated that this precludes the possibility of a slow carry pulse being generated concurrently with a fast carry pulse.

To illustrate the complete function of the index adder, assume that the address register and index register No. 1 are each three bits in length, and that the left-most bit in each register designates the sign of the number in the register. Assume further that the index register contains the number 3 (0 1 1) and that the address register contains –1 (1 1 0). The condition of the register is then as follows:

	SIGN BIT	BIT 1	BIT 2
Index Register No. 1	0	1	1
Address Register	1	1	0

Upon application of command 114 to the transfer gate tubes associated with the index register, the bit 1 and bit 2 flip-flops in the address registers are complemented. Furthermore, the carry pulse resulting from the addition in the bit 2 column is stored in the delay line associated with this bit. At this time, the partial add process is completed and the address register condition is as follows:

		("1")	
Address Register	1	0	1

The "1" in parentheses indicates the bit started in the delay line.

After 0.5 microsecond, the delayed carry pulse is transmitted to the sign bit circuits in the address register, complementing the sign bit flip-flop. Because of the inherent delay in the flip-flop, the carry pulse passes through the fast carry gate tube before the flip-flop can reverse its status. The resultant output pulse (end carry) is supplied to the circuits associated with the bit 2 flip-flop. This end carry pulse is required because the numbers in the address register are expressed in the 1's complement form. The end carry pulse passes through the fast carry gate tube conditioned by the 1 side of the bit 2 flip-flop and also complements this flip-flop. The propagated pulse goes on to complement the bit 1 flip-flop; but, since the bit 1 flip-flop contains a 0 before being complemented, the fast carry gate tube for bit 2 is deconditioned at the time the carry pulse is applied. Consequently, the carry pulse is suppressed at this point. The final result in the address register will be the number 2, correctly completing the subtraction of 1 from 3 leaving the address register as follows:

Address Register	0	1	0

2.4 INTERLEAVE CIRCUITS

The interleave circuits shown in figure 4–10 cause the drum control register to count by 8, 16, or by 64. Since each bit position of a register represents some power of 2, the bit positions which are directly affected are the 9th (2^6), the 11th (2^4), and the 12th (2^3). Further, since the highest address appearing on any drum is 2047 (0 being the first address), bit position 5 (2^{10}) is the highest order bit which need be considered. If all bits from 5 to 15 are 1's and the rest are 0's, the decimal number represented will be 2047. The next addition, whether of 8, 16, or 64, will result in the register holding the number 7, 15, or 63, respectively, if there is no end carry. Accordingly, an end carry circuit is provided which connects any carry pulse generated by the bit 5 circuits to the input circuit of bit 15.

Since all the interleave circuits operate identically, assume that interleaving by 8 is desirable. The selection and IO control element decodes the index interval of the *Select Drum* instruction, which specifies that interleaving by 8 is required. This element then transmits a signal pulse which sets the interleave-by-8 flip-flop in the program element. This setting action conditions the gate tube associated with this flip-flop.

The gate tube, upon being interrogated by command 327, transmits a pulse which complements the bit 12 flip-flop. If the bit 12 flip-flop originally contained a 0, reversing its status is sufficient to increase the contents of the drum control register by 8. If it originally contained a 1, the flip-flop will still reverse its status, but a carry pulse must be generated additionally. To generate this carry pulse, the incoming pulse also inspects a gate tube conditioned by the 1 side of the bit 12 flip-flop. Before the status of the bit 12 flip-flop can change from 1 to 0, the gate tube generates the desired carry pulse. This pulse complements the bit 11 flip-flop, and, if bit 11 was originally a 1, the carry pulse is transmitted to the bit 10 flip-flop; propagation continues until a flip-flop which contains a 0 is encountered. At this point the particu-

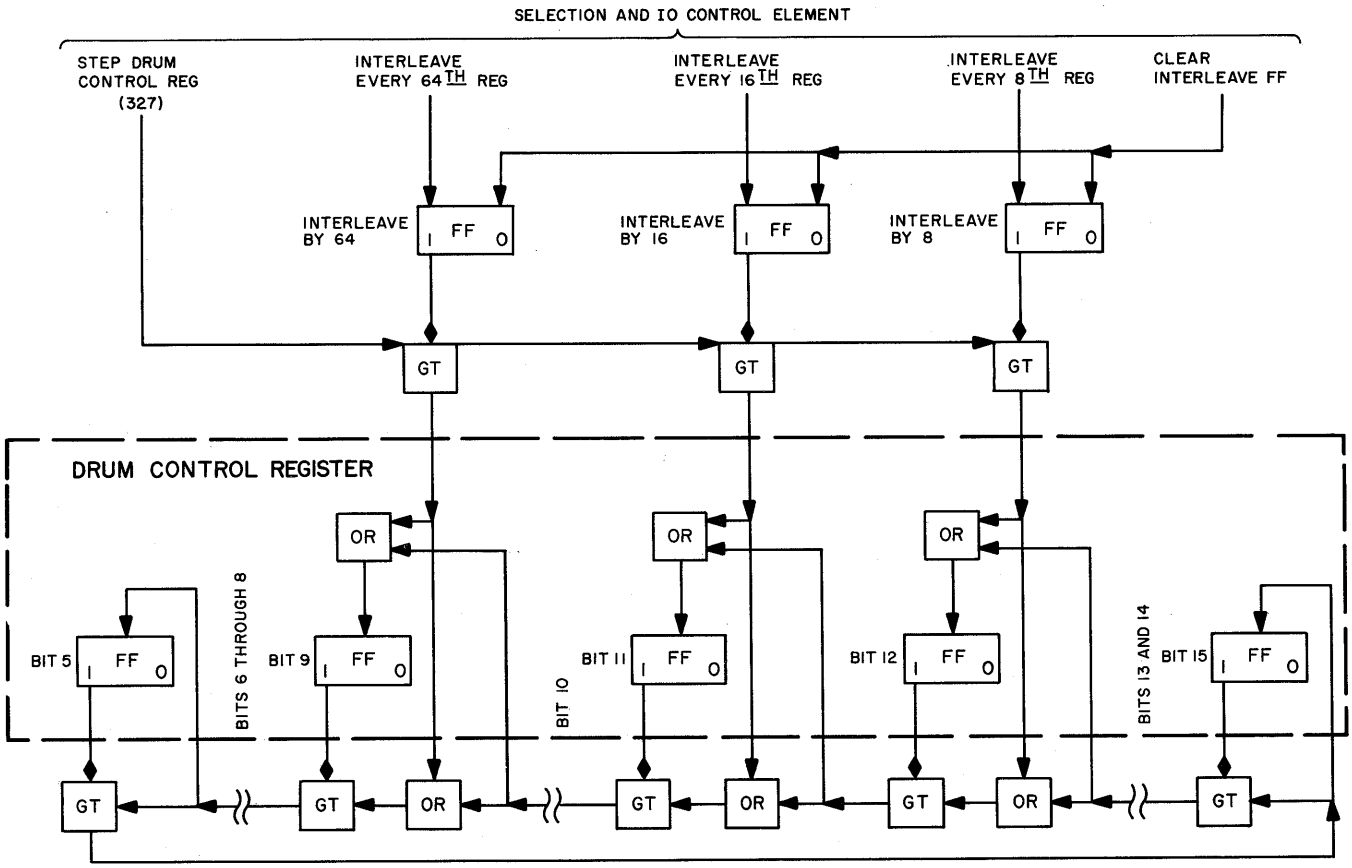


Figure 4–10. Interleave Circuit, Logical Diagram

lar flip-flop is complemented to 1, but the carry pulse is suppressed. If at any time the bit 5 flip-flop generates a carry pulse, this end carry pulse is sent to the bit 15 flip-flop and behaves, therefore, in exactly the same manner as the carry pulse just described. The end carry pulse can occur only if bit 12 and all higher order bits up to bit 5 are 1's.

The final three bits are not restricted. This means that, decimally, the number in the register must lie between 2047 and 2040. If the register did hold 2040 at the time that the interleaving was initiated, the bit 12 flip-flops and all higher order flip-flops up to the bit 5 flip-flop would be complemented to 0, and each one would propagate the carry pulse. The carry pulse from the bit 5 flip-flop would be applied to the bit 15 flip-flop by the end carry circuit. Bit 15 would be changed to a 1, but all other bits would be 0. The drum control register would therefore hold a decimal 1.

2.5 COMPARISON CIRCUITS

The comparison circuits in the program element are used for controling the transfer of information between the Central Computer System and the Drum System. These circuits are of importance only when either the address or the identity mode of information transfer is in effect. Corresponding bit positions in the right IO buffer register and the drum control register are compared. Every bit position from 5 through 15 is provided with a comparison circuit, all of these circuits being identical. The mode of reading or writing (address or identity) determines which of these circuits are to be examined.

When the address mode is being used, bit positions 5 through 15 are compared. When the identity mode is being used, one of three possible groups of bit positions is compared: bit positions 5 through 10, bit positions 11 through 15, or bit positions 14 and 15. Figure 4–11 illustrates the comparison circuits for bit positions 14 and 15. As shown in this diagram, d-c levels from the four flip-flops are combined by means of AND and OR circuits. These combinations are such that the gate tube shown is conditioned only if the contents of either the bit 14 or bit 15 flip-flop in the right IO buffer register do not agree with the contents of the corresponding flip-flops in the drum control register.

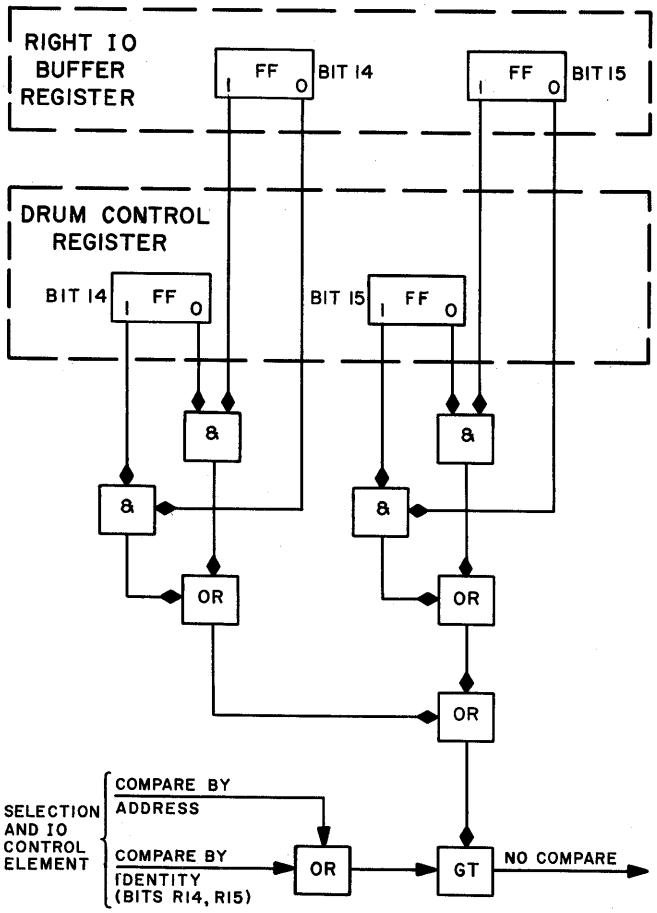


Figure 4–11. Comparison Circuit, Logical Diagram

The gate tube is interrogated by either one of two pulses: the compare by address pulse if the address mode is in effect, or the compare by identity pulse if the identity mode for bits 14 and 15 is in effect. If the two registers agree in these two bit positions, the examining pulse will be suppressed and the imminent information transfer will be permitted to take place. If either or both bit positions do not agree, a no compare pulse will be generated. Generation of this pulse will prevent any information transfer between the Central Computer System and the Drum System.

There are two other gate tubes in the comparison circuits whose functions and circuit connections are very similar to the one shown in figure 4–11. The first of these gate tubes is conditioned by a six-input OR circuit. The inputs for this OR circuit are derived from bit positions 5 through 10 of the two registers in exactly the same manner just described for bit positions 14 and 15. The second gate tube is conditioned by a three-input OR circuit, which supplies information about bit positions 11, 12, and 13. The corresponding OR circuit and gate tube circuit for bit positions 14 and 15 have been previously described. All of these gate tubes are inspected by the compare by address pulse.

The gate tube for bit positions 5 through 10 is examined by the compare by identity 5–10 pulse. The gate tube for bit positions 11, 12, and 13, and the gate tube for bit positions 14 and 15, are examined in parallel by the compare by identity 11–15 pulse. As has been explained, the gate tube for bit positions 14 and 15 is examined by the compare by identity 14–15 pulse. All of these pulses are generated by the selection and IO control element.

CHAPTER 3
OPERATIONAL ANALYSIS

SECTION 1
INTRODUCTION

One of the primary functions of the program element registers is to specify, for any one of the three memory address registers, a memory unit and a memory address which are to be used for either internal Central Computer System operations or word transfers. As mentioned previously, the right half of the instruction word specifies a memory unit by bits R1 to R3, and indicates a memory address by bits R4 through R15. The code used for specifying the various memory units and the bits used to specify the memory address are shown in table 4–2.

TABLE 4–2. MEMORY UNIT AND ADDRESS SPECIFICATION

MEMORY UNIT	MEMORY UNIT SELECTION CODE			BITS USED FOR MEMORY LOCATIONS
	R1	R2	R3	
Core memory No. 1	—	0	0	R4 to R15
Core memory No. 2	—	0	1	R4 to R15
Test memory	0	1	—	R12 to R15
Clock register	1	1	—	———

The significant bits for memory unit selection are R2 and R3 for specifying either core memory No. 1 or No. 2, and R1 and R2 for specifying test storage or clock counter use. No matter what digit is contained in bit R1 (0 or 1), if bits R2 and R3 are 00 or 01, either one of the core memories is selected. In the same manner, if bits R1 and R2 are 01 or 11, the digit contained in R3 has no significance, and either test memory or the clock register is selected. Address register bits R4 through R15 indicate one of 4096 core memory addresses which is used. Three registers in the program element feed the various memory address registers and the memory selector circuits. Two of these registers, the address register and the program counter, are used in internal operations; the third, the IO address counter, is used in IO operations. These registers select a memory unit and a memory address and, in addition, specifically start core memories No. 1 and No. 2.

SECTION 2
INTERNAL OPERATIONS CONTROL

This section deals with those functions of the program element which have to do with information transfer between core memory and the Central Computer System and instruction sequencing.

2.1 MEMORY UNIT SELECTION

The memory unit selector circuits shown in figure 4–12 are an integral part of the address register and the program counter. Either one of these registers may select and start core memories No. 1 or No. 2, and may also select test memory. The address register, however, is the only register which may select the clock register. The *program counter to memory address register* command (91) and the *address register to memory address register* command (71), occurring at PT-1 and OT–1, respectively, are the gating pulses used to initiate memory unit selection.

The code for selecting the clock register is 11—. This means that the bit 1 and bit 2 flip-flops in the address register must both be set to 1. Those gate tubes 2 (GT–2) which are associated with the flip-flops of bits 1 and 2 are conditioned by these flip-flops. When command 71 occurs, the GT–2 associated with the bit 2 flip-flop is pulsed, generating an output which in turn pulses the GT–2 associated with the bit 1 flip-flop. An output pulse is thus generated which selects the clock register.

The code for selecting test memory is 01—. The bit 1 flip-flop must be cleared to 0 and the bit 2 flip-flop set to 1 in both the address register and the program counter to enable either one to select test memory. In the address register, the GT–3 associated with the bit 1 flip-flop and the GT–2 associated with the bit 2 flip-flop are both conditioned by these flip-flops. Command 71 pulses the GT–2 associated with the bit 2 flip-flop. The output generated from GT–2 pulses the GT–3 associated with the bit 1 flip-flop, which in turn generates a pulse that selects test memory. In the program counter, the GT–4 associated with the bit 1 flip-flop and the GT–7 associated with the bit 2 flip-flop are both conditioned by these flip-flops. Command 91 pulses GT–4 associated with the bit 1 flip-flop, generating an output which in turn pulses GT–7. This output pulse from GT–7 selects test memory.

If the bit 2 flip-flop in either the address register or the program counter is cleared to 0, core memory No. 1 or No. 2 is automatically selected. Command 71 pulses address register GT–3, which is conditioned by the bit 2 flip-flop, and command 91 pulses program counter GT–4, which is conditioned by the bit 2 flip-flop. In either case, a pulse is generated which is used to select core memories No. 1 and No. 2.

The three select pulses (one each for the clock register, test memory, and core memories No. 1 or No. 2) thus generated are transferred to the instruction control element for the clock register or directly to the selected core memory.

The code for core memory No. 1 is —00. The bit 2 and bit 3 flip-flops in both the address register and the program counter must be cleared to 0. In the address register, the GT–3's associated with the bit 2 and the bit 3 flip-flops are both conditioned by these flip-flops. Command 71 pulses the GT–3 conditioned by the bit 2 flip-flop, generating an output which in turn pulses the GT–3 conditioned by the bit 3 flip-flop. The output pulse from the latter GT–3 is transferred directly to core memory No. 1, thus starting it. In the program counter, the GT–4's associated with the bit 2 and the bit 3 flip-flops are conditioned by these flip-flops. Command 91 pulses the GT–4 conditioned by the bit 2 flip-flop, generating an output which in turn pulses the GT–4 conditioned by the bit 3 flip-flop. This latter GT–4 now generates an output pulse which is fed directly to core memory No. 1, thus starting it.

Core memory No. 2 is started in a manner almost identical to that used for core memory No. 1. Since the core memory No. 2 is —01, the bit 3 flip-flop, instead of being cleared to 0 as in the previous example, must now be set to 1 in both the address register and the program counter. In the address register, the output generated by the GT–3 conditioned by the bit 2 flip-flop now pulses the GT–2 conditioned by the bit 3 flip-flop, thus generating an output pulse which starts core memory No. 2. In the program counter, the output generated by the GT–4 conditioned by the bit 2 flip-flop now pulses the GT–7 conditioned by the bit 3 flip-flop, thus generating an output pulse which starts core memory No. 2.

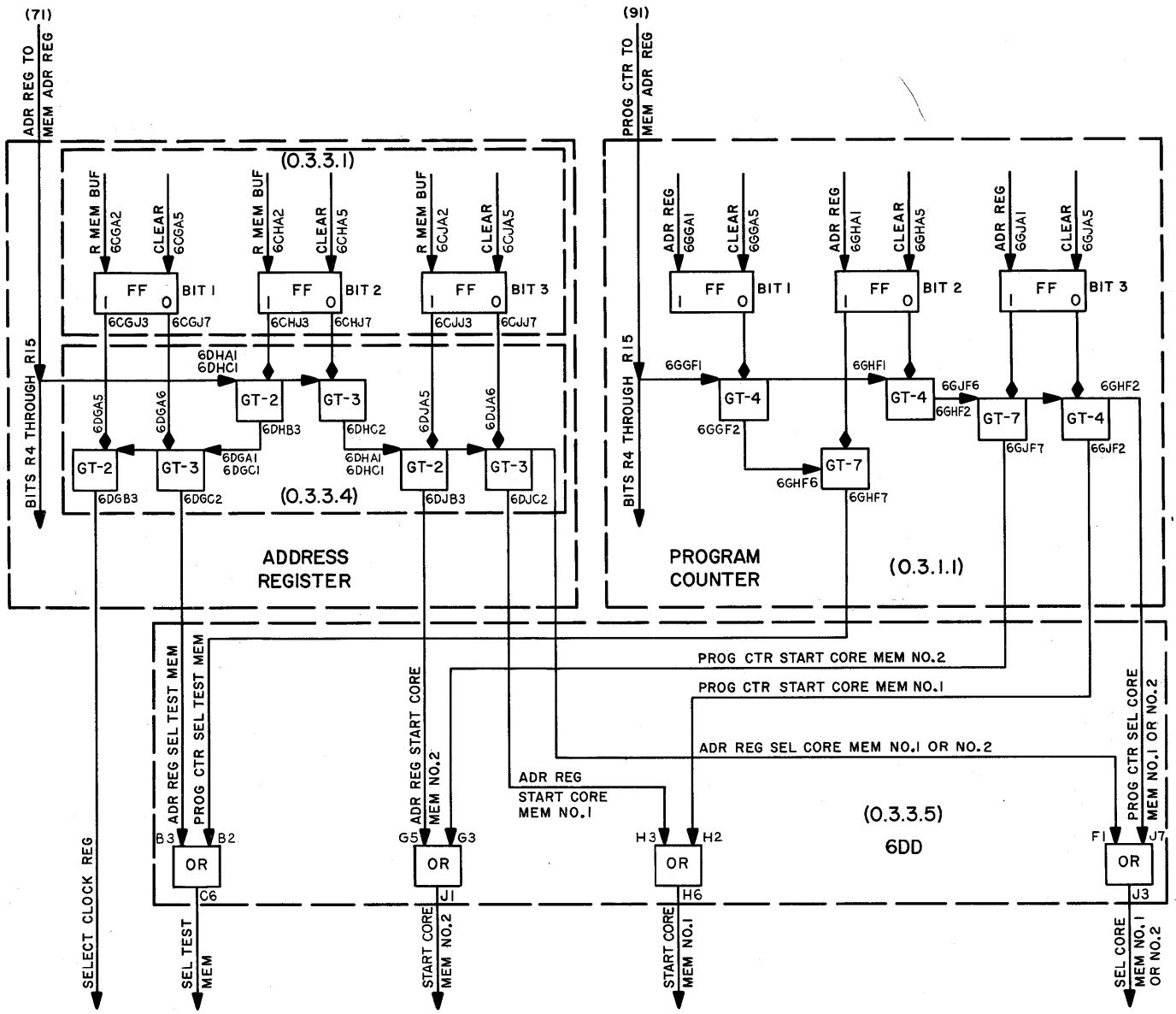
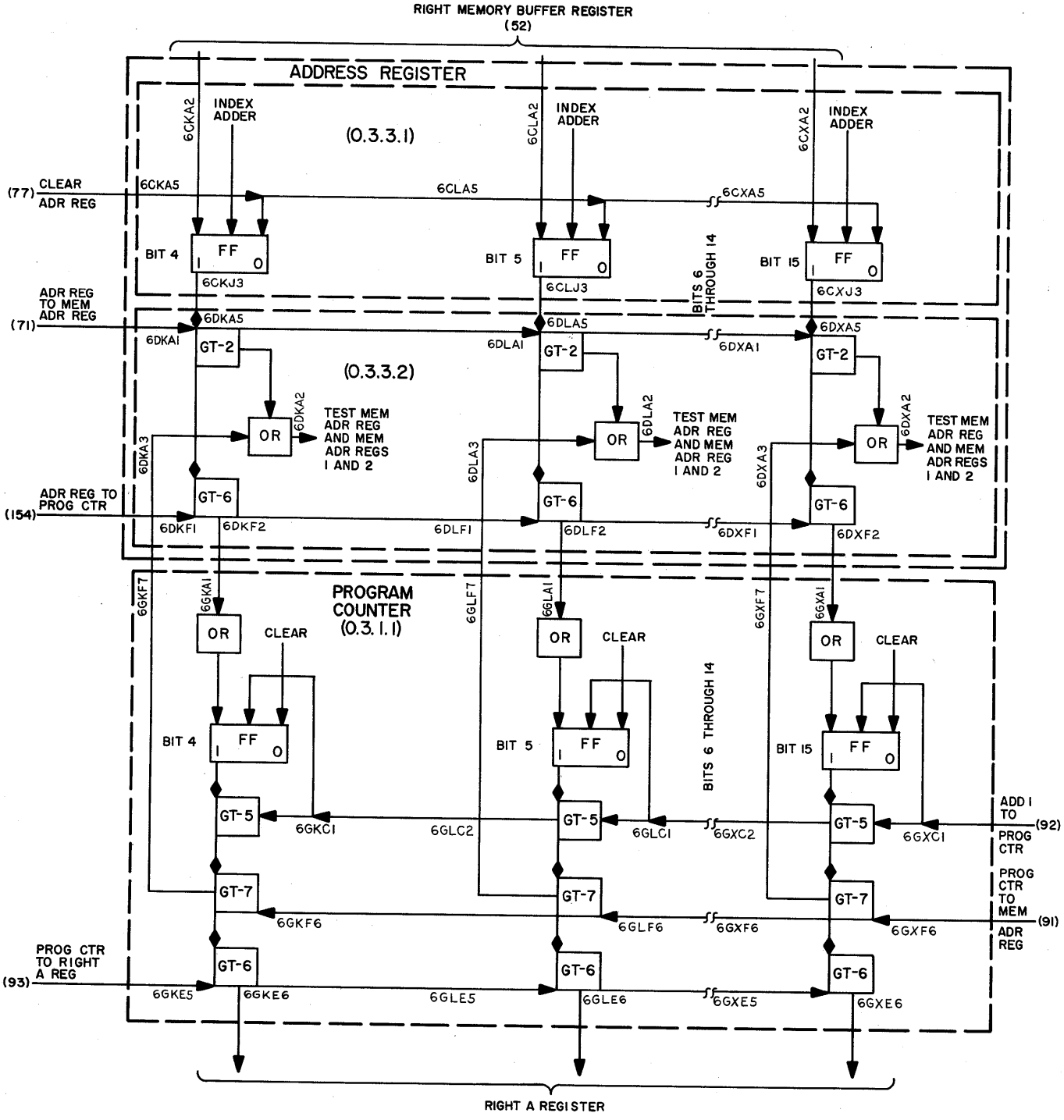


Figure 4–12. Memory Unit Selector, Internal Computer Operation, Logical Diagram

2.2 MEMORY ADDRESS SELECTION

During internal operations, both the address register and the program counter specify the address in core memory which will be used in instruction and operand transfer to and from the Central Computer System. As shown in figure 4–13, the right half of the instruction word (address) is transferred from the right memory buffer to the address register at PT–7 by command 52. The flip-flops in the address register store the word until it is transferred from the address register. Those flip-flops which are set to 1 by the word transferred into the address register condition their associated gate tubes. In the diagram, these are all GT–2's and GT–6's. At OT–1, the command *address register to memory address register* (71) pulses GT–2, thus transferring the address to any one of the three memory address registers, and thereby selecting a memory address from which to transfer an operand.

The program counter, which has the responsibility of program instruction sequencing, stores the instruction address in the flip-flop register. This address is automatically increased by 1 at the beginning of the instruction cycle so that the address of the next instruction is retained in the



COMMAND NUMBER	PULSE TIME
71	OT-1
77	PT-6
91	PT-1
92	PT-7
93	PT-11
154	PT-0

Figure 4–13. Memory Address Selection, Internal Operations, Logical Diagram

program counter. The command *add 1 to program counter* (92) pulses additive counter GT–5 and complements the bit 15 flip-flop, thereby increasing the contents of the flip-flop register by 1. The flip-flop register conditions GT–7's, which are pulsed at PT–1 by command *program counter to memory address register* (91). The instruction address is transferred to any one of the three memory address registers, thereby selecting a memory location containing an instruction word. This instruction word is transferred from core memory to the memory buffer register prior to PT–7, when the next instruction cycle starts.

2.3 INFORMATION FLOW DUE TO THE *BRANCH* INSTRUCTION

When a *Branch* instruction is executed, the information stored in the address register is transferred directly to the program counter by the command *address register to program counter* (154), occurring at PT–0. Previously (specifically at PT–11), the contents of the program counter were transferred to the right A register by the command *program counter to the right A register* (93). Simultaneously, the program counter was cleared in anticipation of receipt of the new address from the address register. When command 91 occurs at PT–1, this new address in the program counter is the one which will be transferred to the memory address registers. When command 92 occurs at PT–7, a 1 is added to this new address so that the program sequence continues, using the new address as the starting address for the new sequence.

SECTION 3

ALERTING THE CENTRAL COMPUTER SYSTEM FOR IO OPERATIONS

The *Load Address Counter (LDC), Select (SEL)* or *Select Drums (SDR),* and the *Read (RDS)* or *Write (WRT)* instructions alert and prepare the Central Computer System for word transfers. These instructions give the Central Computer System all the information necessary for transfers between external devices and core memory.

3.1 *LOAD ADDRESS COUNTER (LDC)* INSTRUCTION

The *LDC* instruction specifies the memory unit and the address in the unit which will be used in subsequent IO operations. The command sequence shown in table 4–3 illustrates information flow in the program element during the execution of this instruction.

TABLE 4–3. COMMAND SEQUENCE FOR *LDC* INSTRUCTION

TIME SEQUENCE	COMMAND NUMBER	COMMAND FUNCTION
PT–7	42, 52	Transfer left- and right-half words from left to right memory buffers to operation and address registers.
	92	Add 1 to program counter.
PT–9	114, 115, 212	Index register No. 1 or No. 2, or right accumulator to address register.
PT–0	31	Clear memory and test memory address registers.
PT–1	41, 53, 91	Clear left and right memory buffer. Program counter to memory address register.
PT–2	148	Clear IO address counter.
PT–3	72	Address register to IO address counter.
PT–6	77, 101	Clear address register, step counter, and operation.

It is evident from table 4–3 that the primary function of the *LDC* instruction is the loading of the IO address counter with the memory address to be used in subsequent IO operations. The relevant commands in this table will be discussed in the following paragraphs.

3.1.1 Memory Unit Selection for IO Operations

The circuits of the IO address counter which are involved in the selection of the memory units are shown in figure 4–14. The outputs of these memory unit selection circuits are fed to the various memory units, via the same OR circuits used by both the address register and the program counter. The selection codes and the circuits used by the IO address counter are identical with those used by the address register and the program counter to specify the various memory units. A detailed analysis of these circuits is given in 2.1 of this Chapter. With the exception of the numbers shown in the gate tube blocks, and command 143, which pulses the various gate tubes at PT–1, the analysis for the IO address counter is identical to the one in 2.1 of this Chapter.

3.1.2 Memory Address Selection for IO Operations

During IO operations, the IO address counter is the only register which specifies the core memory or test memory address to be used in word transfers to and from the external devices. As shown in figure 4–15, the address transferred from the right memory buffer to the flip-flops in the address register by command 52 conditions the GT–7's in the address register. When the GT–7's are pulsed simultaneously by the command *address register to IO address counter* (72) at PT–3, the address in the address register is transferred to the IO address counter, which was cleared at PT–2 in anticipation of receipt of this address. The contents of the IO address counter are transferred to one of the three memory address registers by command 143, which is generated by the selection and IO control element, via the same OR circuit used by the address register and the program counter memory address selection circuits. Each time a word is transferred, the command *step IO address*

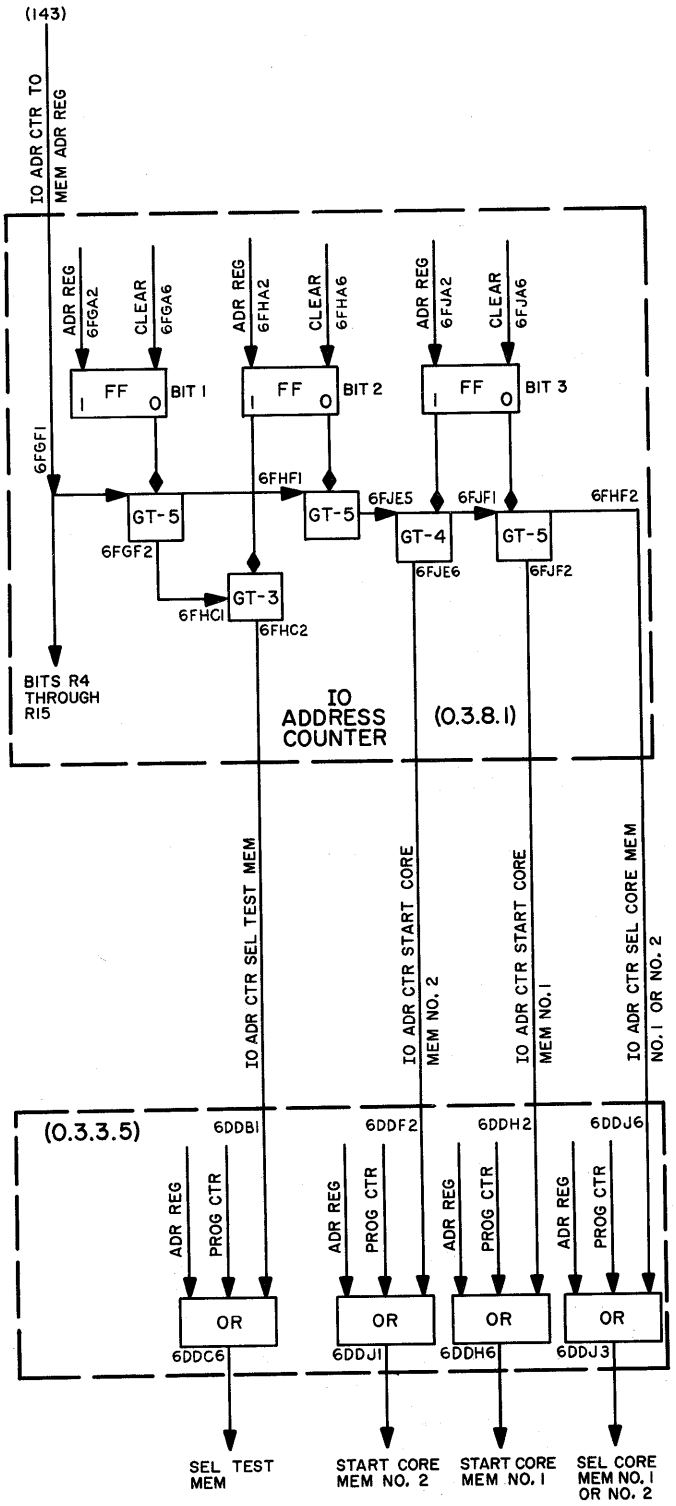


Figure 4–14. Memory Unit Selector, IO Operations, Logical Diagram

counter (324), generated by the selection and IO control element, pulses additive counter GT–3's and complements the bit 15 flip-flop, thereby increasing the contents of the flip-flop register by 1. The additive counter used in the IO address counter is discussed in Chapter 2 of this Part.

3.2 *SELECT (SEL)* OR *SELECT DRUMS (SDR)* INSTRUCTION

During IO operations, information being read out of drums, card machines, manual inputs, or the burst time counter is transferred to the Central Computer System by the way of the IO buffer registers. (See fig. 4–16.) This input information is fed to the flip-flops in the IO buffer register, thus setting these flip-flops, which, in turn, condition the GT–7's. When the command *IO buffer to IO register* pulses the GT–7's, the sign bits through bit 15 are simultaneously transferred to the IO register. This information is subsequently transferred to core memory via the memory buffer registers.

The words being transferred from the drums to core memory are selected from the drum by a search process, consisting of a comparison of the drum address or several bits of a data word (identity) with the contents of the drum control register, which stores the desired drum address or desired identity code bits. This comparison is performed between certain selected bits of the right IO buffer register and the corresponding bits of the drum control register. The desired drum address or identity code is contained in the address portion of the *SDR* instruction, which loads the drum control register. The command sequence shown in table 4–4 illustrates information flow in the program element during the execution of the *SDR* instruction.

The address contained in the right half of the *SDR* instruction word is transferred by command 52 from the right memory buffer register to the flip-flops of the address register, conditioning the GT–1's associated with these flip-flops. This address is then stored in the address register throughout the subsequent OT cycle until PT–3, at which time a transfer is effected. The OT cycle is used in both the *SEL* and *SDR* instructions, allowing the index interval matrix 6 microseconds to rise and settle before the remainder of the instruction is executed; i.e., to give the index interval matrix the necessary time to decode bits L10 to L15. These bits specify the IO unit to be used in IO operations. The drum control register is cleared by command 146 at PT–2; the address contained in the address register is transferred to the drum control register at PT–3 by command 151, thereby loading the drum control register.

COMMAND NUMBER	PULSE TIME
72	PT-3
77	PT-3
143	PT-1
324	BO-2 OR BI-2

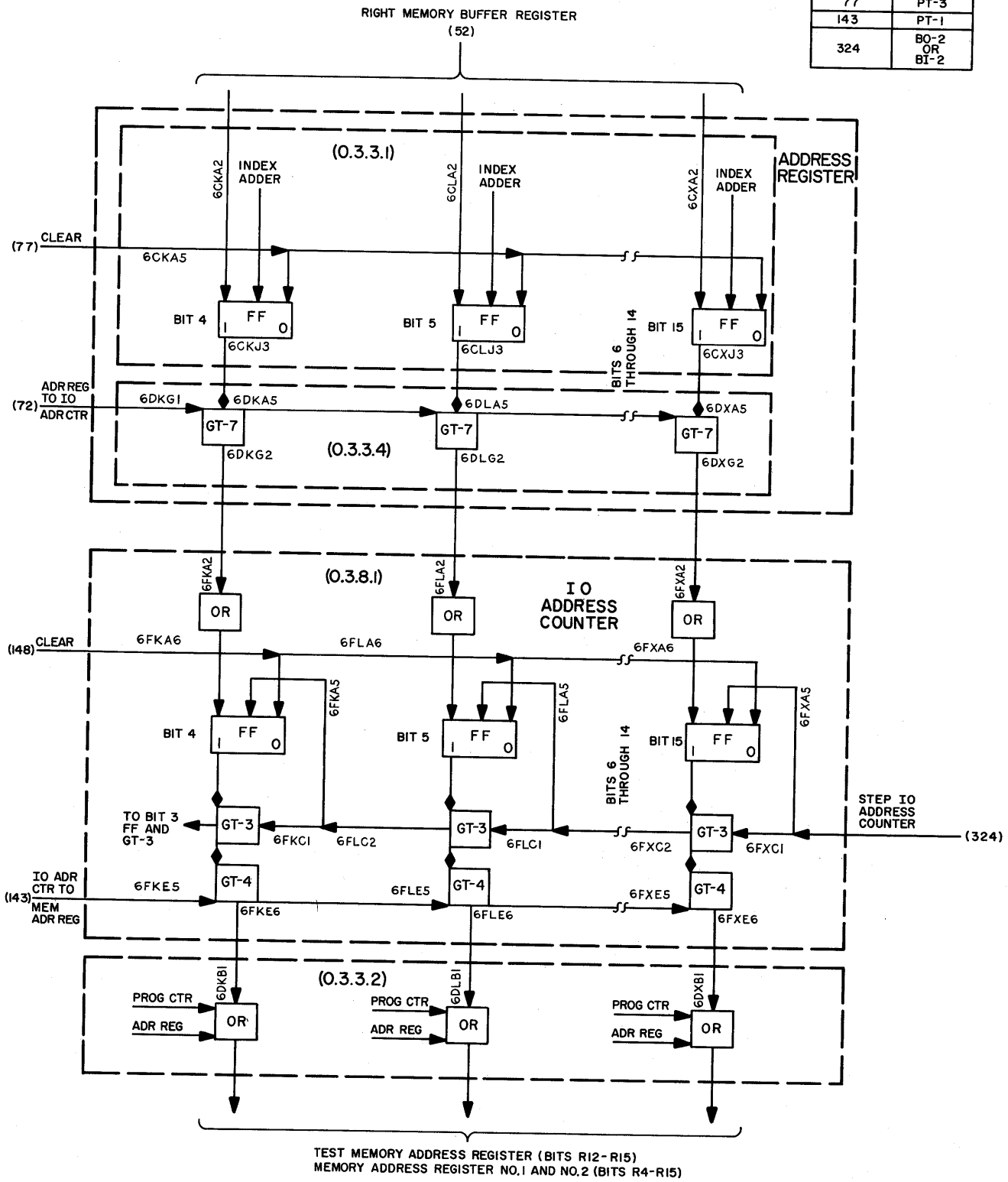
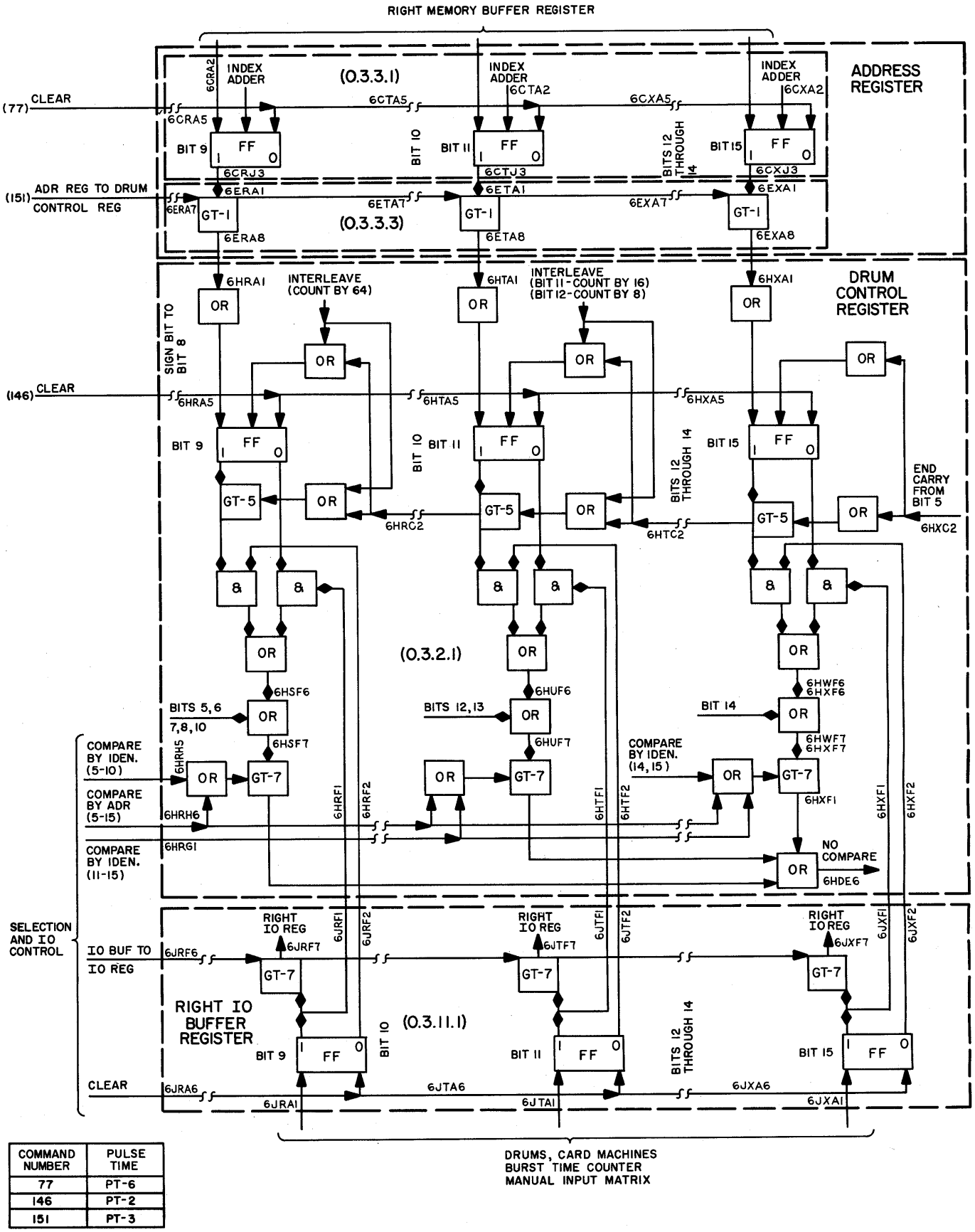


Figure 4–15. Memory Address Selection, IO Operations, Logical Diagram



COMMAND NUMBER	PULSE TIME
77	PT-6
146	PT-2
151	PT-3

Figure 4–16. Word Comparison, Logical Diagram

TABLE 4-4. COMMAND SEQUENCE FOR *SDR* INSTRUCTION

TIME SEQUENCE	COMMAND NUMBER	COMMAND FUNCTION
PT-7	42, 52	Left and right memory buffer to operation and address registers.
	92	Add 1 to program counter.
PT-9	114, 115, 212	Index register No. 1, No. 2, or right accumulator to address register.
PT-11	131	Set PT-OT flip-flop to OT.
OT-0	31	Clear memory and test memory address registers.
OT-1	41, 53	Clear left and right memory buffer.
	71	Address register to memory address register.
OT-11	161	Clear PT-OT flip-flop to PT.
PT-0	31	Clear memory and test memory address registers.
PT-1	41, 53	Clear left and right memory buffers.
	91	Program counter to memory address registers.
PT-2	146	Clear drum control register.
PT-3	151	Address register to drum control register.
PT-5	325	Select pulse for drums.
PT-6	77, 101	Clear address and operation registers.

For purposes of word comparison, drums have three modes of operation; the address mode, the identity mode, or the interleave mode. The address mode is used when words are to be read from or written into consecutive locations or addresses beginning at a specified address. The identity mode is used when words having a particular identification or a specific code are to be read from random locations or addresses. The interleave mode is used when words are to be read from or written into every 8th, 16th, or 64th location or address. The drum control register must be able to compare words when any one of these three modes of drum operation is in use.

In the address mode, the address register loads the drum control register with the drum address of the first word to be transferred in IO operations. Once this word is found and a successful comparison is made (the word in the IO buffer is similar to the desired word in the drum control register), no further words are compared. Words are now read consecutively from the drums into the Central Computer System until a disconnect signal is received from the IO word counter.

In the identity mode, the drum control register is loaded with the identifying code word; every word under the read heads of the drum is then compared with the coded word in the drum control register. The identity code may consist of either six, five, or two bits. These cases are illustrated in figure 4-16. Each word on the drum whose code is identical with the code in the drum control register is transferred into core memory. In this mode, therefore, every word is compared for the identifying code, whereas in the address mode, only the first word undergoes a comparison.

Once the desired address or identity code is loaded into the drum control register, the contents of this register are not stepped, and remain unchanged for the duration of the IO transfer. In the interleave mode, however, the drum control register is stepped, in order to count and compare every 8th, 16th, or 64th drum address. Figure 4-16 shows the interleave circuit inputs to bit 9 (count by 64), to bit 11 (count by 16), and to bit 12 (count by 8). Bits L13 through L15 of the index interval register, which is the part of the left half-word of the *SDR* instruction, determine which one of these three interleave modes is used. When one of the interleave circuit gates in the drum control register is pulsed by the command generators, the appropriate flip-flop in the drum control register will be complemented, thus changing the desired drum address in the drum control register. (See fig. 4-10.) The drums are now searched, and each word on the drum is compared with the word in the drum control register. A word transfer occurs when a successful comparison is made. Note that only bits R5 through R15 are used for comparison. (See fig. 4-16.) In addition, notice that bits 5, 6, 7, 8, 9, and 10 are combined, that bits 11, 12, and 13 are combined, and that bits 14 and 15 are combined, thus setting a d-c level which conditions the GT-7's of these respective groups. The drum control register also acts as an additive counter, with provisions for counting by 8, 16, or 64, to allow interleaved drum operations to take place. Drum comparison was discussed in detail in Chapter 2 of this Part.

3.3 *READ (RDS) OR WRITE (WRT)* INSTRUCTION

The address of the *RDS* or *WRT* instruction specifies the number of words to be transferred during IO operations. The command sequence

shown in table 4–5 illustrates information flow in the program element during the execution of the *RDS* instruction.

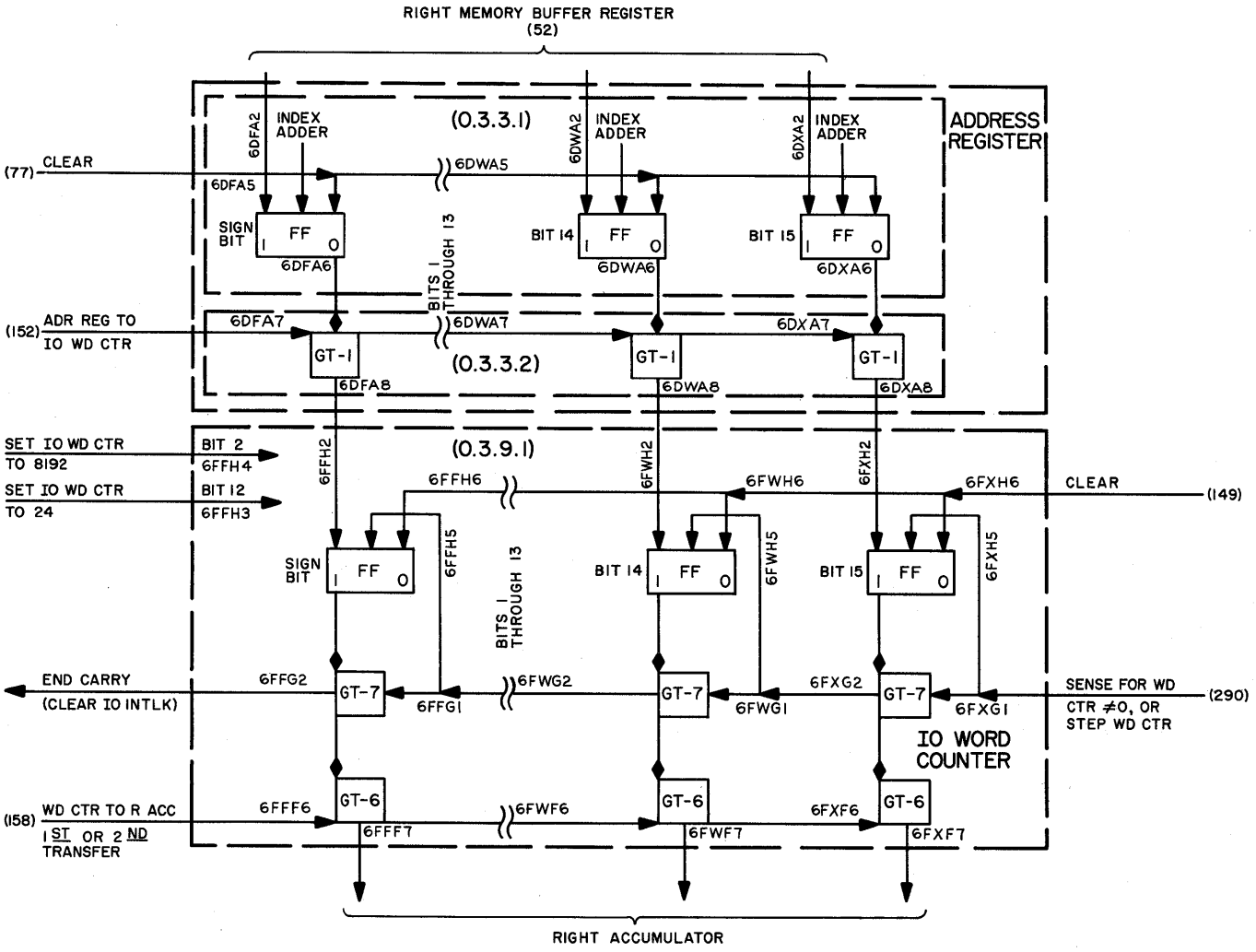
TABLE 4–5. COMMAND SEQUENCE FOR *RDS* INSTRUCTION

TIME SEQUENCE	COMMAND NUMBER	COMMAND FUNCTION
PT–7	42, 52	Left and right memory buffer to operation and address registers.
	92	Add 1 to program counter.
PT–9	114, 115, 212	Index register No. 1, No. 2, or right accumulator to address register.
PT–0	31	Clear memory and test memory address registers.
PT–1	41, 53	Clear left and right memory buffer.
	91	Program counter to memory address register.
	149	Clear IO word counter.
	294	Set IO interlock on.
PT–2	114, 147	Clear IO and right IO registers.
	152	Address register complement to IO word counter.
PT–3	290	Sense IO word counter $\neq 0$.
PT–6	77, 101	Clear address and operation registers.
	180	PT–6 on *RDS*.

The address contained in the right memory buffer register is transferred to the address register at PT–7 of each instruction. (At PT–9, the contents of the address register are modified by the contents of the index registers or the right accumulator register if the operation part of the word calls for an indexing operation.) At PT–1, the IO word counter is cleared and the IO interlock is set on, in preparation for an IO operation. At PT–2, the complement of the word is transferred from the address register to the IO word counter, thus loading the IO word counter with the complement of the number of words to be transferred. At PT–6, the command *PT–6* on *RDS* (180) is fed to the selection and IO control element and further gated, causing the transmittal of a start read pulse to the selected unit taking part in IO operations. If the specified IO device is the IO register, a break is requested for IO transfers.

The IO word counter is loaded only at PT–2 of a *RDS* or *WRT* instruction, when the command *address register complement to IO word counter* (152) pulses the GT–1's in the address register. (See fig. 4–17.) Since the 0 sides of the flip-flops in the address register condition the GT–1's, the number of words to be transferred during IO operations is placed in the IO word counter in complement form. At PT–3, the command *sense the word counter* (290) pulses the carry gate No. 7's, adding 1 to the IO word counter, and thereby reducing this complement by 1. Each time thereafter that a word is transferred into *(RDS)* or out of *(WRT)* the Central Computer System, carry gate No. 7 is stepped by command 290, further reducing this complement by 1. The next-to-last transfer in IO operations will cause each bit in the IO word counter to contain a 1. Since every flip-flop is now up, thus conditioning each carry gate No. 7, the step word counter pulse ripples through every GT–7 tube, generating an end carry as the last word is transferred. This end carry acts as a disconnect signal, clearing the IO interlock so that the next IO operation may be initiated. At the same time that the step word counter pulse ripples through the word counter, it also complements each flip-flop, resetting them all to 0.

Provision is made for setting the IO word counter to the complement of 23 when loading a program from cards, or for setting it to the complement of 8192 when clearing core memory. The *Clear and Subtract Word Counter (CSW)* instruction is used when the remaining number of words to be transferred before an IO operation is complete must be determined. If the IO word counter is not being stepped at PT–1, the command *IO word counter to right accumulator register* (150) effects the transfer of the sign bit through bit 15 to the right accumulator register. If the IO word counter is being stepped at PT–1, transfer from the IO word counter to the accumulator is effected at PT–5 by the command *1st or 2nd word counter transfer to right accumulator register* (158).



COMMAND NUMBER	PULSE TIME
77	PT-6
149	PT-1
152	PT-2
158	PT-1 OR PT-5
290	PT-3

Figure 4–17. Word Transfer Counter, Logical Diagram

SECTION 4
INSTRUCTION INDEXING

4.1 INDEX ADDER

The address register is used for instruction indexing, in conjunction with the index registers and the index adder circuits. Instruction indexing is a process whereby the address part of an instruction is modified after it is placed in and just before it is transferred out of the address register. The corresponding information in core memory, however, is not altered. When the address portion of an instruction is indexed, it can be used either in a repetitive (iterative) cycling loop, table look-up procedure, or for adding a constant to data. Indexing is performed when the original address transferred to the address register is modified by the contents of the index registers, changing it to a new address, so that cycling may take place without requiring a new instruction or additional space in core memory.

The required operations are carried out by the circuit shown in figure 4–18. Note that the gate tubes and the OR circuits in this diagram have been numbered functionally; i.e., gate tubes which perform identical functions, but which are associated with different bit positions, are all identified by the same number. In this discussion, therefore, reference to a numbered gate tube will also apply to all gate tubes having that same number.

Assume that index register No. 1 and the address register constitute complete four-bit registers. (See fig. 4–18.) Assume further that index register No. 1 contains the number 3 (0011), and that the address register contains the number 7 (0111). Figure 4–19 illustrates the sequence of events which will occur as the indexing process progresses. The first two columns show the status of the various flip-flops and the gate tubes which they condition, before the application of command 114. At step 3, command *index register No. 1 to address register* pulses GT–3. Each GT–3 which is conditioned feeds a pulse to OR–1 in the address register at 4. Those GT–3's which are unconditioned (down), such as the ones associated with bit 13 and the sign bit of the index register, have no output pulse feeding the address register. Transfer of bits to the address register for indexing purposes occurs when command 114 for index register No. 1, command 115 for index register No. 2, or command 212 for the right accumulator register takes place. (See fig. 4–18.) The OR–1 outputs in the address register are simultaneously fed to OR–2 and GT–7 at 5, so that the pulse passes through OR–2 and each conditioned GT–76 at 6. The conditioned gates are associated with bits 15, 14, and 13, although OR–1 of bit 13 receives no pulse from the index register, since that bit in the index register is a 0. (See fig. 4–19.)

The output pulses from the OR–2 circuit of bits 15 and 14 complement their respective flip-flops, thus clearing them from 1 to 0. The flip-flops settle to their new condition in approximately 0.5 microsecond (shown by the arrow from 6 to 17 on the sequence of events chart). Since a 1 from the index register has been added to a 1 in the address register, the address register flip-flop now contains the sum of the two, which is equal to 0, and the carry 1 has passed through GT–7 to the 0.5-microsecond delay circuit. When the flip-flops settle to their 0 level, both GT–7 and GT–9 are down, as indicated at 17 on the chart. The carry 1 from the 0.5-microsecond delay circuits of bits 14 and 15 are fed simultaneously, via the delay carry lines, to OR–2 and OR–3 of bits 14 and 13, as indicated at 18 on the chart. The pulse from OR–3 in bit 14 is fed to GT–9 at 19. Since GT–9 is down, as shown at 17, no pulse goes through. The pulse from OR–2 in bit 14 complements the flip-flop again, setting it from 0 to 1 in approximately 0.5 microsecond, as shown at 19, 20, and 30, so that GT–7 and GT–19 are up. The carry 1 from bit 15 has, therefore, complemented the flip-flop of bit 14 in the address register so that it now contains a 1.

The pulse from OR–3 in bit 13 is fed to GT–9 at 19. Since GT–9 is up (see 2), the pulse passes through GT–9 and is fed simultaneously, via the fast carry line, to OR–2 and OR–3 of the sign bit at 20. The pulse from OR–2 of bit 13 complements the flip-flop, clearing it from 1 to 0, as indicated from 20 to 30, so that GT–7 and GT–9 are down.

The carry 1 from bit 14 has, therefore, complemented the flip-flop of bit 13 in the address register so that the flip-flop now contains a 0.

The fast carry pulse passes through OR–2 of the sign bit, complementing the flip-flop and setting it from 0 to 1, as shown from 22 to 32 on the chart. The fast carry pulse to OR–3, however, does not generate a pulse output from GT–9, since GT–9 is down (see 2). The fast carry from bit 13 has complemented the sign bit flip-flop in the address register so that it contains a 1. The sum of the numbers which were originally contained in the flip-flops of the index register and the address register is now contained in the flip-flops of the address register. The sign bit flip-flop now contains a 1, bit 13 contains a 0, bit 14 contains a 1, and bit 15 contains a 0, so that the sum is equal to 1010 in binary form, or is equal to 10 in decimal form. Since the index register originally contained the number 3, and the address register contained address 7, the sum of the two is equal to 10, which agrees with the addition performed in the address register.

Two functions of instruction indexing are to be considered: cycling loops of instructions and table look-up procedure and adding a constant to data.

4.2 PROGRAMMED CYCLING LOOP

A cycling loop is developed when the same sequence of instructions must be used many times in the processing of certain types of data. Only those instructions which are indexable can participate in the process of instruction indexing.

Either index register No. 1 or index register No. 2 may be used for a data cycling loop. The right accumulator register, when it is used for indexing, cannot use the *BPX* instruction. It cannot, therefore, be used directly for cyclic data processing. The right accumulator register may be used, however, to reset either index register No. 1 or No. 2, and these index registers will then be used in a cycling loop. To illustrate the advantages of a cycling loop caused by instruction indexing, a program is laid out without, and then with the indexing feature, as shown in table 4–6. Assume in the table below that this program calls for the addition of 10 numbers; i.e., 10 pieces of data, which are stored in memory locations 200 through 209, and that the sum of the 10 numbers is to be stored in location 300.

The program in table 4–6 without the indexing feature starts with the instruction in memory location 100. This instruction calls for a *Clear and Add (CAD)* operation to be performed on the contents of memory location (address) 209. The *CAD* operation consists of transferring the data or number in address 209 to the right accumulator register via the memory buffer register, right A register, and right adder circuits, as is explained in Part 3.

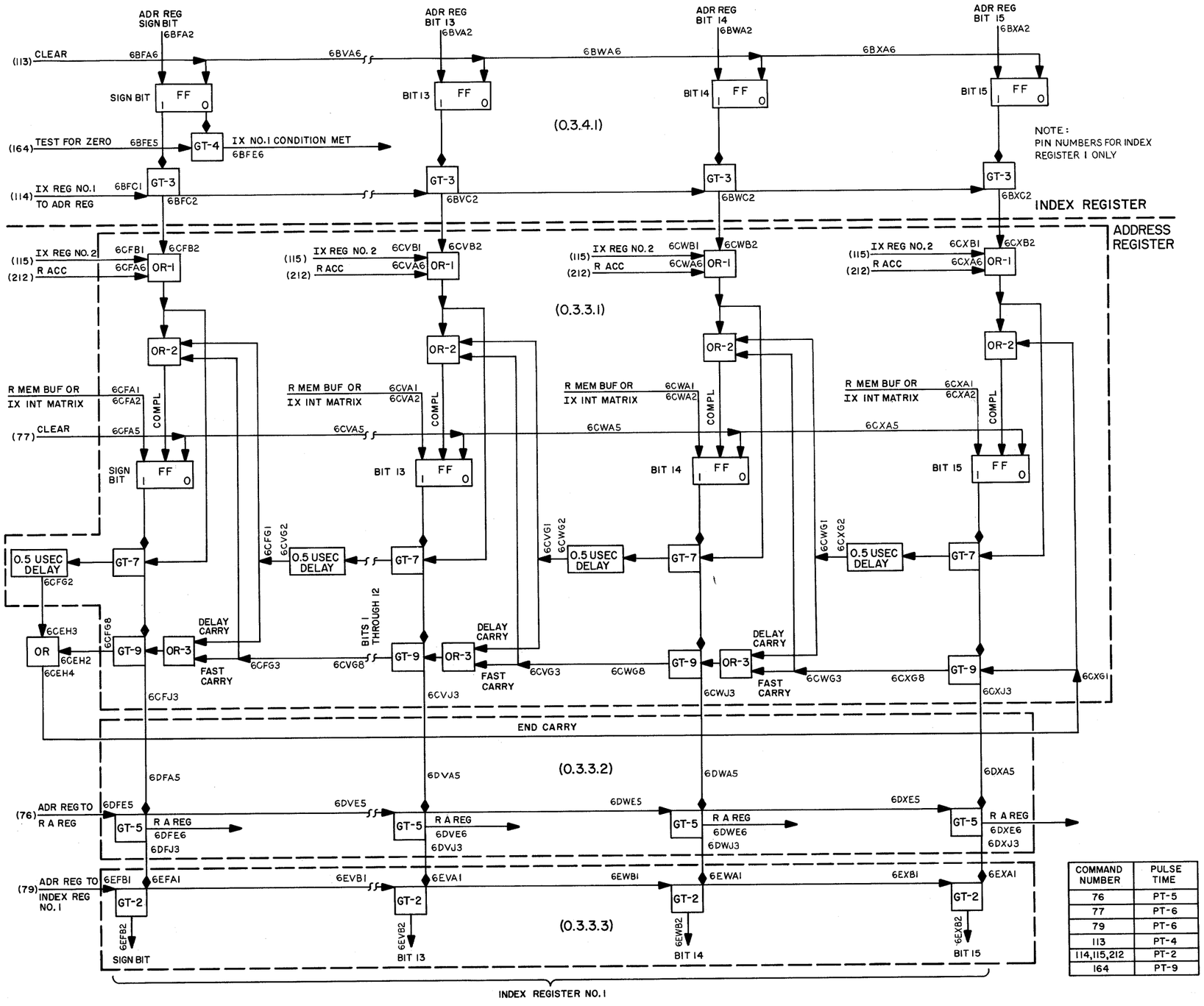
The next program step, which is in core memory address 101, calls for an add operation to be performed on the contents of core memory 208. The contents of address 208 are also transferred to the right accumulator register, and are added to the right accumulator register contents that were previously transferred from address 209. The program steps continue until the sum of the contents of core memory addresses 209 through 200 appears in the right accumulator register. The sum of all these numbers is then stored in core memory address 300. The add process is repeated nine times; 22 core memory locations are used for this program, 11 for storing the instructions, and 11 for storing the data utilized in arithmetic operations.

The program in table 4–6 which contains the indexing feature uses the *Reset Index Register (XIN)* and the *Branch and Index (BPX)* instructions to provide a cycling loop. The instruction commands and their time sequences are shown in table 4–7.

The *XIN* instruction word specifies, by bits L1 to L3, which one of the three index registers is to be used in a cycling loop. If bits L1 to L3 are zero, then no index register is specified.

At PT–7, the *XIN* instruction word is transferred from the memory buffer register to the instruction register. From PT–0 to PT–4, the various registers which will receive the bits of the instruction word are cleared. At the same time, a 1 is added to the program counter for the address of the next instruction, and this address is then transferred to the memory address register.

At PT–6, the address of the *XIN* instruction is transferred from the address register to the designated index register, and the address and operation registers are cleared for the next instruction. The address part of the *XIN* instruction is a number which specifies one less than the number of iterations that are to take place. If 10 numbers are to be added, the process of addition is repeated 10 times, once by the *CAD* instruction of the second step of the outlined program (memory address 100), and nine more times by the repetitive cycling process. The rule for this type of operation states, therefore, that if any number of



COMMAND NUMBER	PULSE TIME
76	PT-5
77	PT-6
79	PT-6
113	PT-4
114,115,212	PT-2
164	PT-9

Figure 4–18. Adder for Instruction Indexing, Logical Diagram

SEQUENCE OF EVENTS

UNIT	0	1	2	3	4	5	6	7 → 17			18	19	20	21	22 → 30 → 32				SUM
BIT 15		IX REG FF SET TO 1	IX REG GT-3 UP	COMMAND 114	GT-3 TO OR-1 IN ADR REG	OR-1 TO OR-2	OR-2 TO FF	COMPL FF FROM 1 TO 0		GT-7, GT-9 DOWN									0
		ADR REG FF SET TO 1	GT-7, GT-9 UP			OR-1 TO GT-7	GT-7 TO 0.5 USEC DELAY												
BIT 14		IX REG FF SET TO 1	GT-3 UP	COMMAND 114	GT-3 TO OR-1 IN ADR REG	OR-1 TO OR-2	OR-2 TO FF	COMPL FF FROM 1 TO 0		GT-7, GT-9 DOWN	DELAY CARRY TO OR-2	OR-2 TO FF	COMPL FF FROM 0 TO 1				GT-7, GT-9 UP			1
		ADR REG FF SET TO 1	GT-7, GT-9 UP			OR-1 TO GT-7	GT-7 TO 0.5 USEC DELAY				DELAY CARRY TO OR-3	OR-3 TO GT-9	NO PULSE OUT							
BIT 13		IX REG FF IS AT 0	GT-3 DOWN	COMMAND 114	NO PULSE OUT						DELAY CARRY TO OR-2	OR-2 TO FF	COMPL FF FROM 1 TO 0				GT-7, GT-9 DOWN			0
		ADR REG FF SET TO 1	GT-7, GT-9 UP								DELAY CARRY TO OR-3	OR-3 TO GT-9	GT-9 FAST CARRY LINE							
BIT S		IX REG FF IS AT 0	GT-3 DOWN	COMMAND 114	NO PULSE OUT								TO OR-2	OR-2 TO FF	COMPL FF FROM 0 TO 1			GT-7 GT-9 UP		1
		ADR REG FF IS AT 0	GT-7, GT-9 DOWN										TO OR-3	OR-3 TO GT-9	NO PULSE OUT					

INDEX REGISTER CONTENTS	0011
ADDRESS REGISTER CONTENTS	0111
SUM	1010

Figure 4-19. Index Adder, Sequence of Events

TABLE 4–6. COMPARISON OF CYCLIC AND NON-CYCLIC PROGRAMS

No Indexing Feature				With Indexing System				
PROG. STEP	MEM. LOC.	INSTRUCTION WORD		MEM. LOC.	INSTRUCTION WORD			
		OPERATION	ADDRESS		IX REG.	OPERATION	IX INT.	ADDRESS
1	100	*CAD*	209	99	1	*XIN*		8
2	101	*ADD*	208	100		*CAD*		209
3	102	*ADD*	207	101	1	*ADD*		200
4	103	*ADD*	206	102	1	*BPX*	1	101
5	104	*ADD*	205	(repeat 101 and 102 nine times)				
6	105	*ADD*	204	103		*FST*		300
7	106	*ADD*	203					
8	107	*ADD*	202					
9	108	*ADD*	201					
10	109	*ADD*	200					
11	110	*FST*	300					

iterations are to take place (nine in this case) one less than that number (eight in this case) is to be placed in the index register.

After the *XIN* instruction in core memory address 99 is executed, the *Clear and Add (CAD)* instruction in core memory address 100 is brought from core memory. The contents of core memory address 209 are then transferred to the memory buffer register, to the address register, and finally to the right accumulator register. Since no index register is specified, the contents of the index register are not added to the contents of the address register. The next program instruction (in memory address 101) specifies index register No. 1, to which will be added the contents of memory address 200. Index register No. 1 contains the number 8, which, when added to address 200 in the address register, produces a new address, 208. The *ADD* instruction now transfers the contents of address 208 to the right accumulator register, adding the contents of 208 to those of 209, which were previously transferred to the right accumulator register. The next program step brings the instruction in core memory address 102 into the memory buffer registers. This *Branch and Index (BPX)* instruction specifies index register No. 1 as well as an index interval of 1. Further, if the branching condition is met, a branch to address 101 will take place.

From the command sequence table of the *BPX* instruction, it is seen that the *BPX* instruction is transferred from the memory buffer registers to the operation and address registers at PT–7, and a 1 is added to the program counter. At PT–9, the specified index register sign bit is tested for 0. If the index register is negative zero, no branch takes place; but, if it is positive, a branch will occur and the branch flip-flop will be set. At the same time, the left and right A registers are cleared before the program counter contents are transferred to the right A register at PT–11. Since index register No. 1 contains a positive value of 8, a branch operation will take place to the address (core memory address 101) contained in the *BPX* instruction. The program counter is therefore cleared at PT–11, and the BPX address is transferred from the address register to the program counter at PT–0; the address register is then also cleared. At PT–1, the branch address in the program counter is transferred to the memory address register, and when the memory cycle takes place, the memory buffer registers are cleared in anticipation of receipt of the instruction from core memory address 101. At the same time, the complement of the index interval is placed in the address register. The contents of the index register, representing the number 8, are transferred to the address register at PT–2, and

TABLE 4-7. COMMAND SEQUENCE OF *RESET INDEX REGISTER* AND *BRANCH AND INDEX* INSTRUCTIONS

INSTRUCTION	TIME SEQUENCE	COMMAND NUMBER	COMMAND FUNCTION
Reset Index Register XIN	PT-7	42	Left memory buffer to operation register.
		52	Right memory buffer to address register.
		92	Add 1 to program counter.
	PT-0	31	Clear memory and test memory address register.
	PT-1	41	Clear left memory buffer.
		53	Clear right memory buffer.
		91	Program counter to memory address register.
	PT-4	113	Clear index register No. 1.
		116	Clear index register No. 2.
	PT-6	78	Address register to index register No. 2.
		79	Address register to index register No. 1.
		77	Clear address register and step counter.
		101	Clear operation register.
Branch and Index	PT-7	42	Left memory buffer to operation register.
		52	Right memory buffer to address register.
BPX		92	Add 1 to program counter.
	PT-9	21	Clear left A register.
		230	Clear right A register.
		164	Test index No. 1 register sign bit for 0.
		170	Set branch flip-flop.
		174	Test index register No. 2 sign bit for 0.
	PT-11	93	Program counter to right A register.
		94	Clear program counter.
	PT-0	154	Address register to program counter.
		31	Clear memory and test memory address register.
		77	Clear address register and step counter.
	PT-1	103	Index interval complement to address register.
		41	Clear left memory buffer.
		53	Clear right memory buffer.
		91	Program counter to memory address register.
	PT-2	114	Index register No. 1 to address register.
		115	Index register No. 2 to address register.
	PT-4	113	Clear index register No. 1.
		116	Clear index register No. 2.
	PT-6	78	Address register to index register No. 2.
		79	Address register to index register No. 1.
		77	Clear address register and step counter.
		101	Clear operation register.
		163	Clear branch flip-flop.

added to the complement of the index interval, which is the complement of the number 1. The complement of 1 reduces the number 8, therefore, to a value of seven. At PT–4, the index register is cleared, and at PT–6, the number 7 in the address register is transferred to the index register. At the same time, the operation and address registers, as well as the branch flip-flop, are cleared for the new instruction located at core memory address 101, which has been transferred to the memory buffer register. Since one *ADD* iteration has been executed, the index register contains a value of 7. The eight steps that follow are repetitions of the above procedure, with the program performing an *ADD* step, going on to core memory address 102, and then branching back to the next *ADD* step at core memory address 101. Every time that the *Branch* instruction is executed, the index register contents are reduced by 1, as specified by the index interval. Since the index register is reduced by 1 for each branch operation and is then added to the contents of the address register, address 200 is first added to 8 in the index register, then to 7, then to 6, etc., until 0 is reached in the index register. The contents of addresses 209 through 200 are, therefore, added consecutively to each other in the right accumulator register.

As long as the index register contents are positive, the branching process continues. When the index register is reduced to negative zero, (negative) i.e., when the *ADD* iterations are completed, since no further branch occurs, the program counter is increased by 1. The instruction brought down from core memory address 103 is a *Store (FST)* instruction which stores the sum of all the numbers in core memory address 300.

This program uses only 17 core memory locations; 5 for storing the instructions, and 12 for storing the data.

Even though the cycling program takes more time than the non-cycling program (because two cycles—*BPX* and *ADD*—are necessary for each addition step), the cycling program uses only 17 memory locations, as opposed to 22 required by the non-cycling program. If more memory space and the use of drums were necessary, the time needed for access to an auxiliary drum memory would be so great that more time would be consumed by a non-cyclic than by a cycling program. The latter program therefore saves considerable memory space and cuts down the number of instructions required in the writing of a program, thereby increasing the potential speed of the Central Computer System.

4.3 INDETERMINATE CYCLING LOOPS

A cycling loop may comprise an indeterminate number of repetitious operations, the actual number of which may be unknown to the programmer. This occurs when the number of iterations to be performed at the beginning of a specific program is unknown and has to be calculated by the Central Computer System. Table 4–8 provides a short program that illustrates how an index register may be set from the right accumulator register. This program will reset the index register from the right accumulator register with the number of words upon which iterative processes such as *ADD, MUL,* etc., must be performed.

The initial instruction is to read 8192 words from the drum, since the number of words to be read from the drums into memory may be unknown. The IO interlock goes on when the *Read* instruction is given. The next instruction, in core memory location 020, senses the IO interlock and branches to core memory location (address) 022 if the IO interlock is still on. Address 022 contains an instruction to branch unconditionally to 020. The program branches alternately between steps 2 and 4 until the drum makes one revolution, at which time a disconnect signal is generated by the drum, clearing the IO interlock. Program step 3 occurs when the IO interlock is cleared. The instruction in memory location 021 causes an unconditional branch to take place to core memory location 023, step 5 of the program. The *Clear and Subtract Word Counter (CSW)* instruction in core memory location 023 transfers the contents of the IO word counter to the right accumulator register.

The IO word counter initially contains the number of words to be transferred (in complement form) during IO operations. The *RDS* instruction places the complement of 8192 into the IO word counter; this number is reduced by 1 for each word transferred into core memory from the drum. After the word transfer is completed in this program, this reduced complement is transferred from the IO word counter to the right accumulator register. Program step 6 in core memory location 024 contains the instruction to add 8192 to the right accumulator register. The *ADD* instruction transfers 8192 from the address register to the right A register and adder circuits, where 8192 is added to the accumulator register contents previously transferred from the IO word counter. Since the right accumulator register contents are in complement form, these contents are subtracted from 8192. The difference of these two numbers equals the actual number of words that have been

transferred into core memory and which will, therefore, be involved in the cyclic data process that will follow in the next program.

Program step 7, in core memory location 025, contains the instruction to *Reset Index Register No. 1 from the Right Accumulator Register (XAC)*, since index register No. 1 is specified by the index indicator. The difference of the two numbers in the right accumulator register is then set into index register No. 1, which is now ready to take part in the cycling loop of program step 8. The timing sequence and the commands involved in the *XAC* instruction are shown in table 4–9.

At PT–7, the *XAC* instruction is brought from the left and right memory buffer registers to the operation and address registers. The instruction is decoded in the instruction matrix, and index register No. 1 is specified. Since the address of this instruction is meaningless, the address register is cleared at PT–9. At PT–10, the contents

TABLE 4–8. PROGRAM, *RESET INDEX REGISTER FROM RIGHT ACCUMULATOR*

PROGRAM STEP	MEMORY LOCATION	INSTRUCTION WORD		
		IX REG.	OPERATION	ADDRESS
1			*RDS*	(8192)
2	020		*BSN* IO Intlk.	022
3	021		*BPX*	023
4	022		*BPX*	020
5	023		*CSW*	
6	024		*ADD*	(8192)
7	025	1	*XAC*	
8	026		Program upon which iterations will be performed.	

TABLE 4–9. COMMAND SEQUENCE OF *RESET INDEX REGISTER FROM RIGHT ACCUMULATOR (XAC)*

INSTRUCTION	TIME SEQUENCE	COMMAND NUMBER	COMMAND FUNCTION
Reset Index Register from Right Accumulator Register (XAC)	PT–7	42	Left memory buffer register to operation register.
		52	Right memory buffer register to address register.
		92	Add 1 to program counter.
	PT–9	77	Clear address register and step counter.
	PT–10	212	Right accumulator register to address register.
	PT–0	31	Clear memory and test memory address registers.
	PT–1	41	Clear left memory buffer register.
		53	Clear right memory buffer register.
		91	Program counter to memory address register.
	PT–4	113	Clear index register No. 1.
		116	Clear index register No. 2.
	PT–6	78	Address register to index register No. 2.
		79	Address register to index register No. 1.
		77	Clear address register and step counter.
		101	Clear operation register.

of the right accumulator register are transferred to the address register. The address register contents are, in turn, transferred to index register No. 1 at PT–6, thus resetting the index register from the right accumulator register. The remainder of the commands in this instruction are ignored, since they are not relevant to this particular discussion.

4.4 TABLE LOOK-UP PROCEDURE

The instruction indexing process allows a programmer many options in the preparation of a program. The procedure given here is only one of many possible variations, affording means for quickly obtaining tabular material from core memory; it is appropriately termed a table look-up procedure.

Index register No. 1, No. 2, or the right accumulator register can be used for instruction indexing when a table look-up procedure is to be employed. The table which is to be referred to may be a table of trigonometric functions, a logarithmic table, etc. These tables must previously be set into core memory in such a manner that the memory address containing a specific value of a function differs from the value of its independent variable itself (for this value of the function) by a constant. The manner in which this is accomplished is illustrated by table 4–10.

In this case, the sine of an angle is located in a memory address which always differs from the angle itself by the factor 100. As an illustration, suppose that the range for an object is given by the expression c sin b, where c is a distance and b is an azimuth angle. Assume that c is 50 miles and b is 4 degrees. The table look-up procedure consists of finding the numerical value of sin b so that the expression may be evaluated by a simple multiplication process. If the angle is the result of a previous computation, it will appear in the right accumulator register; the right accumulator register is therefore clearly indicated as the index register to be used. The programming of a *Clear and Add (CAD)* instruction which specifies 100 as its operand address, and also specifies that index register No. 3 (right accumulator register) is to be used for indexing purposes, adds the angle (4 degrees) to the operand address (100); the result (104) is sent to the memory address register. The outcome is the appearance of the sine of 4 degrees in the accumulator register at the completion of the *CAD* instruction. It remains only to program a *Multiply (MUL)* instruction, specifying the address in which c (50 miles) is contained at the completion of this latter instruction, after which the evaluation of c sin b will appear in the accumulator register.

TABLE 4–10. TABLE LOOK-UP PROCEDURE, DATA STORAGE

MEMORY ADDRESS	CONTENTS
100	sin 0°
101	sin 1°
102	sin 2°
103	sin 3°
104	sin 4°
190	sin 90°

PART 5
SELECTION AND IO CONTROL ELEMENT

CHAPTER 1
INTRODUCTION

In the operation of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals, information must be transferred between the memory element of the Central Computer System and the various input-output (IO) units associated with the computer. These information transfers are called for by the program, but are instituted and co-ordinated by the selection and IO control element. The primary function of the selection and IO control element is to control, synchronize, and direct these transfers. Since this element performs only a control function, the actual words transferred do not pass through it. Figure 5–1 shows the information flow for data received by the Central Computer System from the IO units. Similarly, information flow for data supplied by the Central Computer System to the IO units is illustrated in figure 5–2.

Certain preparatory instructions must be performed before an actual transfer of words is initiated. These instructions supply the prior data necessary to institute the IO word transfers. The selection and IO control element assists in the execution of these instructions (which are in the IO class) and sets up its control circuits accordingly, so that the IO transfers may be properly effected.

In addition, the selection and IO control element incorporates circuits which allow the Central Computer System to perform certain operations, under the direction of the program, upon the electromechanical devices associated with the IO units. For instance, it provides a method of rewinding the magnetic tape units to their starting point or causing the printer to double-space words which are transferred to it. These electromechanical operations are called for by the program (the *Operate* instruction) and may be used

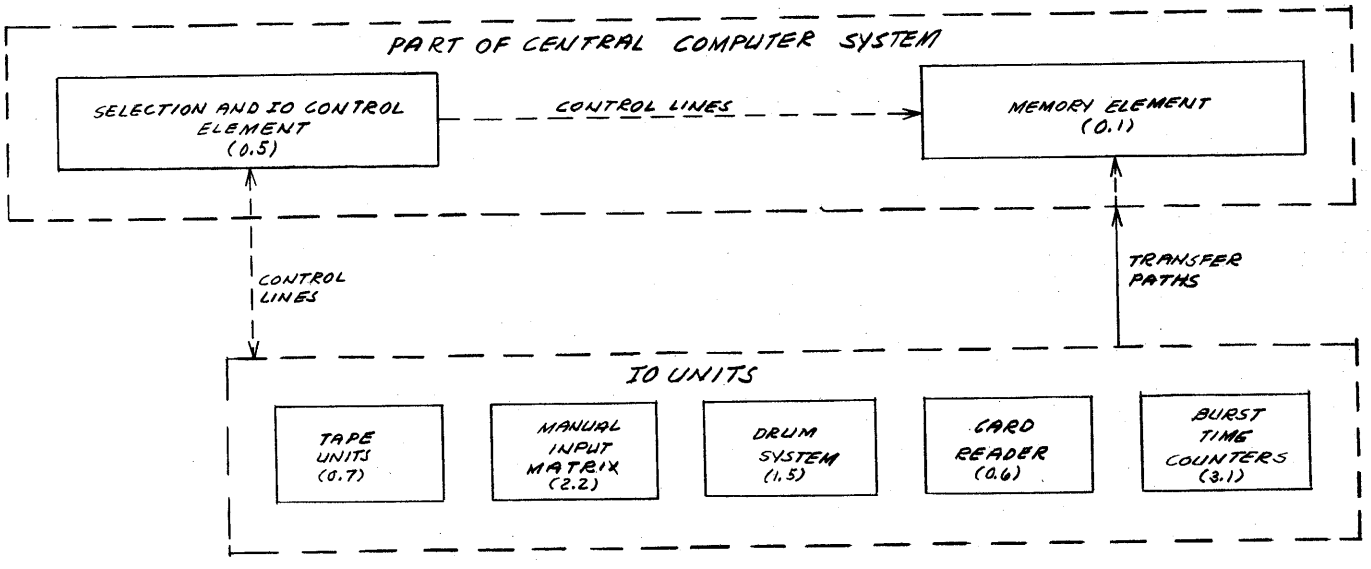


Figure 5–1. IO Unit to Central Computer System, Information Flow

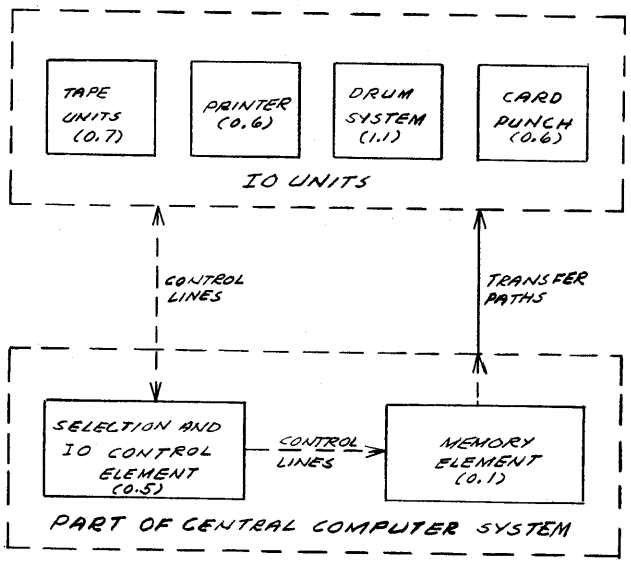


Figure 5–2. Central Computer System to IO Unit Information Flow

either in conjunction with IO transfers or separately. When utilized apart from IO transfers, these operations may be used for such purposes as illuminating neon lights on the maintenance console to indicate the course of the program, activating display cameras in the Display System, starting marginal checking operations, etc.

The selection and IO control element performs the additional function of determining existing conditions in AN/FSQ–7 (XD–1,–2) Combat Direction Centrals and directing the subsequent operations of the Central Computer System in accordance with these conditions. This function is accomplished by circuits which sense for particular conditions when so directed to do so by the program (the *Sense* instruction). In conjunction with IO transfers, this involves the determination of the mechanical or electronic preparedness of the IO units participating in the transfer of words. If the IO units are not prepared, these circuits will cause the Central Computer System to branch its program and thus avoid delaying other computer operations. For operations not associated with IO transfers, the selection and IO control element may sense for such conditions as the presence of a marginal checking excursion, a previous parity or overflow alarm, an error in the Output System, etc. The future course of the Central Computer System program is then directed in accordance with these conditions.

1.1 BLOCK DIAGRAM ANALYSIS

Figure 5–3 illustrates the circuits of the selection and IO control element in simplified block diagram form. Basically, this element consists of a register (index internal register), its associated decoding matrix (PerSelBsn matrix), and numerous control circuits. During the execution of any instruction pertaining to the operations discussed previously (IO transfer, operate, or sense), the index interval register will contain coded data specifying one of the following:

a. The IO unit with which a word transfer is to be effected

b. The electromechanical device, called an operate unit, which is to be activated

c. The condition which the Central Computer System is to determine as existing or not existing

Whenever the content of the index interval register is pertinent, the PerSelBsn matrix will decode the content of the index interval register and generate a single d-c level indicating this specific content. The output of the PerSelBsn matrix, which is the decoded information content of the index interval register, is fed to the control circuits of the selection and IO control element. In response to this decoded information and to appropriate command pulses from the instruction control element, the control circuits generate signals which perform the requisite operation. Thus, if a specific operate unit is to be activated, a command pulse received from the instruction

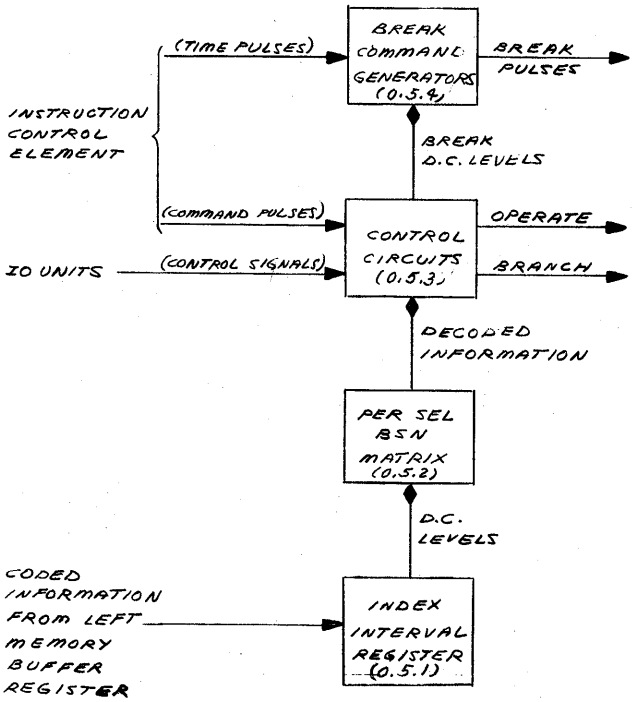


Figure 5–3. Selection and IO Control Element, Simplified Block Diagram

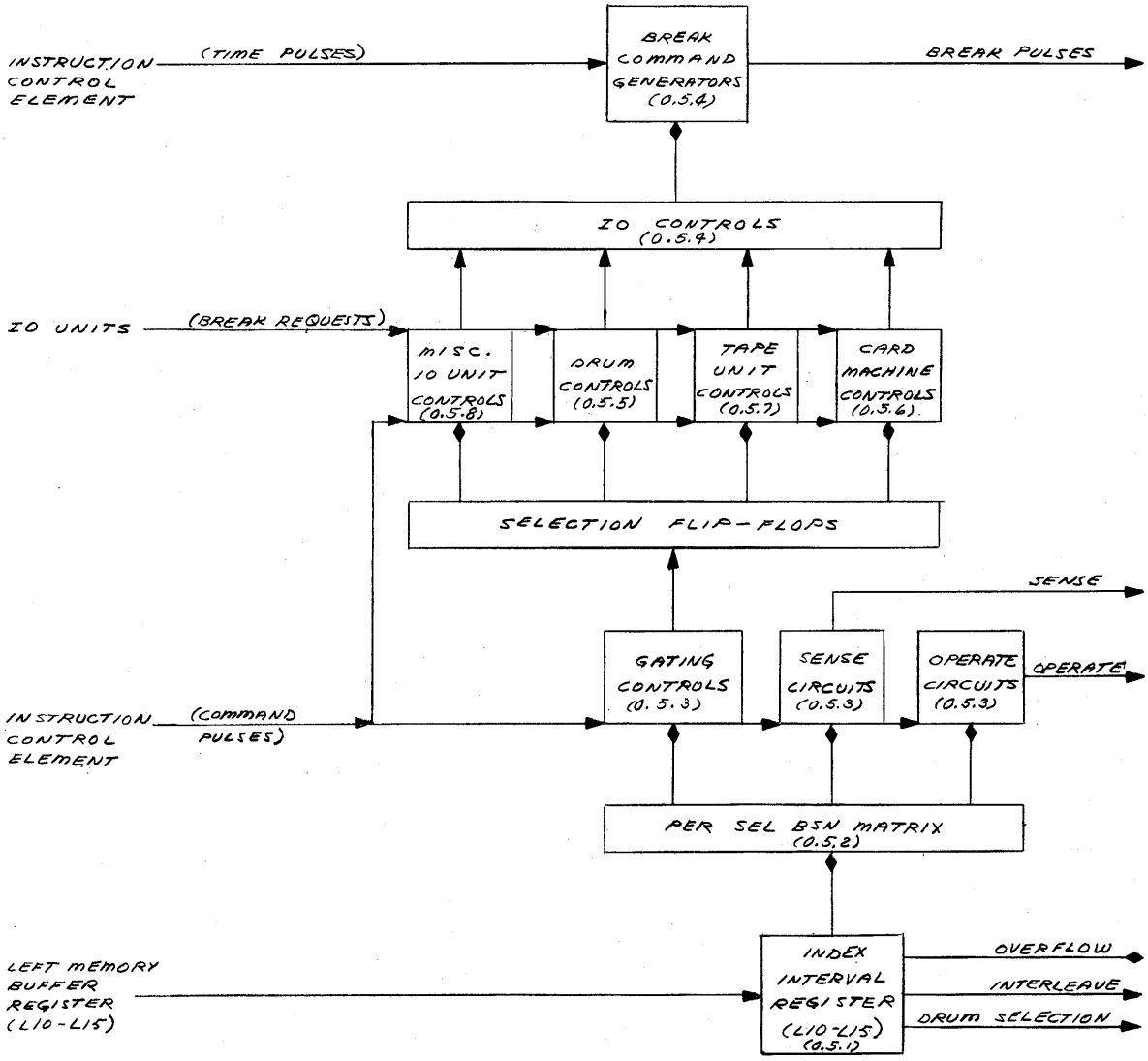


Figure 5–4. Control Circuits, Selection and IO Control Element, Block Diagram

control element will be propagated through the control circuits in such a manner as to appear as an activating signal which is sent to the proper operate unit. In a similar manner, the control circuits will combine a command pulse with the decoded information derived from the PerSelBsn matrix during a *Sense* instruction in such a manner as to sense for the presence or absence of a particular condition. If the condition is present, a command pulse will ultimately appear as a branch signal which, upon being sent to the instruction control element, will cause a branch in the program.

In controlling IO word transfers between the Central Computer System and the IO units, the decoded content of the index interval register, in conjunction with command pulses from the instruction control element, establish certain permanent conditions in the control circuits. When an IO unit is prepared to engage in a word transfer between itself and the Central Computer System, it will generate a signal (break request) which is sent to the control circuits of the selection and IO control element. This signal eventually results in the generation of a break d-c level. This level is applied to and conditions a set of break command generators which then gate time pulses from the instruction control element, to form break pulses. These break pulses initiate, propagate, synchronize and terminate the IO word transfer.

The circuits of the selection and IO control element are shown in block diagram form in figure 5–4. Essentially, the d-c levels of the PerSelBsn matrix drive three different sets of circuits: the gating controls, sense circuits, and operate circuits. These circuits are also driven by command pulses from the instruction control

element, which may be generated by either *Select, Select Drums Operate,* or *Sense* instructions. If an operate unit is to be activated, an operate pulse is gated through the conditioned operate circuits to appear as an operate signal which initiates the proper operate unit. If the existence of some condition is to be determined, a sense pulse will be applied to the sense circuits. If the condition is present, the sense pulse is gated to the instruction control element, where it sets the branch flip-flop and causes the program of the Central Computer System to branch. The sense circuits are also driven by other signal lines which come from the other elements of the Central Computer System and other systems of AN–FSQ–7 (XD–1, –2) Combat Direction Centrals; these are the signal lines which indicate the presence or absence of the various conditions for which the Central Computer System may sense.

When an IO unit is to be selected for participation in IO word transfers, a select pulse is gated through the gating controls to set the appropriate flip-flop in the selection and IO control element. These flip-flops store the information that a particular IO unit has been selected by the program, and also condition the appropriate set of IO unit control circuits.

Future instructions executed by the Central Computer System which pertain to IO word transfers will generate command pulses which are gated through the IO unit control circuits to set up the IO controls in the proper manner. When the selected IO unit is ready to engage in the transfer of a word, it will generate a break request signal which is gated through the IO unit control circuits to activate the IO controls. This results in the generation of a break d-c level which conditions the break command generators. The break command generators then generate break pulses which actually perform the word transfer.

It should be noticed that the contents of the index interval register are bits L10 through L15 of the instruction word. These bits, called the index interval, are received from the left memory buffer register of the arithmetic element. They specify an IO unit to be selected, a condition to be sensed, or an operate unit to be activated. The index interval bits sometimes determine the method by which words are to be transferred from or to the Drum System. In this case, they control an operation called interleaving. (Refer to Chapter 4.) When one of the fields of the Drum System is selected, the index interval bits are transferred to the drum selection register in the Drum System so that this system may activate the selected field. An additional function performed by the Selection and IO control element is control of Central Computer System action when an overflow or parity error occurs. In the case of overflow alarms, the index interval bits direct Central Computer System action by their coded content. These specific topics will be discussed in detail in ensuing sections.

1.2 IO WORD TRANSFER PATHS

The flow of words and the paths that they take between the Central Computer System and the IO units will be considered in this paragraph. These paths and the methods of transfer and control differ among the various IO units of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. The IO units are grouped in four categories, each of which differ widely in the manner and speed with which they process and transfer data. The basic problem of the selection and IO control element is to reconcile the inherent differences of the IO units with the characteristics of the Central Computer System. The various IO units are categorically listed below, in the order in which they will be considered:

a. Card machines
   1. Card reader
   2. Card punch
   3. Printer
b. Drum System
c. Tape units
d. Miscellaneous IO units
   1. Manual input matrix
   2. Burst time counters
   3. IO register
   4. Warning Light System

The card machines are unique in their operation in that they process and transfer two words at a time, whereas the other IO units process and transfer words singly. This miscellaneous category, except for the Warning Light System, consists of IO units which are only capable of transferring information to and are not capable of receiving information from the Central Computer System. The Warning Light System can only receive directions from the computer. All other categories are capable of transferring information in both directions, although a single IO unit in a category may only be capable of transferring information in one direction.

An IO unit generates a break request signal when it is prepared to accept or transmit a word (two words, or a word pair, in the case of the card machines). It is this break request signal

which actually initiates the transfer. When the Central Computer System receives and accepts a break request from an IO unit, it stops its arithmetic and programming operations for a length of time sufficient to completely transfer the requested word. This period of time is 6.0 microseconds, equivalent to one memory cycle, and is called a break cycle. The name is derived from the fact that there is a break in the internal operation of the Central Computer System while the word is being transferred. No mention is made in this paragraph of the control circuits and registers which co-ordinate and effect the word transfer during the break cycle; these will be fully considered in Chapter 2 and, hence, only the actual transfer paths between the Central Computer System and the IO units are discussed in this paragraph. A basic understanding of these transfer paths will greatly facilitate an understanding of the operational analyses to follow.

1.2.1 Break-In Transfers

When words are transferred from a selected IO unit to the Central Computer System, the machine is said to be reading. In this case, the break cycle during which each word is transferred is called a break-in cycle. The starting point for a break-in IO transfer is always the selected IO unit and the destination of the transferred word is always core memory. However, the path of the word between the IO unit and core memory varies for the various IO units of AN/FSQ-7 (XD-1, -2) Combat Direction Centrals. These break-in IO transfer paths are illustrated in figure 5-5.

The most direct transfer path is between the tape units and the Central Computer System. When either of these IO units generates a break request, the word to be stored in core memory will already have been placed in the IO registers of the arithmetic element. During the ensuing break-in cycle, the selection and IO control element will generate break-in pulses which will transfer the word from the IO register to the memory buffer register. The word is then stored in core memory by the action of the memory circuits. Thus, the IO operation of reading from the tape units is quite simple, insofar as the IO transfers are concerned.

The IO operation of reading from one of the fields of the Drum System is rather more complex, in that the word to be read is loaded into the IO buffer register of the program element rather than the IO register. Prior to accepting a word, the Central Computer System usually ascertains

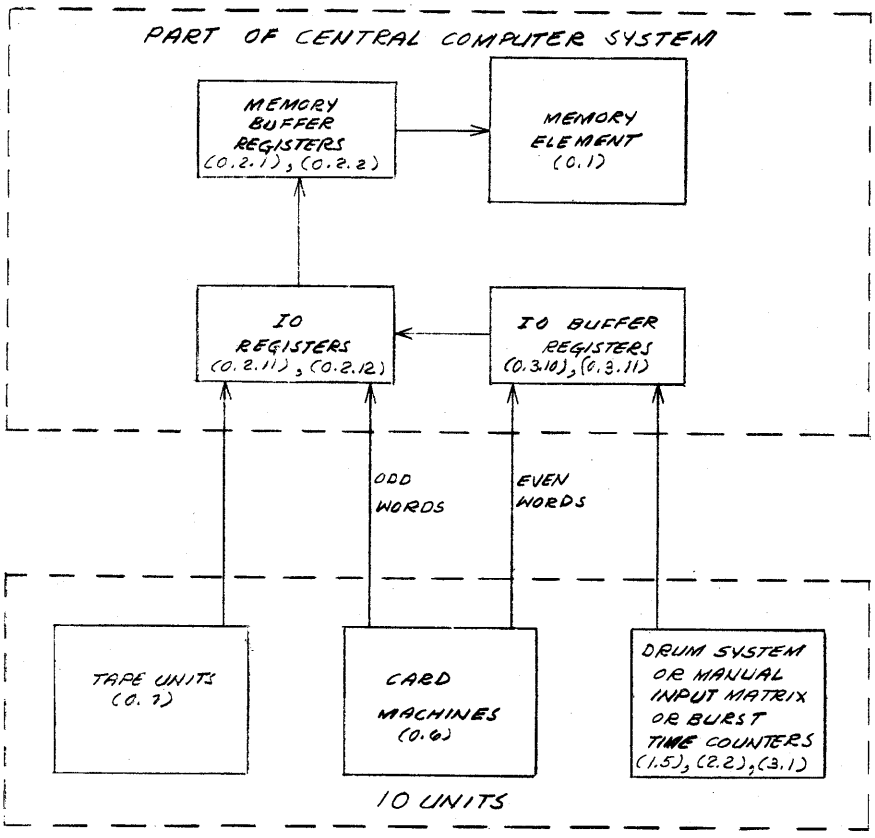


Figure 5-5. Break-In Transfer Paths

if it is the proper one either by status, identity or address. This identification or address search process is discussed in Chapter 4. It will be assumed that the word is to be accepted by the Central Computer System and stored in core memory. In this case, an IO buffer load pulse received from the Drum System will cause the transfer of the word from the IO buffer register to the IO register and institute a break-in cycle. During the break-in cycle, the word is transferred to the memory buffer register and stored in core memory.

The IO operation of reading from the manual input matrix or burst time counters is essentially the same. When the word is placed in the IO buffer register, a break request pulse is received by the selection and IO control element which transfers the word to the IO register and institutes a break-in cycle. The break-in pulses then generated transfer the word to the memory buffer register. It is important to note that a break-in cycle is never instituted until the word to be read has already been placed in the IO register from the IO buffer register.

The operation of reading from the card machines differs from the operations already mentioned, since these machines supply two words, instead of the usual single word. The card reader loads one word into the IO register and the other into the IO buffer register; these two words are called a word pair and require two break-in cycles to be stored in core memory. As the card reader transmits both words to their respective registers, it generates an index pulse, which is sent to the selection and IO control element. This index pulse acts as a break request and also sets an additional flip-flop (second break request FF), indicating that a second break-in cycle must be supplied. The first break-in cycle, initiated by the index pulse, transfers the word in the IO register to the memory buffer register and stores it in core memory. As soon as the IO registers are vacated by the first word, the second word is transferred from the IO buffer register to the IO registers. All this occurs during the first break-in cycle. Also, during the first break-in cycle, a second break request is generated as a direct result of setting the second break request flip-flop by the index pulse. This causes a second break-in cycle to be instituted immediately following the first. During this second break-in cycle, the second word, which is now in the IO register, is transferred to the memory buffer register and stored in core memory. Thus, an index pulse received from the card reader causes two break-in cycles to be instituted, during which the two words received from the card machine are stored in core memory.

1.2.2 Break-Out Transfers

When words are transferred from the core memory of the Central Computer System to a selected IO unit, the system is said to be writing. In this case, the break cycle during which each word is transferred is called a break-out cycle. In the case of writing on the card machines (card punch or printer), two thyratron buffer registers are interposed between the Central Computer

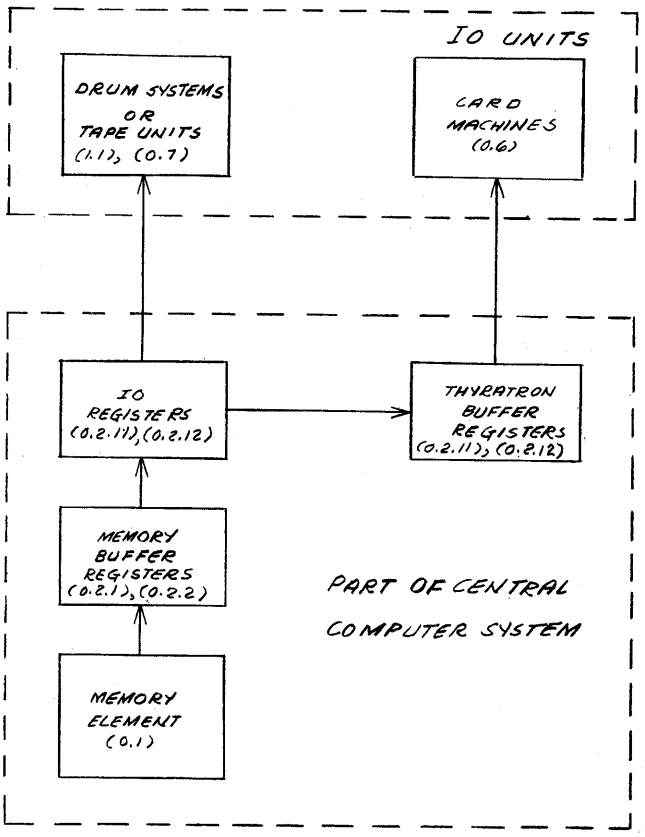


Figure 5–6. Break-Out Transfer Paths

System and the card machines. The thyratron buffer registers provide a means of energizing the proper set of printer or punch magnets of the selected card machine which actually performs the writing. Transfer paths during writing are shown in figure 5–6.

In writing on the tape units, the Central Computer System receives a break request pulse from the tape unit, indicating that it is prepared to write a word. This causes a break-out cycle to be instituted, during which break-out pulses, transfer the word to be written in from the memory buffer register to the IO register. The IO operation of writing on a field of the Drum System is similar,

except that an address search procedure is sometimes necessary prior to transfer to insure that the word will be written in the proper register of the selected field. When the Drum System is ready to write a word, it generates a word demand pulse, which is sent to the selection and IO control element and transfers the word in the IO register to the drum write register. The word demand pulse also generates a break request which institutes a break-out cycle during which a new word is obtained from core memory and transferred, via the memory buffer register, to the IO register. Thus, each word demand pulse transmits the word to be written from the IO register to the Drum System and institutes a break-out cycle during which the IO register is loaded with the next word to be written.

The IO operation of writing on the card machines is complicated by the fact that two words must be transferred to the card machine each time it generates an index pulse. In addition, each word must pass through a thyratron buffer register. This action requires that word transfers be delayed while the thyratrons stabilize. There are two thyratron buffer registers, one transmitting the first word to the card machines and the second transmitting the second word. When the card machine is prepared to write two words, it generates an index pulse which is sent to the selection and IO control element. This index pulse activates one thyratron buffer register so that the first word may be transferred through it. In order to allow the thyratrons to stabilize, a delay of about two machine cycles ensues before a break cycle is instituted. During this first break cycle, the first word is obtained from core memory and transferred through the memory buffer register to the IO register. As soon as the word is placed in the IO register, the activated thyratron buffer register will fire and energize the proper set of printer or punch magnets.

At the end of the first break cycle, the second thyratron buffer register is activated. The activation of each thyratron buffer register is under the control of a thyratron buffer control flip-flop which is set and cleared from the selection and IO control element. After activation of the second thyratron buffer register, another delay of about two machine cycles will ensue to allow time for the thyratrons to stabilize. A second break-out cycle is then instituted by the control circuits of the selection and IO control element, during which the second word to be written is obtained from core memory and transferred, via the memory buffer register, to the IO register. The transfer of the second word into the IO register will fire the thyratrons of the second thyratron buffer register, thus energizing another set of printer or punch magnets. Hence, the IO operation of writing on the card machines involves a system of double break requests and break-out cycles which transfer two words to the card machines.

1.3 BASIC IO PROGRAM

In the course of its program the Central Computer System may be called upon to operate in conjunction with any one of the IO units in AN/FSQ-7 (XD-1,-2) Combat Direction Centrals. This action is called for by the Central Computer System program and results in the setting up of the circuits of the selection and IO control element that were discussed in Section 2, Chapter 2. Words will be subsequently transferred between a selected IO unit and the Central Computer System when requested by the IO unit. This action of setting up the control circuits and carrying out the word transfers is called an IO process. This section discusses the instructions of the Central Computer System program which bring about an IO process.

In instituting and controlling an IO process with one of the IO units the circuits of the selection and IO control element must perform certain basic preparatory procedures at the direction of the program. In addition, certain basic information relating to the desired IO process must be supplied by the program. This programmed information must consist of the following minimum amount of data:

a. The address in core memory from which or to which the first word involved in the first transfer is to be transferred
b. The IO unit with which the Central Computer System is to be linked
c. Whether the Central Computer System is to read words from the IO unit or write words onto the IO unit
d. The number of words which are to be transferred

This information is supplied by a basic pattern of instructions in the program, which set up the proper control circuits and situate the information in the proper registers of the Central Computer System.

If the Central Computer System is to be linked with one of the 39 fields of the Drum System, the sequence of instructions would be:

A. *Load IO Address Counter (LDC)*
B. *Select Drum (SDR)*
C. *Read (RDS)* or *Write (WRT)*

If an IO unit other than a drum field is to function in conjunction with the Central Computer System, the sequence would be:

A. *Load IO Address Counter (LDC)*
B. *Select (SEL)*
C. *Read (RDS)* or *Write (WRT)*

It can readily be seen that the two sequences are identical except for the *Select* and *Select Drum* instructions, the latter replacing the former whenever a drum field is to be involved in the IO process.

The address portion of the *Load IO Address Counter* instruction contains the address in core memory from which the first transferred word is to be obtained or in which the first transferred word is to be stored. This address is loaded into the IO address counter in the program element, where it is stored until needed. The *Select* instruction designates which IO unit is to function during the IO operation. This designation is accomplished by bits L10 through L15 of the instruction. These bits are stored for the duration of the instruction in the index interval register of the selection and IO control element, where they are used to set up the proper control circuits. The address portion of the *Select* instruction is meaningless.

If the IO unit with which the IO operation is to be performed is a drum field, the *Select Drum* instruction is given instead of the *Select* instruction. In the same manner, designation of one of the 39 drum fields is accomplished by the index interval bits. It is often necessary to designate addressable fields: the address on the field from which the first transferred word is to be obtained or in which the first transferred word is to be stored. This designation of address is accomplished by the address portion of the *Select Drum* instruction, which is stored in the drum control register in the program element.

The final instruction, which is the instruction of execution, may be either a *Read* or *Write* instruction, and sends a start signal to the selected IO unit which initiates the actual transfer of information. If the selected IO unit is to transmit information to the Central Computer System, the *Read* instruction is appropriate; however, if the IO unit is to receive information from the core memory of the Central Computer System, the *Write* instruction is applicable. In addition to their function of initiating the IO operation, the address portions of the *Read* and *Write* instructions contain the number of words to be transferred during the subsequent IO operation. This number is loaded (in 1's complement form) into the IO word counter of the program element.

Prior to the institution of an IO operation by the *Read* or *Write* instructions, it is often advantageous to determine whether the IO unit is electronically or mechanically prepared to participate, or if any other condition exists in AN/FSQ–7 (XD–1,–2) Combat Direction Centrals which might impair the proper transfer of information between the Central Computer System and the IO unit. This may be accomplished by the *Sense* instruction, which may be inserted anywhere in the basic pattern of IO instructions except at the end. Usually, however, it is inserted immediately prior to the *Read* or *Write* instructions. If this sensing operation is desirable or necessary, the basic sequence of the IO program then becomes:

A. *Load IO Address Counter (LDC)*
B. *Select (SEL)* or *Select Drum (SDR)*
C. *Sense (BSN)*
D. *Read (RDS)* or *Write (WRT)*

If the *Sense* instruction is programmed and the IO unit selected is not ready to participate in the IO operation, it will cause a branch in control of the program to the instruction contained in the core memory address given by its address portion. This effectively prevents the execution of the initiating *Read* or *Write* instruction. The *Sense* instruction may sense for any one of a number of different existing conditions in the equipment. However, when utilized in an IO program, it will usually sense for the preparedness of the selected IO unit. The *Sense* instruction makes is possible for the Central Computer System to prevent the loss of caluculating time while waiting for the IO unit to be readied for participation in an IO operation.

Another instruction which may be inserted in the basic IO pattern is the *Operate (PER)* instruction. In general, this instruction allows the programmer to perform certain electromechanical operations on an IO unit prior to involving it in an IO operation. For instance, the *Operate* instruction allows the Central Computer System to rewind the tape units, alter the method of punching data onto punch cards with the card punch, or alter the format of printing data onto paper with the printer. Prior to activating an operate unit, it is usually necessary to sense for the preparedness of the unit. The basic IO pattern of instructions would then become:

A. *Load IO Address Counter (LDC)*
B. *Select (SEL)* or *Select Drum (SDR)*
C. *Sense (BSN)*
D. *Operate (PER)*
E. *Read (RDS)* or *Write (WRT)*

1.4 BASIC IO PROCESSES

A single transfer between an IO unit and the Central Computer System core memory is termed an IO operation. Under normal circumstances, the Central Computer System will be executing its program and performing the requisite calculations before, between, and after such transfers. When the selected IO unit is prepared to engage in an IO operation, it notifies the Central Computer System by generating a break request signal. This signal brings the internal operations of the Central Computer System to a halt for a length of time sufficient to effect the transfer. After the transfer is completed, internal operations and calculations are resumed until the next word is transferred. In this manner, the Central Computer System is not slowed down more than is absolutely necessary by the relatively slow word-processing speeds of the IO units. Thus, individual word transfers between the IO unit and the Central Computer System are interspersed between arithmetic operations. In the general case, an IO program will specify that a number of words be transferred rather than only a single word. Although the transfer of a single word is called an IO operation, the transfer of the total number of words required by the program is called an IO process.

Three major problems must be resolved in the course of an IO process:

a. Arithmetic and IO operations must not interfere with one another.
b. The originating point and destination of a word to be transferred must be determined prior to the initiation of the IO operation.
c. The IO process must be terminated as soon as the number of words specified by the *Read* or *Write* instructions have been transferred.

All of these problems are of a switching nature; the selection and IO control element provides the circuits for the requisite switching actions. By means of a break request circuit, it prevents conflict between IO operations and arithmetic calculations of the Central Computer System program. This circuit ensures that any arithmetic operation except a pause is halted if an IO operation is in process. Furthermore, by means of an IO interlock the Central Computer System is prevented from executing any programmed instructions which might affect the control circuits of the selection and IO control element when an IO process has been initiated but not yet completed. By controlling the IO address counter in the program element, the selection and IO control element enables the Central Computer System to calculate the memory address in core memory from which or to which a word is to be transferred in an IO operation. In conjunction with the drum control register of the program element, it also ensures that words transferred between the Central Computer and Drum Systems are taken from or stored in the correct register on the selected field. The requirement for terminating an IO process when all requisite words have been transferred is accomplished by the selection and IO control element and the IO word counter of the program element. Thus, all problems which might cause a conflict between the internal operation of the Central Computer System and an IO process are resolved by the selection and IO control element acting in conjunction with the other elements of the Central Computer System.

The IO program, consisting of the *Load IO Address Counter, Select* or *Select Drum,* and *Read* or *Write* instructions, only sets up the control circuits, provides necessary information, and sends a start signal to the selected IO unit; it does not cause the immediate transfer of a word between the Central Computer System and the IO unit. At PT_2–6 of the *Read* or *Write* instruction, a start signal is generated in the selection and IO control element and sent to the selected IO unit. Either of these instructions also sets the IO interlock and partially enables the break request circuit. The setting of the IO interlock will notify the Central Computer System that an IO process has been programmed, and will prevent execution of any instruction which might interfere with the IO operations (any instruction in the IO class). The break request circuit is a portion of the IO controls mentioned in Section 3 of Chapter 2 and shown in figures 5–5 and 5–6. Since the IO units are relatively slow devices compared to the Central Computer System, the latter will proceed with its program after the end of the *Read* or *Write* instruction.

When the IO unit is ready to take part in an IO operation (word transfer) it will generate a break request signal which is sent to the selection and IO control element and totally enables the break request circuit. As a result, all Central Computer System arithmetic and programming operations, with the single exception of an arithmetic pause, are stopped at the end of the machine cycle during which the break request is received. The memory buffer register is thereby made available to participate in the transfer of a word between core memory and the selected IO unit. The break request circuit also conditions the break

command generators so that break pulses are generated. (See figs. 5–5 and 5–6.) These break pulses are distributed to the other elements of the Central Computer System and effect the transfer of the word. They are generated for a period equal to the length of one machine cycle, 6.0 microseconds, at the end of which the IO operation is complete. The last break pulse generated is used to disable the break request circuit, thus ensuring that the Central Computer System may continue with its program and that no more time than is necessary to properly effect the word transfer will be utilized. The time during which break pulses are generated and a single word is transferred is called a break cycle. A single break cycle is required for every word transferred. At the end of each break cycle, the Central Computer System will return to its program unless another break request has been received, in which case another break cycle will immediately be executed.

At the start of any IO process, the IO address counter contains the address in core memory associated with the first word to be transferred, and the IO word counter contains the 1's complement of the number of words to be transferred. During the first break cycle instituted by the first break request received from the selected IO unit, the content of the IO address counter is transferred to the memory element, thus selecting the memory location associated with the first word transfer. During the break cycle, a one is added to the IO address and word counters by stepping pulses generated in the selection and IO control element. The stepping of the IO address counter increases its content by one, so that it then contains the core memory address associated with the second word to be transferred. The stepping of the IO word counter essentially reduces the magnitude of its content by one, since the IO word counter contains the number of words to be transferred in 1's complement form. This operation of transferring the content of the IO address counter to the memory element and stepping both the IO address and word counters takes place during each break cycle. In this way, each successive IO operation will effect the next higher address in core memory, and the number of words remaining to be transferred will always be correctly represented by the content of the IO word counter. Eventually the IO word counter will be stepped to positive zero, resulting in the generation of an end carry pulse which completely disables the break request circuit in the selection and IO control element. Since the break request circuit is completely disabled, future break requests from the IO unit will be ignored and no more break cycles will be instituted. A disconnect signal is sent to the IO unit at this time, the IO interlock is cleared, and the IO process is terminated.

Each IO device follows a variation of the fundamental process just described. These variations exist because of the different characteristics of the various IO units in the AN/FSQ–7 (XD–1,–2) Combat Direction Centrals. They differ from each other with regard to speed of operation, in manner of receiving or transmitting words, and in the type of information processed. In the Chapters which follow, each major category of IO units will be treated independently. The functions of the various instructions and command pulses will be restricted in each case to the category of IO units under discussion.

1.5 INPUT-OUTPUT TIMING

Two concepts, the break request and the IO pause, are fundamental to a discussion of the timing and synchronization of IO operations. A break request is usually generated by a selected IO unit whenever it is ready to receive or transmit a word. An IO pause occurs whenever the Central Computer System attempts to execute an IO class instruction while the IO interlock is on.

The paths of transfer for IO and arithmetic operations overlap to some extent in that both take place through the memory buffer register of the arithmetic element. This is unavoidable, since the only path available to core memory is through this register. In order to prevent the use of the memory buffer register and core memory by an arithmetic operation when an IO operation is taking place, a break request has been provided. This circuit so synchronizes the action of the Central Computer System with an IO operation that, during any memory cycle, the memory buffer register and the memory element are assigned exclusively to only one type of information processing; i.e., either IO or arithmetic.

Once a *Read* or *Write* instruction has been executed, the execution of all subsequent IO class instructions must be delayed until the existing IO process has been completed. If succeeding IO class instructions were not delayed, they would interfere with the control circuits of the selection and IO control element and associated circuits in other elements in such a manner as to destroy the conditions set up for the proper performance of the existing IO process. This must be avoided, since it would result in improperly controlled transfers.

An IO pause is therefore automatically introduced whenever an IO class instruction appears in the operation register of the instruction control element, if a previous IO process has not yet been terminated (as indicated by the status of the IO interlock).

1.5.1 Break Request

The timing of the Central Computer System is broken down into a continuous series of machine (also called instruction) cycles, each of 6.0-microsecond duration. A single machine cycle may be either a program time, operate time A, or operate time B machine cycle, and any given machine cycle will be immediately followed by another during the normal course of the Central Computer System program. In order to effect the transfer of a word during an IO operation, these machine cycles must be halted so that a break cycle may be instituted. This function of halting the normal Central Computer System machine cycle sequence and initiating a break cycle is accomplished by the break request circuit of the selection and IO control element.

If, during the course of a machine cycle, a break request is received from a selected IO unit, its reception is recorded by the setting of a break request flip-flop. This flip-flop is sensed by the Central Computer System by every TP–11 pulse and, if it is found to be set, a break cycle will be immediately instituted. Thus, the reception of a break request by the Central Computer System has no effect until the end of the current machine cycle during which it is received. At this time, a TP–11 pulse (or a 2-megacycle pulse during an arithmetic pulse), which is the last time pulse generated during any machine cycle, will determine that a break has been requested. The pulse will also cause the immediate institution of a break cycle, during which one word is transferred. During the break cycle, which is equal in length to one machine cycle, break pulses are generated which effect the transfer of a single word between core memory and the IO unit which generated the break request pulse.

The request by an IO unit for a break cycle takes precedence over any other impending machine cycle. Thus, whenever a break is requested by an IO unit during a machine cycle, the succeeding Central Computer System machine cycle will be delayed in its institution until one break cycle has been fully executed. Since none of the IO units of AN/FSQ–7 (XD–1,–2) Combat Direction Centrals are capable of transferring words at a rate of one word per 6.0-microsecond period, no more than one break request will be received by the Central Computer System during any machine or break cycle. Thus a single break request flip-flop, which is cleared as soon as a break cycle is instituted, is sufficient to record all unhonored break requests.

Since a break request may be generated by an IO unit at any time, it is possible that a break request may be received by the Central Computer System when it is in an arithmetic pause. Time pulses are not generated during a pause, and unless other provision is made, the Central Computer System will be unable to sense for the reception of a break request. For this reason, provision has been made to sense the break request flip-flop at a 2-megacycle rate (every 0.5 microsecond) during an arithmetic pause. If any 2-megacycle pulse finds that the break request flip-flop is set, a break cycle will be immediately instituted. Compared with the previous statements, this means that a break request received during an arithmetic pause is immediately honored, but a break request received during a machine cycle is not honored until the end of the machine cycle. It is important to note that the operations performed during the arithmetic pause continue even though a break cycle is being executed, and that if the arithmetic pause ends during the break cycle, the subsequent machine cycle will be delayed until the break cycle is also completed. The ability of the Central Computer System to honor break requests immediately during an arithmetic pause is made possible by the fact that the memory buffer register is not being utilized, and is therefore available for an IO transfer during a break cycle.

1.5.2 Input-Output Pause

Whenever a *Read* or *Write* instruction is executed by the Central Computer System, the IO interlock is set. This interlock affects only succeeding IO class or *Program Stop* instructions, and will prevent their execution by the Central Computer System. By placing the Central Computer System in an IO pause, the IO interlock prevents the instructions of any possible succeeding IO program from interfering with a current IO process. This IO pause occurs whenever an IO class or *Program Stop* instruction appears in the operation register of the instruction control element before the completion of the previous IO process.

The IO pause is similar to the arithmetic pause and utilizes essentially the same circuits. (See fig. 5–7.) Although the circuits are located in the instruction control element, they are included here since they have a direct effect on the action

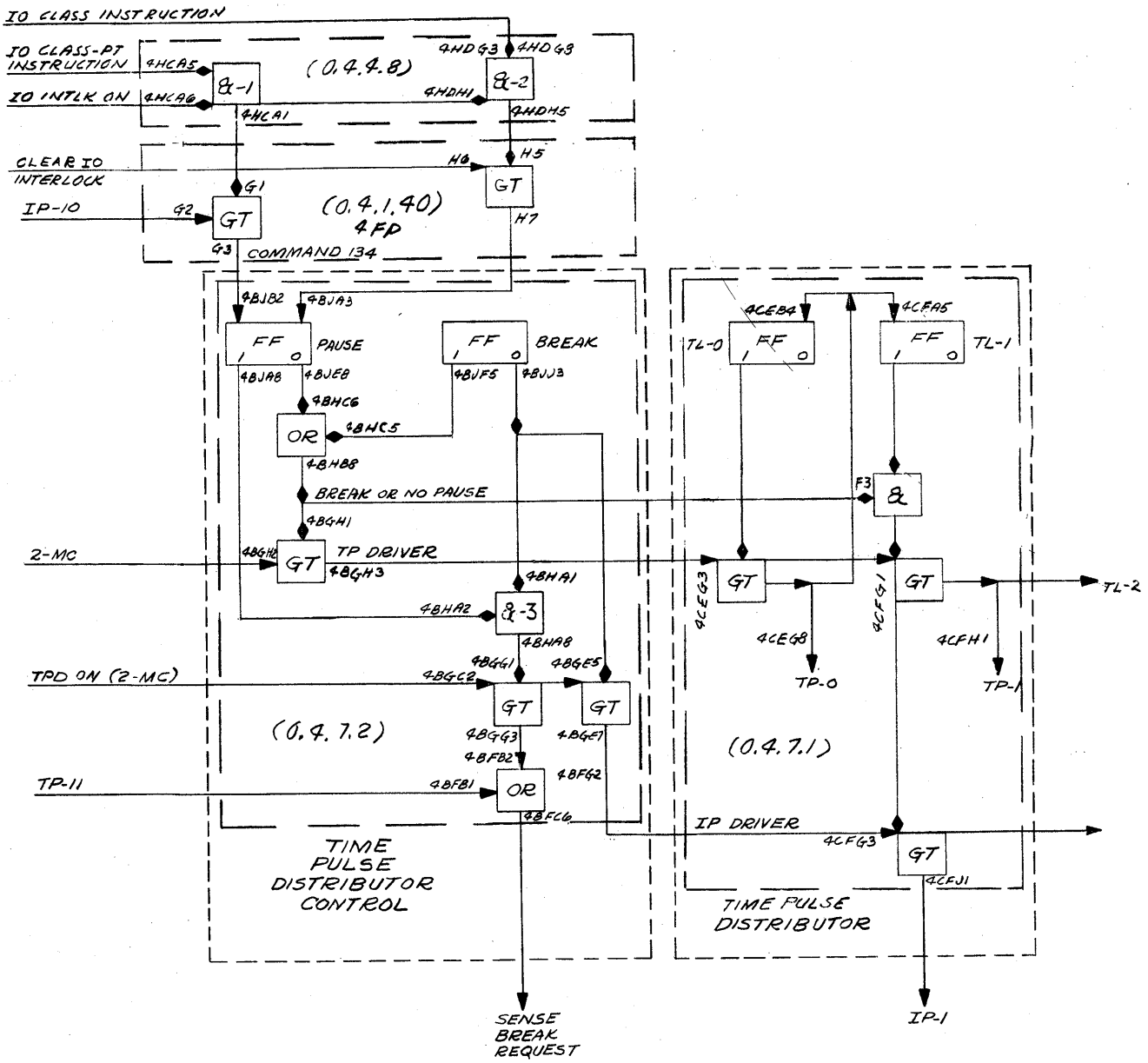


Figure 5–7. IO Pause Circuit, Logical Block Diagram

of the selection and IO control element. The break flip-flop shown here performs the same indicating functions as the corresponding break flip-flop in the selection and IO control element. In addition to controlling the IO pause, the circuits shown in the figure are responsible for the timing and synchronization of all IO processes, in conjunction with the break request circuit of the selection and IO control element.

It can readily be seen from figure 5–7 that the time pulse distributor is driven by two pulsed lines coming from the time pulse distributor control. One of these lines is called the TP driver line; it carries 2-megacycle pulses which cause the time pulse distributor to cycle from stage to stage and generate TP pulses. If the TP driver line is inactive, the time pulse distributor will not cycle and no TP pulses will be generated. The other line is called the IP driver line and results in the generation of IP pulses which are used to execute the instructions of the Central Computer System. It is important to note that IP pulses will be generated only if the TP driver line is active and TP pulses are being generated. Thus, TP pulses may be generated without IP pulses, but IP pulses must be generated coincident with TP pulses. Control of the IP and TP driver lines is accomplished by the break and pause flip-flops in the time pulse distributor control.

The break flip-flop shown in figure 5–7 indicates whether or not a break cycle is presently in progress. This flip-flop is set when a break cycle begins and is cleared when the break cycle ends. This setting and clearing action is accomplished by the break request circuit in the selection and IO control element. In the discussion to follow it will be assumed that the IO interlock has been set by some previous *Read* or *Write* instruction of an IO program. It will be assumed that an IO process is in progress although the Central Computer System may or may not, at any specific time, be engaged in an IO operation (break cycle).

Between break cycles and IO word transfers, the Central Computer System will continue to execute its program. The case to be considered now is the appearance of an IO class instruction in the operation register between break cycles. In this instance, the break flip-flop will be cleared, since no break cycle is being executed. The pause flip-flop will also be cleared, since the Central Computer System is not in an arithmetic pause. Since the IO interlock was turned on by the previously executed IO program, one input of AND circuit No. 1, shown in figure 5–7, will be conditioned. The appearance of the IO class instruction in the operation register will fully condition the AND circuit during its program time cycle. This will result in the conditioning of a gate tube which will be sensed by an IP–10 pulse and result in the setting of the pause flip-flop. This gated IP–10 pulse is the *pause* command (134). The setting of the pause flip-flop will decondition the gate tube feeding 2-megacycle pulses to the TP driver line and stop the time pulse distributor. Thus, no TP or IP pulses will be generated, and the Central Computer System will be unable to execute further the IO class instruction in the operation register. This is the IO pause condition.

It is characteristic of the IO pause condition that the Central Computer System is completely halted so far as its program is concerned. However, break cycles can still occur. Once put into an IO pause, the Central Computer System will remain in the state until the current IO process is terminated. It should be noted that during the IO pause AND circuit No. 3 will be fully conditioned by the 1 in the pause flip-flop and the 0 in the break flip-flop. This will gate 2-megacycle pulses to the selection and IO control element to sense for a break request, in lieu of a TP–11 pulse which is not being generated. When the current IO process is completed, a clear IO interlock signal will be received from the selection and IO control element. This pulse is gated through by the fully conditioned AND circuit No. 2 to clear the pause flip-flop. Thus, the circuits of the time pulse distributor control will be returned to their original states, IP and TP pulses will be generated, and the Central Computer System will continue with the execution of the IO class instruction in the operation register.

In the most general case, the Central Computer System will receive break requests from the IO units when it is in neither an arithmetic nor an IO pause. Thus, prior to the reception of the break request, both the pause and break flip-flops will be cleared. In this case, both the TP and IP driver lines will be conditioned. AND circuit No. 3 will be deconditioned by the 0 in the pause flip-flop and every TP–11 pulse will be sent to the selection and IO control element to sense for a break request. When the IO unit does request a break, it will be detected by this TP–11 pulse and the break flip-flop will be set. The setting of the break flip-flop will decondition the IP driver line, stopping the generation of IP pulses, and preventing the further execution of the instruction presently in the operation register. However, TP pulses will continue to be generated and will be sent to the selection and IO control element, where they ultimately perform the functions and operations required during a break cycle. When the break cycle is completed the break flip-flop will be cleared; this will cause the time pulse distributor to again generate IP pulses, and the Central Computer System will continue with its program.

The operation of instituting a break cycle when the Central Computer System is in an arithmetic or IO pause is slightly different from the operations just discussed. This modification is necessary since neither IP nor TP pulses are available during a pause, and the TP pulses are necessary to execute the break cycle. Prior to the reception of a break request during an arithmetic or IO pause, the break flip-flop will be in the 0 state and the pause flip-flop will be in the 1 state. The 1 in the pause flip-flop will prevent the activation of the TP driver line; the time pulse distributor will therefore be stopped. In this case, AND circuit No. 3 will be conditioned and 2-megacycle pulses will be gated to the selection and IO control element to sense for a break request. When a break request by an IO unit does occur, the break flip-flop will be immediately set. This energizes the TP driver line, and TP pulses will be generated for the execution of the break cycle. However, the 1 in the break flip-flop will decondition the gate tube feeding the IP driver line so that the Central Computer System will still be unable to execute

any instruction. When the break cycle is completed, the break flip-flop will be cleared, but the 1 in the pause flip-flop will continue to prevent the generation of TP and IP pulses; the Central Computer System will therefore continue in the pause condition.

The salient points of the previous discussion are summarized in table 5–1. It should be understood that the break flip-flop is only set after a break request from an IO unit and remains set only for the duration of the break cycle. Similarly, the pause flip-flop is only set during the arithmetic pause instituted in the Central Computer System by certain instructions, or during an IO pause when an IO class instruction is in the operation register when the IO interlock is on. It is important to note that IP pulses are only generated when both the break and pause flip-flops are in the 0 states. In a similar manner, TP pulses are consistently generated, except when the Central Computer System is in the pause and no break condition. It is only in this condition that the Central Computer System senses for break requests at a 2-megacycle rate; at all other times a TP–11 pulse performs the sensing operation.

TABLE 5–1.
BREAK AND PAUSE CHARACTERISTICS

PAUSE FLIP-FLOP STATUS	BREAK FLIP-FLOP STATUS	TP PULSES	IP PULSES	BREAK CYCLE	INSTRUCTION BEING EXECUTED
0	0	Yes	Yes	No	Yes
0	1	Yes	No	Yes	No
1	0	No	No	No	No
1	1	Yes	No	Yes	No

1.6 DECODING

Every action performed by the selection and IO control element depends on a decoding procedure involving the index interval register and the PerSelBsn matrix. By means of this procedure, the intent of the programmer (as expressed by the index interval of a *Select, Select Drum, Sense,* or *Operate* instruction) is interpreted by the Central Computer System. Interpretation consists of conditioning a set of gate tubes which control the appropriate circuits. In this discussion attention will be directed to the mechanics of transforming the index interval codes into a set of conditioned gate tubes.

1.6.1 Index Interval Register

The circuits associated with the index interval register are illustrated in figure 5–8. The information to be held by this register is received from the left memory buffer register at PT_1–7 of the particular instruction being executed. The significance of the index interval bits is a function of the instruction with which they appear. The d-c levels generated by the flip-flops of the index interval register control the action of three groups of gate tubes and also energize the PerSelBsn matrix. The first set of gate tubes are effective only during a *Branch and Index* instruction. The *index interval complement to address register* command (103) of this instruction transfers the 1's complement of the index interval to the address register in the program element. The second group of gate tubes is provided in order to transfer the contents of the index interval register to the drum selection register in the Drum System whenever the *select pulse for drum* command (325) of the *Select Drum* instruction is issued. This transfer will supply the Drum System with the identity of the drum field selected.

A third decoding procedure, employing the third set of gate tubes associated with the index interval register, provides information concerning drum interleave operations. The *PT–6 on read* command (180) of a *Read* instruction or the *PT–6 on read or write* command (182) of a *Write* instruction examines the three gate tubes in this group. A 1 in any of the last three flip-flops in the index interval register indicates that some interleaving method has been specified. If such a case exists, command 180 or 182 will produce a signal which will set the interleave flip-flops in the selection and IO control element and the program element. The d-c levels from the 1 sides of the flip-flops for bits 14 and 15 are utilized to specify the action to be taken should an accumulator overflow condition arise. The major function served by the index interval register in the decoding process, however, is to condition the PerSelBsn matrix.

1.6.2 PerSelBsn Matrix

As soon as the index interval register has settled, its resultant d-c levels drive the PerSelBsn matrix. It is a feature of the design of this matrix that, during any one instruction, only one of its outputs will be positive. The matrix is composed of 63 six-way AND circuits; each AND circuit provides one of the 63 outputs of the matrix. Not all of these outputs are in use at the present time. Seven levels are reserved as spares and four levels condition gate tubes whose output pulses perform no useful function, but have been provided for future expansion. A portion of the PerSelBsn

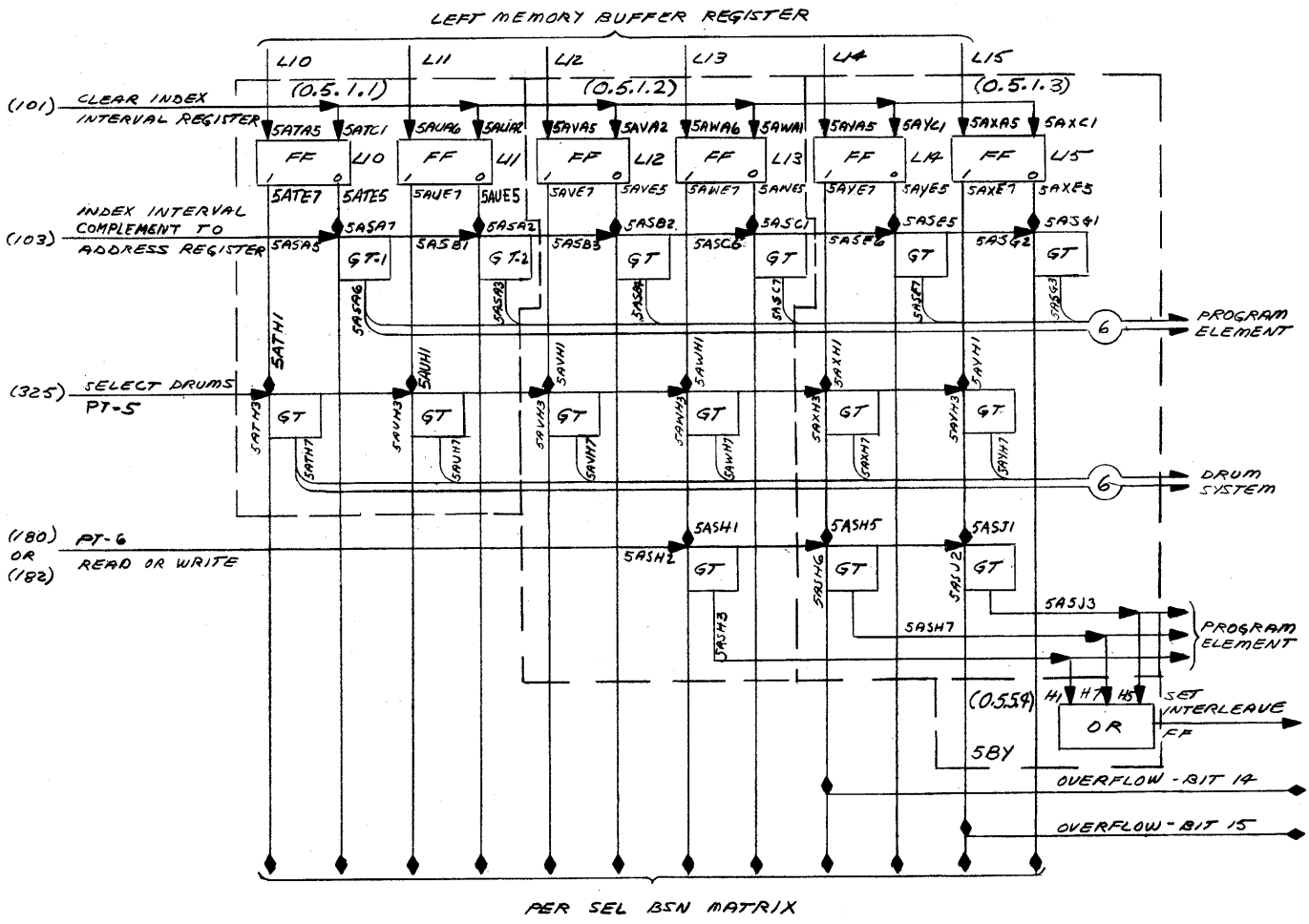


Figure 5–8. Index Interval Register, Logical Block Diagram

matrix is shown in figure 5–9. In order to obtain a positive output level for a specific index interval, the corresponding levels from the index interval register are mixed in a six-way AND circuit. Each possible combination of the index interval bits has a different AND circuit assigned to it exclusively.

Because several index interval codes are duplicated for different instructions, the gate tubes conditioned by a single output line of the PerSelBsn matrix are not all used for the same instruction. The distinction as to which gate tubes are to be used for a given instruction is made by the command pulses originating in the instruction control element. During any particular instruction, only those conditioned gate tubes which pertain to the instruction are pulsed. The other gate tubes conditioned by the PerSelBsn matrix remain dormant.

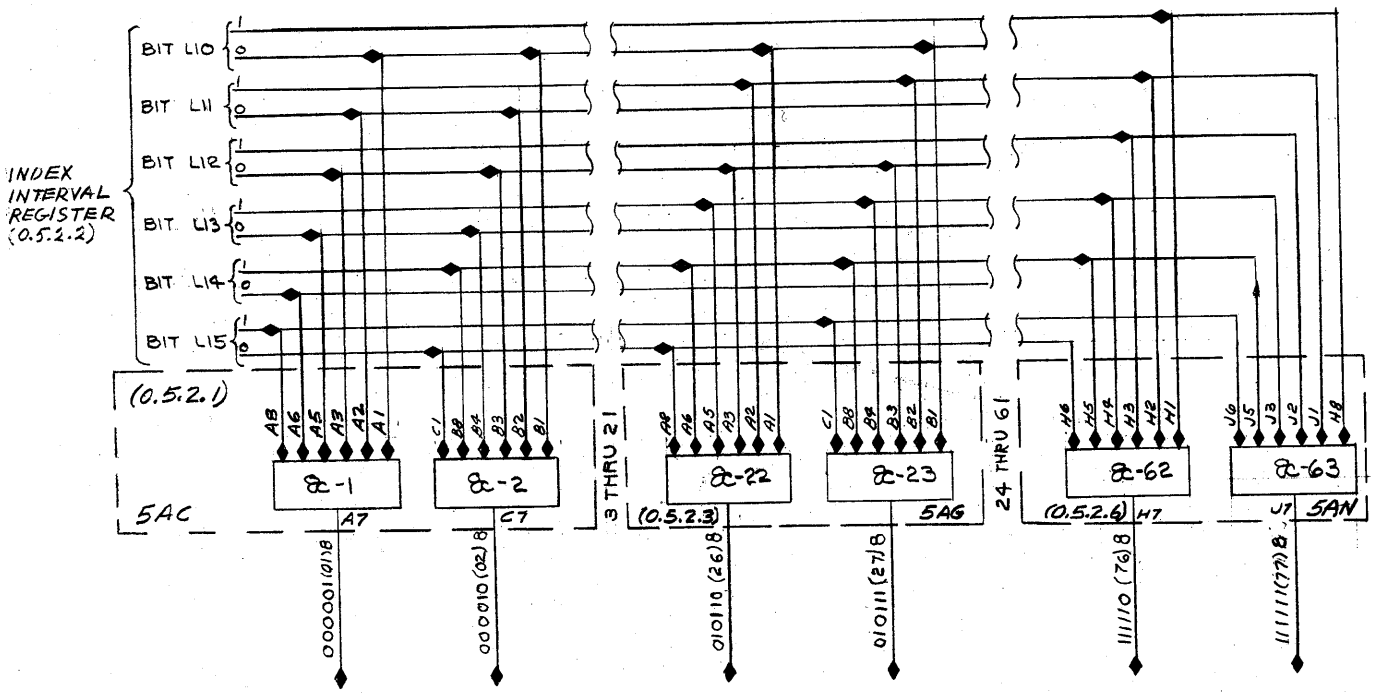


Figure 5–9. PerSelBsn Matrix, Logical Block Diagram

CHAPTER 2

BREAK PULSE GENERATION

SECTION 1

GENERAL

In Combat Direction Central AN/FSQ–7, Central Computer System time is divided into six-microsecond intervals called memory cycles. A memory cycle may be defined as the time required by core memory to completely cycle through its basic pattern of operation. This pattern repeats itself without interruption except during an arithmetic or IO pause, when the succession of memory cycles is temporarily suspended until the end of the pause.

Since access to core memory is shared by both the Central Computer System and the IO units, any given memory cycle can be assigned to only one or the other. If a memory cycle is assigned to the Central Computer System, a machine cycle will also take place during which arithmetic operations are performed. On the other hand, if a memory cycle is assigned to one of the IO units, the Central Computer System will execute a break cycle in co-ordination with the IO unit. A break cycle may therefore be defined as a six-microsecond interval of time during which one memory cycle of core memory is assigned to transfer a single word between core memory and the IO unit. If the information transfer is from core memory to the IO unit (writing), the break cycle is further defined as a breakout (BO) cycle; if the transfer is from the IO unit to core memory (reading), it is defined as a break-in (BI) cycle.

For the duration of any break cycle, the arithmetic circuits of the Central Computer System are inactive except possibly for those operations associated with an arithmetic pause. The method by which this is accomplished was discussed in 1.5.2 of Chapter 1 in which it was shown that during a break cycle time pulses are always generated by the time pulse distributor of the instruction control element. It is these time pulses which actually perform the transfer and control functions necessary to an IO operation during a break cycle. Before being utilized for this purpose, however, they are gated through a set of break command generators. The break command generators will be discussed in Section 3 of this Chapter. Essentially, the break command generators gate time pulses as either break-out pulses or break-in pulses. Thus, with each time pulse of the time pulse distributor (excluding TP–0 which is not used) may be associated a break pulse. To distinguish them from one another and from the time pulses, a special notation is used. Thus, a TP–7 pulse gated as a break-in pulse would be notated as BI–7, or a TP–3 pulse gated as a break-out pulse would be notated BO–3. During each break-in cycle, a continuous series of break-in pulses from BI–1 to BI–11 are generated corresponding to TP–1 through TP–11 of the associated memory cycle. An identical arrangement of break-out pulses, BO–1 through BO–11, are generated during a break-out cycle.

The speed of the Central Computer System in transferring and processing words exceeds that of any of the IO units. Because of this, break cycles do not ordinarily appear in continuous trains. An indication to the Central Computer System that a break cycle must be supplied in order to transfer a word is given by the IO unit in the form of a break request pulse. Each type of IO unit has its own method of generating break requests. These methods will be discussed in the next paragraph. Upon receipt of a break request, the Central Computer System will either accept immediately (2-megacycles operation on pause and no break), accept at the end of the memory cycle in progress (TP–11), or reject. If accepted, the Central Computer System will synchronize the action of the subsequent break with the computer operations presently being performed. This synchronization function is accomplished by the break request circuit which will be described in mediately (2-megacycle operation on pause and Section 2 of this Chapter. If the break request is accepted, the break flip-flops in the instruction control and selection and IO control element will be set. The function of the former break flip-flop was discussed in 1.5.2 of Chapter 1. The break flip-flop in the selection and IO control element, in co-ordination with read and write flip-flops which are set by *Read* or *Write* instructions, will condition the break command generators. The break

command generators then develop either the BI or the BO pulses which activate the IO control circuits and effect the request word transfer.

1.1 BREAK REQUEST GENERATION

Three flip-flops in the selection and IO control element are directly responsible for the production of either break-in or break-out pulses: the break, read, and write flip-flops. When both the break and read flip-flops are set, break-in pulses are generated. When both the break and write flip-flops are set, break-out pulses are generated. The logic of the computer never permits the read and write flip-flops to be in the 1 status concurrently. The read or the write flip-flops are always set during the *Read* or the *Write* instruction, respectively. The read flip-flop remains set until the end of the IO process. Unless the card printer, the card punch or the warning lights, are in use, this is also true of the write flip-flop. Therefore, with the exception of the card machine writing operation, break pulse generation is essentially controlled by the break flip-flop alone. The break flip-flop is in turn controlled by the break request flip-flop.

Attention must therefore be directed to the action of the break request flip-flop. This flip-flop is examined at every TP–11 during a machine or a break cycle, and at a 2-megacycle rate during arithmetic or IO pauses. It is cleared every BO–1 or BI–1 and also at selected times during the IO process. The subject of interest in this paragraph, however, is the manner of its setting. No matter which IO unit is linked with the Central Computer System for an IO transfer, the break request flip-flop is always set by a break request pulse. In short, no IO transfer can be accomplished unless a break cycle is assigned and no break cycle can be assigned unless a break request pulse is generated. The method of generating these break requests varies from IO unit to IO unit, but one aspect of their generation is common to all IO units. The break request pulse will never be generated until a start read or a start write pulse has been transmitted to the selected IO unit by the Central Computer System. For some IO units, such as a drum field during a writing operation, the original break request pulse is produced by the Central Computer System; for others, such as the card reader, every alternate break request pulse is generated by the computer. The following paragraphs present a detailed analysis of the method used by each IO unit to develop its break request pulses.

1.1.1 Card Machines

All card machine transfers are made in 64-bit groups. The 64 bits are in reality two 32-bit words so that each card machine transfer involves a word pair. When the card reader is being used, one word pair is processed every nine milliseconds. When the card printer or the card punch is the selected IO unit, several dummy break-out cycles must be generated between and during individual word transfers, but since a break-out cycle is measured in microseconds, the approximate rate of transfer is still one word pair every nine milliseconds. The dummy break-out cycles used in card machine operation are merely time-consuming devices, and no positive action takes place in the Central Computer System while they are in effect. The extra time is granted because the two thyratron buffer registers associated with the card machines each require a settling time of 10 microseconds after their shield grids are conditioned and an additional 10 microseconds of settling time after their control grid levels are raised.

The following paragraphs will all assume that the IO word counter is not equal to zero, that a *Read* or *Write* instruction has been properly executed, and that the IO interlock remains on throughout the process.

1.1.1.1 Read

The card reader, the only card machine which can transmit information to the Central Computer System, generates its break requests in a manner different from either the card printer or the card punch. The circuits required for this generation are shown in figure 5–10. The operation of these circuits depends upon the receipt of the first break request pulse from the card reader. This pulse will result in the first break-in cycle being assigned, during which the first 32-bit word or a word pair is transferred from the reader to the Central Computer System, and it will also establish the conditions necessary for a second break request pulse to be generated by the computer. The second break request pulse will demand a second break-in cycle, during which the remaining word of the word pair will be transferred to the Central Computer System. This process is repeated once every nine milliseconds, the first or odd pulse being generated by the card reader and the second or even pulse being generated by the Central Computer System.

The card reader index pulse, generated by the reader every time a new card row is positioned under the read brushes, is sent to the Central Computer System as the odd break request pulse.

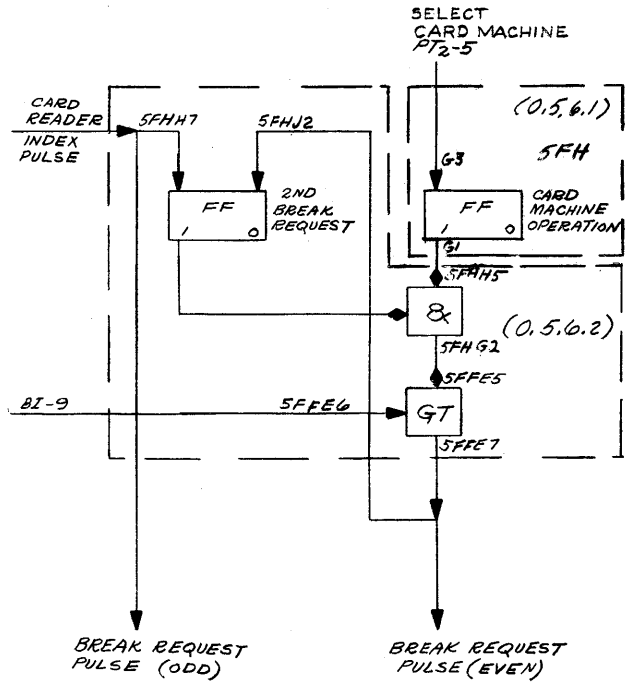


Figure 5–10. Break Request Generation—Card Reader, Logical Block Diagram

In addition to setting the break request flip-flop (not shown in the figure), it also sets the second break request flip-flop. As a result of the first break request, a break-in cycle is eventually assigned. During this cycle, the left word on the card row being read is transferred through the IO register and the memory buffer register to its destination in core memory. The right word on the card row has been placed in the IO buffer register by the original card reader to Central Computer System transfer occurring when the index pulse was generated. At BI–7 of the first break-in cycle, the IO registers, already cleared of the left word, receive the right word from the IO buffer register. The fact that both the card machine operation and second break request flip-flops are set conditions a gate tube examined at BI–9. The output pulse developed by this gate tube clears the second break request flip-flop and is also sent to the break request circuits as the second break request pulse. The resultant break-in cycle transfers the right word of the original word pair from the IO register to its predetermined destination in core memory via the memory buffer register. After the second break-in cycle is finished, the Central Computer System continues its program until the next index pulse appears about nine milliseconds later when the process is repeated and a new word pair is transferred and stored in core memory.

1.1.1.2 Write

The printer and the card punch are similar to the card reader in that they transfer word pairs. However, the time required to prepare the word pair for actual printing or punching is longer than that required to prepare a word pair for reading. Whereas two successive break-in cycles are sufficient to transfer a word pair from the card reader to core memory, approximately 10 memory cycles and two break-out cycles are required to transfer a word pair from core memory to the printer or card punch. The additional time is dictated by the relatively slow action of the thyratron buffer registers used in card machine writing operations. A series of delaying flip-flops are therefore an integral part of the break request generation circuits for the printer or the card punch. These flip-flops cause the Central Computer System to assign dummy break-out cycles, during which the thyratron buffer registers have time to settle after their condition is altered. The same circuits are used for both printer and punch since their respective modes of operation are very similar. While the ensuing discussion will assume that the printer has been selected, it will also be applicable to the card punch.

The break request generation circuits are shown in figure 5–11. A short time after receiving the start write pulse from the Central Computer System, the printer transmits an index pulse to the selection and IO control element. The reception of this pulse indicates to the Central Computer System that the printer is ready to write a word pair. The index pulse performs three functions:

a. Sets the break request flip-flop (odd break request pulse)

b. Clears the write flip-flop

c. Initiates the action of the break request generation circuits

The break request flip-flop will be sensed at TP–11 of the existing memory cycle (or by the next 2-megacycle pulse if the computer is in a pause). As a result, the break flip-flop will be set and, unless some provision is made, a break cycle will be assigned immediately. To prevent the break cycle, the index pulse also clears the write flip-flop, thus deconditioning the break-out command generators and, even though the break flip-flop is set, no break-out pulses are issued. Another function of the index pulse is to establish the proper starting conditions for the break request generation circuits. Accordingly, it sets the index and the No. 1 flip-flops.

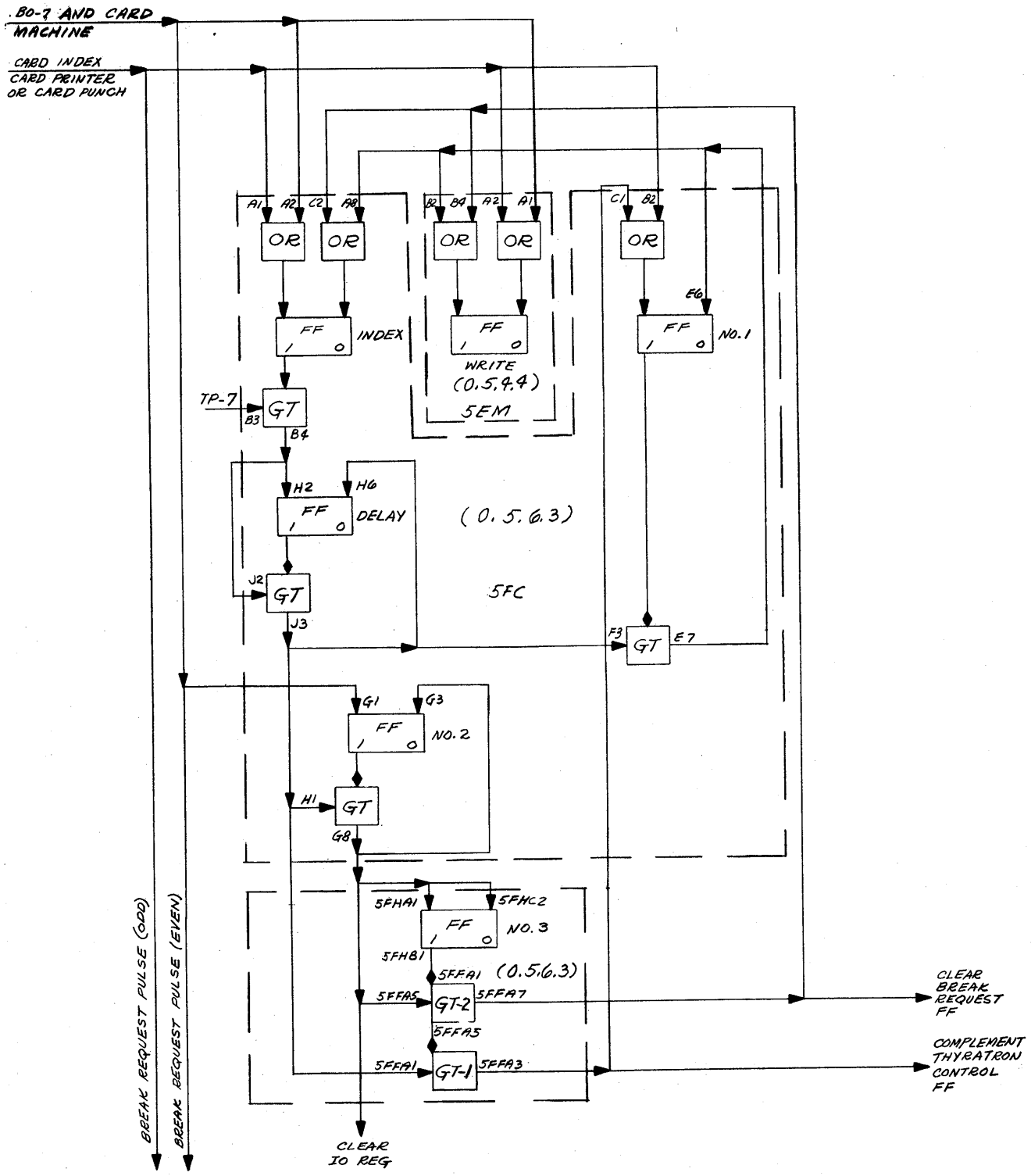


Figure 5–11. Break Request Generation, Card Printer, Card Punch, Logical Block Diagram

Because the break flip-flop is set, the time pulse distributor generates time pulses even if the computer is in an arithmetic or IO pause. The first TP–7 pulse to appear after the index pulse examines the gate tube conditioned by the 1 side of the index flip-flop. Since this flip-flop was set by the index pulse, the gate tube propagates the TP–7 pulse, which then simultaneously sets and interrogates the delay flip-flop. The shift in status of the delay flip-flop from 0 to 1 is not fast enough to permit the TP–7 pulse to proceed any further.

The following TP–7 pulse will appear during the next memory cycle and will be passed by both the index and the delay flip-flops. Upon being issued by the gate tube associated with the 1 side of the delay flip-flop, the pulse clears the delay flip-flop and examines the No. 1, No. 2, and No. 3 flip-flops. With this circuit arrangement, only every alternate TP–7 will be passed. This means that the status of the No. 1, No. 2, and No. 3 flip-flops will be examined once every two memory cycles. At this point of the operation, only the No. 1 flip-flop conditions its gate tube. The pulse resulting from this gate tube clears the index and No. 1 flip-flops and sets the write flip-flop. Setting the write flip-flop fully conditions the break-out command generators, and the next memory cycle will be a break-out cycle. During the two memory cycles just completed, the first thyratron buffer register has had sufficient time to raise its shield grid potentials to the proper level.

During the break-out cycle, a 32-bit word will be transferred from core memory via the memory buffer register to the IO register. The resultant d-c levels of the IO registers will condition the control grids of the first thyratron buffer register to fire.

Another two memory cycles elapse because of the action of the delay flip-flop. At TP–7 of the second of these, flip-flops No. 1, No. 2, and No. 3 are interrogated again. Only flip-flop No. 3 conditions its gate tube (GT–2) and the pulse resulting reverses the status of the thyratron control flip-flop and sets the No. 1 flip-flop. Reversing the status of the thyratron control flip-flop causes the shield grids of the other thyratron buffer register to be raised. Except for the No. 3 flip-flop, now in the 1 state, the break request generation circuits are in their original condition. Six memory cycles and one break-out cycle have transpired.

The process is repeated for the second word of the word pair to be transferred, beginning at TP–7 of the seventh memory cycle. This cycle, because of the action of the delay flip-flop, is a dummy cycle, during which the shield grids of the second thyratron buffer register are raised to their proper level. The next TP–7 pulse is passed by both the index and delay flip-flops, and, upon examining the No. 1, No. 2, and No. 3 flip-flops, is passed by the gate tube associated with the 1 side of the No. 3 flip-flop (GT–1). The propagated pulse clears the index, delay, and No. 1 flip-flops and sets the write flip-flop. At the end of the current cycle, since both the break and write flip-flops are set, the forthcoming memory cycle is assigned as the second break-out cycle. At BO–1, the break request flip-flop is cleared. At BO–7, the index and No. 2 flip-flops are set, the write flip-flop is cleared, and a break request is made. The break request at this point assures that the next memory cycle will be a dummy break-out cycle and not an instruction cycle. Otherwise, the process might be terminated prematurely. During the break-out cycle, the second word is transferred to the IO register and is used to raise the control grid levels of the second thyratron buffer register. Time must again be allowed for these thyratrons to fire. The action of the delay flip-flop again causes a dummy break cycle to be assigned for this purpose. The final TP–7 required for the card machine writing process occurs during the 10th memory cycle. It is passed by the index and delay flip-flops to the No. 1, No. 2, and No. 3 flip-flops. The No. 1 flip-flop does not condition its gate tube and the examining pulse for this flip-flop is suppressed. The pulse that examines the No. 2 flip-flop is passed and complements the No. 3 flip-flop, returning it to its original status. This pulse also clears the IO registers and the No. 2 flip-flop. The No. 3 flip-flop has both its gate tubes inspected and since it remains in the 1 status long enough for the gate tubes to pass these examining pulses, the following action takes place:

A. GT–1
   1. Clears break request flip-flop
   2. Sets write flip-flop
   3. Clears index flip-flop

B. GT–2
   1. Reverses status of thyratron control flip-flop
   2. Sets No. 1 flip-flop

On the assumption that no arithmetic pauses exist, the break flip-flop will be cleared at TP–11 of this memory cycle. If it were not, a break-out cycle would be assigned because the write flip-flop is set.

The word pair is now in the two thyratron buffer registers. The break request generation circuits are in their original state. The two 32-bit words are now printed or punched under the control of the particular card machine in use. The next index pulse will not appear for at least nine milliseconds, and, in this interim, the computer continues with its program. At BO–1, the first break pulse of the break-out cycle, the break request flip-flop is cleared. During the break cycle, this flip-flop is sensed by a TP–11 pulse which will be gated through to clear the break flip-flop. This insures that the generation of

break pulses will be stopped at the end of the break-out cycle so that no immediate second break cycle will occur.

Thus far, two memory cycles and part of one break-out cycle have elapsed. One of the words to be transferred has already been allowed to raise the control grid levels of one of the two thyratron buffer registers. Some time must be allowed for the thyratrons to actually fire. Accordingly, at BO–7 of the break-out cycle, the index and the No. 2 flip-flops are set and the write flip-flop is cleared. The BO–7 pulse also requests a break (even break request pulse) so that the break flip-flop is set and the write flip-flop is cleared at the end of the first break-out cycle.

Because of the action of the delay flip-flop, the first TP–7 to appear after the break-out cycle is suppressed. At TP–7 of the following memory cycle, the delay flip-flop again develops a pulse which examines the No. 1, No. 2, and No. 3 flip-flops. At this stage, only the No. 2 flip-flop conditions its gate tube, and the output of this gate tube clears the IO register and the No. 2 flip-flop. It also complements the No. 3 flip-flop. Although the pulse also examines a gate tube conditioned by the 1 side of the No. 3 flip-flop (GT–1), the complementing action does not reverse the status of the No. 3 flip-flop fast enough (0 to 1) and the pulse does not proceed any further. Therefore, after four memory cycles and one break-out cycle, one of the thyratron buffer registers is fully loaded and the flip-flops shown in figure 5–11 are in the states listed in table 5–2.

TABLE 5–2. FLIP-FLOP STATES FOR ONE LOADING THYRATRON REGISTER

FLIP-FLOP	STATE
Index	Set
Delay	Clear
No. 1	Clear
No. 2	Clear
No. 3	Set
Write	Clear
Break	Set

1.1.2 Drum System

As in the case of the card machines, the Drum System employs two methods for generating break request pulses, one for reading and the other for writing. In the method used for reading, every break request pulse is generated by the Drum System. In the method used for writing, the Central Computer System generates the first two break request pulses. Furthermore, during a reading process, the first break request pulse is not generated until a drum address search is completed, whereas, for writing, two break request pulses and their attendant break-out cycles immediately follow the *Write* instruction. The drum address search termination is necessary only for the third and all succeeding break request pulses. During a read operation, the fact that the address search is terminated is evidenced by the transmittal of a CD–1 load pulse from the Drum System to the selection and IO control element. During a write operation, address search termination is indicated to the selection and IO control element by the receipt of a word demand pulse from the Drum System.

1.1.2.1 Read

The circuits required for the generation of a break request pulse during an addressable drum field reading operation and after the address search are shown in figure 5–12. A CD–1 load pulse, coincident in time with the transfer of the word to be read from its drum register to the IO buffer registers, sets the IO buffer load sync flip-flop after first being delayed for one microsecond. The delay must be introduced so that the IO buffer registers can fully settle before an attempt is made to transfer the word out of them. After the sync flip-flop has settled, the first 2-megacycle pulse to inspect its associated gate tube is passed and sets the IO buffer load flip-flop. The gate tube conditioned by the 1 side of the IO buffer load flip-flop is also examined at a two-megacycle rate. The first pulse passed through this gate tube clears the IO buffer load flip-flop as well as its synchronizing flip-flop, and inspects the gate tube associated with the drum operation flip-flop. The drum operation flip-flop is always in the set status during the drum reading operation and the interrogating pulse is therefore passed. It next examines the accept flip-flop. This flip-flop was set at the conclusion of the address search and remains in the 1 side throughout the IO process. Its gate tube, upon being interrogated by the signal from the drum operation flip-flop, produces an output signal which sets the IO buffer status flip-flop.

Interpreting the 1 side of the flip-flop as meaning full, and the 0 side as meaning empty, the status of the IO buffer status flip-flop at this point correctly reflects the status of the IO buffer registers. The IO register status flip-flop, interpreted in the same way, also correctly represents the status of the IO registers since these registers are empty and the flip-flop is cleared. The

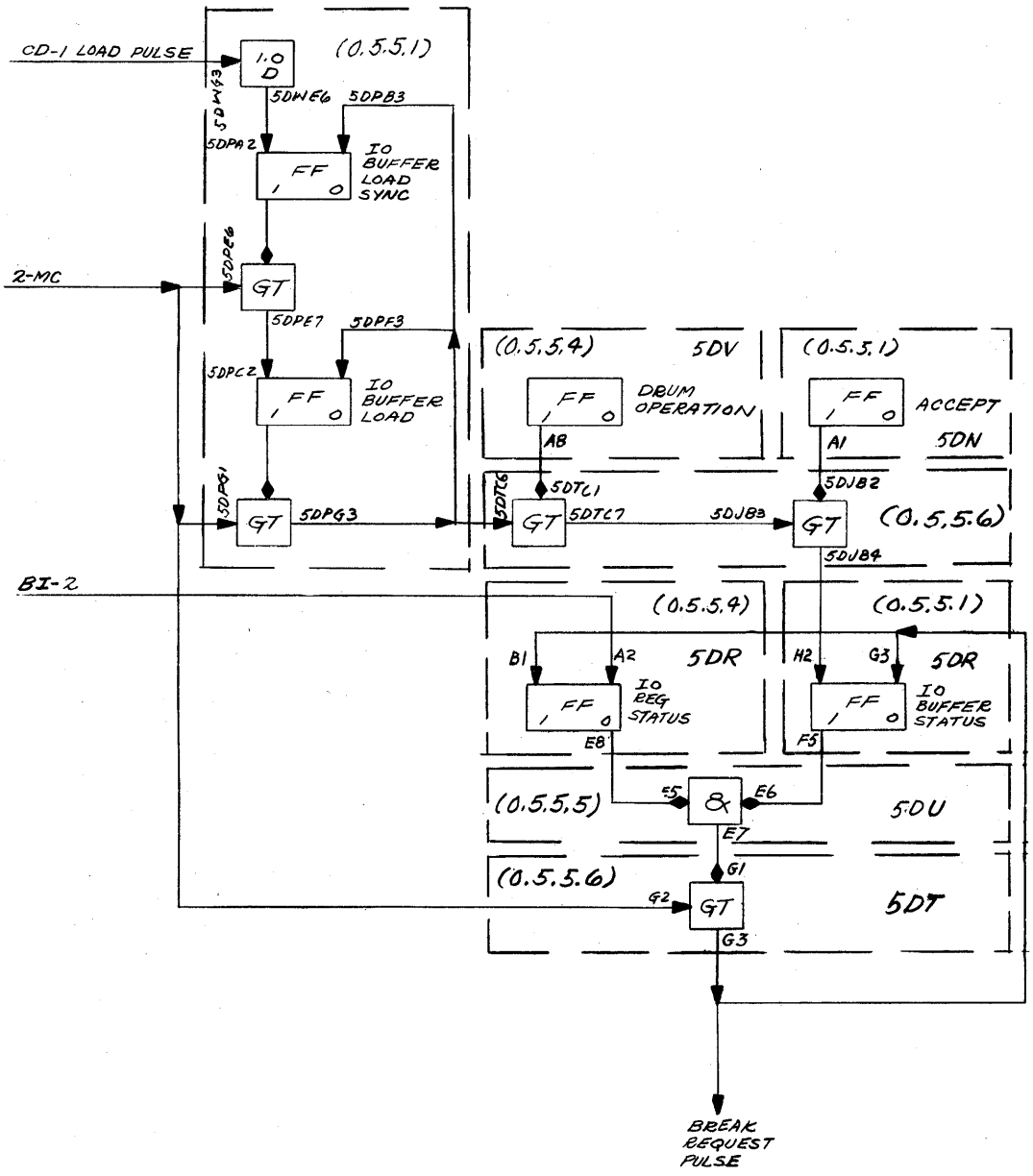


Figure 5–12. Break Request Generation, Drum System, Read, Logical Block Diagram

combination of full IO buffer registers and empty IO registers during a drum reading operation is always the signal that a break-in cycle is required. Accordingly, the outputs of the 1 side of the IO buffer status flip-flop and the 0 side of the IO register status flip-flop are combined in an AND circuit whose output conditions a gate tube constantly inspected by two-megacycle oscillator pulses. The output of this gate tube is the desired break request pulse. In addition to setting the break request flip-flop, the break request pulse reverses the status of the IO register status and the IO buffer status flip-flops, ensuring that only one break request will be made for each word transferred from the selected field to the IO buffer registers.

During the ensuing break-in cycle, the word is transferred from the IO registers to its destination in core memory. Initiation of the process to generate the next break request pulse awaits the appearance of the next CD–1 load pulse 10 microseconds later. At this time a new word will be transferred from the selected drum field to the IO buffer registers and the entire procedure will be repeated.

1.1.2.2 Write

The generation of any break request after the second during a drum write process is always

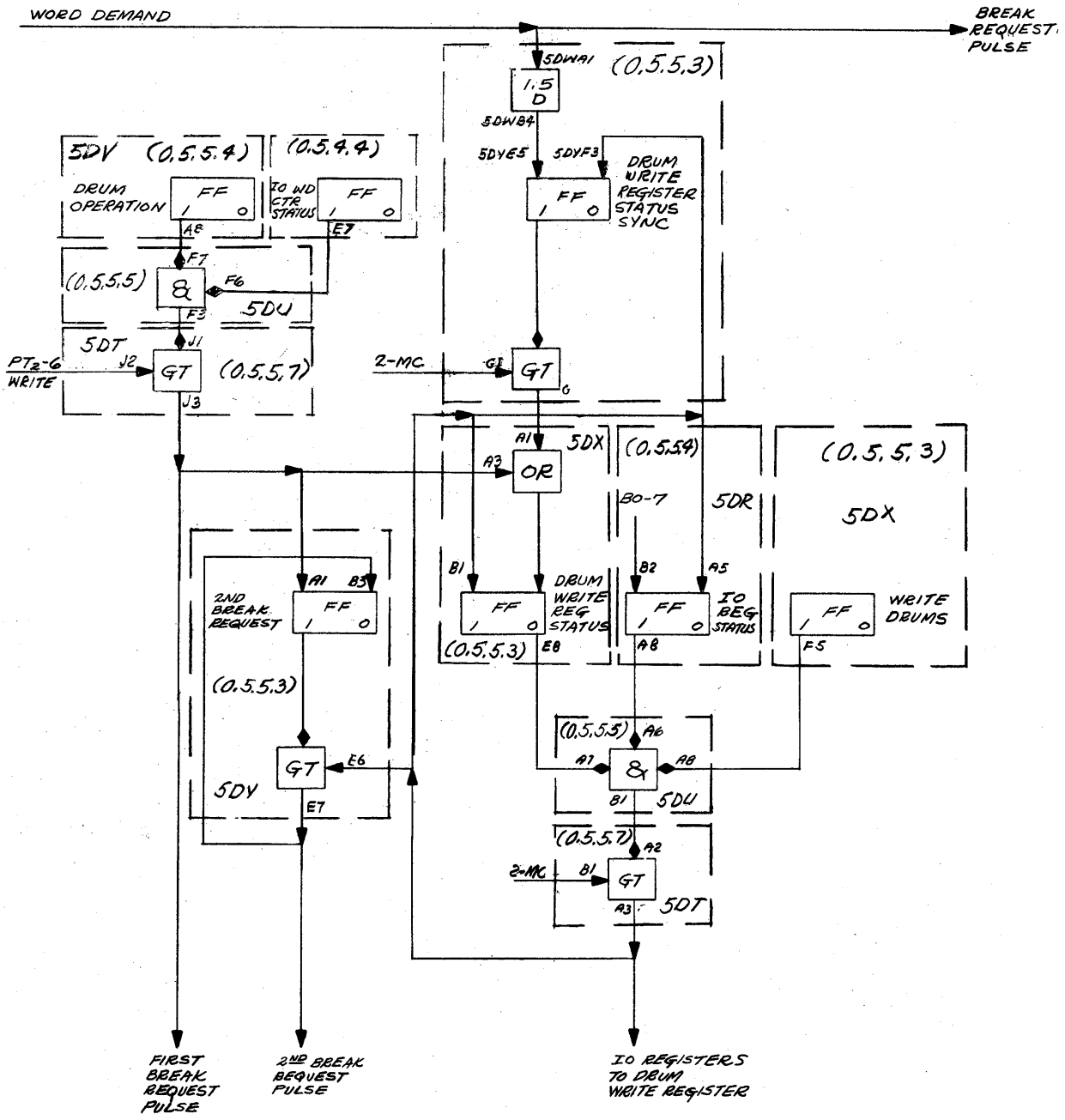


Figure 5–13. Break Request Generation, Drum System, Write, Logical Block Diagram

made by a word demand pulse. The word demand pulse is developed by the Drum System only after a word has been written on the selected drum field. However, since a word to be written must pass through the IO registers and the drum write register before being written, two break request pulses must be generated by the computer prior to the actual drum writing operation. The two resultant break-out cycles are used to fill the IO registers and the drum write register with the first two words of the impending IO transfer.

The circuits responsible for the development of the first two break requests and all subsequent requests are depicted in figure 5–13. The drum operation and the IO word counter status flip-flops are both set before PT_2–6 of the *Write* instruction. Accordingly, the gate tube pulsed by the PT_2–6 write signal is conditioned at the time of its examination. The resultant output pulse becomes the first break request pulse. In addition, the output pulse sets the second break request flip-flop and clears the drum write register status flip-flop.

The PT_2–6 pulse of the *Write* instruction is also sent to the Drum System as a start write pulse; however, a 120-microsecond delay, deliberately introduced to allow the drum writing circuits time to settle, prevents the Drum System from writing for this length of time. There is therefore ample time to complete the first two break-out cycles.

Note

If a *Write* instruction is immediately followed in the program by another *Write* instruction, there is no second 120-microsecond delay. In this case, the drum circuits do not write until the write register status flip-flop in the Drum System is set.

During the break-out cycle assigned as a result of the first break request pulse, the first word of the transfer is sent from core memory through the IO registers to the drum write register. At BO–7 of this cycle, the IO register status flip-flop is cleared and, since the write drum flip-flop was previously cleared, the three-way AND circuit shown in the figure is fully conditioned. The output of the AND circuit conditions a gate tube examined at a 2-megacycle rate. An output pulse is immediately developed which simultaneously examines the second break request flip-flop, sets the drum write register status flip-flop, clears the IO register status flip-flop, and activates the transfer of the word now in the IO registers to the drum write register. The examination of the second break request flip-flop results in the generation of a second break request pulse. The second break request flip-flop is cleared by this pulse. By setting the drum write register status flip-flop and clearing the IO register status flip-flop, a correct representation of circuit conditions is given since the drum write register is now full and the IO registers are now empty.

The break-out cycle now assigned need only transfer a word from core memory to the IO register. The further transfer from the IO registers to the drum write register is inhibited at this stage of operation because the three-way AND circuit controlling this transfer is deconditioned since the drum write register status flip-flop is in the 1 state. At BO–7, the IO register status flip-flop is again set, indicating that the IO registers contain the second word of the impending transfer. At the end of the second break-out cycle, therefore, the first word of the transfer is in the drum write register, the second is in the IO registers, and the transfer awaits completion of the 120-microsecond delay introduced by the Drum System circuits.

At the end of the prescribed delay, an address search is conducted after which the word in the drum write register is transferred to the drum surface. A break-out cycle is needed to transfer the second word of the IO transfer from the IO registers to the drum write register, and also to bring another word from core memory to the IO registers. Accordingly, after the first word is written, a word demand pulse is sent to the selection and IO control element and this pulse requests a break. It also sets the drum write register status sync flip-flop. In the usual manner, the drum write register status flip-flop is itself cleared a short time later. At BO–7, the IO register status flip-flop is set, permitting the IO registers to drum write register transfer. At the same time, the next word in core memory is brought to the IO registers. The same pulse that activates the transfer also reverses the indications of the IO register and drum write register status flip-flops. As was the case for the drum read operation, the status of two registers, as reflected by the appropriate flip-flops, controls the IO transfer. For the drum read process, the IO registers and the IO buffer registers are the controlling registers; for the drum write operation, the IO register and the drum write register are the controlling registers.

1.1.3 Tape Units

The Central Computer System has six tape units associated with it. However, at any given time, only one of them can be operative in an IO transfer. The units are identical in design and operation and the following discussion is applicable to any of them. All tape units can participate in both read and write operations.

The tape unit has a timing system of its own which is asynchronous to Central Computer System timing. The tape unit synchronizes its internal operations by means of an electronic clock which is part of the tape timing system. In response to a start read or start write signal from the selection and IO control element, the tape unit prepares the tape buffer register to accept or transmit a word. As soon as the register is ready, a break request pulse is automatically sent to the selection and IO control element where it examines the tape operation flip-flop as shown in figure 5–14. It should be noted that in the case of tape units, each break request is directly requested by the tape unit.

1.1.4 Miscellaneous IO Units

Four miscellaneous IO units are to be considered in connection with the IO function of the

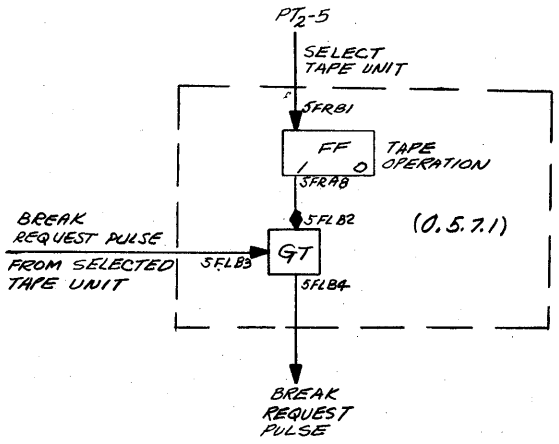


Figure 5–14. Break Request Generation—Tape Units, Logical Block Diagram

selection and IO control element. Two of these miscellaneous units, the manual input matrix and the burst time counter, utilize the same circuit to generate their break request pulses. The third is the IO register, ordinarily a part of the transfer path for other IO units, and the fourth is the Warning Light System.

1.1.4.1 Manual Input Matrix and Burst Time Counters

Both the manual input matrix and the burst time counter can only supply information to the Central Computer System, but cannot accept it. As relatively fast IO devices, the burst time counter supplies the Central Computer System with the words at the rate of one word every 10 microseconds and the manual input matrix at the rate of one word every 20 microseconds. Since the two IO units share essentially the same circuits, the design of their break request generation circuits must be for the faster rate; i.e., one word per 10 microseconds. This speed is exactly that of the Drum System, and the information flow path, therefore, must include the IO buffer register. Accordingly, a part of the Drum System break generation circuits is utilized.

The required circuit configuration is shown in figure 5–15. Assume that the manual input matrix was selected by the *Select* instruction. At PT_2–5 of that instruction, the manual input matrix flip-flop was set to 1, thereby conditioning its associated gate tube. As soon as the manual input matrix is ready to send a word, it generates a break request pulse. This pulse passes through the gate tube conditioned by the manual input matrix flip-flop and sets the IO buffer load sync flip-flop. At the same time, the word is being

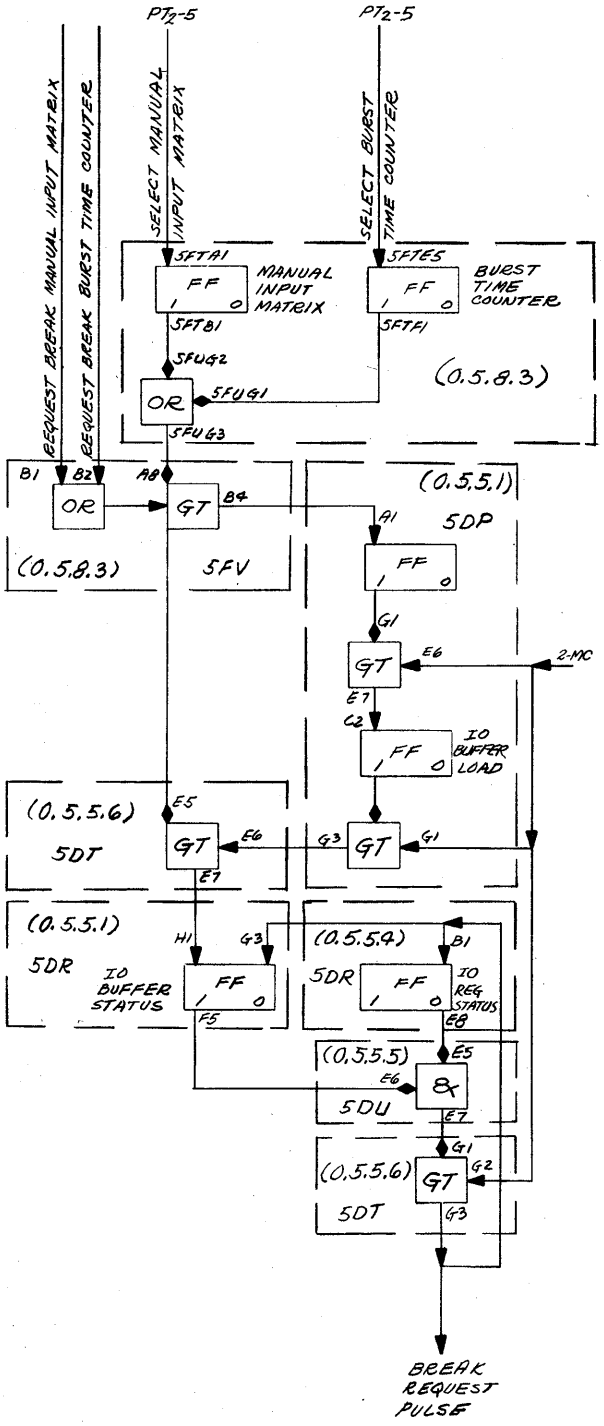


Figure 5–15. Break Request Generation, Burst Time Counters and Manual Input Matrix, Logical Block Diagram

transferred to the IO buffer registers. The IO buffer load flip-flop is set as soon as the gate tube conditioned by the IO buffer load sync flip-flop allows an examining 2-megacycle pulse to pass.

The IO buffer load flip-flop in turn conditions a gate tube examined by 2-megacycle oscillator pulses. The first pulse to pass is gated through a gate tube conditioned by the manual input matrix flip-flop and sets the IO buffer status flip-flop. Since the IO register status flip-flop is cleared at this point, setting the IO buffer status flip-flop fully conditions the AND circuit controlling the action of the gate tube whose output is the desired break request pulse. The gate tube is constantly inspected by the 2-megacycle oscillator pulses. Upon being conditioned, it generates a break request pulse which, in addition to setting the break request flip-flop, reverses the status of the IO buffer register and IO register status flip-flops in preparation for the next break request from the manual input matrix.

1.1.4.2 IO Register

The IO register, ordinarily a transfer point for IO operations involving other IO units, can itself be selected as an IO device to be linked with the Central Computer System. When used in this capacity, it cannot accept information from core memory and consequently, only a *Read* instruction can properly follow a *Select* (IO register) instruction.

The first break request is a result of the action of the *Read* instruction. (See fig. 5–16.) PT_2–6 of this instruction examines a gate tube conditioned by the 1 side of the IO register flip-flop. If the IO register has been selected, this flip-flop will be in its 1 status and the gate tube will be conditioned. The pulse which is propagated becomes the break request pulse required for assigning a break-in cycle.

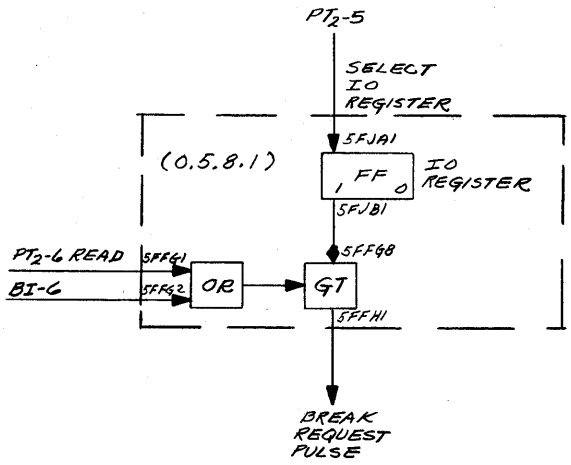


Figure 5–16. Break Request Generation—IO Register, Logical Block Diagram

Following the initial break request, no more *Read* instruction pulses will be available. To transfer the required number of words, the IO register flip-flop is examined during every break-in cycle, the examination resulting in a further break request. This function, originally undertaken by PT_2–6 of the *Read* instruction, is assigned to BI–6 during each break-in cycle.

1.1.4.3 Warning Light System

The Warning Light System is used to warn operators at different consoles of certain Central Computer System actions or special programming operations. This is accomplished by transferring eight words from the Central Computer System to eight 32-bit registers in the Warning Light System, called the warning light registers. The contents of these registers, which are the words written by the Central Computer System, specify neons and audible alarms which are to be turned on and off at the various consoles. The Warning Light System is selected by a *Select* (warning lights) instruction, and the break-out cycles which write the eight words, are initiated by a *Write* instruction.

The first break request is a result of the *Write* instruction. (See fig. 5–17.) Pt_2–6 of this instruction examines a gate tube conditioned by the 1 side of the warning lights flip-flop at PT_2–5. If the warning lights have been selected, this flip-flop will be in the 1 status and the gate tube will be conditioned. The pulse which is propagated becomes the break request pulse required for assigning a break-out cycle. Following the initial break request, no more *Write* instruction pulses will be available. To transfer the required number of words, the warning lights flip-flop is examined during every break-out cycle, the examination resulting in a further break request. This function, originally undertaken by PT_2–6 of the *Write* instruction, is assigned to BO–8 during each break-out cycle.

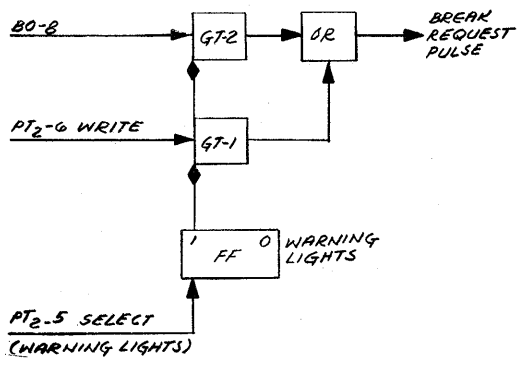


Figure 5–17. Break Request Generation—Warning Light System—Logical Block Diagram

SECTION 2

BREAK REQUEST CIRCUIT

No matter which IO unit originates a break request, disposition of the request is controlled by the circuits shown in figure 5–18. A break request can logically appear only if a *Read* or *Write* instruction has preceded it; no break request can be accepted after the IO word counter, by going to 0, has indicated that no more words are to be transferred.

Accordingly, a restriction is placed on the break request pulse before permitting it to set the break request sync flip-flop. The restriction consists of making the request pulse examine the status of the IO word counter as reflected by the IO word counter status flip-flop. If the IO word counter contains anything but 0, the flip-flop will be in the 1 state and will condition the gate tube examined by the break request. The resultant pulse will set the break request sync flip-flop.

The break request sync flip-flop is examined at a 2-megacycle rate, but the interrogating pulses are delayed by an amount sufficient to bring them into coincidence with the 2-megacycle pulses which, during a pause, inspect the break request flip-flop itself. The two pulse trains would not otherwise be coincident because the one that examines the sync flip-flop is derived directly from the 2-megacycle oscillator, while the pulse train which examines the break request flip-flop,

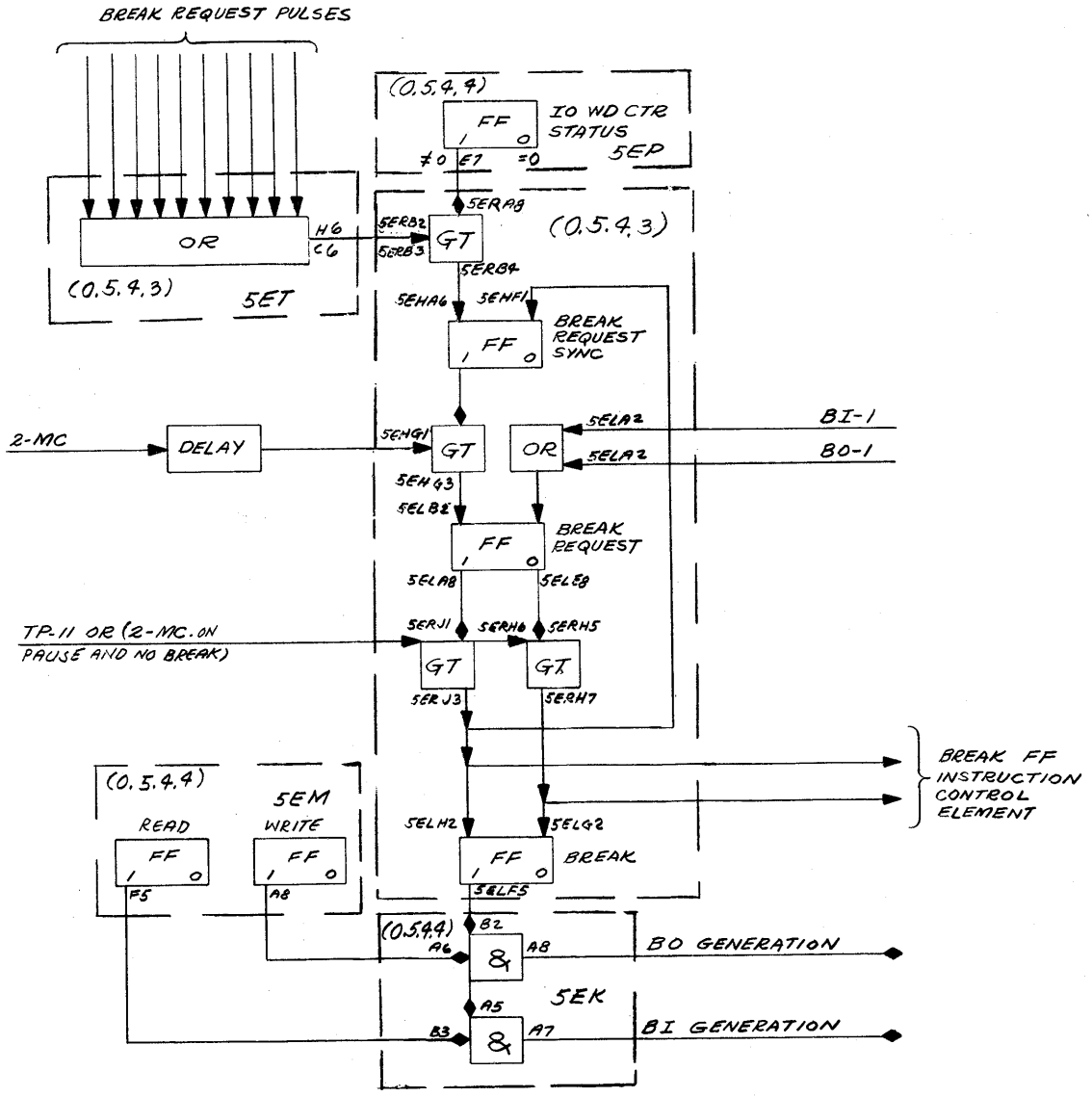


Figure 5–18. Break Request Circuits, Logical Block Diagram

although also originated by the 2-megacycle oscillator, is passed through a number of transformers and gates, each introducing a certain delay, before being applied to the sensing circuit shown in figure 5–18. Without this coincidence, partial pulses, with their attendant difficulties, could easily result.

As a result of the interrogation, the break request flip-flop is set whenever a break request has been made. If the Central Computer System is not in a pause, the flip-flop is examined at TP–11 once during each memory cycle. During a pause, it is sensed by 2-megacycle pulses from the oscillator because TP–11 is not generated under a pause and no break condition. Since the action of the break request circuits is the same under both conditions of examination, it is assumed for the present that no pause exists. At TP–11 of an existing memory cycle, therefore, the indication of a break request as given by the break request flip-flop is transferred to the break flip-flops (one in the selection and IO control element and the other, not shown in figure 5–18, in the instruction control element). For the indication to be positive (break request exists), the break request pulse must have appeared before TP–9; otherwise, the break request flip-flop will not be set in time, and the request will not be recognized until TP–11 of the succeeding memory cycle.

The purpose and function of the break flip-flop in the instruction control element appears in 1.5 of Chapter 1. The break flip-flop in the selection and IO control element operates in conjunction with either the read or the write flip-flop to generate a d-c signal level for the break command generators. Since only one of these two flip-flops (read and write) can properly be in the 1 state at the time the break flip-flop is set, only one d-c signal, the break-in (BI) generation level for reading or the break-out (BO) generation for writing, will be positive at any one time. As a result, either BI or BO pulses are generated for one memory cycle. The first pulse in either sequence is used to clear the break request flip-flop, preparing it for the next break request pulse. The break flip-flop will remain set until, a memory cycle having been assigned as a machine cycle because of the absence of a break request, TP–11 examines the break request flip-flop. As a result, the break flip-flop is cleared.

SECTION 3

BREAK COMMAND GENERATION

Each of the d-c signal levels controlled by the break request circuit conditions 11 gate tubes. Those conditioned by the BI generation line are termed the break-in command generators; those conditioned by the BO generation line, the break-out command generators. (See fig. 5–19.) These command generators are identical in operation to those in the instruction control element. The examining pulses, however, instead of being instruction pulses, are the time pulses generated by the time pulse distributor in that element. Each time pulse from TP–1 to TP–11 examines a separate gate tube pair, each pair comprised of a break-in and a break-out command generator. Only one command generator of the pair can be conditioned at any time, and every other pair must have the corresponding command generator conditioned. With this agreement, the break command generators will produce either a set of break-in or a set of break-out pulses for each break cycle.

The function of the break pulses is to activate the necessary transfer and counting operations during the break cycle since no instruction pulses are generated for the duration of this cycle. Table 5–3 lists the various break pulses, giving for each its assigned task. The functions given apply regardless of the IO unit selected except in those instances where the unique IO unit for which the pulse is applicable is given in parenthesis. As an example, BO–7 always initiates a parity count but will only set the IO register status flip-flop if a Drum System-Central Computer System IO transfer is to take place. If a card machine has been selected instead, the pertinent function of BO–7 is to request a second break cycle which will follow the one assigned whenever an index pulse is sent from the card machine to the Central Computer System.

One break-out pulse (BO–5) and three break-in pulses (BI–2, BI–3, BI–4) are sent to the instruction control element. The BO–5 pulse is effective only if test memory has been selected as the memory source for an IO transfer; if it has, the pulse effects the transfer of the test memory contents to the memory buffer registers. The BI–2 pulse is used to clear the test memory if it has been selected as the memory destination for an IO transfer. This pulse serves a different purpose if core memory has been designated for the IO transfer; it must inhibit the read function of the memory cycle. If this were not done, the core memory would automatically read the contents of a core memory address into the memory buffer registers at some time before BI–7, and the IO

TABLE 5–3. BREAK PULSE FUNCTIONS

BREAK PULSE	FUNCTION
Break-Out	
BO–1	IO address counter to memory address register. Clear break request flip-flop
BO–2	Step IO word counter
BO–3	Not used
BO–4	Not used
BO–5	To instruction control element
BO–6	Not used
BO–7	Memory buffer register to IO register Parity count Request 2nd break (card machines) Set IO register status FF (Drum System)
BO–8	IO register—tape buffer register (tapes)
BO–9	Not used
BO–10	Not used
BO–11	Clear IO interlock (Drum System) Disconnect tapes if IO word counter=0 (tapes)
Break-In	
BI–1	Same as BO–1
BI–2	IO register-memory buffer register Step IO word counter Step IO address counter Clear IO register status FF (Drum System)
BI–3	To instruction control element Parity count
BI–4	To instruction control element
BI–5	Not used
BI–6	Request break (IO register)
BI–7	Parity check IO buffer register to IO register (card machines)
BI–8	Not used
BI–9	Request 2nd break (card machines)
BI–10	Not used
BI–11	Clear IO interlock (IO register, burst time counter, manual input matrix, mechanical clock)

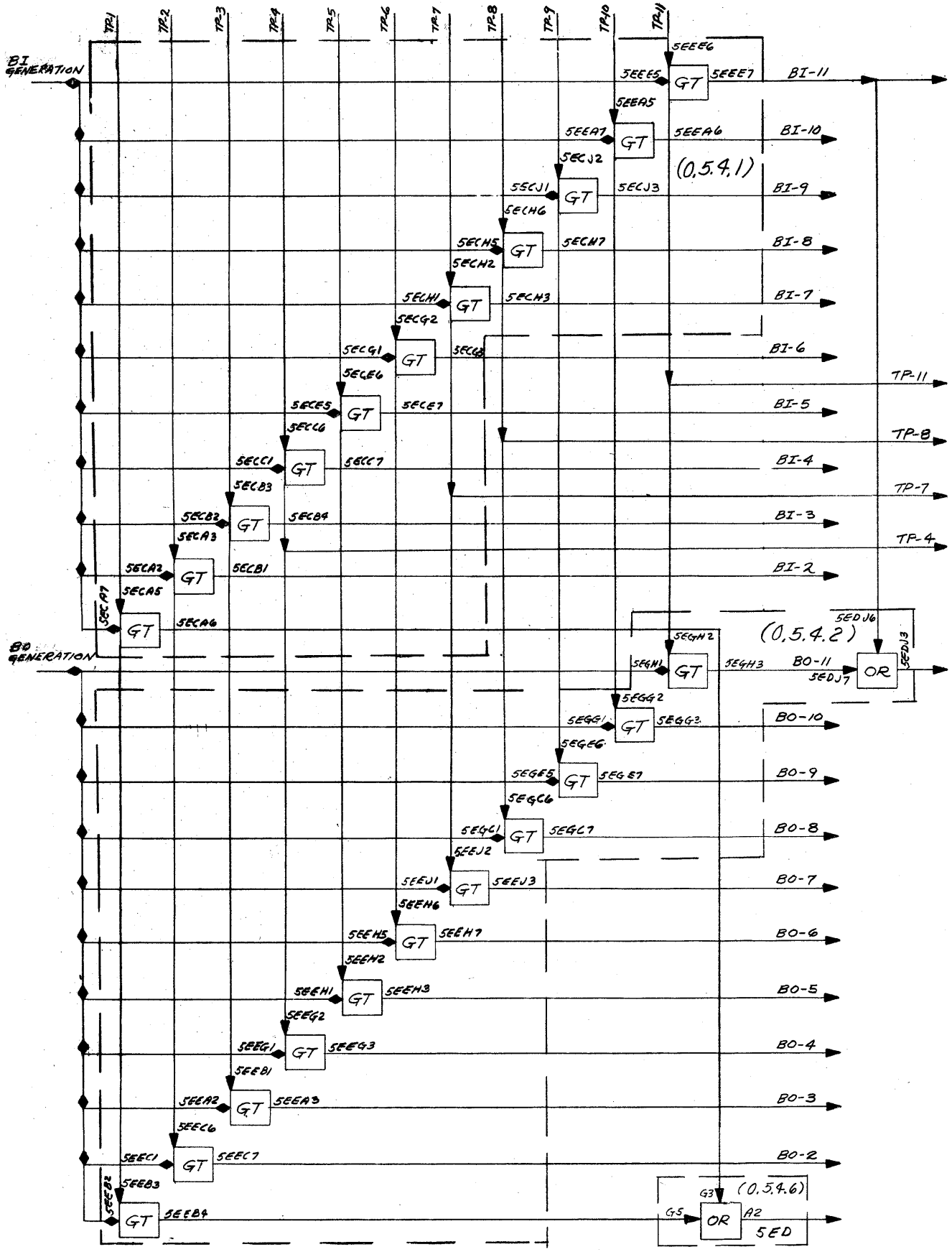


Figure 5-19. Break Command Generators, Logical Block Diagram

word transferred to this register at BI–2 would be lost. The result would be that the word written into core memory during the write portion of the memory cycle (BI–7 to BI–11) would be a meaningless combination of the desired IO word and the original contents of the core memory address specified as the destination for the IO word. The inhibiting action still allows the contents of the desired core memory address to be destroyed by the destructive core memory read-out process, but does not allow these bits to enter the memory buffer registers. In this manner, the core memory address is readied to receive the IO word during the write portion of the memory cycle. The BI–4 pulse transfers the contents of the memory buffer register to the test register if a break-in cycle involving test memory is in effect.

CHAPTER 3

CARD MACHINES

Three card machines are associated with AN/FSQ-7 (XD-1,-2) Combat Direction Centrals: the card reader, the card printer, and the card punch. The card reader is used when information is to be transferred from punched cards to the Central Computer System (read); the card punch and the card printer are used to transform information generated by the Central Computer System into card or printed form (write). The card machines as a group have the ability, therefore, to both read and write, although each card machine individually can perform only one of these two functions.

Word transfers between any card machine and the Central Computer System occur in pairs; i.e., two full binary words are simultaneously transferred. The card machines differ in this respect from other IO units, all of which process words singly. The time interval between successive transmissions of word pairs is established by the electromechanical design of the card machine involved and is in all cases much longer than the basic memory cycle. Since relatively few memory cycles are required to completely process the transferred information in the Central Computer System itself, the time intervals between word transfers (less the time required for processing) are devoted to continuance of the existing program.

In this section attention will be directed to those circuits of the selection and IO control element which, by responding to the basic pattern of an IO program, will effect an information transfer between the Central Computer System and any card machine. A complete block diagram of all card machine control circuits, except for the sense and operate circuits, is illustrated in figure 5–22. The ensuing paragraphs will discuss separate portions of these circuits in detail, and this figure will aid in an understanding of the overall picture and of the interconnections between the separate circuits.

The procedure for generating break request pulses is identical for all card machine IO units. (Refer to Chapter 2.) Furthermore, since the *Load IO Address Counter* instruction is common to all IO programs and has been discussed in Chapter 1, it will be assumed that the instruction is always properly programmed at the start of each IO process. Hence the following discussions will assume that the IO address counter in the program element has been loaded with the starting address in core memory associated with the first word to be transferred.

3.1 PUNCH CARDS

The punch cards used with the card reader and the card punch are standard IBM cards, having 80 columns on each card and 12 punch positions in each column. The columns are divided into three separate groups: 1 through 16, 17 through 48, and 49 through 80. Columns 17 through 48 and columns 49 through 80 are each used to store 32-bit binary words. Since there are 12 punch positions in each column, a card may be used to store as many as 24 words. The distribution of these words on a card is shown in figure 5–20. Columns 1 through 16 are used for storing auxiliary information either in the form of 16-bit binary half-words or by the Hollerith code.

When the IO process involves reading or punching, the 12 rows are transferred one row at a time; i.e., two 32-bit words are processed simultaneously. Referring again to figure 5–20, the transfer sequence is words 1 and 2, then 3 and 4, etc. An index point is associated with each row of information on the punched card. The card is at the index point when one full row of information is under the brushes of the card reader (or the magnets of the card punch). Thirteen index points constitutes a card machine cycle, twelve for the twelve rows of binary words and the thirteenth denoting the time at which the rear edge of the card passes from under the brushes. Thus, during a card machine cycle, one full card will be either read or punched by the Central Computer System.

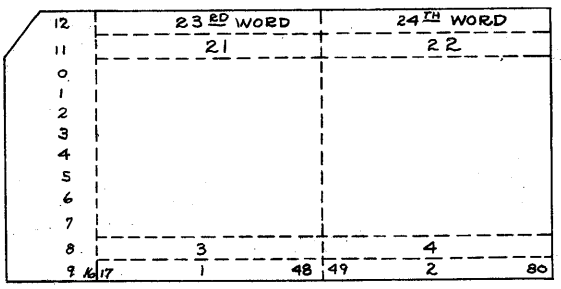


Figure 5–20. Standard IBM Punch Card

3.2 INFORMATION FLOW

The information paths linking the card reader and core memory are illustrated in figure 5–21 and are used following the execution of a *Select* (card reader), *Read* program pattern. As each index point is reached on a card being read, a pair of words are simultaneously transferred from the card reader to the Central Computer System, one going to the IO buffer registers and the other to the IO registers. At the same time, a break request pulse is transmitted by the card reader to the selection and IO control element. The break request pulse actuates the circuits which assign the next memory cycle as a break-in cycle. During the break-in cycle, the word in the IO registers is transferred to the memory buffer registers, the IO registers being cleared during this process, and the word in the IO buffer registers is copied into the IO register. An additional break-in cycle is required in order to place the second word in the memory buffer registers and from that point into core memory. A second break request pulse, generated by the Central Computer System rather than the card reader in this case, results in the assignment of another memory cycle as a break-in cycle. During this cycle the remaining word is sent from the IO registers to core memory, via the memory buffer registers. The memory buffer registers are not available to the arithmetic circuits of the Central Computer System for the duration of these two break cycles; consequently, all arithmetic operations are temporarily suspended. If, however, an arithmetic pause had been in effect at the time that the first break request appeared, the pause operation would be allowed to continue uninterrupted because all pause operations, being of a repetitive and automatic nature, do not require access to the memory buffer registers.

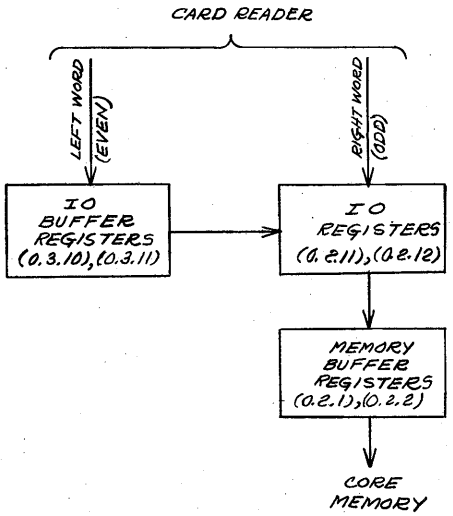


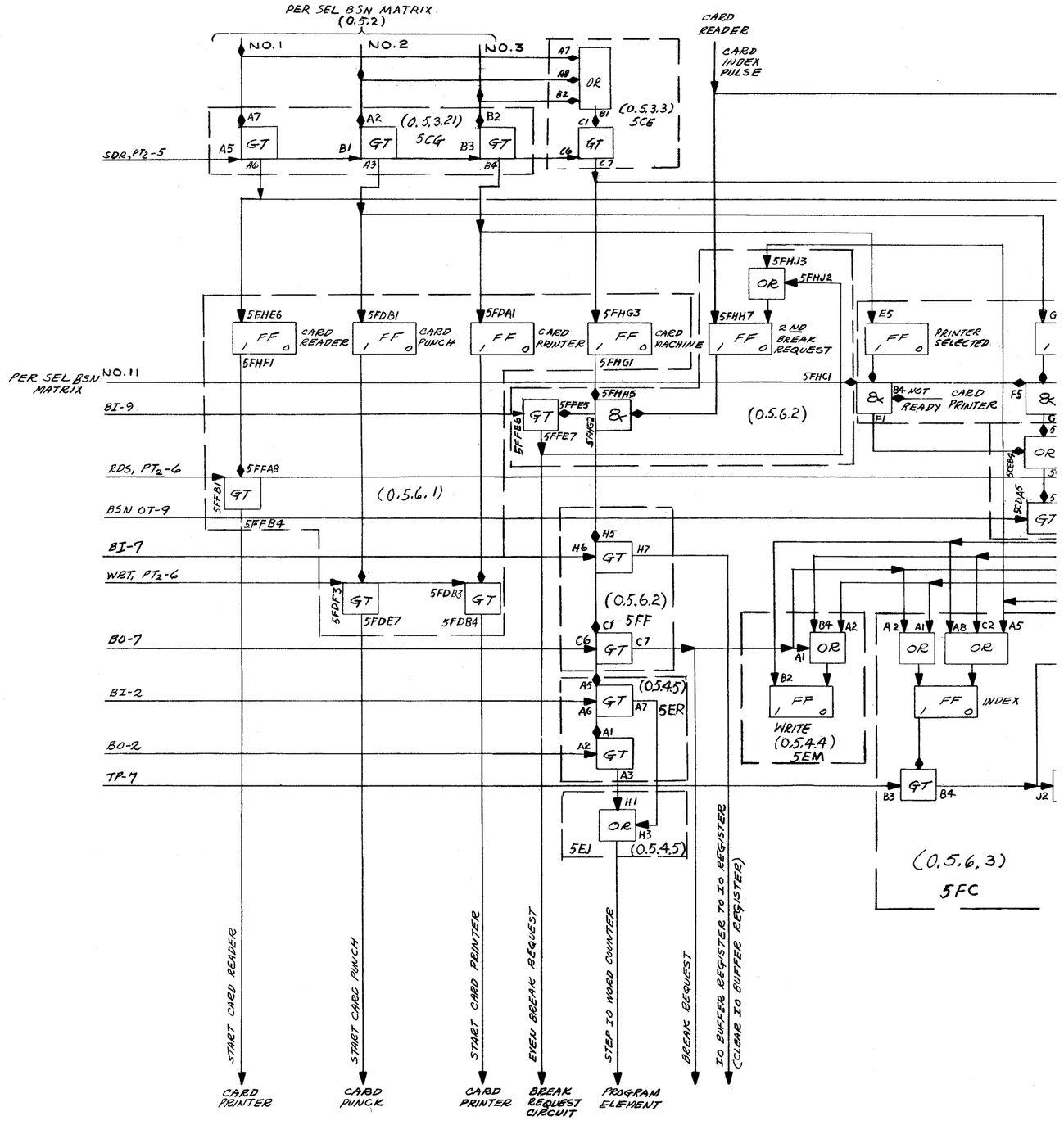
Figure 5–21. Information Flow—Card Reader

Once the two words have been placed in core memory, the Central Computer System continues its normal program. Approximately nine milliseconds later, another row of the card will be under the card reader brushes. A break request pulse will again be generated and the foregoing process will be repeated. When the requisite number of words have been read, a disconnect pulse generated by the Central Computer System will stop the card reader and no more cards will be read. This pulse is sent to all card machines. However, if the end of the required word block occurs somewhere in the middle of a card, the disconnect pulse will not be generated until the entire card has passed under the brushes. The remaining words on this card will not be read because the break request pulse generated at each index point will not be honored by the selection and IO control element circuits. No more memory cycles can therefore be assigned as break-in cycles.

When the Central Computer System is executing a *Write* instruction involving the card printer or the card punch, the information flow is along the channels shown in figure 5–23. A break request pulse sent by the card machine when it is at an index point will result in a word being transferred from core memory to the IO registers, via the memory buffer registers. This word then conditions the control grids of two thyratron buffer registers. The thyratron control flip-flop determines which thyratron buffer register is to be used as the word destination. When the flip-flop is in the 1 side, the 32 thyratrons comprising register A will have their shield grids conditioned; when it is in the 0 side, the 32 thyratrons comprising register B will have their shield grids conditioned.

Assume that for this first word the thyratron control flip-flop is in its 1 state. The 1's of the word in the IO registers will condition the control grids of thyratrons in the corresponding bit positions of both thyratron buffer registers. However, the shield grids of all these thyratrons are under the direction of the thyratron control flip-flop, which only raises the shield grid levels of the thyratrons in register A. Accordingly, only the thyratrons in register A will conduct.

A second break-out cycle is now generated by the Central Computer System, during which the status of the thyratron control flip-flop is re-


PER SEL BSN MATRIX (0.5.2)
NO.1
NO.2
NO.3
CARD READER
CARD INDEX PULSE
OR
(0.5.3.3) 5CE
(0.5.3.21) 5CG
GT
SDR, PT2-5
5FHJ3
5FHJ2
5FHE6
5FDB1
5FDA1
5FHG3
5FHH7
E5
CARD READER
CARD PUNCH
CARD PRINTER
CARD MACHINE
2 ND BREAK REQUEST
PRINTER SELECTED
5FHF1
5FHG1
5FHC1
PER SEL BSN MATRIX
NO.11
BI-9
5FHH5
5FFE5
5FFE6
5FFE7
5FHG2
(0.5.6.2)
B4 NOT READY
CARD PRINTER
RDS, PT2-6
5FFA8
5FFB1
(0.5.6.1)
BSN 0T-9
5FFB4
BI-7
WRT, PT2-6
5FDF3
5FDE7
5FDB3
5FDB4
BO-7
BI-2
BO-2
TP-7
5FF
(0.5.4.5)
5ER
5EJ
WRITE (0.5.4.4) 5EM
INDEX
(0.5.6.3) 5FC
START CARD READER
START CARD PUNCH
START CARD PRINTER
EVEN BREAK REQUEST
STEP IO WORD COUNTER
BREAK REQUEST
IO BUFFER REGISTER TO IO REGISTER (CLEAR IO BUFFER REGISTER)
CARD PRINTER
CARD PUNCH
CARD PRINTER
BREAK REQUEST CIRCUIT
PROGRAM ELEMENT

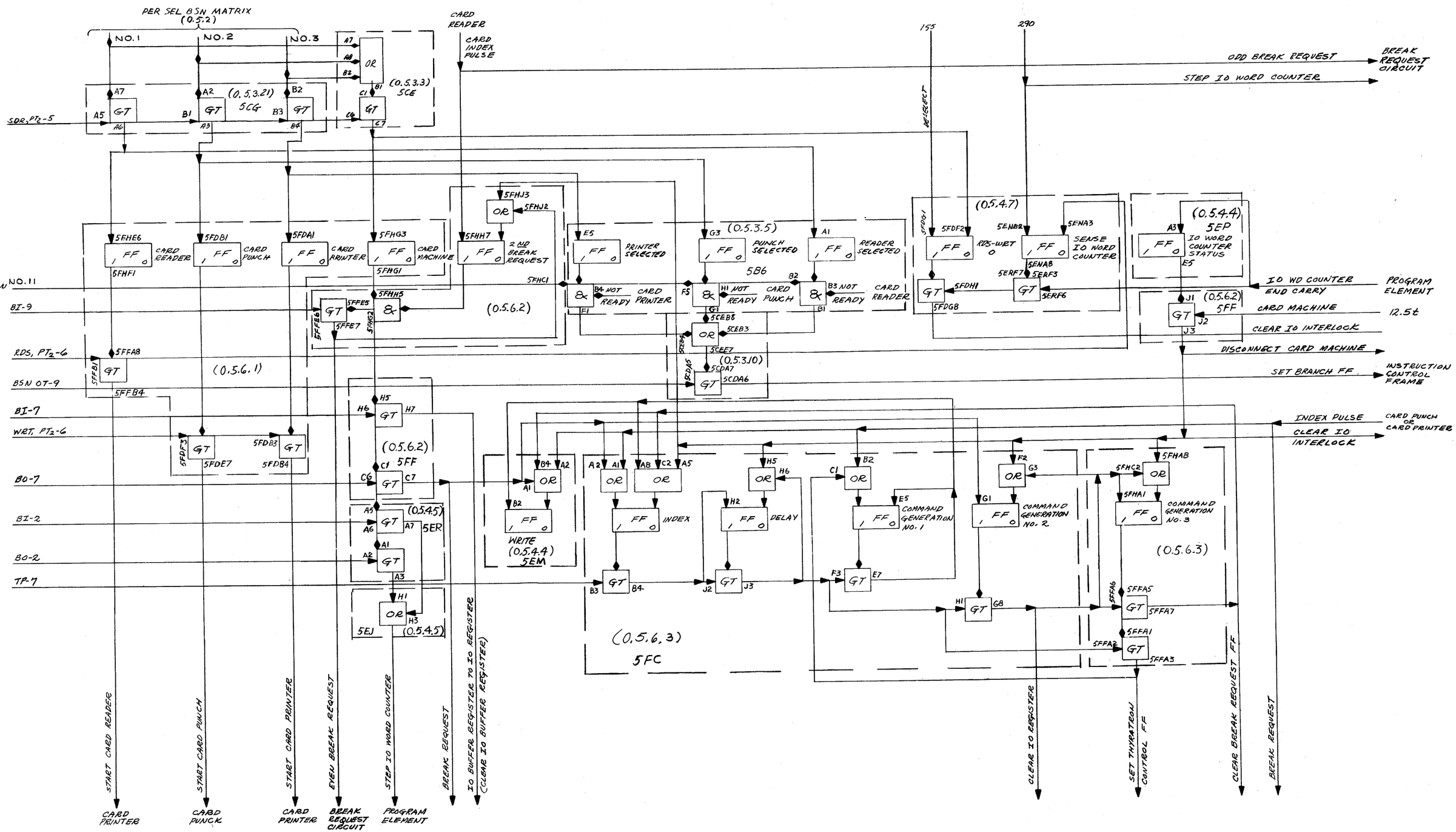


Figure 5-22. Card Machine Control Circuits, Logical Block Diagram

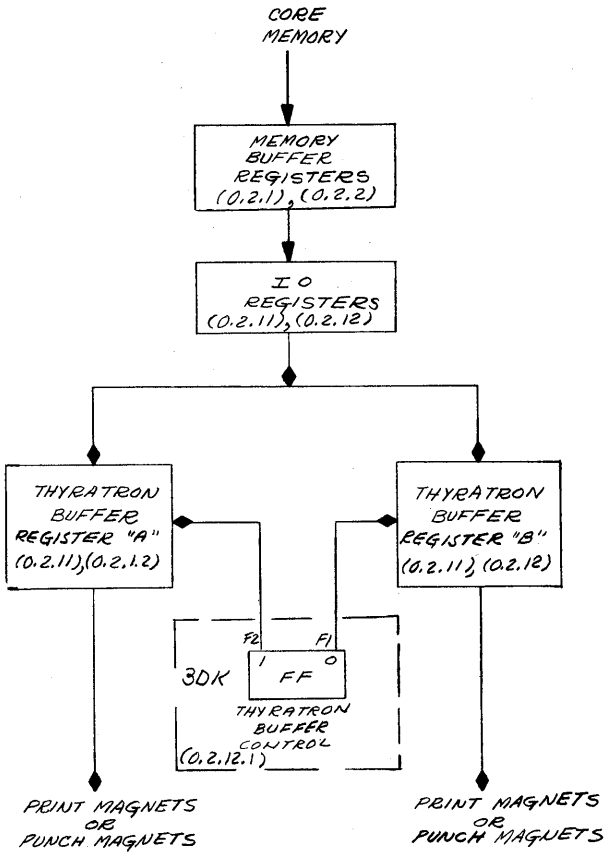


Figure 5–23. Information Flow, Card Printer and Card Punch

versed. Thyratron buffer register B will be made to fire, the procedure being identical to that of register A. When both thyratron buffer registers are thus loaded, the information they contain is transferred to the print magnets in the case of the card printer, and to the punch magnets in the case of the card punch. The Central Computer System resumes its normal program until, at the next index point, a break request pulse is again generated by the card machine. During the interval between word pair transfers, the plate voltage for the thyratron buffer registers is momentarily interrupted so that all thyratrons are reset to their unfired state before the next break sequence occurs.

3.3 CARD MACHINE SELECTION

A card machine is chosen by the *Select* instruction as the IO unit for an IO process. The index interval of this instruction designates the selected IO unit according to the following octal code:

01	Card Reader
02	Card Punch
03	Card Printer

The purpose of this instruction is to establish such circuit conditions that all other IO units are deselected and only the desired one has its circuits conditioned. Once an IO unit has been selected, it remains so until deselected by a subsequent *Select* or *Select Drum* instruction which specifies some other IO unit.

Figure 5–24 illustrates the circuits immediately affected when the card reader is selected. The index interval contents are decoded by the PerSelBsn matrix, and output No. 1 of this matrix conditions the two gate tubes shown in the figure. At OT–5 of the instruction, command 155 *(deselect)* clears all flip-flops shown in the figure except the read-write 0 flip-flop, which it sets. This same deselect pulse also resets all flip-flops in the selection and IO control element so that after resetting, the IO circuits of this element are in a neutral condition. At PT_2–5, command 156 *(select)*, after being generated by the instruction control element and transmitted to the selection and IO control element, examines the status of the two gate tubes originally conditioned by the No. 1 output of the PerSelBsn matrix. Since both gate tubes have their suppressor grid levels raised, the result of the interrogation by command 156 is that all flip-flops in figure 5–24 are set except the read-write 0 flip-flop, which is cleared.

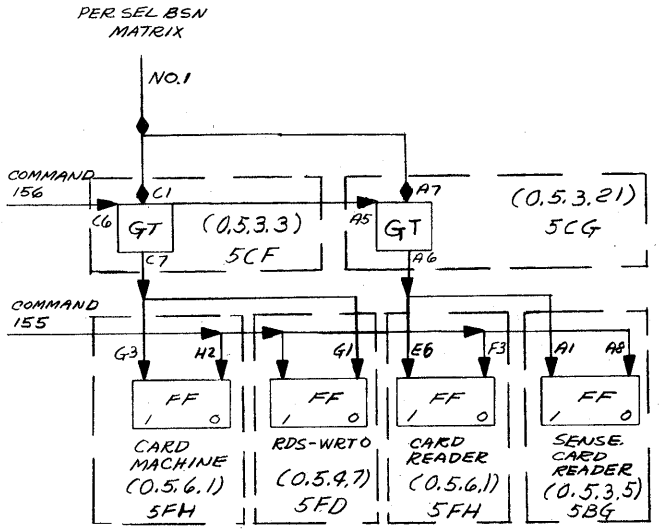


Figure 5–24. Selection Circuits, Card Reader, Logical Block Diagram

The card reader flip-flop is used to gate the start read pulse (occurring during the subsequent *Read* instruction) to the card reader. The card machine flip-flop is used to gate break-in (BI) pulses, which carry on the actual IO transfer

following a *Read* instruction. The read-write 0 flip-flop is used to inform the Central Computer System whether such an instruction *(Read 0* or *Write 0)* is a logical one for the IO unit selected. In the case of card machines it is logical, and therefore the flip-flop is cleared. The sense card reader flip-flop is used to inform the sense circuits that the card reader has been selected and that a branch procedure should take place if, the card reader upon being sensed, is not ready to partake in an IO operation.

Selection of the card printer or the card punch is accomplished in an identical manner, and figure 5–24, with slight variations, is applicable in both instances. To select the card punch, the No. 2 output of the PerSelBsn matrix is energized; for the card printer, the No. 3 output level must be raised. If the card punch is selected, the card reader and sense card reader flip-flops shown in figure 5–24 are replaced by the card punch and sense card punch flip-flops, respectively. If the card printer is selected, substitution of the card printer and sense card printer flip-flops for the card reader and sense card reader flip-flops is necessary. In conjunction with these flip-flops, the break control circuits described in Chapter 2 also affect the card machine-Central Computer System word transfer.

3.4 SENSE

The *Select* instruction may be followed by a *Sense* instruction to determine if the selected card machine is ready to engage in an IO operation. This instruction will sense the appropriate card machine flip-flop and associated sense circuits to see if certain conditions exist in the selected card machine which would prevent proper IO word transfers. Examples of conditions which impair or prevent an IO operation are: the power is off, a fuse is blown, the card reader stacker is full, a card feed failure has occurred, etc. If any of these conditions are present when an associated *Sense* (IO not ready) instruction is executed, the instruction will branch the program to prevent the IO operation from being initiated with a defective card machine.

Part of the circuits of the selection and IO control element which perform the sensing operations associated with the card machines are shown in figure 5–25. In order to carry out the sensing operation, the index interval bits (L10 through L15) of the *Sense* instruction must be octal 11, corresponding to a binary content of 001001. In this case the instruction is called *Sense* (IO not ready) and the decoding of the index interval content during the instruction will result in the energizing of output No. 11 of the PerSelBsn matrix. This will partially condition the three AND circuits shown in the figure. Each of these AND circuits is associated with a sense flip-flop. Basically, the *Sense* (IO not ready) instruction applies to all three card machines; however, in operation it will only sense for conditions existing in the particular card machine which has been selected by the previous *Select* instruction. In the discussion to follow it will be assumed that the card reader was selected and, hence, only the sense card reader flip-flop is set; the sense card punch and sense card printer flip-flops having been cleared by the *deselect pulse* (command 155) of the *Select* instruction. (Refer to 2.3 of this Chapter.)

Since the sense card reader flip-flop is in the 1 status and output No. 11 of the PerSelBsn matrix is positive due to the *Sense* (IO not ready) instruction, two of the conditions for the card reader AND circuit are met. The third and crucial condition to this AND circuit is a d-c level from the card reader itself. If this d-c level is absent (negative), it indictaes that the card reader is ready or able to be involved in an IO operation. However, if this d-c level is present (positive), it indicates that the card reader is not ready and should not be linked to the Central Computer System for an IO operation. In this latter case, the card reader AND circuit will be fully conditioned and in turn condition the gate tube shown in figure 5–25. This gate tube is inspected by an OT–9 pulse, or *sense pulse* (command 140), which is then gated to set the branch flip-flop in the instruction control element. It is important to note that if the card reader is ready, the card reader AND circuit will not be fully conditioned and the sense pulse will be inhibited so that the branch flip-flop will not be set. If the branch flip-flop is set, the Central Computer System will not execute the instruction in the next sequential memory location, which is usually the instruction of execution *(Read* or *Write)* of the IO program. Instead, it will execute the instruction contained in the memory location whose address is given by the address portion of the *Sense* instruction. Thus, the *Sense* instruction effectively prevents the Central Computer System from activating a card machine for an IO operation which is not capable of properly performing its function of transferring words.

Two other outputs of the PerSelBsn matrix, Nos. 31 and 32, are also associated with the card machine sensing operations. These provide the

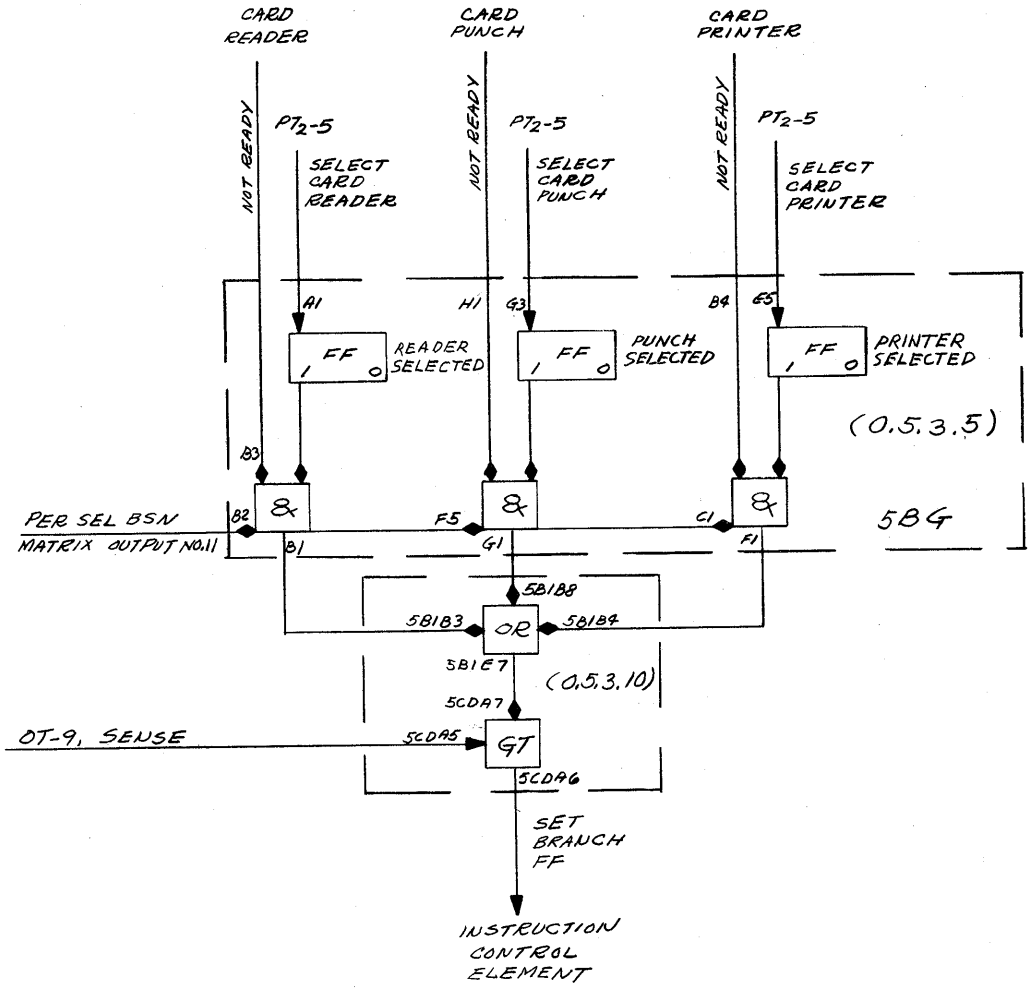


Figure 5–25. Sense Card Machine Circuit, Logical Block Diagram

Central Computer System with a means of sensing certain entry hubs on the card printer, and are associated with the *Sense* (printer No. 1) and *Sense* (printer No. 2) instructions. If a specified entry hub on the card printer is being impulsed, the *Sense* instruction will execute a branch in the program. If the specified entry hub (No. 1 or No. 2) is not being impulsed, the program will continue sequentially. The circuits which perform this sensing operation are shown in figure 5–26, and are similar in operation to the circuits just described, since the same OR circuit and gate tube are used for both circuits.

3.5 OPERATE

Insofar as the card machines are concerned, the *Operate* instruction need only be introduced into the IO program if the method of presentation for information to be printed or punched is to be altered from single spacing to double spacing. Moreover, since it is only applicable to a writing or break-out operation, the *Operate* instruction will never be contained in an IO program for the card reader. In any case, the index interval bits designate which operate units are to be energized.

For the card printer, there are 10 separate index interval codes from octal 51 through 62, corresponding to 10 separate operate units. Each of these operate units is an exit hub on the printer control panel. When an *Operate* instruction specifying one of these 10 operate units is executed in the Central Computer System, one of the associated 10 outputs of the PerSelBsn matrix will be energized. These 10 outputs are terminated with 10 gate tubes, only one of which will be conditioned during the instruction. At OT–9 of the *Operate* instruction, all of these gate tubes are sensed by the *sense operate gate tube* command (104) from the instruction control element. However, only the single, conditioned gate tube will develop an output pulse. This pulse is then sent to the card printer, where it is used to pick up the pilot selector specified by the *Operate* instruction index interval code.

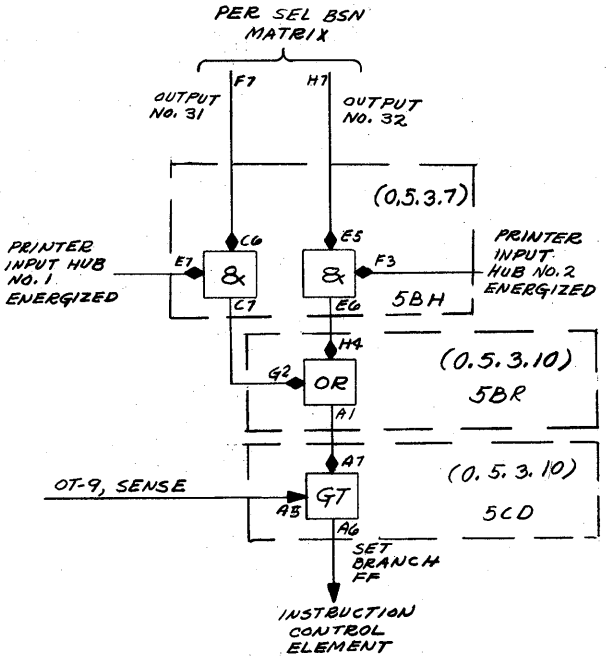


Figure 5–26. Sense Circuit—Card Printer, Logical Block Diagram

For the card punch, there are only two operate units corresponding to index interval codes octal 73 and 74. An *Operate* $(73)_8$ instruction causes information normally punched into columns 17 through 32 of a card to be punched into columns 1 through 16. An *Operate* $(74)_8$ instruction causes the information in columns 1 through 16 of the card to be punched into the corresponding columns of succeeding cards (gang-punching). The circuits which generate the operate signals for these two instructions are similar to those for the card printer. Outputs octal 73 and 74 of the PerSelBsn matrix are terminated with two gate tubes which are sensed by command 104, thus generating the specified operate pulse to the card punch. Table 5–4 lists the card printer and punch operate units and gives their related PerSelBsn octal matrix output number and binary index interval code.

3.6 READ OPERATION

The only card machine which can be read by the Central Computer System is the card reader. This IO unit reads punched cards through electromechanical action and transfers two words at a time to the Central Computer System, one word to the IO buffer register and the other to the IO register. The required instruction in a card reader IO program is therefore the *Read* instruction. If a *Write* instruction is incorrectly programmed,

TABLE 5–4. CARD MACHINE OPERATE UNITS

OPERATE UNIT	INDEX INTERVAL CODE	PERSELBSN MATRIX OUTPUT NO.
Card Printer		
Printer (1)	101 001	51
Printer (2)	101 010	52
Printer (3)	101 011	53
Printer (4)	101 100	54
Printer (5)	101 101	55
Printer (6)	101 110	56
Printer (7)	101 111	57
Printer (8)	110 000	60
Printer (9)	110 001	61
Printer (10)	110 010	62
Card Punch		
Punch (1)	111 011	73
Punch (2)	111 100	74

no starting pulse will be sent to the card reader. However, the IO interlock will be turned on during the execution of the *Write* instruction and, since the IO word counter cannot be reduced to 0, the interlock will not be turned off. The Central Computer System will continue with its normal program until a subsequent IO class instruction appears. Because of the IO pause incurred by the presence of the IO interlock, the Central Computer System cannot execute the new IO class instruction and, therefore, will remain in the IO pause condition indefinitely. This condition is referred to as hang-up. The only way to rectify a Central Computer System hang-up is to clear the IO interlock by manual intervention at the maintenance console.

3.6.1. *Read* Instruction

If the instruction following the *Select* (card reader) and *Sense* instructions has properly been the *Read* instruction, the transfer of words from the card reader to the Central Computer System can be effected. At PT_2–1 of the instruction, the instruction control element generates the commands *clear IO word counter* (149) and *set IO interlock on* (294). Command 149 is sent to the program element, where it clears the IO word counter. Command 294 is sent to the selection and IO control element. Its functions are illustrated in figure 5–27. By setting the IO interlock, this command eliminates the possibility of initiating

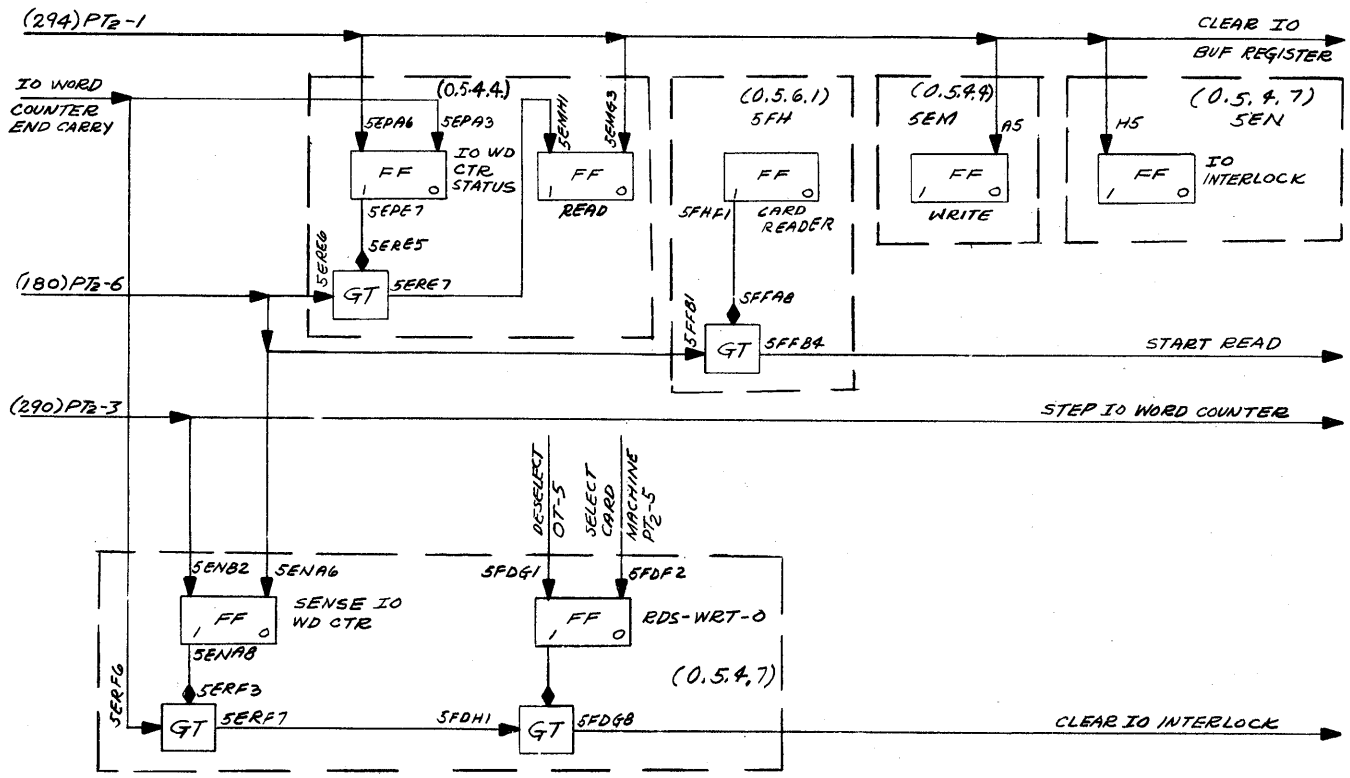


Figure 5–27. Read Instruction—Card Reader, Logical Block Diagram

a conflicting IO program. By clearing both the read and the write flip-flops, the command nullifies the conditions set up by any previous IO process, which could have been either a read or write operation. By setting the IO word counter status flip-flop, an indication is supplied to the Central Computer System that the contents of the IO word counter are different from zero. The IO word counter contents are actually made different from zero one-half-microsecond later (at PT_2–2) by the *address register complement to IO word counter* command (152), which transfers the complement of the address register contents to the IO word counter.

At PT_2–3, the *sense IO word counter ≠0* command (290) sets the sense IO word counter flip-flop in the selection and IO control element, sets the CSW flip-flop in the instruction control element, and steps the IO word counter in the program element. The IO word counter is stepped to see if an end carry will result. If it does, the IO word counter contained negative zero, the instruction is *Read 0*, and therefore no words should be transferred. However, whenever a card machine or a tape unit is selected, a *Read 0* instruction is a logical one, being intended to make the Central Computer System skip a punched card or a tape record. The IO interlock, therefore, should not be cleared, the card machine or tape unit should be allowed to run without interruption. If the selected IO device is a drum field or miscellaneous IO unit, the *Read 0* instruction has no meaning, and the IO interlock should be immediately cleared. The read-write 0 flip-flop is used in conjunction with the sense IO word counter flip-flop and with the IO word counter end carry pulse to implement this interpretation of the programmer's intent. The read-write 0 flip-flop is set at OT–5 of every *Select* or *Select Drums* instruction, but it is cleared at PT_2–5 of the same instruction only if the selected IO device is a card machine or a tape unit.

An end carry pulse, when and if it is generated by the IO word counter, is returned to the selection and IO control element. As shown in figure 5–27, it immediately clears the IO word counter status flip-flop to make its reading concur with the actual status of the counter. This pulse simultaneously examines a gate tube conditioned by the 1 side of the sense IO word counter flip-flop. The examination takes place at approximately PT_2–5, owing to the inherent delays encountered by the original stepping pulse in propagating through the 16 stages of the counter and in transmission between units. Since the sense IO word counter flip-flop was set by the

same pulse that stepped the word counter (command 290), the examining pulse is passed and examines a gate tube conditioned by the 1 side of the read-write 0 flip-flop. In this instance, since the card reader has been selected, the gate tube will not be conditioned and the pulse is suppressed. This suppression avoids clearing the IO interlock and enables the card reader to skip a card without terminating the IO process. Note that the sensing procedure just described reduces the contents of the IO word counter by 1. If it is assumed that n words are to be transferred, the IO word counter now contains n–1, although no words have yet been transferred.

At PT_2–6, the *PT–6 on read* command (180) senses the gate tube associated with the card reader flip-flop. (See fig. 5–27.) The flip-flop was set during the *Select* instruction and therefore this gate tube is conditioned. The resultant pulse is sent to the card reader as a start read pulse. Command 180 also sets the read flip-flop and clears the sense IO word counter flip-flop. The sense IO word counter flip-flop is cleared because it has already served its purpose and must be readied for any subsequent *Read* or *Write* instruction. The read flip-flop is set so that the break-in command generators will be partially conditioned. The start read pulse sent to the card reader causes the first card in the card hopper to be fed under the brushes.

3.6.2 Break Request Generation

The IO process must now await the arrival of a break request pulse from the card reader. (Refer to 1.1 of Section 1.) As explained in that paragraph, index points of the card machine cycle are associated with each of the 12 rows of card information. At each index point, two words (columns 17 through 80) are simultaneously transferred to the Central Computer System input registers and an odd break request pulse is sent to the selection and IO control element. The odd break request pulse sets the break request flip-flop and the second break request flip-flop. The break request flip-flop is examined by the TP–11 pulse of the existing memory cycle or, if the Central Computer System is in an arithmetic pause, by the next 2-megacycle pulse. Because the flip-flop is in the 1 state, this results in setting two break flip-flops, one in the selection and IO control element and the other in the instruction control element. The function of the instruction control element break flip-flop is to ensure that time pulses will be continuously available throughout the entire IO process regardless of the presence or absence of an arithmetic pause condition. (Refer to 1.5.2 of Chapter 1.) The fact that both the read flip-flop and the selection and IO control break flip-flop are set results in the generation of a break-in memory cycle.

3.6.3 Break-In Memory Cycle

Generation of the start read pulse is under the control of the card reader flip-flop. Generation of the activating pulses during a break-in memory cycle is under the control of the card machine flip-flop. (See fig. 5–28.) At BI–1, the memory buffer registers and the break request flip-flop are cleared, the former in anticipation of the receipt of the first word to be transferred, and the latter to permit further break requests to be recorded. At the same time, the IO address counter contents, which indicate the address of the first core memory register to be used, are transferred to the memory address register in the memory element. The next pulse, BI–2, steps the IO address counter, changing its contents to indicate the second required core memory address. Subject to the condition that the card machine flip-flop be in the 1 side, the BI–2 pulse also steps the IO word counter, changing its contents to $-n+2$. The BI–2 pulse also effects the transfer of the first word from the IO registers to the memory buffer registers, from which point the normal action of the memory circuits will serve to place the word in the core memory location specified by the memory address register. The IO register is cleared during the transfer.

At BI–3, a parity count is performed on the word in the memory buffer registers (first word of transfer) and a parity bit is assigned. At BI–7, the word and its attached parity bit are transferred to core memory. The BI–7 pulse transfers the second word from the IO buffer registers to the IO registers, subject to the condition that the card machine flip-flop be in the 1 side. (See fig. 5–28). The IO buffer registers are cleared during this process. The next break-in pulse to affect the card reader IO process is BI–9. This pulse examines a gate tube which is conditioned whenever both the card machine and the second break request flip-flops are in the 1 status. Since this is the case, the BI–9 pulse is passed and immediately clears the second break request flip-flop. It is also sent to the break request circuit as the even break request pulse. If the IO word counter is not equal to 0 (it is assumed that it is not), the even break request pulse will set the break request flip-flop. Upon being examined by the following TP–11 pulse (coincident with BI–11), the

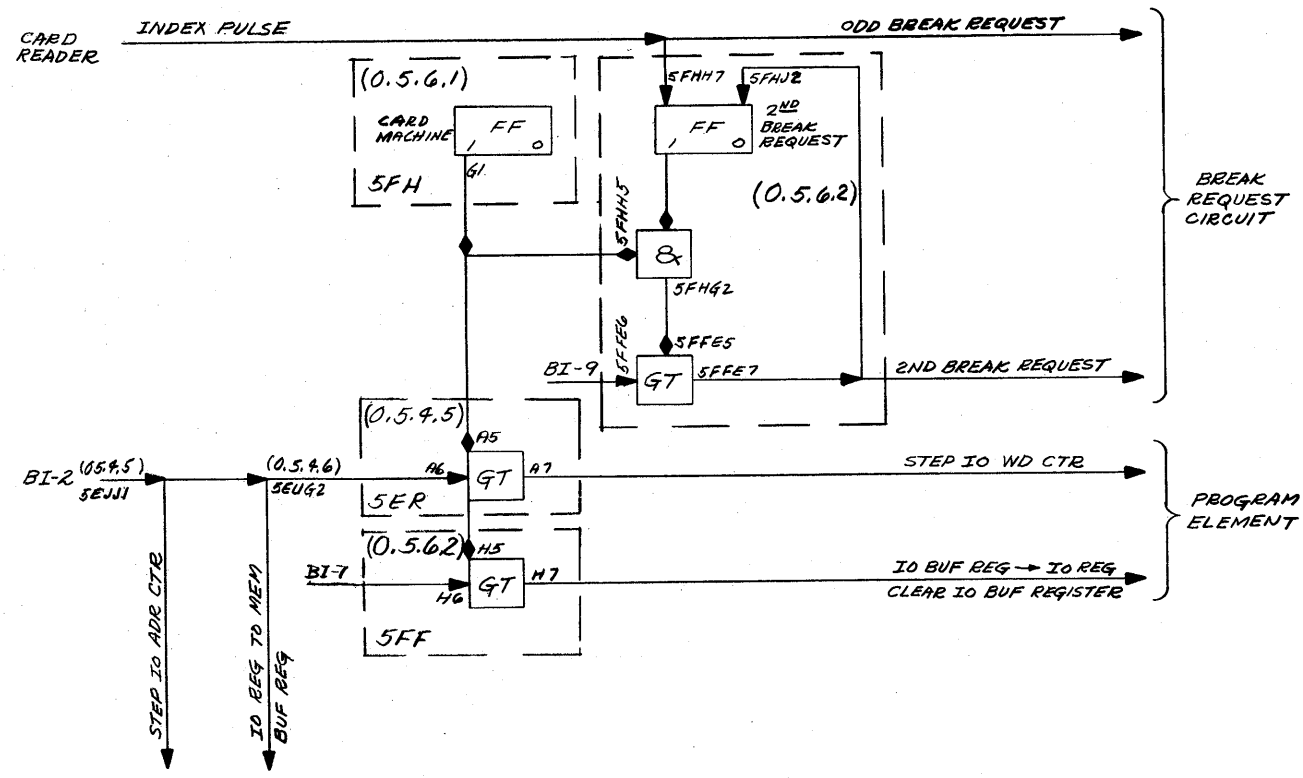


Figure 5–28. Break-In Cycle Controls—Card Reader, Logical Block Diagram

break request flip-flop will cause the Central Computer System to assign the forthcoming memory cycle as a break-in cycle.

At the start of the second break-in cycle, the first word of the transfer is already in core memory and the second is in the IO register. The memory address register still contains the core memory address for the first word, the core memory address for the second word being held by the IO address counter. The IO word counter reads —n+2. The break, read, and break request flip-flops are all set. At TP–0, the memory address register is cleared so that it may receive the second core memory address from the IO address counter at BI–1. Until the appearance of BI–9, the procedure during the second break-in cycle is identical with that of the first break-in cycle. By the time that BI–9 occurs, the second word has been placed in its core memory location. The BI–9 pulse is suppressed because the second break request flip-flop is now in the 0 status. Consequently, no even break request pulse is developed, the break request flip-flop which was cleared at BI–1 is not reset, and the two break flip-flops are cleared when the break request flip-flop is sensed at TP–11. The ensuing memory cycle is therefore assigned as a machine cycle. The Central Computer System continues its arithmetic program until the next card reader index pulse is received, about nine milliseconds later. At that time, the entire process will be repeated and another word pair will be placed into core memory. Note that, because the IO address counter is stepped in increments of 1, the successive words of the IO transfer will be placed in consecutive core memory locations. In the interval between the first and second card reader index pulses, the IO address counter will contain the desired address for the third word to be transferred and the IO word counter will contain —n+3.

3.6.4 IO Process Termination

The IO process continues in this manner until, as a result of stepping the IO word counter, an end carry pulse is developed which signifies that the IO word counter contents are now 0 and that no more break-in cycles are to be assigned. The pertinent circuits are shown in figure 5–29. Because of inherent delays, the end carry pulse will not appear in the selection and IO control element until approximately BI–4, where it will clear the IO word counter status flip-flop. This action disables the break request circuit so that no more break requests will be honored. The break request flip-flop, cleared by the BI–1 pulse of the last break-in cycle, will remain in the 0 status. Examination of the break request flip-flop at TP–11 will cause the break flip-flop to be cleared. All further break requests (in the form

of card reader index pulses) will also be ignored because the IO word counter status flip-flop is in the 0 status. The IO interlock flip-flop, however, remains in the 1 status, prolonging the IO pause and thereby preventing any new IO instruction from being performed. As the trailing edge of the card on which the last word to be transferred was located passes under the read brushes of the card reader, a 12.5-t pulse of the card machine cycle is sent to the selection and IO control element. This pulse passes through the gate tube conditioned by the 0 side of the IO word counter status flip-flop, simultaneously clears the IO interlock flip-flop through a synchronizing circuit, and disnects the card reader. The first action terminates the IO pause and the second stops the card reader.

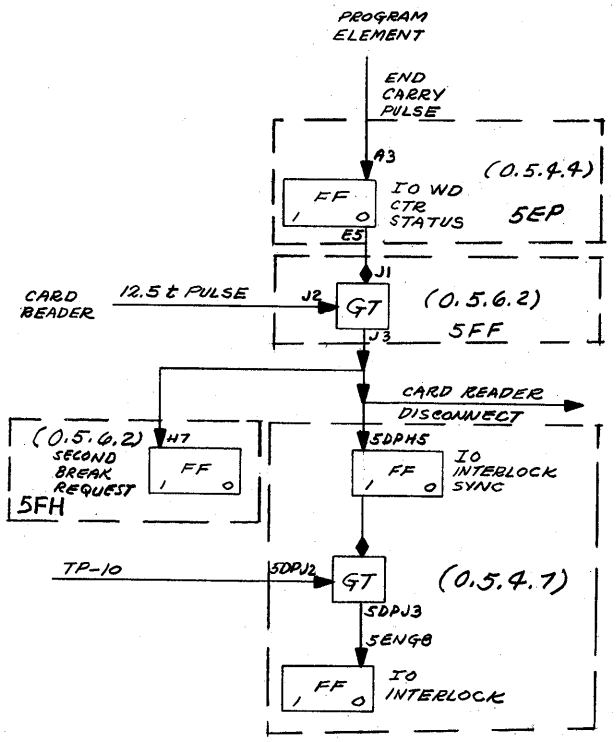


Figure 5–29. IO Process Termination Circuits—Card Reader, Logical Block Diagram

3.7 WRITE OPERATION

The card printer and the card punch are the only two card machines which can transform information generated by the Central Computer System into readable form. The operation of the card reader; i.e., a card passes under the punch magnets and, as each of the 12 card rows position themselves under the magnets, a word pair is transferred to columns 17 through 80 of the card. The distinguishing characteristic of this type of transfer is that after every two break-out cycles, the word pair appears in its final form (a series of punches in a card row).

The card printer operates differently in several respects. First, by using the Hollerith code, the binary information in core memory may have to be translated to an alphabetical representation as its final form. Secondly, since each line written by the card printer contains 24 words, the results of 24 break-out cycles must be stored before allowing the printer to write. Both of these problems are resolved by an analyzer which accepts the transferred words from the two thyratron buffer registers, decodes them when necessary, and times the printing action. As far as the Central Computer System is concerned, these problems do not have to be considered if the thyratron buffer registers in the card printer are considered to be the output devices.

3.7.1 *Write* Instruction

The proper instruction to follow a *Select* (card punch) or *Select* (card printer) instruction is the *Write* instruction. The execution of this instruction is very similar to the execution of the *Read* instruction. (Refer to 3.6.1 of Chapter 3.) The only difference is that command 182 replaces command 180 at PT_2–6. Whereas command 180 set the read flip-flop and inspected the card reader flip-flop, command 182 sets the write flip-flop and examines both the card punch and card printer flip-flops. Either the card punch or the card printer flip-flop, depending on which card machine has been selected, will, upon being examined, cause generation of a start write pulse. This pulse is then sent to the appropriate card machine and the Central Computer System will continue with its program while awaiting the first card index pulse.

3.7.2 Break Request Generation

Break requests for both the card printer and the card punch are controlled by the same circuits. The action of these circuits, described in detail in 1.1.1 of Section 1, will be briefly summarized in the following paragraphs. The purpose of these circuits is twofold: to provide the pulses even though the Central Computer System is in an arithmetic pause, and secondly, to introduce the time delays required by the thyratron buffer registers for stabilization.

A card index pulse received from either the punch or the printer sets the index and break request flip-flops and clears the write flip-flop (set at PT_2–6 of the *Write* instruction). As a result

of setting the break request flip-flop, the break flip-flop is also set, causing the generation of time pulses regardless of the presence or absence of an arithmetic pause condition. However, since the write flip-flop has been placed in the 0 status, no break-out pulses will be generated.

Because of the action of the delay flip-flop, two memory cycles will elapse before a TP–7 clears the No. 1 and index flip-flops and sets the write flip-flop. As a result of setting the write flip-flop, a break-out memory cycle will be executed. However, before it is executed, 16 microseconds will have passed, during which time the shield grids of one 32-bit thyratron buffer register will have been conditioned.

The break-out memory cycle is now executed and, at BO–7, the write flip-flop is again cleared and the break request, index, and No. 2 flip-flops are set. Setting the break request flip-flop assures the generation of time pulses for the ensuing operations. Another two memory cycles elapse before a TP–7 pulse clears the IO registers. This delay insures proper firing of the thyratrons. The TP–7 pulse also sets the No. 3 flip-flop and clears the No. 2 flip-flop. Two more memory cycles elapse before the shield grids of the other 32-bit thyratron buffer register are conditioned as the result of a TP–7 pulse which sets the thyratron control flip-flop. (The thyratron control flip-flop was originally cleared by the card index pulse.) The same pulse also sets the No. 1 flip-flop.

The process described above is repeated and the second word of the transfer is used to condition the other thyratron buffer register during the second break-out cycle. Note that in a card machine write operation, break request pulses as such do not exist. Break-out cycles are assigned by leaving the break flip-flop continuously in the 1 status and setting the write flip-flop whenever a break-out cycle is required. The entire process for a single word pair is still initiated, however, by the card index pulses sent from the selected card machine to the Central Computer System at a rate of approximately one every nine milliseconds.

3.7.3 Break-Out Memory Cycle

Two memory cycles elapse between the end of the memory cycle (during which the card index pulse appears) and the initiation of the first break-out cycle. At BO–1 of this cycle, the break request flip-flop is cleared and the IO address counter contents are transferred to the memory address register. At BO–2, both the IO address counter and the IO word counter are stepped. In the time interval between BO–1 and BO–6, the core memory circuits place the word in the address specified by the memory address register into the memory buffer registers. The BO–7 pulse transfers this word to the IO registers. The same pulse also sets the break request, index, and No. 2 flip-flops and clears the write flip-flop. Since the shield grids of the first thyratron buffer register were conditioned before the appearance of the break-out cycle, and the control grids of the same thyratrons are connected to the IO register flip-flops, the first thyratron buffer register will fire and energize its associated magnets in either the printer or the punch. The first word is therefore completely processed.

Since the write flip-flop is in the 0 status at the end of the first break-out cycle, the following memory cycle will not be a break-out cycle, even though the break flip-flop is set. In fact, six dummy memory cycles are developed by the action of the break request generation circuits before the second break-out cycle is assigned. In the course of these six dummy cycles, the IO registers are cleared and the thyratron control flip-flop status is reversed. As a result, the IO register is ready to receive the second word and the shield grids of the second thyratron buffer register are conditioned. At TP–7 of the sixth dummy cycle, the write flip-flop is set, thereby forcing the Central Computer System to assign the forthcoming memory cycle as the second break-out cycle. The functions performed by this second series of BO pulses are identical with those performed during the first break-out cycle.

After the BO–7 pulse, the second thyratron buffer register fires and a second set of punch or printer magnets is energized. The write flip-flop is cleared during this break-out cycle, but the break flip-flop is not. The ensuing two memory cycles are therefore also dummy cycles, during which no arithmetic operations take place. The time is utilized to reset the break command generation circuits, to clear the IO registers, and finally, during the second dummy cycle, to clear the break flip-flop. The forthcoming memory cycle is therefore a machine cycle, enabling the Central Computer System to return to its arithmetic program while it awaits the next card index pulse.

3.7.4 IO Process Termination

The word pair transfers continue until the IO word counter is reduced to 0. The resulting end carry pulse clears the IO word counter status flip-flop, disabling the break request circuit. No further break requests will be honored. The IO

interlock flip-flop, as in the case of card reader operation, is not cleared until the 12.5-t index pulse is received from the selected card machine. The 12.5-t pulse, after being gated by the 0 side of the IO word counter status flip-flop, disconnects the selected card machine, clears the No. 2, No. 3, index, and delay flip-flops, and clears the IO interlock.

CHAPTER 4
DRUM SYSTEM
SECTION 1
GENERAL

The Drum System of AN/FSQ–7 (XD–1, –2) Combat Direction Centrals contains 39 separate fields located on six physical drums. Each field has a maximum storage capacity of 2048 words, and is considered and treated as a distinct IO unit by the Central Computer System. In general, the fields of the Drum System differ from one another in respect to their mode of reading and writing data, presence or absence of parity bits in their stored word content, and in the type of information each field stores. Furthermore, some fields can only accept information from the Central Computer System (write), whereas others can only supply information (read). Because of the different inherent characteristics of these various fields, the method of instituting and carrying out IO processes with the Central Computer System varies from field to field. The various IO processing properties of these 39 fields are listed in table 5–5.

In relation to IO transfers between the Central Computer and the Drum Systems, the function of the selection and IO control element is threefold:

a. Decoding
b. Selection
c. IO control

The decoding function involves determination of which drum field has been selected by the existing program. This is accomplished by the decoding of the index interval bits of the *Select Drum* instruction by the PerSelBsn matrix and within the drum units. In the performance of its selection function the selection of an IO control element sets up the necessary control circuits which are applicable to the particular field selected. The final function of the element is to initiate, control, and terminate the actual flow of words between the selected field and the Central Computer System in the particular manner applicable to the characteristics of the selected field. As in all IO operations, the selection and IO control element must also synchronize the assignment of the memory buffer registers and core memory between arithmetic and IO operations. Figure 5–30 is a complete logical block diagram of all the circuits in the selection and IO control element which apply to the IO operation of transferring words from the Drum System to the Central Computer System. Similarly, figure 5–31 shows all of the circuits that apply to IO word transfers from the Central Computer System to the Drum System. These two figures may be referred to throughout the following discussions to show the interconnections between the circuits and to provide an overall picture of the drum control circuits.

1.1 DRUM OPERATION MODES

There are three distinct modes of determining the location of data on the various drum fields: the address mode, the identity mode, and the status mode. The mode to be used for reading or writing on most drum fields is predetermined by the design of the field although some fields may be read by status or identity depending on the *SDR* code. As soon as a drum field has been selected, the circuits of the selection and IO control element establish the proper conditions for processing the desired information on that drum field in the mode dictated by its design. For those drum fields which exist on drums processed by the address mode, an additional variation is provided. This variation is termed the interleave method; it enables the Central Computer System to read or write into addresses which are spaced by a constant numerical increment. There are three interleave methods, making it possible to affect every 8th, 16th, or 64th drum register.

The address mode is used for both reading and writing operations. Each word on an addressable drum has an address assigned to it. These addresses are used to find data stored in these fields. When using the address mode, it is only necessary to locate the address for the first word to be transferred. The addresses required for the remainder of the words are then selected consecutively and automatically. If an interleave method is also requested, the method remains the same but only those drum registers separated by the selected numerical increment are processed.

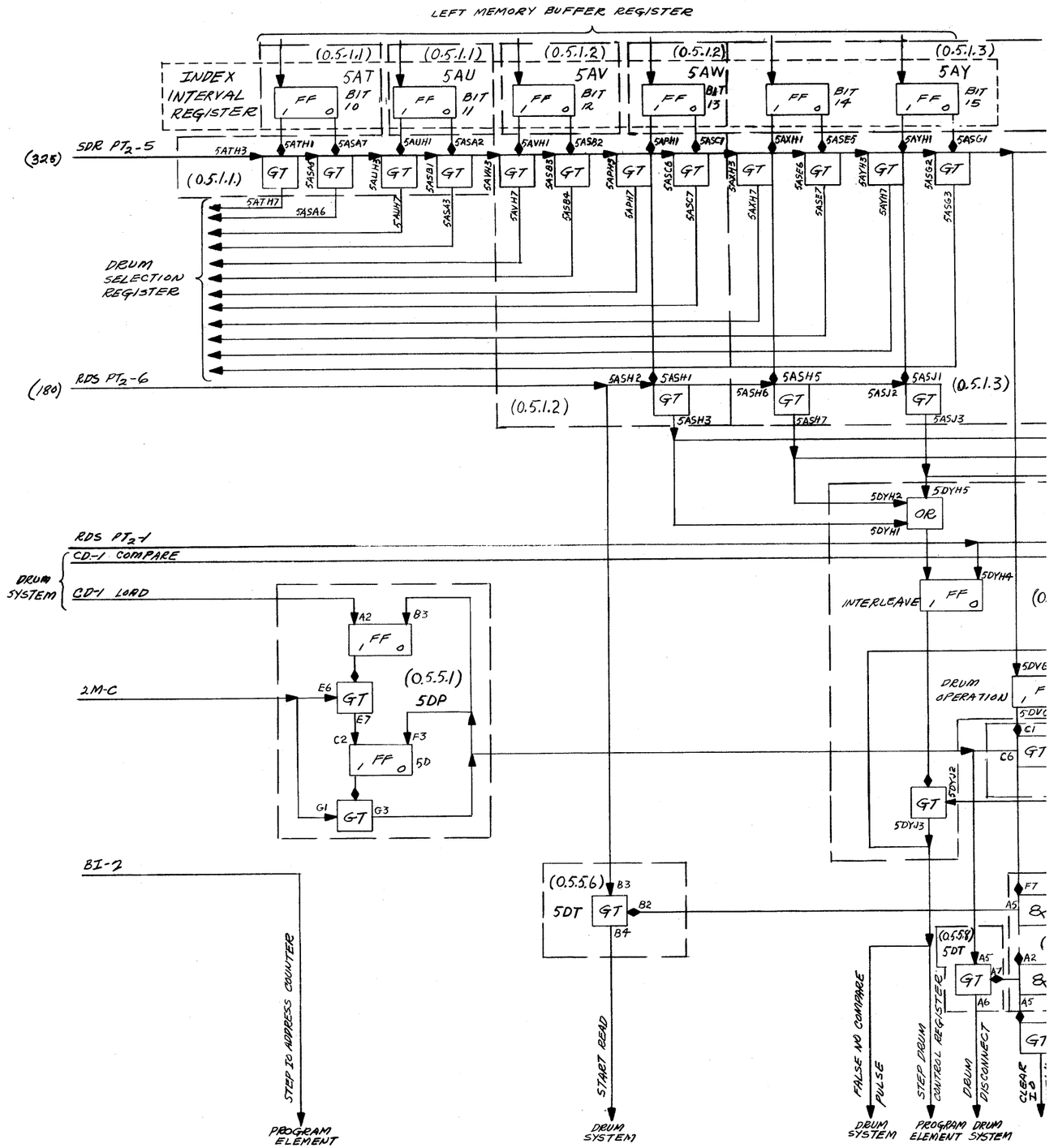
TABLE 5–5. DRUM SELECTION CODES

DRUM FIELD	OCTAL CODE	READ	WRITE	MODE
Auxiliary memory No. 1	02	Yes	Yes	By address
Auxiliary memory No. 2	03	Yes	Yes	By address
Auxiliary memory No. 3	04	Yes	Yes	By address
Auxiliary memory No. 4	05	Yes	Yes	By address
Auxiliary memory No. 5	06	Yes	Yes	By address
Auxiliary memory No. 6	07	Yes	Yes	By address
Auxiliary memory No. 7	10	Yes	Yes	By address
Auxiliary memory No. 8	11	Yes	Yes	By address
Auxiliary memory No. 9	12	Yes	Yes	By address
Auxiliary memory No. 10	13	Yes	Yes	By address
Auxiliary memory No. 11	14f	Yes	Yes	By address
Auxiliary memory No. 12	15	Yes	Yes	By address
Intercommunication (other)	16	Yes	No	By address
Digital display test	17	Yes	No	By identity (*)
Spare No. 1	20	Yes	Yes	By address
Spare No. 2	21	Yes	Yes	By address
Manual input	22	Yes	No	By status (**)
Manual input	23	Yes	No	By identity
Crosstelling	24	Yes	No	By status (**)
Crosstelling	25	Yes	No	By identity
Intercommunication (own)	26	Yes	Yes	By address
Digital display	27	Yes	Yes	By address
Output buffer odd	30	No	Yes	By status (***)
Output buffer even	31	No	Yes	By status (***)
Misc. radar input No. 1	32	Yes	No	By status (**)
Misc. radar input No. 2	33	Yes	No	By identity
Radar input No. 1	34	Yes	No	By status (**)
Radar input No. 1	35	Yes	No	By identity
Radar input No. 2	36	Yes	No	By status (**)
Radar input No. 2	37	Yes	No	By identity
Crosstelling marker	40	No	Yes	(****)
Track display No. 1	41	Yes	Yes	By address
Track display No. 2	42	Yes	Yes	By address
Track display No. 3	43	Yes	Yes	By address
Track display No. 4	44	Yes	Yes	By address
Track display No. 5	45	Yes	Yes	By address
Track display No. 6	46	Yes	Yes	By address
SD (test)	47	Yes	No	By identity (*)
Radar data No. 1	60	Yes	Yes	By address
Radar data No. 2	61	Yes	Yes	By address
Radar data No. 3	62	Yes	Yes	By address
Radar data No. 4	63	Yes	Yes	By address
Radar data No. 5	64	Yes	Yes	By address
Radar data No. 6	65	Yes	Yes	By address
Radar data No. 7	66	Yes	Yes	By address
Radar data No. 8	67	Yes	Yes	By address
Radar data No. 9	70	Yes	Yes	By address

**Read output side by identity for test purposes*
***Write on output side by status demand for test purposes*
****Read output side by status demand for test purposes*
*****Write in the crosstelling marker channel by address*

The status mode can be used for both reading and writing operations. Each storage location on fields read or written by the status mode has an extra bit assigned to it, termed the status bit. If the status bit is a 1, it indicates that the location is full. A 0 indicates that the location is empty. In reading a drum field by the status mode, the Central Computer System only reads data from those locations whose status bits are 1's. Similarly, in writing on a drum field by the status mode, the Central Computer System only places data in those locations whose status bits are 0's.

The identity mode can only be used for a reading operation. It is characterized by the appearance of identification bits in each word stored on a field processed by this mode. When this method is in effect, the Central Computer System searches the selected field and reads those words whose status bits are 1's. Of all these words, only those which have identification bits identical to those given by the address part of the *Select Drum* instruction are accepted by the Central Computer System. The identity mode can therefore be considered a variation of the status mode, in which only those words having the desired identification bits are acceptable.


LEFT MEMORY BUFFER REGISTER
INDEX INTERVAL REGISTER
(0.5.1.1) 5AT BIT 10
(0.5.1.1) 5AU BIT 11
(0.5.1.2) 5AV BIT 12
(0.5.1.2) 5AW BIT 13
BIT 14
(0.5.1.3) 5AY BIT 15
FF
GT
OR
(326) SDR PT2-5
(180) RDS PT2-6
(0.5.1.1)
(0.5.1.2)
(0.5.1.3)
DRUM SELECTION REGISTER
5ATH3
5ATH1
5ASA7
5AUH3
5AUH1
5ASA2
5AVH3
5AVH1
5ASB2
5AWH3
5AWH1
5ASC1
5AXH3
5AXH1
5ASE5
5AYH3
5AYH1
5ASG1
5ATH7
5ASA6
5ASA4
5AUH7
5ASA3
5ASA1
5AVH7
5ASB4
5ASB3
5AWH7
5ASC7
5ASC6
5AXH7
5ASE7
5ASE6
5AYH7
5ASG3
5ASG2
5ASH2
5ASH1
5ASH6
5ASH5
5ASJ2
5ASJ1
5ASH3
5ASH7
5ASJ3
5DYH2
5DYH5
5DYH1
5DYH4
INTERLEAVE
RDS PT2-1
CD-1 COMPARE
DRUM SYSTEM
CD-1 LOAD
2M-C
(0.5.5.1) 5DP
5DT
5DYJ2
5DYJ3
DRUM OPERATION
(0.5.5.6) 5DT
(0.5.5.8) 5DT
BI-2
STEP TO ADDRESS COUNTER
PROGRAM ELEMENT
START READ
DRUM SYSTEM
FALSE NO COMPARE PULSE
STEP DRUM CONTROL REGISTER
DRUM DISCONNECT
PROGRAM ELEMENT
DRUM SYSTEM
CLEAR

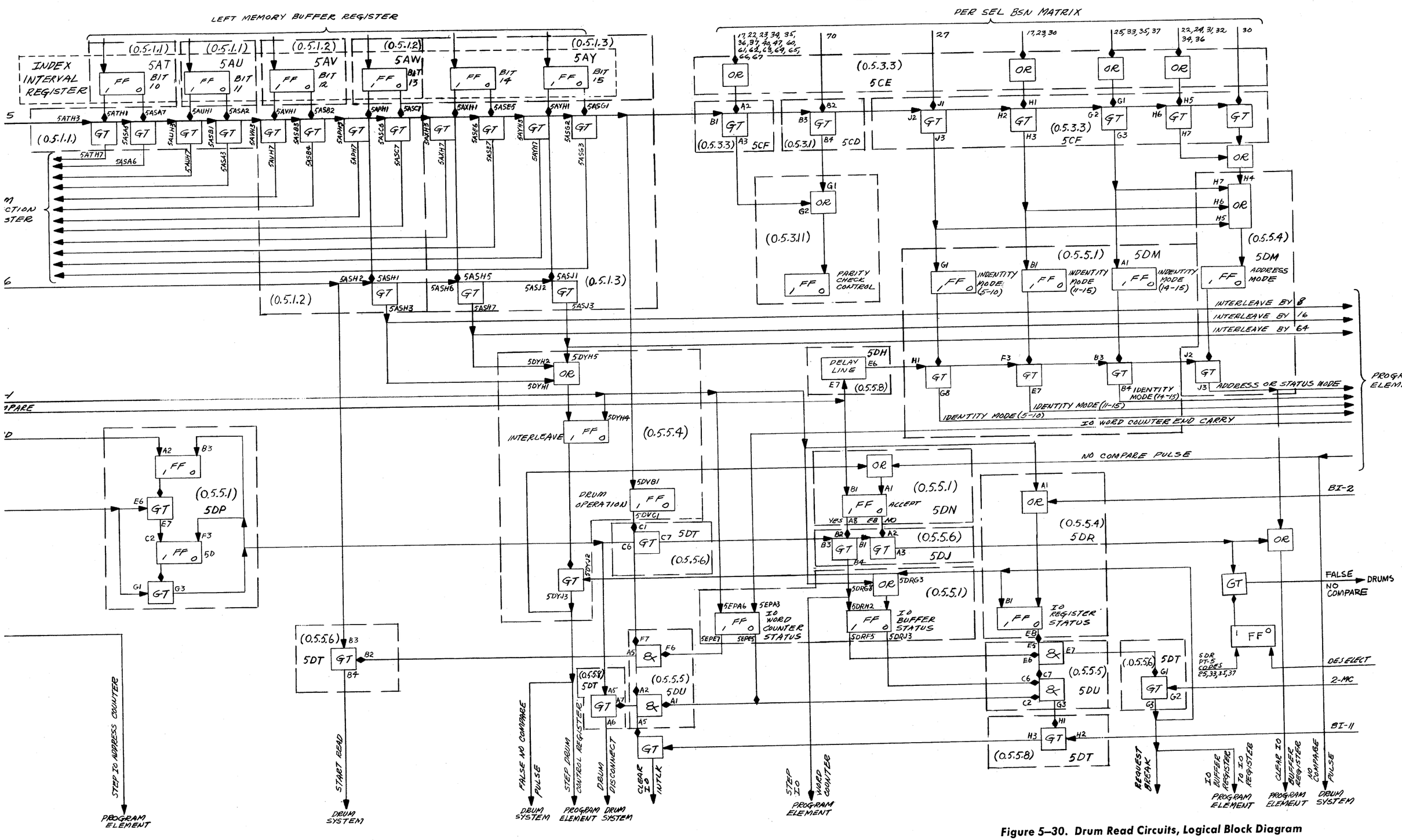
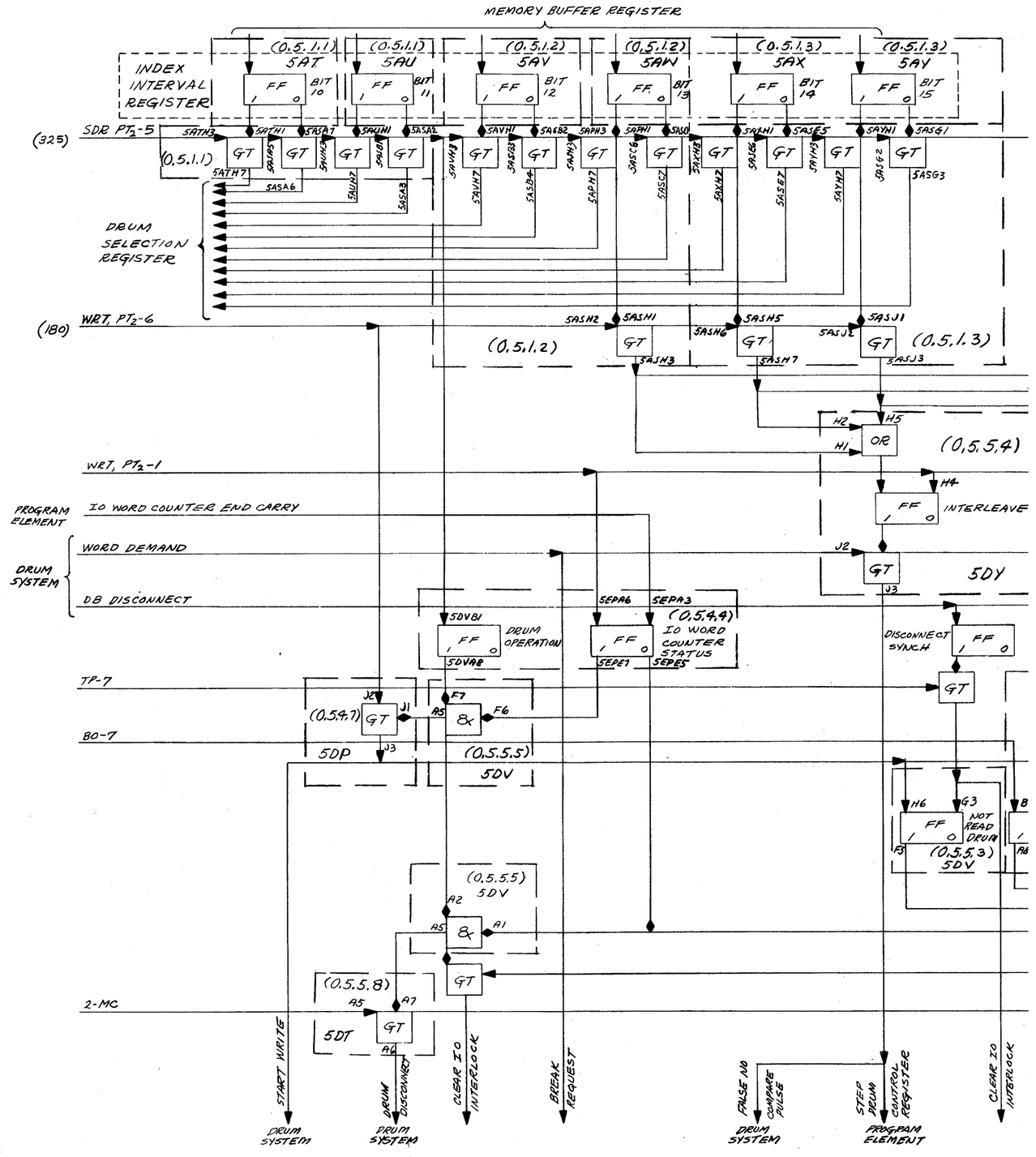


Figure 5–30. Drum Read Circuits, Logical Block Diagram


MEMORY BUFFER REGISTER
INDEX INTERVAL REGISTER
(0,5,1,1) 5AT
(0,5,1,1) 5AU
(0,5,1,2) 5AV
(0,5,1,2) 5AW
(0,5,1,3) 5AX
(0,5,1,3) 5AY
FF
BIT 10
BIT 11
BIT 12
BIT 13
BIT 14
BIT 15
(325)
SDR PT2-5
(0,5,1,1)
GT
DRUM SELECTION REGISTER
(180)
WRT, PT2-6
(0,5,1,2)
(0,5,1,3)
OR
(0,5,5,4)
WRT, PT2-1
PROGRAM ELEMENT
IO WORD COUNTER END CARRY
INTERLEAVE
DRUM SYSTEM
WORD DEMAND
DB DISCONNECT
5DY
5DVB1
DRUM OPERATION
5DVA8
(0,5,4,4)
IO WORD COUNTER STATUS
5EPE5
DISCONNECT SYNCH
TP-7
(0,5,4,7)
8x
(0,5,5,5)
5DV
5DP
B0-7
NOT READ DRUM
(0,5,5,3)
(0,5,5,8)
5DT
2-MC
START WRITE
DRUM SYSTEM
DRUM DISCONNECT
CLEAR IO INTERLOCK
BREAK REQUEST
FALSE IO COMPARE PULSE
STEP DRUM CONTROL REGISTER
CLEAR IO INTERLOCK
PROGRAM ELEMENT

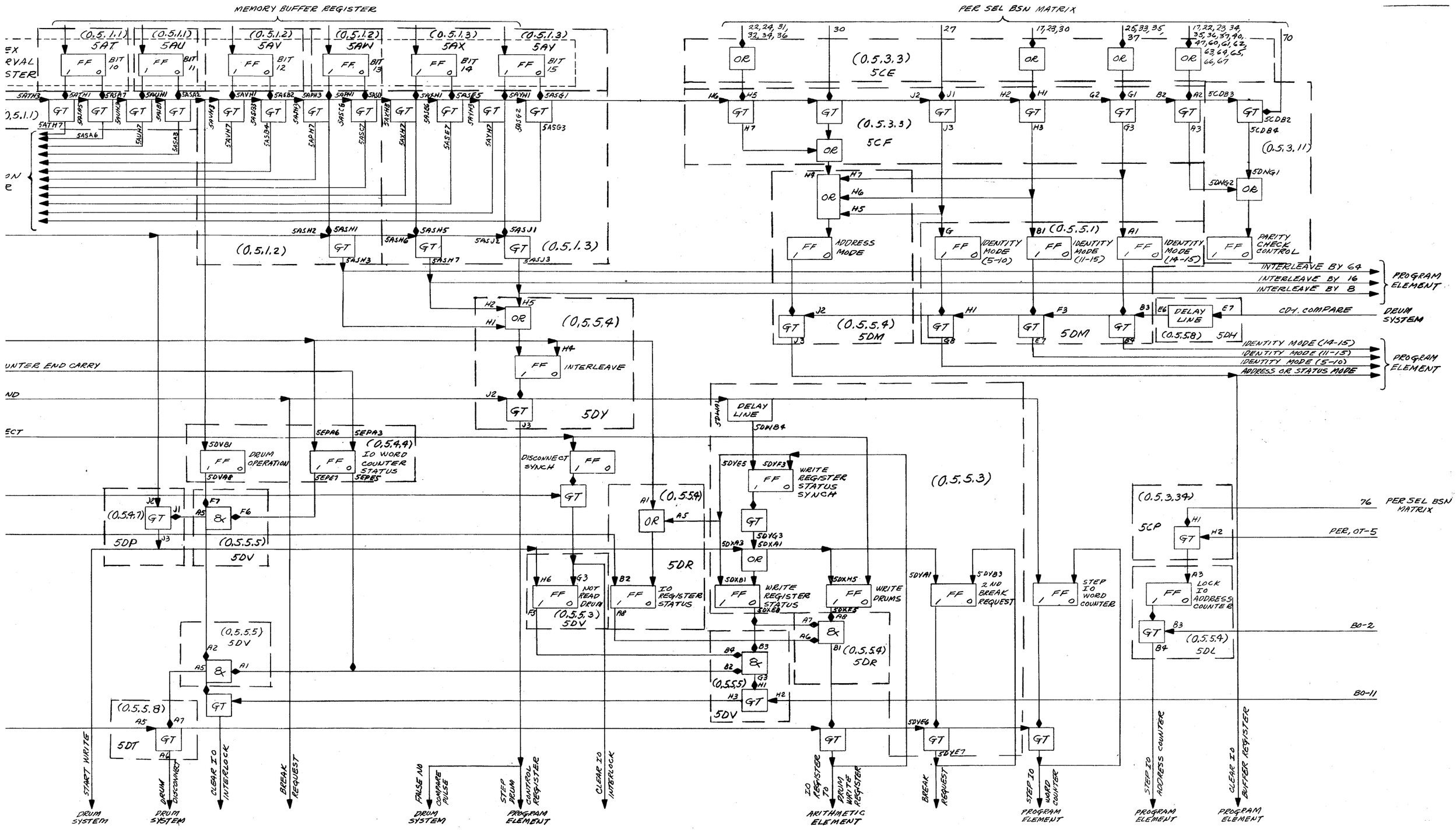


Figure 5-31. Drum Write Circuits, Logical Block Diagram

1.2 INFORMATION FLOW

Two transfer paths exist between the Central Computer System and the Drum System, one for reading and the other for writing. The Drum System will transmit information at the rate of one word every 10 microseconds during a reading operation. The IO buffer registers are required to prevent these incoming words from interfering with each other. During a writing operation, word transmission originates in the Central Computer System and the rate at which the information is sent is well within the time limitation established by the writing speed of the Drum System. Consequently, the IO buffer registers are not needed for information flow during a drum writing operation, but are used for address comparison.

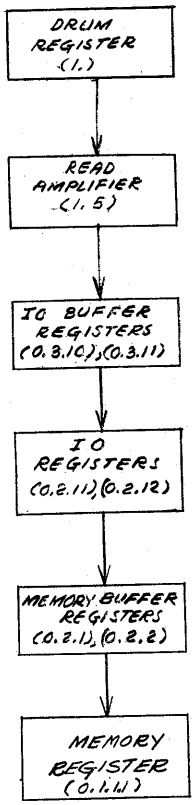


Figure 5–32. Drum Read Information Flow

The path employed for reading is shown in figure 5–32. A binary word (32 or 33 bits, depending on location) is transferred initially from the drum register in the selected drum field to the IO buffer registers. Approximately two microseconds will elapse before the word being transferred settles in the IO buffer registers. Further transfer of the word to the IO registers is effected by the next 2-megacycle oscillator pulse, but, because of synchronizing delays, as much as one microsecond may be required before the word appears in IO registers. The time required to transmit the word one step further (to the memory buffer registers) depends upon whether the break request pulse, generated as soon as the word appears in the IO registers, occurs before or after TP–9 of the existing memory cycle. If it occurs before TP–9, the next memory cycle is designated as a break-in cycle, during which the IO registers-memory buffer registers transfer will be effected. If it occurs after TP–9, the break request flip-flop may not be fully set when it is sensed at TP–11, and a complete 6-microsecond memory cycle may pass before the required break-in cycle is assigned. This discussion precludes the existence of an arithmetic pause condition because, in this case, the break request flip-flop is sensed at a 2-megacycle rate instead of a TP–11. A break-in cycle will therefore be assigned immediately after the break request flip-flop is set.

Including the time required for the IO registers-memory buffer registers transfer, a delay of approximately nine microseconds is possible before the IO registers complete their transfer to the memory buffer registers and thereby make ready to accept another word from the IO buffer registers. A total time of 12 microseconds will have elapsed since the first word was read from the drum surface. The second word, automatically transferred 10 microseconds after the first, must therefore be placed in the IO buffer registers and not in the IO registers if the two words are not to be superimposed in the latter register.

When the Central Computer System is writing on a drum field, the original information exists in a core memory register and its ultimate destination is a drum register on a drum field. The word passes through three registers: the memory buffer registers, the IO registers, and the drum write register. (See fig. 5–33.) The drum write register is physically located in the Drum System and serves much the same purpose during a break-out cycle that the IO buffer registers serve during a break-in cycle. The first word to be transferred is placed in the memory buffer registers. From there it is successively transferred to the IO registers and the drum write register. These transfers occur as the result of a break request made by the circuits of the selection and IO control element during the execution of a *Write* instruction. In the course of the first break-out cycle a second break request is made by the selection and IO control element circuits. During the resultant second break-out cycle, a second word is transferred from core memory to the IO registers, passing through the memory buffer registers as an

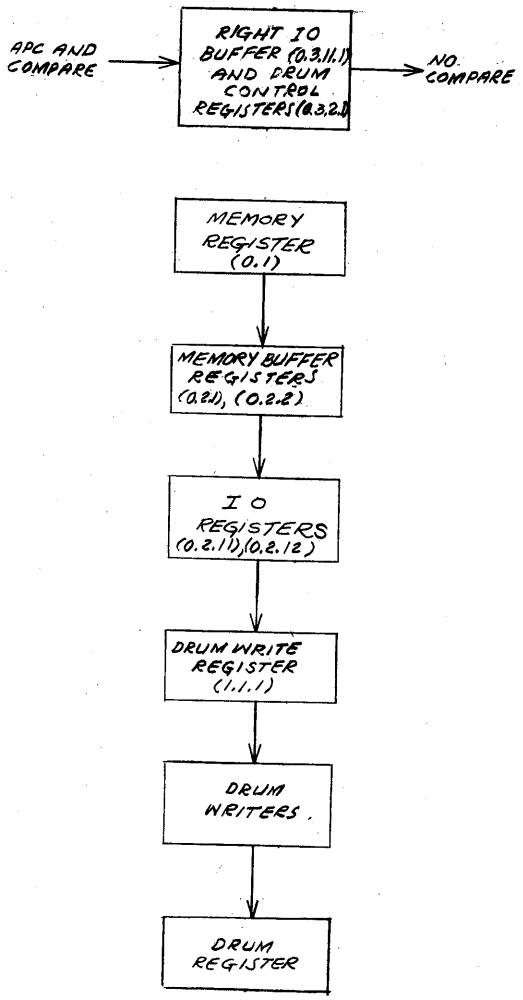


Figure 5–33. Drum Write Information Flow

intermediate transfer point. Both processes, one placing a word in the drum write register and the other placing a word in the IO registers, are completed before a word is written on the drum surface. Once the first word is written, the Drum System will generate a single break request pulse after each writing process.

1.3 FIELD SELECTION

Field selection involves a decoding process performed by the PerSelBsn matrix and the index interval register. (Refer to Chapter 1 of this Part. In brief, the function of the index interval register is to store bits L10 through L15 of any instruction word. The information thus recorded varies as the Central Computer System instruction changes. During a *Select Drum* instruction, the index interval register identifies the drum field that is to be used in the forthcoming IO process. Accordingly, at PT_2–5 of this instruction, the contents of the index interval register are transferred to the drum selection register in the Drum System, where they establish the proper conditions for reading or writing on the desired field. During a *Read* or *Write* instruction, the last three bits of the index interval register are used to specify the interleaving constant. A 1 in bit L13 indicates that the interleaving constant is to be 64; a 1 in bit L14 indicates that the constant is to be 16; and a 1 in bit L15 indicates that the constant is to be 8. Interleaving is not to be performed if none of these bits contains a 1. The interleaving information is transferred to the appropriate circuits of the program and selection and IO control elements at PT_2–6 of either the *Read* or *Write* instruction.

The PerSelBsn matrix accepts information in the form of d-c levels from the index interval register and, utilizing diode AND circuits, produces a single output for a given content of the register. Several of these outputs (listed in table 5–5) are associated with drum fields. Each output conditions a set of gate tubes which, upon being pulsed in the course of the *Select Drum* instruction, alter the status of certain flip-flops so that the characteristics of the selected drum field will be reflected by the status of these flip-flops. In the following step-by-step analysis of drum selection, attention will be directed only to those circuits pertinent in an IO transfer between the Drum System and the Central Computer System.

A drum field is designated as the IO unit to be linked with the Central Computer System by the *Select Drum* instruction. This instruction supplants the *Select* instruction in the basic IO program pattern. The index interval of the *Select Drum* instruction assigns one of the available drum fields in accordance with the code given in table 5–5. In the address mode the starting address on the drum field is identified by the address part of the same instruction. The starting address in core memory is held by the IO address counter as a result of the *Load IO Address Counter* instruction which precedes the *Select Drum* instruction in the IO program sequence.

Execution of the *Select Drum* instruction (and all other IO class instructions) is delayed if the IO interlock is on. The conditions giving rise to an IO pause of this nature are given in 1.5.2 of Chapter 1. If the IO interlock is on, a previously scheduled IO process is still in progress. If the *Select Drum* instruction were not delayed, all selection controls would be reset before completion of the existing IO process. Such design would result in incomplete or incorrect execution of any IO transfer which was followed too closely in time by an IO class instruction. The time delay brought about by the IO pause has been made automatic to

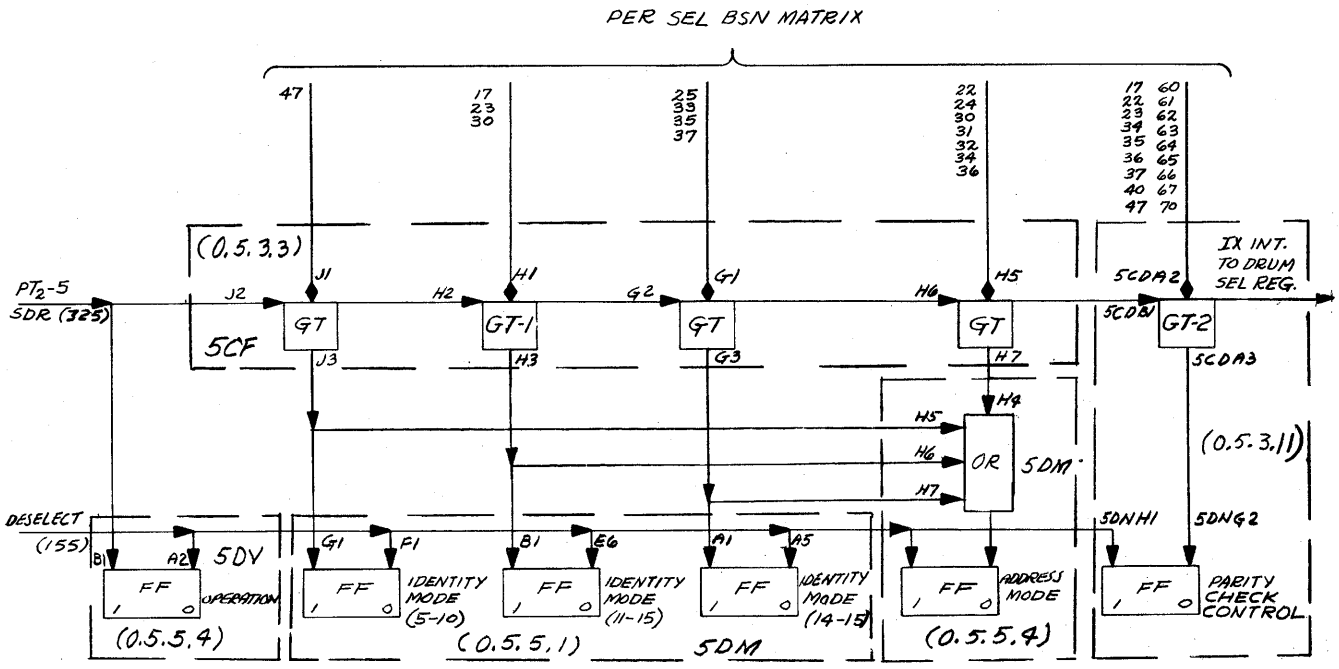


Figure 5–34. Drum Selection Controls, Logical Block Diagram

relieve the programmer of an added restriction.

As soon as the IO interlock is cleared, the *Select Drum* instruction proceeds. The *deselect pulse* command (155) is generated at OT–5 by the instruction control element and is sent to both the selection and IO control element and the Drum System. In the Drum System, it prepares the drum circuits to select a new drum field by resetting all circuits affected by the previous Drum System-Central Computer System information transfer. (Refer to the CD element Drum System, for full details concerning this function of the *deselect* command.) In the selection and IO control element, the chief function of command 155 is to deselect the IO unit that was last in use and to place the drum selection controls in their neutral positions. The drum selection controls located in the selection and IO control element are shown in figure 5–34. Six flip-flops are used; one (drum operation) indicates that a Drum System-Central Computer System transfer has been requested by the program, and the other five reflect the inherent characteristics of the selected drum field. When the flip-flops are in their neutral condition, it is assumed that the drum field to be selected is processed by the address mode and that it has a parity bit associated with each of its registers. These assumptions are altered in the course of the *Select Drum* instruction to conform with the characteristics of the drum field actually selected.

At PT_2–2, the drum control register in the program element is cleared by the *clear drum control register* command (146) in preparation for receipt of the address part of the *Select Drum* instruction. This transfer occurs at PT_2–3. The *select pulse for drums* command (325) is issued next by the instruction control element. This command alters, if necessary, the original assumptions concerning mode of operation and parity bit existence made by the drum selection controls. Command 325 also causes transfer of the index interval register contents to the drum selection register.

The index interval register will condition a single output of the PerSelBsn matrix; i.e., the one associated with the selected drum field. To illustrate the action of the drum selection controls, assume that output No. 23 of the PerSelBsn matrix is raised in accordance with the contents of the index interval register. Table 4–1 gives the characteristics of this drum field, which is read by identity bits 11 through 15, and does not have parity bits associated with its registers. Accordingly, gate tubes 1 and 2 are conditioned by the PerSelBsn matrix output. (See fig. 5–34.) When the *select* command is given, all gate tubes shown in the figure are interrogated. With gate tubes 1 and 2 conditioned, the interrogation will result in the identity mode (11 through 15) flip-flop being set and the address mode and parity check control flip-flops being cleared. The circuits conditioned by these flip-flops act in accordance with the properties of drum field No. 23.

SECTION 2
READ OPERATION

The *Sense* and *Operate* instructions are not used for drum IO processes. If it is desired that a transfer of words from the Drum System to the Central Computer System is desired, the *Select Drum* instruction is supplanted next in the instruction register by the *Read* instruction.

2.1 *READ* INSTRUCTION

The *Read* instruction is decoded by the instruction control element in the usual way. At PT_1–10, if the IO interlock is on, the *pause* command (134) sets the pause flip-flop in the time pulse distributor control, thereby instituting an IO pause on the Central Computer System until the IO interlock is cleared. The memory address register is cleared at PT_2–0 and is thus prepared to accept the core memory address of the first transferred word from the IO address counter. At PT_2–1, commands 149 *(clear IO word counter)* and 294 *(set IO interlock on)* are generated. Command 149 clears the IO word counter and command 294 is sent to the selection and IO control element. The effect of command 294 on that element is illustrated in figure 5–35. As shown in the figure, it affects seven flip-flops in the following way: it clears the IO buffer status, the IO register status, the interleave, the write, and the read flip-flops, and it sets the IO interlock and the IO word counter status flip-flops. Command 294 is also used to clear the IO buffer register and the interleave flip-flops in the program element. When interpreted, the flip-flops represent the status of the Central Computer System at this stage of the operations; the IO interlock is on, the IO buffer and IO registers are empty, interleaving has not yet been specified, neither a read nor write operation has yet been established, and the IO word counter contents specify the number of words to be transferred. Some of these flip-flops anticipate the actual condition they represent; e.g., the IO word counter at this point is still empty, but the necessary transfer from the ad-

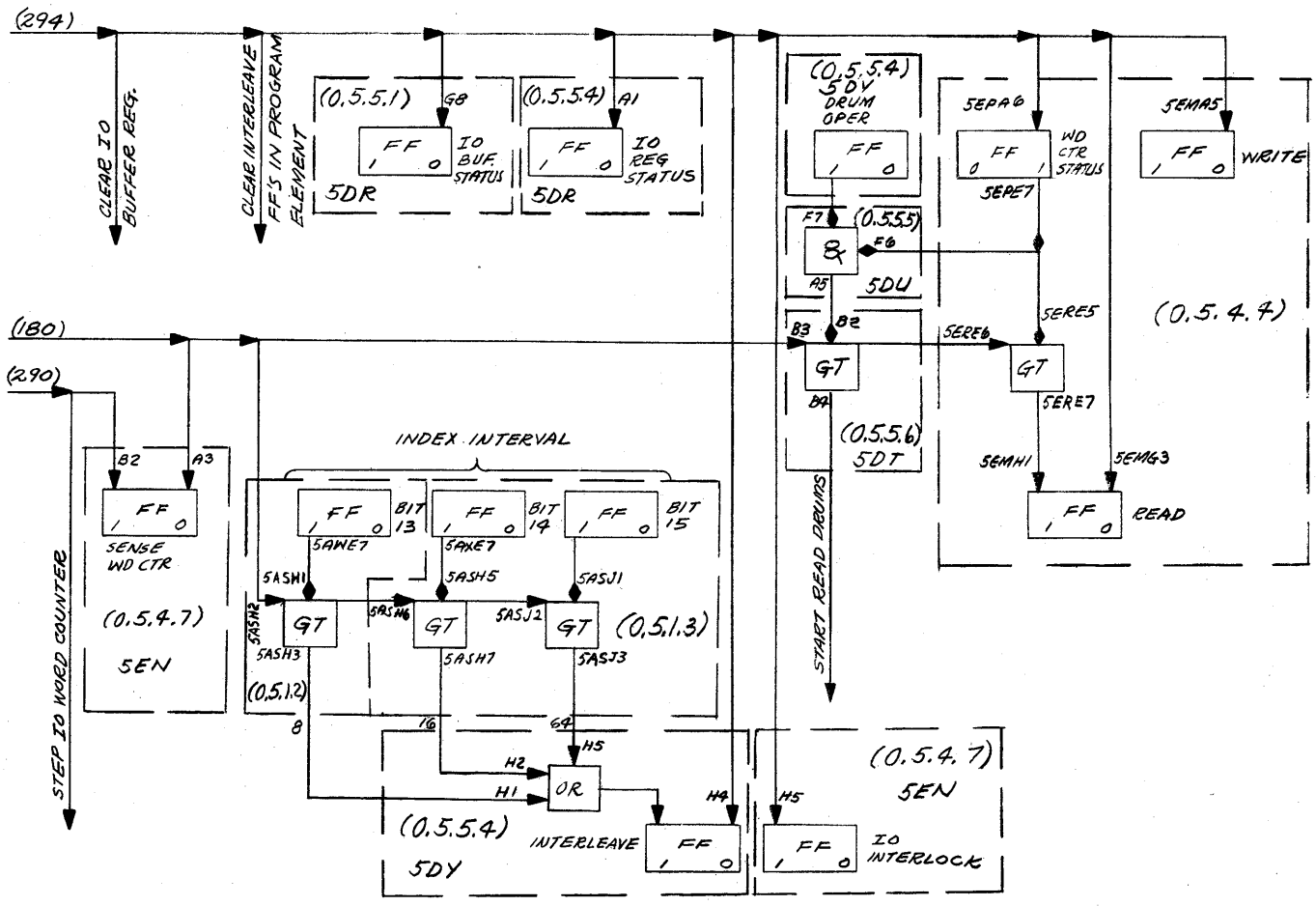


Figure 5–35. Read Instruction—Drum System, Logical Block Diagram

dress register to the IO word counter is made in time to prevent such anticipation from disrupting the operating sequence. One-half microsecond later, at PT_2–2, the IO register is cleared by commands 144 *(clear left IO register)* and 147 *(clear right IO register)*, and the address register contents, specifying the number of words to be transferred, are sent to the IO word counter by the *address register complement to IO word counter* command (152). Command 290 *(sense IO interlock 0)*, generated at PT_2–3, sets the sense IO word counter flip-flop and steps the IO word counter at the same time.

The reason for stepping the IO word counter at this time has been given in 3.6.1 Chapter 3. The stepping pulse for the IO word counter will return to the selection and IO control element as an end carry pulse only if the counter contained negative zero before it was stepped. Since the IO word counter specifies how many words are to be transferred, an end carry pulse would indicate that a *Read 0* instruction had been given. This is not a legitimate instruction when using a drum field as an IO unit. The IO interlock must be immediately cleared and the drum controls disabled. The method of accomplishing these desired results without at the same time inhibiting the execution of a legitimate *Read 0* instruction during card machine and tape IO operations is shown in figure 5–36. As previously mentioned, command 290 simultaneously sets the IO word counter flip-flop and steps the IO word counter itself. An IO word counter end carry, indicating that a *Read 0* instruction is being executed, immediately clears the IO word counter status flip-flop to indicate the true status of that counter, and clears the write drums flip-flop to disable the drum control circuits. It is also propagated by the gate tube conditioned by the 1 side of the sense IO word counter flip-flop to the gate tube associated with the 1 side of the read-write 0 flip-flop. The read-write flip-flop is in the 1 status, having been set by the deselect pulse occurring at OT–5 of the *Select Drum* instruction. If a card machine or a tape unit has been selected instead of a drum field, the read-write 0 flip-flop would have been cleared again at PT_2–5 of the *Select Drum* instruction. Since the present discussion concerns drum IO operation, the end carry pulse is passed and immediately clears the IO interlock.

The final command of the *Read* instruction, command 180 *(PT–6 on read)*, is issued at PT_2–6 and is sent to the selection and IO control element. As shown in figure 5–35, command 180 examines the last three flip-flops of the index interval regis-

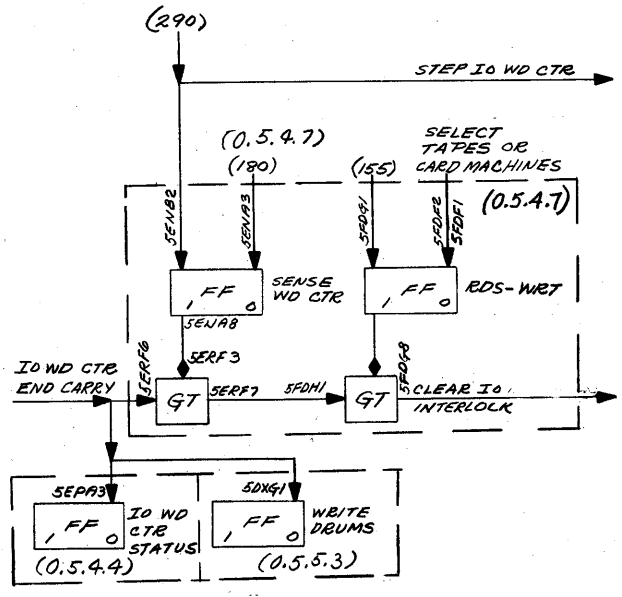


Figure 5–36. Read 0 Instruction Determination—Drum System, Logical Block Diagram

ter for an interleaving indication. If a 1 appears in any one of these three flip-flops, its associated gate tube propagates a pulse which sets the two interleave flip-flops, one in the selection and IO control element and the other in the program element. (Refer to Part 4, for a description of the function of the interleave flip-flop in that element.) Command 180 clears the sense IO word counter flip-flop and, subject to the condition that the IO word counter status flip-flop is in its 1 state, it also sets the read flip-flop. Consequently, the read flip-flop will not be set unless the IO word counter contents are other than zero. The start read pulse for the Drum System is also developed by command 180 after first inspecting the status of the drum operation flip-flop.

As a result of the combined action of the *Select Drum* and the *Read* instructions, and with the assumption that the address mode is applicable and that no IO word counter end carry pulse has appeared, the flip-flops which are of interest for the drum reading operation are in the states shown in table 5–6.

TABLE 5–6. FLIP-FLOP STATUS AFTER EXECUTION OF *READ* INSTRUCTION

SET	CLEAR
IO interlock	IO buffer status
IO word counter status	IO register status
Drum operation	Interleave
Address mode	IO buffer load
Read	IO buffer load sync

2.2 ADDRESS SEARCH

When utilizing any drum field as an IO unit, and if this field is not being processed by the status mode, the problem of address search will always arise. The drum registers, located on a constantly rotating drum surface, are never in a static location during IO operations. However, the read-write heads are stationary, which necessitates reading or writing on the drum registers as they pass under the read-write heads. Since the rotation of a drum is not synchronized with the operation of the Central Computer System, the request for a word transfer associated with a specified field register address cannot be given at any arbitrary time. Hence, it is necessary to ascertain that the specified field register is currently under the read-write heads at the time that the word transfer is requested. This determination of address location on the selected drum field is performed by an address search process prior to the transfer of any words. At the conclusion of the address search, it will have been ascertained that the desired field register is under the read-write heads and the first word transfer will be effected.

If, as in the address mode, a pattern of consecutive drum registers is desired, the address search is completed as soon as the starting field register is positioned under the read-write heads. From that time until the end of the IO transfer, the pattern is followed automatically. If, as in the identity mode, each word must be examined for acceptability, the address search is reduced to a process whereby the word in each full field register, as it appears under the read-write heads, is accepted or rejected on the basis of the identity bits it contains. If, as in the status mode, no drum register need be specified as the starting address and no limitation is placed on each individual word, the address search is not required and registers are indiscriminately processed in accordance with their status bits.

The interleaving process, which is a variation of the address mode, also exhibits a basic pattern. This pattern, subject to the option of the programmer, is one in which every 8th, 16th, or 64th register is read. When this process is being used, the circuits of the selection and IO control element find the starting address in exactly the same manner as if the address mode were in use. However, as each consecutive register now appears under the read-write heads, a false no compare pulse is generated except when the positioned drum register is a desired one. When a desired drum register is positioned a data transfer takes place and a CD–1 load pulse is sent in lieu of a CD–1 compare pulse. In this way, the execution of this type of IO transfer is similar to that of an address mode transfer, with this exception: the address search is not discontinued upon location of the starting address. In effect, the starting address keeps changing by a constant amount as soon as the previous address is found. This design forces the address search to continue until the termination of the IO transfer. A summary of address search considerations during drum reading operations is given in table 5–7.

TABLE 5–7. ADDRESS SEARCH CONSIDERATIONS

PROCESS	MODE	ADDRESS SEARCH
Read	Address	Yes, until first address found
	*Interleaving	Yes, intermittent until end of IO transfer
	Identity	No, data are compared, not address
	Status	No

** Interleaving is a variation of the address mode.*

The address search is initiated upon receipt of a CD–1 compare pulse from the Drum System. This pulse will appear after the start read pulse, generated by command 180 of the *Read* instruction, is sent to the Drum System. The selection and IO control element circuits which are activated by this pulse are shown in figure 5–37. These circuits operate in conjunction with the comparison circuits in the program element. The signals which control the status of the drum operation flip-flop and the mode flip-flops have already been described in Section 1 of Chapter 4. At the start of the address search, the drum operation flip-flop and one mode flip-flop will be in the 1 states. All others will be cleared. If the drum field selected is processed by the status mode, only the drum operation flip-flop will be set. However, since no address search is necessary for this mode, the circuits shown would not be functionally useful.

The first case to be considered is the drum read operation using the address mode. Accordingly, the drum operation and address mode flip-flops are assumed to be set and all other flip-flops are assumed to be cleared. The generation of the start read pulse sent to the Drum System is described in 2.1 of Section 2, Chapter 4. Upon receipt of this pulse, the Drum System introduces a 120-microsecond delay before returning a CD–1 compare pulse to the Central Computer System.

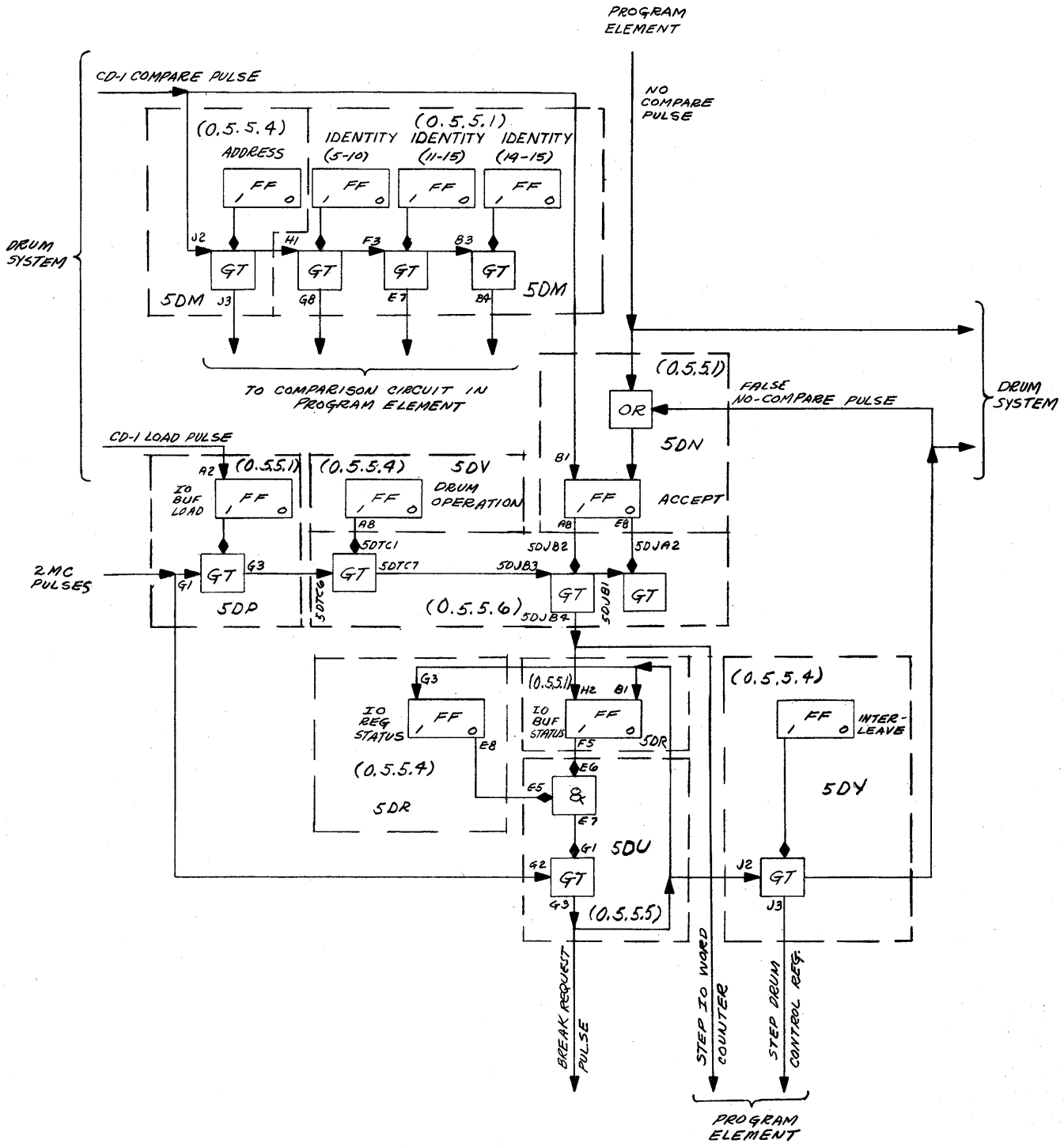


Figure 5–37. Address Search Circuits, Logical Block Diagram

Simultaneously, the contents of the angular position counter, reflecting the address of the drum register currently under the read-write heads, are transferred to the right IO buffer register for comparison purposes. (Refer to Part 4, for a complete description of the drum comparison circuits.) A 120-microsecond delay is necessary to inhibit actual reading operations until the drum reading circuits are raised to their proper voltage levels.

As shown in figure 5–37, the CD–1 compare pulse immediately sets the accept flip-flop and, after a one-microsecond delay, examines the four mode flip-flops. Without this one-microsecond delay, an interrogating signal would be sent to the comparison circuits before the current drum ad-

dress has fully settled in the right IO buffer register. After the delay, the CD–1 compare pulse is propagated only by the gate tube conditioned by the 1 side of the address mode flip-flop. This output pulse is transmitted to the comparison circuits in the program element and is also used to clear the right IO buffer register after the comparison is made. The register clearing function is delayed for one microsecond to allow time for the comparison. If the current drum register address, as held by the right IO buffer register, does not agree with the required starting address as held by the drum control register, a no compare pulse is generated in the program element and is transmitted to the selection and IO control element. The pulse clears the accept flip-flop and is transmitted to the Drum System, where it prevents transmission of the contents of the field register whose address was compared. The drum continues to rotate and the contents of the angular position counter are increased by 1. As soon as the next consecutive drum register is positioned under the read-write heads, the process repeats and continues to repeat until the comparison circuits, by not generating a no compare pulse, indicate that the starting address is held in the right IO buffer register. The accept flip-flop is therefore not cleared. The Drum System then transmits CD–1 load pulses because it now is permitted to load the contents of the field register into the IO buffer registers. From this point until the termination of the IO process, no more CD–1 compare pulses will be generated. A CD–1 load pulse will be transmitted each time the contents of a field register are transferred to the IO buffer registers. The accept flip-flop will remain set, thus successfully concluding the address search.

However, if the interleave method of transfer is in effect, the address search is not concluded. Only the proper address for the first word has been found. To determine the succeeding addresses, use is made of the break request generation circuits described in 1.1.2.1 of C–2, S–1. As explained in that paragraph, the CD–1 load pulse transmitted by the Drum System after the first address has been found sets the IO buffer load flip-flop, thereby conditioning a gate tube inspected by 2-megacycle oscillator pulses. The first pulse to be propagated by this gate tube clears the IO buffer load flip-flop and, passing through a gate tube conditioned by the 1 side of the drum operation flip-flop, examines the accept flip-flop. The accept flip-flop was left in the 1 status because of the successful address search just completed. Therefore, the gate tube associated with its 1 side develops a pulse which sets the IO buffer status flip-flop. The same pulse is sent to the program element, where it steps the IO word counter, indicating that one information word is in the process of being transferred from the Drum System to the Central Computer System.

Since the IO register status flip-flop is in the 0 status, setting the IO buffer status flip-flop fully conditions the AND circuit shown in figure 5–37. Note that if the 1 side is interpreted as meaning full and the 0 side as empty, these flip-flops reflect the actual status of the IO registers and IO buffer registers. The AND circuit is conditioned, therefore, whenever the IO registers are empty and the IO buffer registers are full. A gate tube conditioned by the AND circuit is constantly examined by 2-megacycle pulses from the oscillator and the resultant reverses the indication of the two flip-flops, transfers the word in the IO buffer registers to the IO registers, requests a break from the Central Computer System, and inspects the interleave process. The break-in cycle resulting from the break request will transfer the first word, now in the IO registers, to its final destination in core memory. The two status flip-flops now indicate that the IO registers are full and the IO buffer registers are empty. Thus far the procedure is exactly that of an address mode transfer. The distinction between address mode and interleave method arises because, when reading by the address mode, the interleave flip-flop will be in the 0 state, and when reading by the interleave method, the interleave flip-flop will be in the 1 state.

The inspecting pulse for the interleave flip-flop interrogates a gate tube conditioned by the 1 side of the flip-flop. Since it is assumed that interleaving has been specified, an output pulse is developed which will step the drum control register by 8, 16, or 64, depending on the request made by the *Read* instruction. Selection of the proper constant is made in the program element. To prevent the operation from continuing as an ordinary address mode transfer, the same output pulse clears the accept flip-flop and is sent to the Drum System as a false no compare pulse. Upon receipt of this signal, identical to a no compare pulse as far as the Drum System is concerned, the Drum System will designate its next CD–1 pulse as another CD–1 compare pulse. In effect, a new address search will be initiated. The CD–1 load pulse required to set the IO buffer load flip-flop will not be transmitted again until the new starting address, as given by the drum control register, agrees with the angular position counter, as given

by the contents of the right IO buffer register. As soon as they do agree, the address mode process just described is repeated to the point where the interleave flip-flop is again inspected. By this time, the second word of the transfer will be in the IO registers. Again, the next 7, 15, or 63 drum registers will be treated as incorrect addresses. In this way, only every 8th, 16th, or 64th register is read from the drum field into core memory.

If the identity mode is required, several significant differences in the process just described must be considered. First of all, the initial transfer to the right IO buffer register by the Drum System is not an address as given by the angular position counter but an actual word taken from the drum register beneath the read-write heads. To properly reflect this fact, CD–1 load and CD–1 compare pulses are sent simultaneously. Secondly, since the word in the right IO buffer register is useful if it does compare, this register cannot be cleared immediately after the comparison process, as it was during an address mode operation.

Upon completion of the delay after the start read pulse, the Drum System sets the accept flip-flop and asks for a comparison by examining the four mode flip-flops. Simultaneously, a CD–1 load pulse sets the IO buffer load flip-flop. This results in an examination of the accept flip-flop immediately after the first word is transferred to the right IO buffer register. It will be remembered that this examination did not take place during address mode operation until the starting address had been found. At the same time, whichever identity mode flip-flop has been set enables a pulse to activate the appropriate comparison circuits in the program element. Because of synchronizing delays in setting the IO buffer load flip-flop with the CD–1 load pulse, there is time to compare the specified identity bits before the first 2-megacycle pulse examines the accept flip-flop. If the words (one in the drum control register and the other in the right IO buffer register) compare by identity, the accept flip-flop is not cleared and, upon being interrogated, causes the IO word counter to be stepped, the IO buffer registers content to be transferred to the IO registers, the IO buffer status flip-flop to be set, and a break to be requested. The interleave flip-flop will always be cleared during the identity mode, inhibiting any false no compare pulse. This process is very similar to the address mode procedure. However, if a no compare pulse is generated, the accept flip-flop will be in its 1 state when interrogated. Logically, this means that the word currently in the right IO buffer register is not acceptable and must be erased. Consequently, the gate tube conditioned by the 0 side of the accept flip-flop generates a signal which clears the IO buffer registers. The requisite operation for transferring the word is not performed, and the Central Computer System awaits the next set of CD–1 load and CD–1 compare pulses, indicating that the contents of the next drum register are ready to be examined. In this way, consecutive drum registers are examined until the desired number of words, as specified by the IO word counter, have been processed.

The final method of reading to be considered is the status mode. If the status mode is in effect, all of the mode flip-flops will be in their 0 states and no comparison procedure will be carried out. A CD–1 load and a CD–1 compare pulse arrive simultaneously each time a full field register is encountered by the read-write heads of the Drum System. These pulses behave in the same manner as described previously. Consequently, the accept and the IO buffer load flip-flops will always be set when a word exists in the IO buffer registers. This word is always transferred to core memory.

2.3 BREAK REQUEST GENERATION

As a result of the various address search procedures given in the preceding paragraphs, a word will always be placed in the IO registers. To complete the transfer to core memory, a break request must be made and a break-in cycle assigned. The break request generation circuits have been fully described in Chapter 2, but a short synopsis will be given here. Assume once again that the address mode is in effect and that the address search has been successfully terminated. The first word of the impending IO transfer has been placed in the IO buffer registers by the drum reading circuits. At the same time, a CD–1 load pulse has been sent to the selection and IO control element.

The CD–1 load pulse is delayed one microsecond before it sets the IO buffer load sync flip-flop, thereby conditioning a gate tube continuously examined by 2-megacycle oscillator pulses. The delay allows the IO buffer registers to settle in their new state before the transfer operation begins. The first output pulse from the conditioned gate tube sets the IO buffer load flip-flop which, in turn, conditions another gate tube examined by the 2-megacycle oscillator pulses. The output of this second gate tube clears the IO buffer load flip-flop and its synchronizing flip-flop and, after being propagated by a gate tube conditioned by the 1 side of the drum operation flip-flop, examines the accept flip-flop. This

flip-flop is set as the result of the successful address search previously conducted and consequently conditions the gate tube associated with its 1 side. The examining pulse from the IO buffer load flip-flop is propagated by this gate tube, and is used to step the IO word counter and to set the IO buffer status flip-flop to its 1 (full) side.

Note that the pulse which steps the IO word counter is completely synchronous to the action of the Central Computer System; in particular, it may occur while a *Clear and Subtract IO Word Counter* instruction is being executed. To avoid stepping the IO word counter at a time when the contents of this counter are being transferred to the right accumulator register, recourse is made to the CSW control circuit. However, to avoid disrupting the continuity of the present discussion, the functional description of this circuit is deferred to 3.2 of C–4, 5–3.

The 1 side of the IO buffer status flip-flop and the 0 side of the IO register status flip-flop, both energized, activate an AND circuit whose output signal, indicating full IO buffer registers and empty IO registers, conditions a gate tube interrogated at a 2-megacycle rate. The output of this gate tube performs four functions: it reverses the indications of the IO buffer status and the IO register status flip-flops, transfers the contents of the IO buffer registers to the IO registers, examines the interleave circuits, and requests a break. The reversal of the two status flip-flop indications confirms the current status of the IO register and IO buffer register. The result of examining the interleave circuits is described in 2.2 of Section 2, Chapter 4 and the action of the break request circuits following the setting of the break request flip-flop is described in Chapter 2.

2.4 BREAK-IN MEMORY CYCLE

As soon as the break request is honored, a memory cycle is assigned to complete the transfer of the word now in the IO register to its core memory destination. At BI–1, the contents of the IO address counter are sent to the memory address register, and the word destination in core memory is thereby established for the memory circuits. At BI–2, the IO word counter is stepped, the IO register status flip-flop is cleared, and the word in the IO registers is transferred to the memory buffer registers, where it will be completely processed before the conclusion of the assigned break-in cycle. The BI–3 pulse is sent to the arithmetic element, where it performs a parity count, and to the instruction control element, where it prevents the memory buffer register from writing into core memory during the break-in cycle. The BI–4 pulse is also sent to the instruction control element. If test memory is in operation, this pulse transfers the contents of the memory buffer registers to the test registers. BI–7 performs a parity check and BI–11 checks the status of the transfer by simultaneously examining the status of the IO word counter and the break request flip-flop. BI–5, BI–8, and BI–10 are not issued, and BI–6 and BI–9 do not affect a drum reading operation. At the end of the break-in cycle, the first word will have been placed in its core memory location.

2.5 IO PROCESS TERMINATION

The Drum System will asynchronously send another CD–1 load pulse 10 microseconds after the first load pulse and the entire process will be repeated. With the next to last word in the IO registers, the IO word counter, after being stepped by the BI–2 pulse, will contain negative zero. When the last word is in the IO registers, the counter, after being stepped at BI–2 of the last break-in cycle, will contain positive zero and will develop an end carry pulse. The end carry pulse will clear the IO word counter status flip-flop and no new break requests will be honored. As soon as the IO word counter status flip-flop is placed in the 0 side, it fully conditions a two-input AND circuit whose other condition is that the drum operation flip-flop be set. (See fig. 5–38.) The

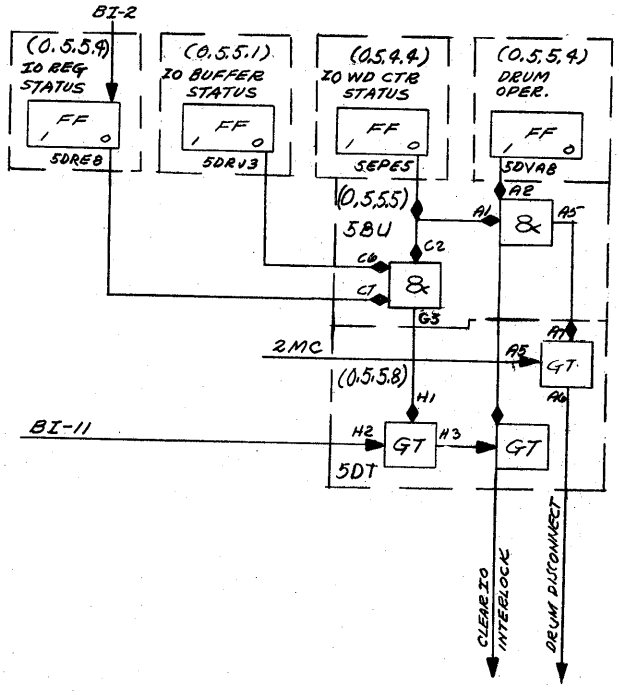


Figure 5–38. IO Termination Circuit-Drum Read, Logical Block Diagram

output of the AND circuit conditions a gate tube examined at a 2-megacycle rate. The output signal of this gate tube is sent to the Drum System as a drum disconnect pulse and disables the reading circuits of that system. Following the drum disconnect pulse, the last word is processed. At BI–11, the IO interlock flip-flop is cleared and the Central Computer System continues with its normal program, starting with the TP–0 pulse of the succeeding memory cycle.

SECTION 3
WRITE OPERATION

A drum writing operation is scheduled by following the *Select Drum* instruction with a *Write* instruction. At the end of the *Write* instruction and as a direct result of its action, two break-out cycles are executed. As a result of the break-out cycles, the first word to be written will be in the drum write register and the second in the IO registers. An address search will be conducted, and as soon as the proper field register is positioned under the read-write heads of the selected drum field, the word in the drum write register will be transferred to the drum surface. After the writing process is completed, a word demand pulse will be transmitted to the selection and IO control element by the Drum System. This pulse will perform five functions:

a. Request a break
b. Set the step IO word counter flip-flop
c. Set the write register status sync flip-flop
d. Inspect the interleave circuits
e. Clear the drum write register

As a result, the word in the IO registers will be transferred to the drum write register and another word will be taken from core memory and placed in the vacated IO registers. The Drum System will write the second word in accordance with the mode in effect. After writing, it will transmit another word demand pulse to the selection and IO control element. This procedure is repeated until the required number of words have been placed on the drum surface. After the last word is written, the IO interlock will be cleared and the Drum System will be disconnected.

3.1 *WRITE* INSTRUCTION

As applicable to a drum writing operation, the *Write* instruction is very similar to the *Read* instruction. (Refer to 2.1 of C–4, S–2.) (See fig. 5–39.) At PT_2–1, command 294 sets the IO interlock and the IO word counter status flip-flops and clears the IO buffer status, the IO register status, the interleave, the read, and the write flip-flops. It also clears the IO buffer registers and the interleave flip-flops in the program element. Command 290 is issued at PT_2–3 and sets the sense IO word counter flip-flop. A complete discussion of the function of this command is given in 3.6.1, Chapter 3. Command 182 is generated at PT_2–6, requests a break, and examines the last three flip-flops of the index interval to determine if interleaving has been specified. If it has, the interrogation by command 182 results in the setting of the interleave flip-flop. At the same time, the appropriate signal is sent to the program element. Another function of command 182 is to set the second break request, the write drums, the not read drums flip-flop, and at the same time to transmit a start write pulse to the Drum System. This function is conditional, in that the drum operation and IO word counter status flip-flops must both be set before it can be realized. Finally, command 182, upon examining a gate tube conditioned by the 1 side of the IO word counter status flip-flop, develops a pulse which sets the write flip-flop. Note that a break request has been made and the second break request flip-flop has been set in the course of this instruction.

3.2 ADDRESS SEARCH

An address search during a drum writing operation need only be conducted if the mode required is either the address mode or the interleaving mode. The status mode does not require an address search because the Central Computer System, when writing on a drum field processed by this mode, will write in each empty field register encountered. The identity mode is not used when transferring information from the Central Computer System to a drum field. The address search description given here is for an address mode operation.

As stated in 3.1 of C–4, S–3, two break-out cycles will follow the termination of the *Write* instruction. At the conclusion of the second cycle, one word will be in the drum write register and a second in the IO registers. With the drum write register filled, a CD–1 compare pulse is sent to the selection and IO control element. The pulse examines the mode flip-flops and sets the accept flip-flop. Since the identity mode is not applicable during drum writing operations and the status mode is only indirectly indicated by the mode flip-flops (all mode flip-flops cleared), attention need be directed only to the address mode flip-flop. If it is in the 1 state, either an address mode or interleave transfer is in progress. If it is in the 0 state, a status mode transfer is in progress. The status mode requires no address search. An address search, therefore, will only be conducted if the address mode flip-flop is in the 1 state.

The gate tube conditioned by the 1 side of the address mode flip-flop converts the CD–1 com-

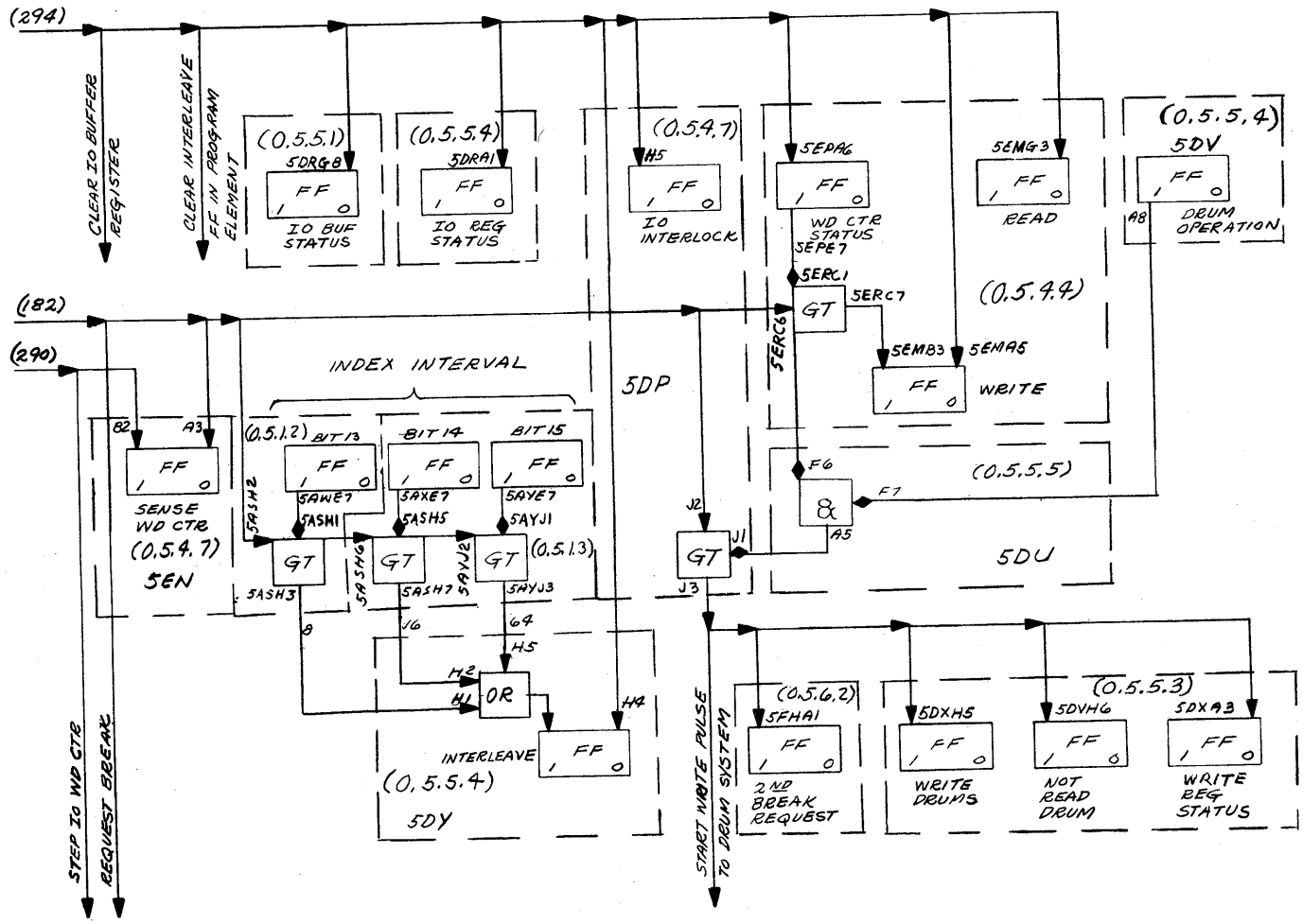


Figure 5–39. Write Instruction—Drum System, Logical Block Diagram

pare pulse into an examining pulse sent to the comparison circuits in the program element. If the contents of the drum control register and the right IO buffer register do not agree, a no compare pulse will be returned to the selection and IO control element. In that event, the pulse clears the accept flip-flop, and is transmitted to the Drum System, where it inhibits writing and the word held in the drum write register. It also assigns the next CD–1 pulse as another compare pulse. The process is repeated until the Drum System, because of the absence of a no compare pulse, is informed that the proper address has been found. The word in the drum write register is accordingly written on the drum surface and a word demand pulse is transmitted to the selection and IO control element.

With the address search completed, the word demand pulse must establish the proper conditions for continuing the IO process. In the Drum System, the pulse clears the drum write register, in preparation for receipt of the next word to be written. The action of the word demand pulse in the selection and IO control element is illustrated graphically by figure 5–40. As shown in the figure, the pulse immediately asks for a break. The resultant break-out cycle, when assigned, will be the third break-out cycle devoted to the drum writing process.

By setting the write register status sync flip-flop, the pulse causes the write register status flip-flop to be eventually cleared, thereby conditioning one input of the three-way AND circuit whose other two conditions are that the IO register status and the write drum flip-flops both be in the 1 state. The IO register status flip-flop is placed in the 1 state during the second break-out cycle after the *Write* instruction, and the write drums flip-flop is always in the 1 state during a drum writing process. Therefore, the AND circuit is fully conditioned when the write register status flip-flop is placed in the 0 state.

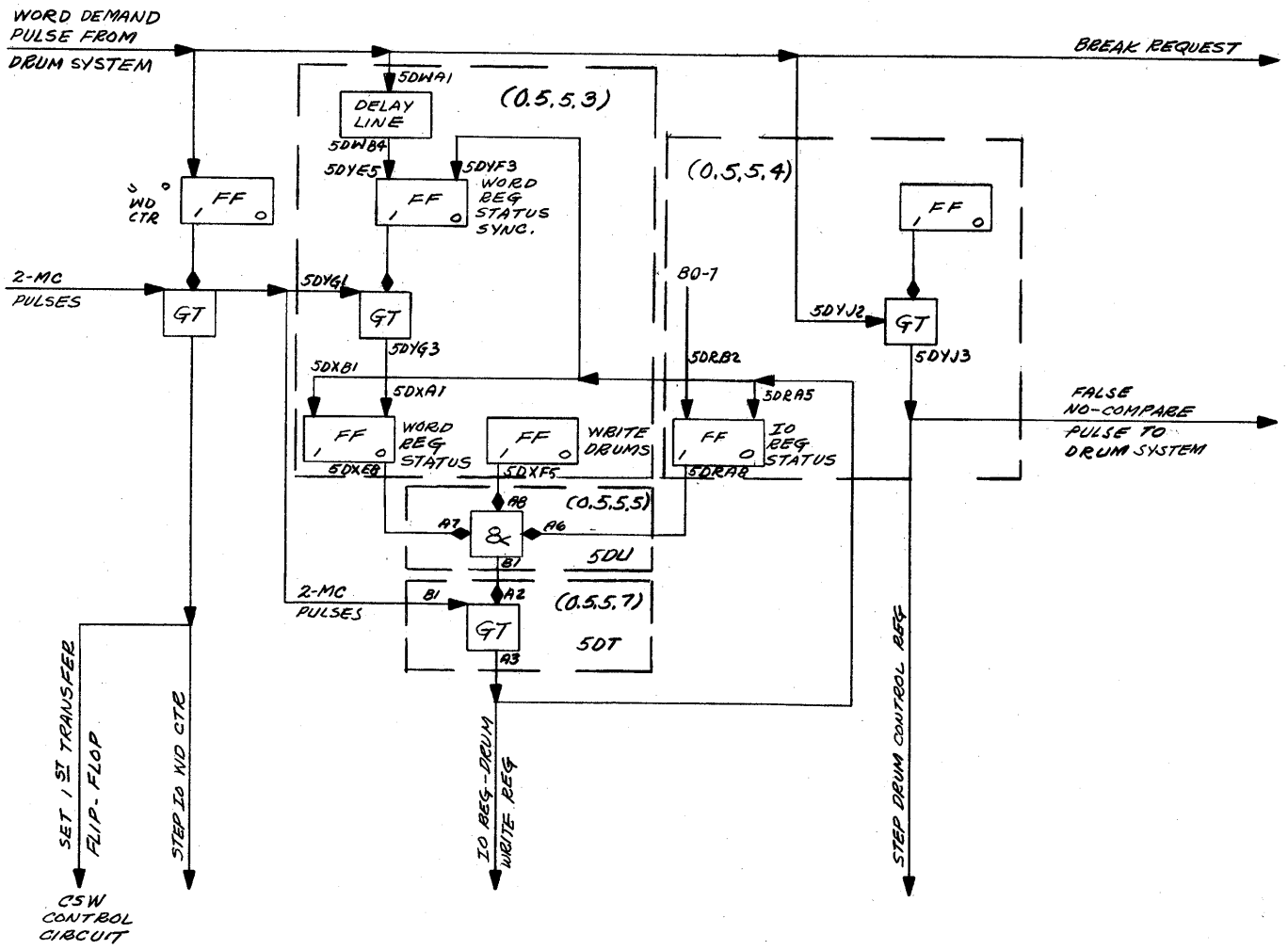


Figure 5–40. Word Demand—Drum Write, Logical Block Diagram

The output of the AND circuit conditions a gate tube examined by 2-megacycle oscillator pulses. The first pulse to be propagated initiates the transfer of the word in the IO registers to the drum write register and, following the transfer, also gives a correct representation of the status of these two registers by reversing the indications of the write register status and IO register status flip-flops.

By setting the step IO word counter flip-flop, the word demand pulse allows the IO word counter to be stepped by the next 2-megacycle pulse. It must be noted that, because the word demand pulse is asynchronous with Central Computer System action, the stepping pulse for the IO word counter is also asynchronous. If it should happen to coincide in time with the program time cycle of a *Clear and Subtract Word Counter* instruction, there is a possibility that the instruction will attempt to transfer the contents of the IO word counter while they are in the process of being stepped. Such a situation would invalidate the instruction. Therefore, the step IO word counter pulse engendered by the word demand pulse is also sent to the CSW control circuit. (See fig. 5–41.) All components illustrated in this figure are located in the instruction control element.

The logic of the CSW control circuit permits transfer of information between the IO word counter and the right accumulator register at either IP–1 or IP–5. If the word counter is being stepped at IP–1, the transfer is delayed until IP–5. If the word counter is not being stepped, the transfer is allowed to take place at IP–1 and the IP–5 transfer is inhibited. Consequently, the IO word counter-right accumulator register transfer is safely effected during the *Clear and Subtract Word Counter* instruction regardless of the time at which the IO word counter is stepped.

In their reset or neutral condition, both the first and second transfer flip-flops are cleared. To realize the results described above, the step IO word counter pulse first sets the first transfer

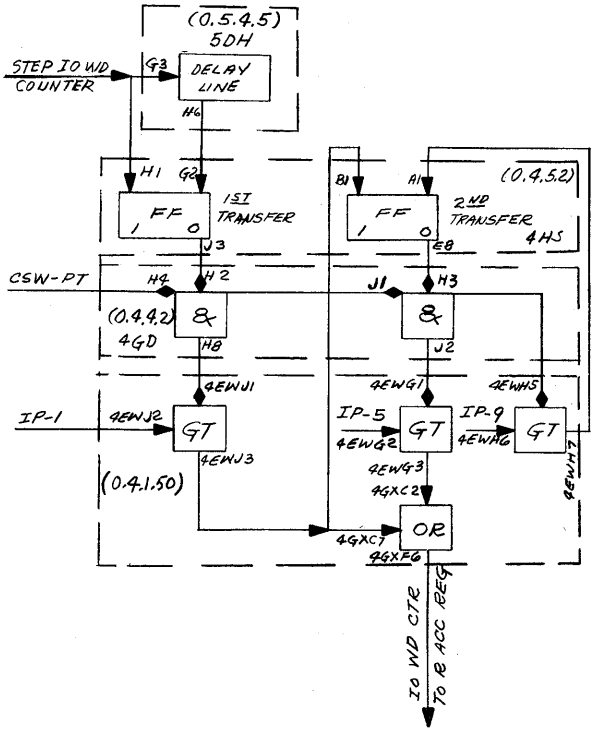


Figure 5–41. CSW Control Circuit, Logical Block Diagram

flip-flop, thus disabling the transfer circuit associated with an IP–1 pulse. After a delay of one microsecond, the first transfer flip-flop is cleared, since IP–1 can no longer have any detrimental effect. Since the CSW–PT line shown in the figure must be energized for the problem to exist at all, the AND circuit associated with the 0 side of the second transfer flip-flop conditions a gate tube inspected at IP–5. As a result of the inspection, the word in the IO word counter is transferred to the right accumulator register. The IP–9 pulse, examining a gate tube conditioned by the fact that the Central Computer System is in a CSW–PT cycle, develops a pulse which clears the second transfer flip-flop. In the case just described, this clearing action is superfluous because the second transfer flip-flop is already in the 0 state. However, assume that at the time the IP–1 pulse appears, the first transfer flip-flop has not been set (IO word counter not being stepped). The desired transfer will then be effectd by the IP–1 pulse. To disable a further transfer at IP–5, the second transfer flip-flop must be set. The clearing pulse for the second transfer flip-flop developed by the IP–9 pulse is now necessary to return the CSW control circuit to its neutral condition.

The final function of the word demand pulse is to examine the status of the interleave flip-flop. If this flip-flop is set, false no compare pulses must be generated to ensure writing in only those registers requested by the interleaving code. A full explanation of the generation and use of the false no compare pulses is given in 3.2 of C–4, S–3.

3.3 BREAK REQUEST GENERATION

The various methods of generating break request pulses during a drum writing procedure are treated in detail in 1.1.2.2 of C–2, S–1. The material presented there is summarized in this paragraph for the sake of continuity.

Two break-out cycles are requested as a direct result of the *Write* instruction. The first break request is made by the drum write signal sent to the Drum System at PT_2–6 of the instruction. The same signal also sets a second break request flip-flop. (This flip-flop is distinct from the second break request flip-flop used for card machine transfers.) Shortly after BO–7 of the first break-out cycle, a signal is developed which transfers the first word from the IO registers to the drum write register. At the same time, the status of the second break request flip-flop is examined. As a result of the examination, a second break request pulse is generated.

The third break request pulse does not appear until the 120-microsecond delay introduced by the Drum System at the time the first request is concluded. The request is a result of a word demand pulse which is sent to the selection and IO control element after the first word has been written. All succeeding break requests are made in the same manner. Consequently, a break request is generated every time a word is written on the drum surface.

3.4 BREAK-OUT MEMORY CYCLE

The first two break-out cycles assigned during a drum write operation are slightly different from those which follow. Since the execution of these two cycles has been completely described in 1.1.2.2 of C–2, S–1 the present discussion will only deal with the break-out cycle which follows a word demand pulse.

The generation of a word demand pulse results in the setting of the break request flip-flop, ensuring the eventual assignment of a break-out cycle. The cycle is initiated after the TP–0 pulse clears the memory address register. The pulse at BO–1 clears the break request flip-flop and transfers the IO address register contents to the memory address register. The BO–2 pulse steps the IO address counter. At BO–7, the contents of the memory buffer registers are sent to the IO

registers, a parity count is performed, and the IO register status flip-flop is set. The fact that the drum write register is empty, that the IO registers are full, and that a drum write process is in effect is represented in the selection and IO control element by three flip-flops: the IO register status flip-flop (set by the BO–7 pulse), the write register status flip-flop (cleared by the word demand pulse), and the write drums flip-flop (set during the *Select Drum* instruction). The outputs of these three flip-flops are combined in an AND circuit whose output conditions a gate tube constantly examined by 2-megacycle pulses from the oscillator. During a drum write process, therefore, this gate tube will develop a signal pulse whenever the IO registers are full and the drum write register is empty. The signal pulse is used to transfer the word in the IO registers to the drum write register. As a result, during any break-out cycle a word is transferred from core memory to the drum write register. At the end of the cycle, the IO registers will be empty and the drum write register will be full. A constant check on the status of the transfer is maintained by BO–11, which examines the IO word counter status during every break-out cycle.

3.5 IO PROCESS TERMINATION

As soon as the IO word counter is stepped from negative zero to positive zero, an end carry pulse will be developed. The end carry pulse indicates that no more words are to be transferred to the drum surface. The circuit shown in figure 5–42 illustrates how the process is terminated. The drum operation and not read drum flip-flops are set for the duration of the drum write process. The write register status flip-flop, cleared by every word demand pulse, is not reset before BO–7, at which time the IO register-drum write register transfer takes place. Therefore, as soon as the IO word counter status flip-flop is cleared by the end carry pulse, both AND circuits shown in the figure are conditioned. The AND circuit associated with the drum operation and the IO word counter status flip-flop condition a gate tube examined by 2-megacycle pulses from the oscillator. The output pulse from this gate tube disconnects the Drum System from the Central Computer System. The IO interlock is not cleared until BO–11, at which time the gate tube associated with the three-way AND circuit in the figure propagates a pulse which, after being passed by the gate tube conditioned by the drum operation flip-flop, clears the IO interlock.

The IO process termination depends upon a drum disconnect signal generated by the Central Computer System. In the special case of the output buffer (OB) drum, the IO process can also be terminated by a disconnect pulse generated by the Drum System. This pulse can appear during either a reading or writing operation. The OB drum is read by status. As soon as its full complement of registers have been examined for the appropriate status bits, an OB disconnect pulse is sent to the selection and IO control element. As shown in figure 5–43, the pulse clears the write drums flip-flop and sets the drum disconnect sync flip-flop. By clearing the write drums flip-flop, the circuits which effect the transfer of a word from

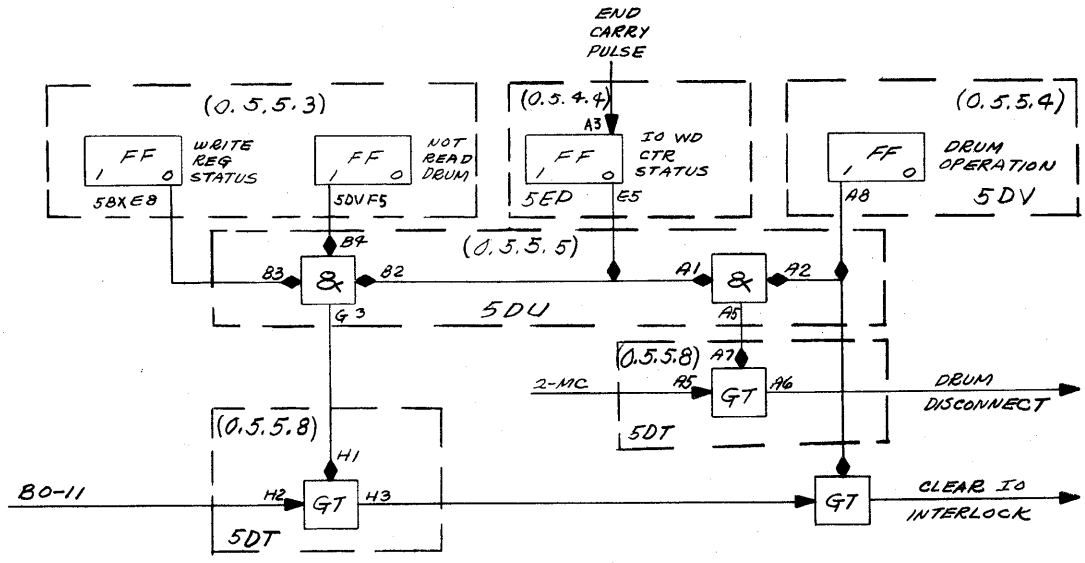


Figure 5–42. IO Termination Circuit—Drum Write, Logical Block Diagram

the IO register to the drum write register are disabled, and no more words will flow between the Central Computer System and the OB drum. By setting the drum disconnect sync flip-flop, the IO interlock and not read flip-flops are not cleared until the next TP–7 pulse.

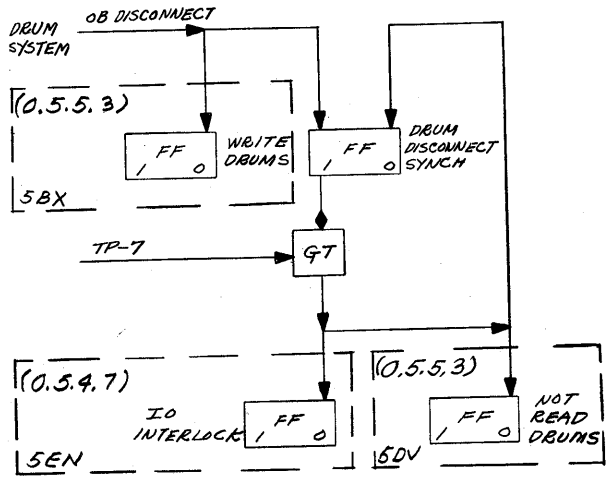


Figure 5–43. OB Disconnect Circuit, Logical Block Diagram

CHAPTER 5

TAPE AND MISCELLANEOUS IO UNITS

Four magnetic tape units are associated with the Central Computer System, each comprised of a tape adapter and a tape drive. There are also four miscellaneous IO units used by the computer for its IO operations: the manual input matrix, the burst time counters, the IO register, and the mechanical clock. Whereas both reading and writing operations are permissible between the Central Computer System and the tape units, only reading operations can be performed when the computer is linked with any one of the four miscellaneous units. An additional distinction between these two categories is that all words involved in a Central Computer System tape unit transfer have parity bits associated with them, while all of the miscellaneous IO units send words to the computer without an assigned parity bit.

5.1 TAPE UNITS

Word bits on a tape are arranged in six transverse columns; the first five columns contain six bits each, and the final column contains three bits. This does not include the synchronizing bit found in each column. Upon receiving a word from the Central Computer System, the tape circuits split it into the groups required for each tape column. In transmitting a word, the tape circuits reverse the process, composing a 32-bit word from the tape columns before sending it to the IO registers in the arithmetic element. Both of these operations are controlled by the tape timing system. This system operates asynchronously to the Central Computer System. The tape unit is therefore permitted to generate its break requests and disconnect pulses independently of Central Computer System timing. The necessary synchronization is accomplished in the selection and IO control element.

5.1.1 IO Process Preparation

To select a tape unit as the IO device, the index interval of the *Select* instruction must contain in binary form one of the four numbers from 11_8 to 14_8. At OT–5 of this instruction, the deselect command clears the tape operations flip-flop in the selection and IO control element. At PT_2–5 of the instruction, the same flip-flop is set, conditioning the various circuits for the tape operation shown in figure 5–44.

To avoid possible Central Computer System hang-up, two separate *Sense* instructions must follow the *Select* (tapes) instruction in the program sequence. These are the *Sense* (10_8) and *Sense* (11_8) instructions. (See fig. 5–45.) During the first instruction, the No. 10 output of the PerSelBsn matrix conditions a gate tube associated with the tapes not prepared flip-flop. If, at some time before OT–9 of the instruction, a tapes not prepared signal has been received by the selection and IO control element, the IO interlock will be cleared and the tapes not prepared flip-flop set. The outputs of the conditional gate tube and the tapes not prepared flip-flops are combined by an AND circuit. The output of this AND circuit conditions a gate tube inspected at OT–9. As a result, the branch flip-flop will be set and the computer will transfer its operations to some other area. If the tapes not prepared signal is not received, the branch flip-flop is not set and the second required *Sense* instruction, *Sense* 11_8, is executed. The No. 11 output of the PerSelBsn matrix is energized as a result and the previous procedure is repeated for the tapes not ready flip-flop. Since the not ready condition for a tape unit is of a more transitory nature than the tapes not prepared condition, the IO interlock would not be cleared should a tapes not ready signal occur. The Central Computer System still branches but the IO interlock remains set, thus ensuring that no other IO process will take precedence.

The four operate units associated with the tape units are listed in table 5–8 along with their octal codes.

TABLE 5–8. OPERATE CODES FOR TAPE UNITS

OPERATE UNIT	BINARY CODE	PERSELBSN MATRIX OUTPUT
Set prepared	110111	67_8
Backspace tape	111000	70_8
Rewind tape	111001	71_8
Write end of file	111010	72_8

Each operate unit is energized by programming an *Operate* instruction containing the appropriate index interval. All operate units except the set prepared unit set the IO interlock while they are energized. At the conclusion of the desired operation, a tape disconnect pulse is gen-

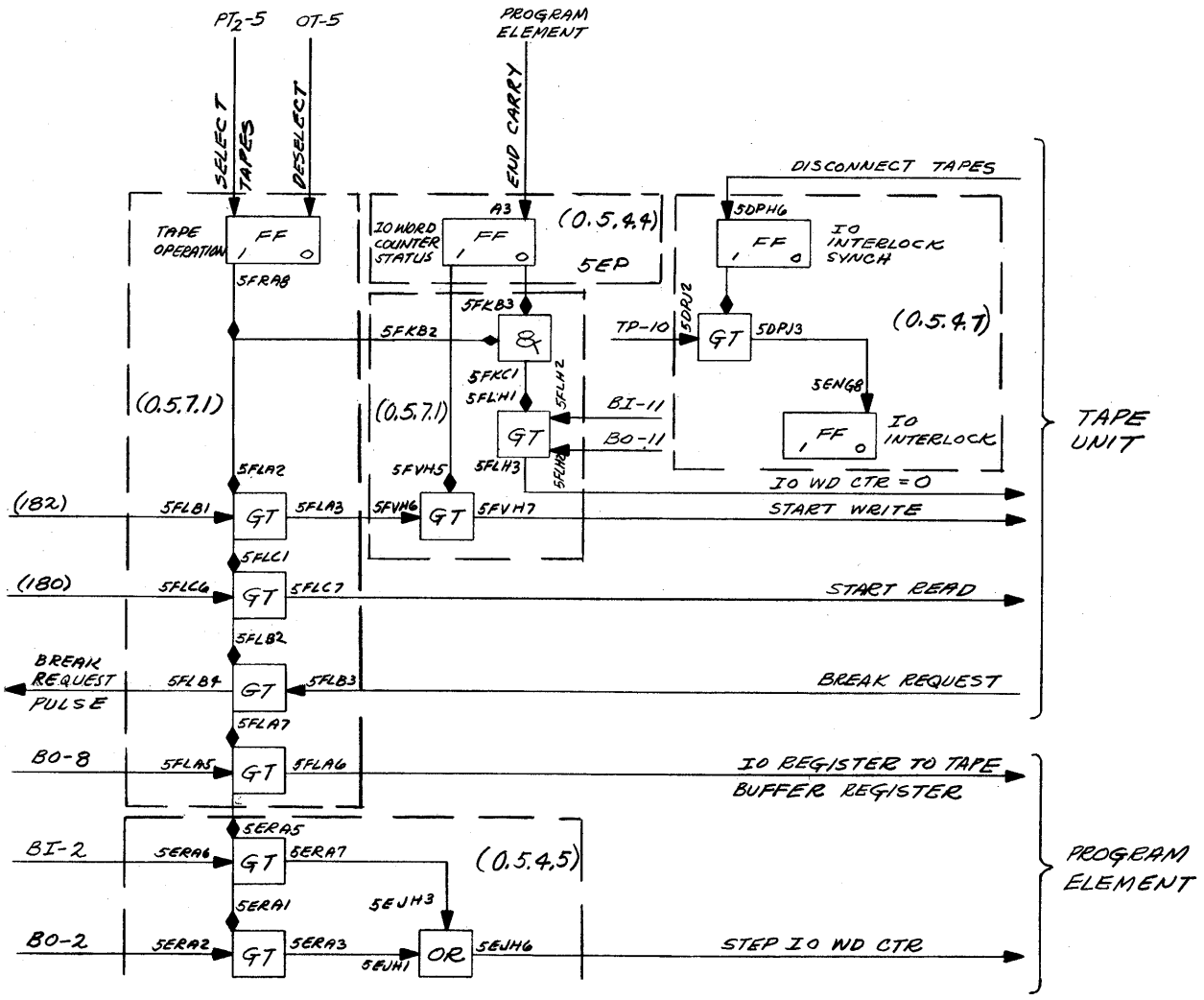


Figure 5–44. Tape Unit Control Circuits, Logical Block Diagram

erated by the operative tape unit and this pulse, returned to the selection and IO control element, clears the IO interlock. The representative circuit used for the *Operate* instruction is shown in figure 5–46. The set prepared operate unit does not require the presence of the IO interlock because it merely allows the tape unit to read two consecutive files on the same tape without interruption. Without this facility, the tape unit would stop at the end of the first file.

After the required *Sense* and *Operate* instructions have been executed, a *Read* or *Write* instruction is placed in the operation register. Assume that it is a *Read* instruction which is to be performed. In the course of the instruction, the IO interlock and the IO word counter status flip-flop will be set. A *Read 0* instruction is guarded against as described in 3.6.1 of Chapter 3. At PT_2–6, command 180, passing through a gate tube conditioned by the 1 side of the tape operations flip-flop, is sent to the selected tape unit as a start read pulse.

5.1.2 IO Process

Upon receipt of the start read pulse, the tape unit will compose a 33-bit word in the tape buffer register. As soon as this word is ready, a break request pulse is sent to the selection and IO control element at the same time that the word itself is transferred to the IO registers. The break request pulse passes a gate tube conditioned by the 1 side of the tape operations flip-flop and, if the IO word counter status flip-flop is set, sets the break request flip-flop. Refer to Chapter 2 for a complete description of the break request circuits.

The succeeding memory cycle is assigned as a break-in cycle. At TP–0, the memory address register is cleared. At BI–1, the memory address register receives the contents of the IO address counter. As shown in figure 5–44 BI–2 steps the IO word counter. BI–2 also effects the transfer

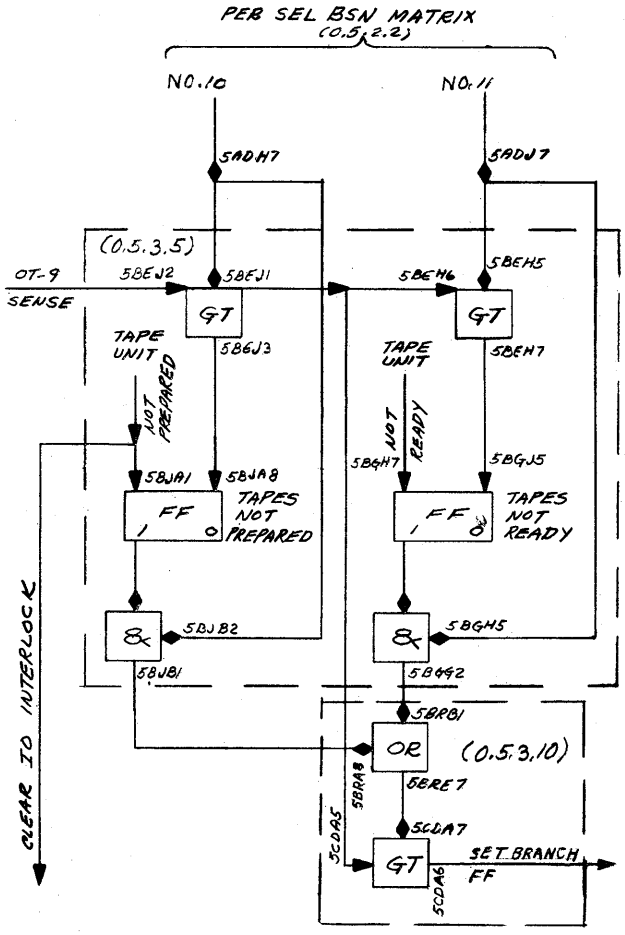


Figure 5–45. Sense Circuits—Tape Units, Logical Block Diagram

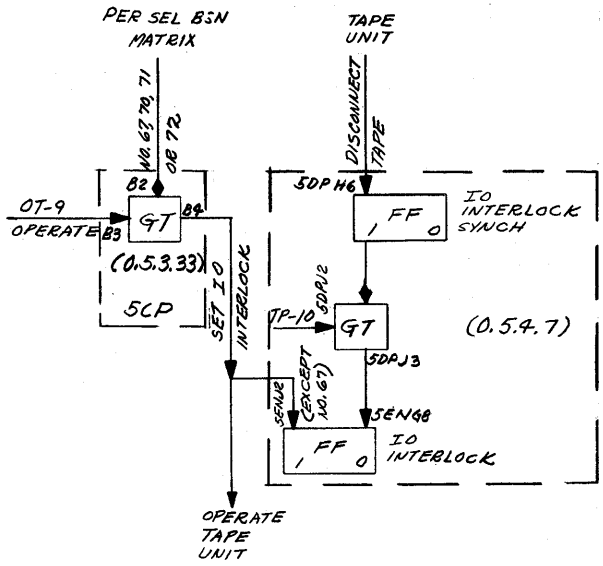


Figure 5–46. Operate Circuits—Tape Units, Logical Block Diagram

of the word in the IO registers to the memory buffer registers. At BI–3, the IO address counter is stepped. The normal action of the core memory circuits serves to place the word now in the memory buffer registers into the core memory location specified by the memory address register. This sense procedure is iterated as long at the IO word counter contents are different from 0. As soon as 0 does appear in the IO word counter, the IO word counter status flip-flop in the selection and IO control element is immediately cleared by the resultant and carry pulse. An AND circuit associated with the 0 side of the IO word counter status flip-flop and the 1 side of the tape operations flip-flop is thereby conditioned. The gate tube conditioned by the output of the AND circuit is examined at BI–11 and generates a signal which informs the selected tape unit that no more words are to be transferred. Accordingly, the tape unit returns a tape disconnect pulse to the selection and IO control element. The pulse sets the IO interlock sync flip-flop. At TP–10, the IO interlock is cleared and the IO process is terminated.

The sequence of operations for a writing operation with the tape units is very similar to the one just described for reading. The only significant difference is that the transfer of the word between the Central Computer System and the selected tape unit always occurs at BO–8 rather than during an asynchronous process.

5.2 MISCELLANEOUS IO UNITS

This discussion will not consider the Warning Light System. See 6.3 of Chapter 6 for a complete description of this system. If a miscellaneous IO unit has been selected, neither the *Sense* nor the *Operate* instruction is applicable. Furthermore, since a miscellaneous IO unit can only transmit information to the Central Computer System, a *Select* instruction can only be followed by a *Read* instruction. The proper IO program from any miscellaneous IO unit is therefore *Load IO Address Counter, Select,* and *Read.* The index interval codes applicable to the miscellaneous IO units are listed in table 5–9.

TABLE 5–9. SELECTION CODES FOR MISCELLANEOUS IO UNITS

IO UNIT	BINARY CODE	PERSELBSN MATRIX OUTPUT
Manual input matrix	000110	06_8
Burst time counters	010001	21_8
IO register	000100	04_8
Mechanical clock	000101	05_8

5.2.1 Manual Input Matrix and Burst Time Counters

The manual input matrix and the burst time counters share essentially the same IO circuits. These circuits are illustrated in figure 5–47. The manual input matrix receives information from the photoelectric pickup unit and from its own keyboard. No more than 120 words can be transferred from this unit to the Central Computer System during a single IO process. Each word must pass through the IO buffer registers before being placed in the IO registers. The three burst time counters determine the time of transmission of outgoing messages. No more than eight words can be transferred from this unit to the Central Computer System during a single IO process. Each word must pass through the IO buffer registers as in the case of the manual input matrix.

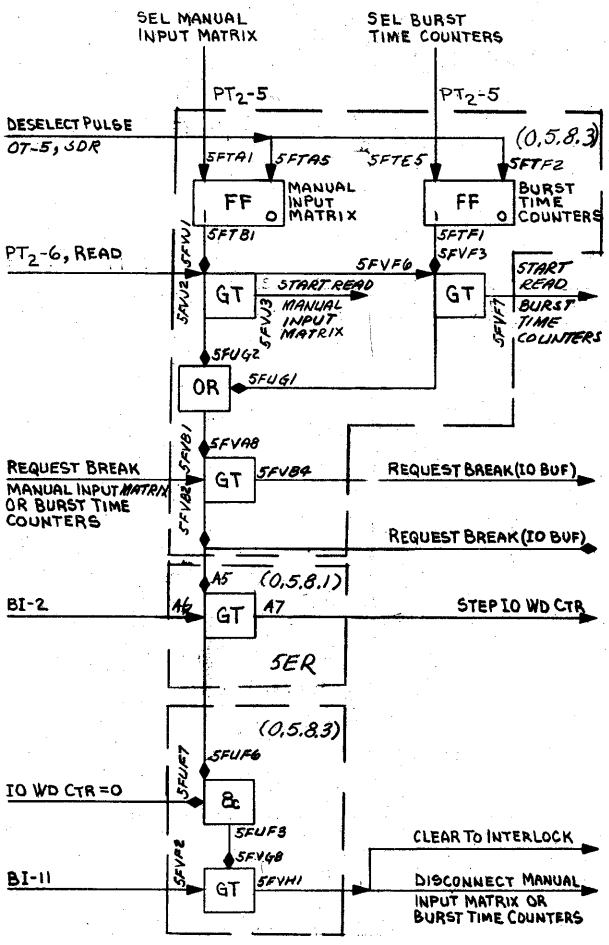


Figure 5–47. Manual Input Matrix and Burst Time Counters Control Circuits, Logical Block Diagram

An analysis of the IO operation of the manual matrix follows, but the same analysis can be applied to the burst time counters. During the *Select* instruction, the manual input matrix flip-flop is first cleared at OT–5 by the select command and then set at PT_2–5. Execution of the *Read* instruction results in setting the IO interlock and the IO word counter status flip-flop. At PT_2–6 of the instruction, a start read pulse is gated by the 1 side of the manual input matrix flip-flop and sent to the manual input matrix. The matrix returns a break request pulse and at the same time places its first word in the IO buffer registers. Since the speed of this IO device is comparable to that of the Drum System, the Drum System break request circuits are also used for the manual input matrix. Note from the figure that two signals are required for this purpose, a d-c signal level and a break request pulse. Refer to Chapter 2 for an analysis of the break request generation using the Drum System circuits.

The break-in cycle is subsequently assigned and during its six microseconds, the IO word counter and the IO address counter are stepped by BI–2 and BI–3, respectively; the word in the IO buffer registers is transferred successively to the IO registers and the memory buffer registers; and the break request flip-flop is cleared. Each new word is treated in the same way, the rate of their arrival being constant and dictated by the design of the manual input matrix. As soon as the IO word counter contents become 0, the end carry pulse resulting from the final stepping operation for this counter clears the IO word counter status flip-flop. The 0 side of this flip-flop and the 1 side of the manual input matrix flip-flop fully condition a two-way AND circuit whose output gates a clear IO interlock and disconnect manual input matrix signal at BI–11.

5.2.2 IO Register

The IO register as an IO unit provides a means of clearing specific locations in core memory. When used for this purpose, the IO register is first cleared. Consequently, all core memory addresses specified during an IO process involving the IO register are cleared.

The IO control circuits for the IO register are shown in figure 5–48. After the *Load IO Address Counter* instruction has specified the number of transfers to be performed, the *Select* (IO register) instruction sets the IO register flip-flop, fully conditioning the two gate tubes shown on the figure and partially conditioning a two-way

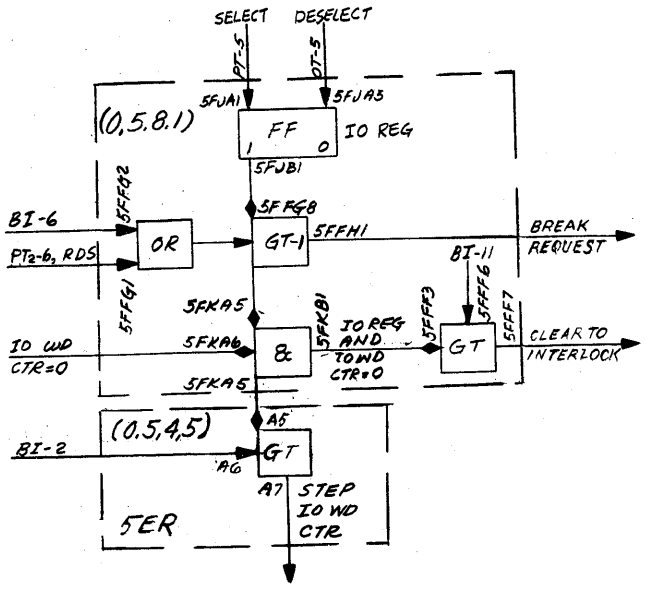


Figure 5–48. IO Register Control Circuits, Logical Block Diagram

AND circuit. The second condition for the AND circuit is that the IO word counter contain 0. The subsequent *Read* instruction sets the IO interlock and the IO word counter status flip-flop. At PT_2–6, a break request is generated by GT–1 and as a result, a break-in cycle is assigned. During the cycle, the core memory location specified by the IO address counter at the beginning of the process is cleared. The IO word counter contents are reduced by 1 at BI–2, and the IO address counter is stepped at BI–3. Since the IO register is not capable of generating its own break request pulses, this function is assigned to the BI–6 pulse of the break-in cycle. The generation of the break request assures that the next memory cycle will also be assigned as a break-in cycle. A continuous train of break-in cycles are assigned in this manner until the appearance of an end carry pulse from the IO word counter gives indication that the required number of clearing operations have been performed. The end carry pulse clears the IO word counter status flip-flop, fully conditioning the AND circuit shown in figure 5–48. The next BI–11 pulse therefore clears the IO interlock, thus terminating the IO process.

CHAPTER 6

MISCELLANEOUS DATA

This section contains an analysis of those circuits which were not considered in the preceding sections since they were not directly involved in the basic IO process. These circuits may be grouped as follows: miscellaneous sense units, miscellaneous operate units, Warning Light System, and parity circuits. The sense and operate units function exactly as did the units of this type described in Chapters 3 and 4. The present discussion will only complete the consideration of these units by defining all the additional sense and operate units which have been provided. The Warning Light System is comprised of a group of lights which inform console operators of important circuit conditions. The status of any one of these lights is under the control of the selection and IO control element. The parity circuits appearing in this element are used in conjunction with the parity circuits of the memory buffer registers in the arithmetic element to detect errors when a word is transferred into or out of core memory. In addition, the parity circuits dictate the action to be taken by the Central Computer System should a parity error occur.

6.1 MISCELLANEOUS SENSE UNITS

Of the 20 sense units provided for Combat Direction Central AN/FSQ-7 only four have been described in the preceding sections: the tape not prepared, the IO unit not ready, the printer No. 1, and the printer No. 2. The remaining 16 sense units are listed in table 5-10. Each sense unit is a flip-flop which reflects some circuit condition. When a sense unit is specified by the index interval of a *Sense* instruction, the specific sense flip-flop is inspected. If the status of the flip-flop represents an undesirable condition, the Central Computer System unconditionally branches to the instruction contained in the address given by the right half-word of the *Sense* instruction.

The definition of an undesirable condition, in a limited and restricted sense, was given in the previous paragraph; i.e., if the Central Computer System is forced to branch, the status of the specific sense unit reflects an undesirable circuit condition. For the four condition lights, this condition consists of their being energized. Accordingly, execution of a *Sense* instruction specifying any one of the condition lights will cause the program to branch if the light is energized.

TABLE 5-10. SENSE UNIT SELECTION CODE

SENSE UNIT	BINARY CODE	PERSELBSN MATRIX OUTPUT	REMARKS
Condition light No. 1	000001	01_8	Branch if on and extinguish light
Condition light No. 2	000010	02_8	Branch if on and extinguish light
Condition light No. 3	000011	03_8	Branch if on and exinguish light
Condition light No. 4	000100	04_8	Branch if on and extinguish light
Left overflow	001010	12_8	Branch if on and reset
Right overflow	001011	13_8	Branch if on and reset
IO interlock	001100	14_8	Branch if on
Memory parity	001101	15_8	Branch if on and reset
Drum parity	001110	16_8	Branch if on and reset
Tape parity	001111	17_8	Branch if on and reset
Marginal checking excursion on	010000	20_8	Branch if on
Sense switch No. 1	010001	21_8	Branch if on
Sense switch No. 2	010010	22_8	Branch if on
Sense switch No. 3	010011	23_8	Branch if on
Sense switch No. 4	010100	24_8	Branch if on
Output alarm	011011	33_8	Branch if on and reset

However, the light will be extinguished in the process. The sense switches, which are two-position switches located on the maintenance console of the Central Computer System, can also be sensed. If the switch is in the OFF position, sensing the flip-flop for this switch does not disturb the program sequence. If the switch is in the ON position, the program will branch.

The presence of an overflow condition in either the left or right accumulator register can

also be considered an undesirable condition. Therefore, if either condition exists and a *Sense* instruction specifying the sense unit corresponding to the overflow is being executed, the Central Computer System will undergo a branch of control. By this same procedure, a branch of control will take place if the IO interlock is on, if a marginal checking excursion is being performed, or if an output alarm or parity error is present.

6.2 MISCELLANEOUS OPERATE UNITS

The operate units not discussed are listed in table 5–11. Each operate unit can be energized by making its binary code bits the index interval of an *Operate* instruction. As a result, an energizing signal is sent from the selection and IO control element to the selected operate unit.

TABLE 5–11. OPERATE UNIT SELECTION CODE

OPERATE UNIT	BINARY CODE	PERSELBSN MATRIX OUTPUT
Condition light No. 1	000001	01_8
Condition light No. 2	000010	02_8
Condition light No. 3	000011	03_8
Condition light No. 4	000100	04_8
Operate area discriminator phototube No. 1	001111	17_8
Operate area discriminator phototube No. 2	010000	20_8
Marginal checking—start excursion	010001	21_8
Marginal checking—stop excursion	010010	22_8
Situation display camera No. 1	011001	31_8
Start digital display	011101	35_8
Lock IO address counter	111101	75_8
Reset scan counter	111110	76_8
Step scan counter	111111	77_8

Four condition lights, already mentioned in the description of the sense units, have been provided as operate units. These lights are located on the maintenance console. The energizing signal developed is used to turn on the appropriate light. The reset scan counter and step scan counter operate units are used in the control of the scan counter associated with the radar data drum in the same manner that the two operate units for the area discriminators are used to control the operation of these discriminators. Two operate units have been incorporated to enable programmed control of marginal checking excursions.

The final operate unit to be considered, the lock IO address counter, is used only in conjunction with a writing operation. If an *Operate* (75_8) instruction is given immediately before a *Write* instruction, the IO address counter is not stepped during the subsequent IO operation. As soon as the IO operation is completed and the IO interlock turned off, the lock IO address counter operate unit is also turned off and does not affect succeeding IO processes. This operate unit therefore causes the contents of a single register in core memory to be written in the successive registers of the IO unit. One of the more obvious applications of this operate unit is to clear a drum field by specifying a core memory address which contains 0 and selecting the drum field that is to be cleared.

Figure 5–49 depicts the method used to accomplish the desired result. At OT–9 of the *Operate* instruction, the gate tube conditioned by output No. 75 of the PerSelBsn matrix generates the pulse which clears the lock IO address counter operate unit. The gate tube through which BO–2 must pass before stepping the IO address counter is thereby disabled and the contents of the IO address counter remain unchanged as long as this condition persists. The IO process resulting from the subsequent *Write* instruction otherwise preceeds in the usual manner. As soon as the IO word counter, by generating an end carry pulse, indicates that the IO process is terminated, the lock IO address counter operate unit is cleared to avoid automatically restricting any succeeding IO process.

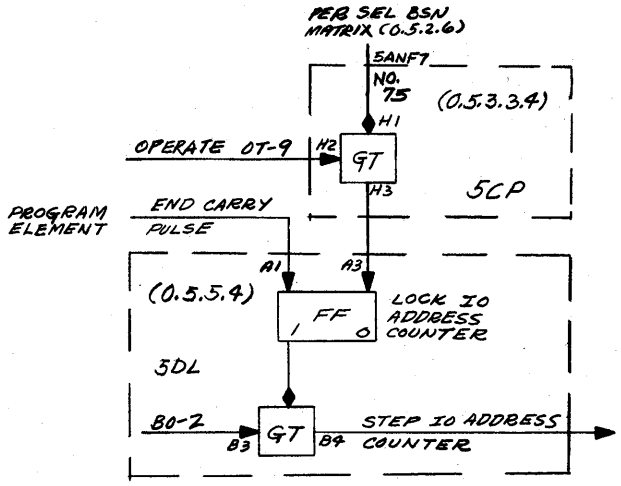


Figure 5–49. Lock IO Address Counter Control Circuit, Logical Block Diagram

6.3 WARNING LIGHT SYSTEM

Warning lights are used to warn operators at different consoles of certain Central Computer System actions. Provision is made for 256 lights, divided into eight groups of 32 lights each. Each group can be considered a 32-bit register and each register an IO unit. Accordingly, to make use of the Warning Light System, a *Load IO-Address-Counter, Select* (10_8), *Write* program must be scheduled. The circuits required in the selection and IO control element to implement the use of the Warning Light System are shown in figure 5–50.

The principle of operation depends upon the fact that complementing a flip-flop an even number of times does not alter its original status. The IO registers will be loaded with a new word after every two memory cycles. The first of these cycles is a break-out cycle during which the word in the IO registers is transferred to one of the warning light registers; the second is a dummy cycle. The original status of the warning light registers is immaterial; at the end of the IO process, each will duplicate the status of the core memory word assigned to it.

The address of the first core memory word to be used is placed in the IO address counter by the *Load IO Address Counter* instruction. During the *Select* instruction, the warning light flip-flop is set and the auxiliary warning light flip-flop is cleared. See figure 5–50. After the *Write* instruction, the IO interlock and the IO word counter status flip-flop are set, and the IO word counter contains the complement of the right half-word of the *Write* instruction. The magnitude of the number in the IO word counter cannot exceed eight because there are only eight warning light registers, but it can have any intermediate value between one and eight.

The final pulse of the *Write* instruction, PT_2–6, requests a break, sets the counter control flip-flop, and sets the warning light register counter to one by setting its No. 1 flip-flop and clearing the other two. The register matrix converts the reading of the warning lights register counter into a positive d-c level on matrix output 100. The AND circuit conditioning GT–1 is therefore energized.

As a result of the break request, the next memory cycle is assigned as a break-out cycle. The IO word counter and the IO address counter are stepped at BO–2 and BO–3, respectively. After the stepping process is completed, the IO word counter contains n–1 and the IO address counter contains the address of the core memory word which is to affect the second warning light register. The word for the first warning light register has already been placed in the memory buffer registers. This first word is transferred from the memory buffer registers to the IO registers at BO–7.

As seen in figure 5–50, the BO–8 pulse is gated by the warning light flip-flop. Since this flip-flop was set at PT_2–5 of the previous *Select* instruction, the BO–8 pulse is passed, effecting the transfer of the word in the IO registers to all eight warning light registers. To clarify the complementing process, assume that there are only three warning light registers, four lights composing each register, and that the IO registers contain only four flip-flops. The initial and final conditions are ascribed as given in table 5–12.

TABLE 5–12. INITIAL AND FINAL CONDITIONS OF WARNING LIGHT REGISTER FLIP-FLOPS

WARNING LIGHT REGISTER	FF–1	FF–2	FF–3	FF–4
Initial condition				
No. 1	On	Off	On	Off
No. 2	Off	On	Off	On
No. 3	Off	Off	On	Off
No. 4	On	On	Off	Off
Desired condition				
No. 1	Off	On	Off	Off
No. 2	On	Off	Off	Off
No. 3	Off	Off	Off	On
No. 4	On	On	On	On

Note that the status of each flip-flop is represented by the actual condition of the light which it controls. To arrive at the desired conditions, the IO register must receive the following four words from the memory buffer register: first word, 0100; second word, 1000; third word, 0001; and fourth word, 1111. As explained previously, each flip-flop of the IO register which contains a 1 will complement the corresponding bit position in all eight warning light registers. If an IO register flip-flop contains a 0, the corresponding bit position in the warning light registers will be left unchanged. Consequently, the status of the warning light registers after the complementing process initiated by BO–8 is completed is as shown in table 5–13.

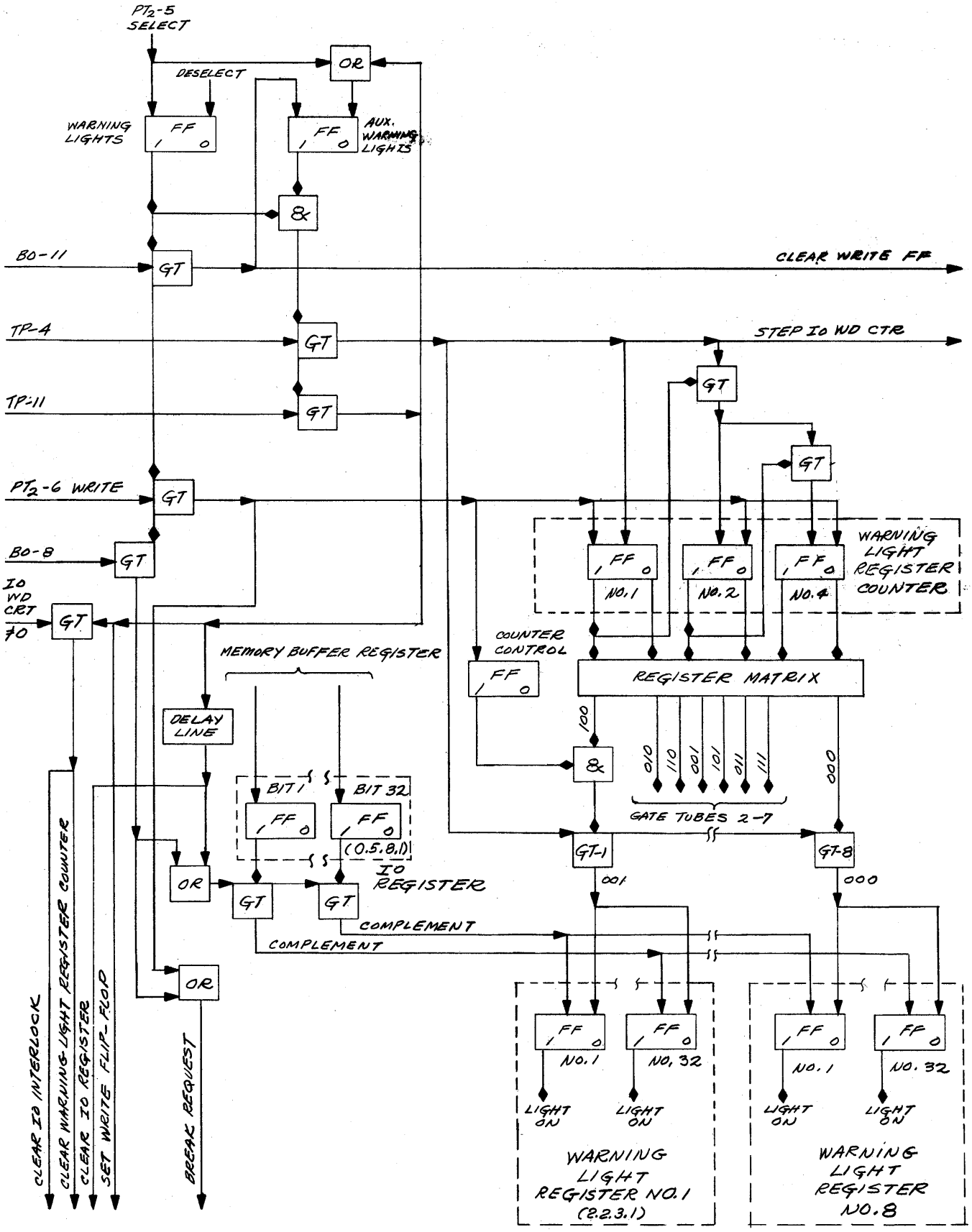


Figure 5–50. Warning Light Circuits, Logical Block Diagram

TABLE 5–13. CONDITION OF WARNING LIGHT REGISTER FLIP-FLOPS AFTER FIRST COMPLEMENTING PROCESS

WARNING LIGHT REGISTER	FF–1	FF–2	FF–3	FF–4
No. 1	On	On	On	Off
No. 2	Off	Off	Off	On
No. 3	Off	On	On	Off
No. 4	On	Off	Off	Off

The warning light register flip-flops are slow-speed flip-flops, therefore about two microseconds must elapse before they can be operated upon again. Since there is not enough time between BO–8 and BO–11 to provide the two microseconds, a dummy cycle must be assigned. Accordingly, the BO–8 pulse requests a break. To prevent the generation of break-out pulses during the dummy cycle, the write flip-flop is turned off at BO–11. The BO–11 pulse also sets the auxiliary warning light flip-flop, thereby conditioning the circuits which are to be inspected during the dummy cycle.

Since both the warning light and the auxiliary warning light flip-flops are set, the AND circuit associated with the auxiliary warning light flip-flop is enabled. Its output conditions two gate tubes, one inspected at TP–4 of the dummy cycle and the other at TP–11. At TP–4, the IO word counter is stepped and the eight gate tubes conditioned by the register matrix are simultaneously interrogated. At this stage of the process, GT–1 is conditioned and, as a result, warning light register No. 1 is cleared. The status of the other warning light registers remains unchanged. The TP–4 pulse will also stop the warning light register counter by complementing its No. 1 and No. 2 flip-flops. The resultant output from the register matrix will therefore be 010, conditioning GT–2 (not shown in the figure) and deconditioning GT–1. However, the clearing pulse generated by GT–1 for warning light register No. 1 is developed before this gate tube is disabled. The second core memory word will not be placed in the IO register during the dummy pulse because of the absence of break-out pulses. At TP–11, the flip-flops of the warning light registers will again be complemented in accordance with the contents of the IO register (0100). Including the effect of the clearing operation at TP–4, the status of the warning light registers will now be as shown in table 5–14.

TABLE 5–14. CONDITION OF WARNING LIGHT REGISTER FLIP-FLOPS AFTER DUMMY CYCLE EXECUTION

WARNING LIGHT REGISTER	FF–1	FF–2	FF–3	FF–4
No. 1	Off	On	Off	Off
No. 2	Off	On	Off	On
No. 3	Off	Off	On	Off
No. 4	On	On	On	On

It should be noted that the desired configuration exists in warning light register No. 1 and that the other registers have regained their original status. The TP–11 pulse also clears the auxiliary warning light flip-flop, resets the write flip-flop, and clears the IO register.

With the break and write flip-flops both set, the following memory cycle will be another break-out cycle. During this cycle and its attendant dummy cycle, the second warning light register will have its contents changed to conform with the dictates of the word in the IO register. Following the TP–11 pulse of the dummy cycle, the first two warning light registers will be correctly conditioned, GT–3 will be conditioned by the register matrix, and the remaining registers will still retain their initial contents. Each TP–11 pulse that is generated examines the status of the IO word counter. As soon as this counter contains 0, a gate tube conditioned by the IO word counter status flip-flop will propagate the TP–11 pulse. As a result, the IO interlock and the counter control flip-flop will be cleared.

6.4 PARITY CIRCUITS

The parity circuits in the Central Computer System (XD–1,–2) provide a means of detecting errors which arise in computer words as they are transferred from one register in the system to another. These errors can only be detected while the word is in the memory buffer register. The error detection circuits are not infallible, however, since they will only detect errors in which an odd number of bits have been reversed. Thus, if one bit in a word is altered, the parity circuits associated with the memory buffer register will detect this error and generate an alarm. However, if two bits are altered, the parity circuits will not detect the error and no alarm will be generated. In general, the probability of multiple errors in a word transfer is small so that the parity circuits still provide a convenient and efficient means of error detection for the Central Computer System.

A complete parity operation requires three steps: a first parity count, a second parity count, and a parity check. The first parity count examines the 32 bits of the normal Central Computer System word and assigns to it a 33rd bit, called a parity bit. This parity bit is assigned in accordance with the number of 1's in the original 32-bit word. If the count indicates an even number of 1's in the original word, a parity bit of 1 is assigned; if the count indicates an odd number of 1's a parity bit of 0 is assigned. With this method, a 33-bit word will always contain an odd number of 1's. In some cases, the first count is performed, not by the memory buffer register, but by some input unit. The second parity count again counts the number of 1's in the original 32-bit word. The result of the first count, as indicated by the parity bit, is then compared with the second count and the result of the comparison is retained by a parity count flip-flop. If the word has been transferred with no introduced errors, the two parity counts should agree. However, if the word has had an odd number of errors introduced during transfer, the two counts will disagree. By examining the status of the parity count flip-flop, the parity check operation determines whether or not an error has occurred. In the most usual case, a word resides in a core memory register between the times of the first and second counts.

The parity circuits of the Central Computer System which perform these operations are depicted in figure 5–51. Details of the counting circuits are not given here since their operation is fully described in Part 3, Arithmetic Element. A complete description of parity circuit operation with respect to word transfers between core memory and the memory buffer registers is also given therein. Hence, the discussion in this manual will be limited to a consideration of the parity operation during IO transfers.

6.4.1 Break-Out Parity Operation

If a word is to be transferred out of core memory to an IO unit, a break cycle is instituted in the Central Computer System and the word is read out of its memory register into the memory buffer register. Since all words stored in core memory have a parity bit associated with them, it is only necessary now to perform the second parity count and a parity check. The parity bit is placed in the parity count and parity storage flip-flops at the same time that the 32-bit word is loaded into the memory buffer register. To accomplish the second parity count operation, a BO–7 pulse of the break-out cycle is applied to the parity count circuit. As described in Part 3, an even (odd) count pulse will be derived from this circuit if the 32-bit word in the memory buffer registers contains an even (odd) number of 1's.

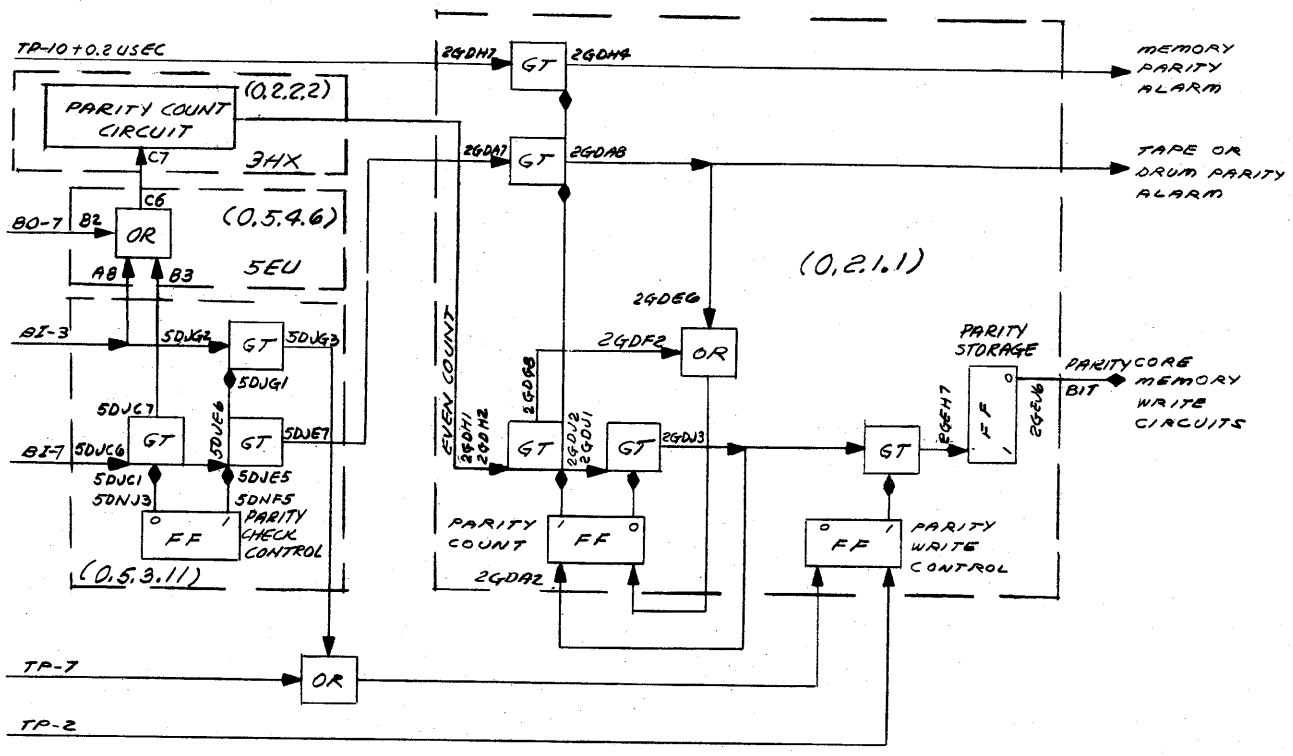


Figure 5–51. Parity Error Detection Circuits

In this case, the parity bit and therefore the parity count flip-flop will contain 1(0). The even count pulse is then gated to clear the parity count flip-flop, which will prevent a TP–10 0.2-microsecond pulse from being propagated as a memory parity alarm. TP–10, delayed 0.2 usec, has been used instead of TP–11 in order to retain the word in the memory buffer in the event of a memory parity alarm. This is typical of the parity operation when no transfer error has occurred. The parity bit retained in the parity storage flip-flop will be written back in core memory at the same time as the original word in the memory buffer registers is restored in its proper core register.

Now, if an error has occurred in the transfer of the word such that an odd number of 1's exist where previously these were an even number, the parity count will be odd and an even count pulse will not issue from the parity count circuit. Hence the parity count flip-flop will remain set and the succeeding TP–11 pulse will be gated through the gate tube on the 1 side of the parity count flip-flop to appear as a memory parity alarm. This is typical of parity circuit operation whenever an error has occurred in the transfer of a word presently being checked. It should be noted, however, that the original parity bit is still re-stored in core memory even though an error has occurred. This insures that any future attempts by the Central Computer System to read this word will result in a parity alarm.

The parity circuit operates in essentially the same manner if the parity bit is 0 and the correct 32-bit word contains an odd number of 1's. If the transfer is correct, no even count pulse will be derived from the parity count circuit, the parity count flip-flop will remain cleared, and a memory parity error will not be generated. If an error has occurred in the word transfer, an even count pulse will be generated which is gated by the 0 in the parity count flip-flop thus setting the parity count flip-flop from 0 to 1. The succeeding TP–11 parity check pulse is then gated as a memory parity error.

It is apparent that the status of the parity count flip-flop after the second parity count operation is performed always indicates whether or not the original and new parity count agree. If the parity count flip-flop contains 0, they do agree and no alarm is generated; if the parity count flip-flop contains 1 after the second parity count operation, the two counts do not agree and an alarm is generated. It is important to note that although the parity write control flip-flop is set at TP–2, it is cleared at TP–7 in time to inhibit any count pulse issued at TP–7 from altering the status of the parity storage flip-flop. Since the delay through the parity count circuit is over 0.5 microsecond, any count pulse issued at TP–7 will arrive too late to pass through the parity write control flip-flop gate tube. As previously stated, this insures that the original parity bit will be retained in core memory.

6.4.2 Break-In Parity Operation

Words to be transferred to the Central Computer System from an IO unit require the assignment of a break-in cycle. In such a case, the parity circuit must differentiate between incoming words which consist of 32 bits (no parity bit) and 33 bits (with a parity bit). As shown in figure 5–51, the necessary distinction is made by the parity check control flip-flop. This flip-flop is set when incoming words have a parity bit and is cleared when incoming words do not have a parity bit. It is also necessary that the parity circuit prevent alteration of the parity storage flip-flop while a word, including its parity bit, is being written into core memory and, in the case of an incoming 33-bit word, before it is written into core memory. If this was not prevented, a parity bit might be assigned to an incorrect word in such a manner that the indication of the error by the parity bit would be canceled. This problem is eliminated by the action of the parity write control flip-flop which will allow the original parity bit to be retained with any word which gives rise to a parity error. This insures that this word, upon being read from core memory at some later time, will invariably cause the generation of a parity alarm.

6.4.2.1 IO Words of 32 Bits

All of the IO units associated with AN/FSQ–7 (XD–1,–2) Combat Direction Central except the tape units and certain drum fields process 32–bit words which do not have a parity bit. For words of this length, two parity counts must be performed before the parity check. The first count assigns a parity bit and the second verifies it. If this order of operations is to be accomplished, the parity check control flip-flop must be cleared before the end of the *Read* or *Write* instruction which completes the setting up of the control circuits in the selection and IO control element for the IO process. This indicates to the parity circuits that neither a tape unit nor a drum field processing 33-bit words has been selected. The circuits which accomplish this are depicted in figure 5–52.

Initially, a gate tube conditioned by the 0 side of the tape operation flip-flop is sensed by the

PT_2–1 pulse of the *Read* or *Write* instruction. Any output pulse from this gate tube then senses a gate tube conditioned by the 0 side of the drum operation flip-flop. It is apparent that if an output pulse is derived from this second gate tube, both the tape and drum operation flip-flops contain 0, indicating that neither a drum field nor a tape unit has been selected and thus precluding the possibility of any incoming words having 33 bits. Accordingly, this pulse is used to clear the parity check control flip-flop. Now, if the PT_2–1 pulse of the *Read* or *Write* instruction is gated by the tape operation flip-flop but inhibited by the drum operation flip-flop, a drum field is the selected IO unit and the status of the parity check control flip-flop will depend on the specific field selected by the previous *Select Drum* instruction. As described in Chapter 4, the deselect pulse at OT–5 of the *Select Drum* instruction sets the parity control flip-flop unconditionally. At PT_2–5 this flip-

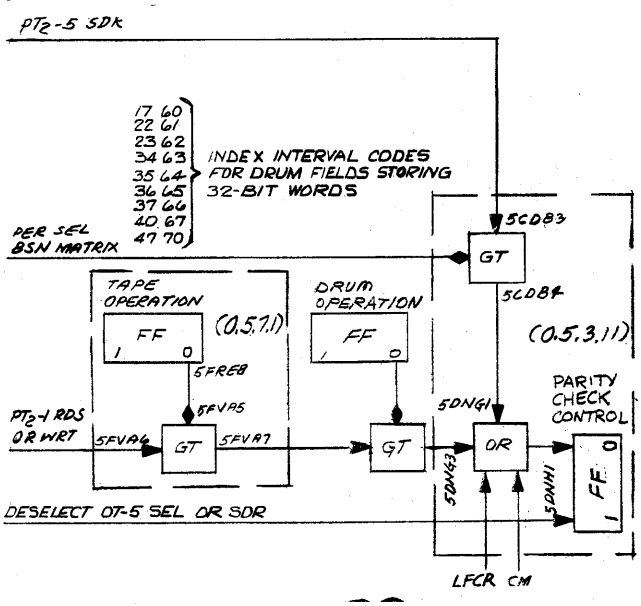


Figure 5–52. Parity Check Control Flip-Flop Circuits

flop is cleared if the selected field has no parity bit. On the other hand, this flip-flop remains set if the field selected has a parity bit associated with its stored words. Consequently, at the end of the *Read* or *Write* instruction, the parity check control flip-flop will contain a 0 if the incoming words are 32 bits long or will contain a 1 if the incoming words are 33 bits long and possess a parity bit.

Assume that a 32-bit word is to be placed in core memory. Since there are no parity bits associated with the incoming words, the parity check control flip-flop will be in the 0 status throughout the break-in cycles. At TP–1 the memory buffer registers, parity count flip-flop, and parity storage flip-flops are cleared. At TP–2 the parity write control flip-flop is set, thereby conditioning its associated gate tube. Simultaneously, the 32-bit word in the IO registers obtained from the IO unit is transferred to the memory buffer registers. Since this word has no parity bit, the parity count and parity storage flip-flops will remain cleared. At BI–3 a parity count is performed by pulsing the parity count circuit. If the word contains an even number of 1's, an even count pulse will be derived which sets the parity count and parity storage flip-flops, thus assigning a parity bit of 1 to the word. If the word contains an odd number of 1's, these two flip-flops will remain cleared, thus assigning a parity bit of 0 to the word.

At BI–7 a pulse is gated through the gate tube conditioned by the 0 side of the parity check control flip-flop to effect a second parity count. At the same time, the parity write control flip-flop is cleared to prevent alteration of the parity storage flip-flop by the second parity count. If the second count agrees with the first, the parity count flip-flop is cleared. In this case, the parity storage flip-flop will contain a 1 if the number of 1's in the word is even or 0 if the number of 1's is odd. Hence the assigned parity bit is correct for the word. If the counts disagree, the opposite will be true. In this case, the parity count flip-flop will contain a 1 and the parity storage flip-flop will contain 1 if the number of 1's in the word, as indicated by the second count is odd, or 0 if the number of 1's is even. Hence, if the two counts agree, a correct parity bit is assigned; if the two counts disagree, an incorrect parity bit is assigned. This insures that an incorrect word will generate an alarm if it is reread from core memory. At TP–11 the status of the parity count flip-flop is examined and if it is found to be set, a memory parity alarm is generated. The disposition of these alarms will be discussed in 6.4.3. of this Chapter. The action of the parity circuit for 32-bit words is summarized in table 5–15.

6.4.2.2 IO Words of 33 Bits

Assume that a 33-bit word is to be transferred from an IO unit to core memory. Since this word contains a parity bit, no first parity count operation need be performed and the parity check control flip-flop will contain a 1. At BI–3 a second parity count operation is performed and a pulse is gated through the gate tube on the 1 side of the parity check control flip-flop to clear the parity write control flip-flop. Because of this clearing action, the parity storage flip-flop will not be affected by the result of the parity count and the original parity bit will be stored in core memory along with the word itself. The parity check is performed by

TABLE 5–15. PARITY CIRCUIT ACTION FOR 32-BIT IO WORDS

TIME	1ST COUNT	2ND COUNT	PARITY COUNT FF	PARITY STORAGE FF	PARITY WRITE CONTROL FF	REMARKS
			For Even-Even Counts			
TP–1			0	0	0	
TP–2			0	0	1	
BI–3	Even		1	1	1	1st parity count
TP–7			1	1	0	
BI–7		Even	0	1	0	2nd parity count
TP–11				No alarm		Parity check
			For Odd-Odd Counts			
TP–1			0	0	0	
TP–2			0	0	1	
BI–3	Odd		0	0	1	1st parity count
TP–7			0	0	0	
BI–7		Odd	0	0	0	2nd parity count
TP–11				No alarm		Parity check
			For Even-Odd Counts			
TP–1			0	0	0	
TP–2			0	0	1	
BI–3	Even		1	1	1	1st parity count
TP–7			1	1	0	
BI–7		Odd	1	1	0	2nd parity count
TP–11				Alarm given		
			For Odd-Even Counts			
TP–1			0	0	0	
TP–2			0	0	1	
BI–3	Odd		0	0	1	1st parity count
TP–7			0	0	0	
BI–7		Even	1	0	0	2nd parity count
TP–11				Alarm given		

the BI–7 pulse after it has been gated by the 1 in the parity check control flip-flop. The action of the parity circuit for 33-bit words is summarized in table 5–16.

6.4.3 Parity Alarms

When a parity alarm occurs as a result of a parity error, the Central Computer System can be forced either to stop or continue its program. Three switches on the maintenance console enable the operator to control the action of the Central Computer System whenever a drum, tape, or memory parity alarm is generated by the parity circuits. When the switch is in the STOP position, the Central Computer System will stop if the parity error associated with the switch occurs; when in the CONTINUE position, the Central Computer System will continue uninterrupted with the program if the parity error occurs. If the Central Computer System is allowed to continue, insertion of the appropriate *Sense* instruction in the program will cause the computer to branch the control of its program. The circuits which accomplish these operations are shown in figure 5–53.

When a memory parity alarm is generated by the parity circuit at TP–10 +0.2 microsecond, the alarm pulse sets the memory parity alarm flip-flop and senses a gate tube conditioned by the MEMORY PARITY switch on the maintenance console. If this switch is in the CONTINUE position, the gate tube will be deconditioned and the alarm pulse will be inhibited. In this case, the Central Computer System continues with its program. However, the 1 in the memory parity alarm flip-flop is an indication that the alarm has occurred. On the other hand, if the switch is in the STOP position, the gate tube will be conditioned and the alarm pulse will be propagated. The resultant signal is then sent to the tape units as a disconnect signal and to the instruction control element where it clears the TPD control flip-flop. This action will stop the time pulse distributor and interrupt the execution of the instruction or break cycle presently in progress. Since the TP–1 pulse

TABLE 5–16. PARITY CIRCUIT ACTION FOR 33-BIT IO WORDS

TIME	1ST COUNT	2ND COUNT	PARITY COUNT FF	PARITY STORAGE FF	PARITY WRITE CONTROL FF	REMARKS
			For Even-Even Counts			
TP–1			0	0	0	
TP–2			1	1	1	
BI–3		Even	0	1	0	2nd parity count
BI–7				No alarm		
			For Odd-Odd Counts			
TP–1			0	0	0	
TP–2			0	0	1	
BI–3		Odd	0	0	0	2nd parity count
BI–7				No alarm		
			For Even-Odd Counts			
TP–1			0	0	0	
TP–2			1	1	1	
BI–3		Odd	1	1	0	2nd parity count
BI–7				Alarm given		
			For Odd-Even Counts			
TP–1			0	0	0	
TP–2			0	0	1	
BI–3		Even	1	0	0	2nd parity count
BI–7				Alarm given		

of the next memory cycle is not generated, the memory buffer register is not cleared and the word which caused the parity error will remain in this register available for examination.

If the Central Computer System is allowed to continue after the occurrence of the memory parity error, the computer may be made to branch its program by a *Sense* (memory parity) instruction. Since the memory parity alarm flip-flop is set, the AND–1 circuit shown in figure 5–53 will be partially conditioned. If a *Sense* instruction with index interval content of octal 15 is executed, this AND circuit will be fully conditioned and its output will enable a gate tube. This gate tube is sensed at PT–9 of the *Sense* instruction and, if conditioned, will generate an output pulse which sets the branch flip-flop in the instruction control element. This causes the Central Computer System to branch its program to the memory location given by the address portion of the *Sense* instruction. It should be noted that if no memory parity error has occurred prior to the execution of the *Sense* instruction, the memory parity alarm flip-flop will be in the zero status, preventing the generation of the alarm stop signal.

If a parity alarm pulse occurs at BI–7, the word giving rise to the alarm must be an incoming 33-bit word from either a tape unit or a drum field. To determine which IO unit originated the word, two gate tubes are simultaneously sensed. One of these is conditioned by the 1 side of the tape operation flip-flop and the other by the 1 side of the drum operation flip-flop. The specific gate tube which is conditioned propagates a pulse which performs two functions: it sets the appropriate parity alarm flip-flop and senses a gate tube conditioned by the associated parity switch on the maintenance console. The subsequent action of the circuit for both types of alarms is identical to that for the memory parity alarm just discussed.

In order to have generated a tape or drum parity alarm pulse, the parity count flip-flop in the parity circuit must have been set. By referring to figure 5–51, it can be seen that the next TP–10 +0.2 microsecond pulse would ordinarily be gated as a memory parity alarm and set the memory parity alarm flip-flop. If this were permitted, any succeeding *Sense* (memory parity) instruction would cause a branch in the program even though a memory parity error has not occurred. To prevent this false indication of a memory parity error when a tape or drum parity error occurs, the tape or drum parity alarm pulse is also used to clear the parity count flip-flop. Since this occurs at BI–7, the next TP–10 +0.2 microsecond pulse will be inhibited by the cleared parity count flip-flop and no false memory parity alarm will be generated.

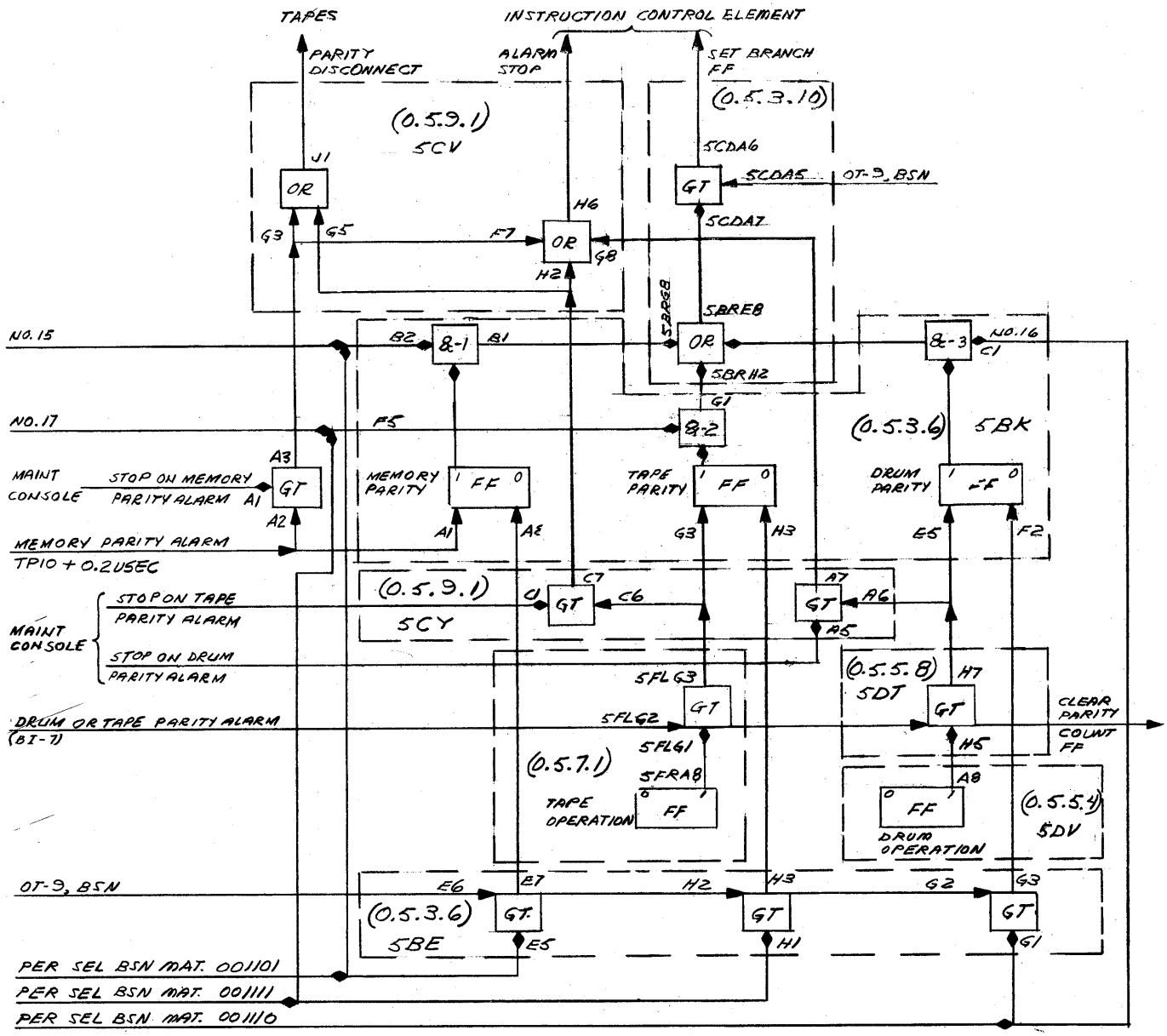


Figure 5–53. Parity Alarm Circuits

PART 6
CORE MEMORY ELEMENT

CHAPTER 1
INTRODUCTION

1.1 GENERAL

The magnetic core memory of the Central Computer System consists of two identical elements, each with a storage capacity of 4096 words of 33 bits each, giving the internal magnetic core memory a total capacity of 8192 words. The nominal memory cycle is 6.0 microseconds. Drum fields are set aside for auxiliary storage to supplement the magnetic core memory; however, the auxiliary storage is considered to be a part of the Drum System and is not described in this Part.

The principal component of each magnetic core memory unit is a three-dimensional array consisting of 147,456 ferrite cores. This array is composed of 36 horizontal planes, each containing 4096 cores in a 64 by 64 array. Only 33 of the planes are used at any one time; the remaining three are spares. The number of planes used (33) is equal to the number of bits (32) in the Central Computer System word, plus the parity bit.

The magnetic core memory, as compared with other types of storage, is compact, requires no periodic regeneration of stored information, and does not lose stored information when the power is turned on and off in the normal sequence. In addition, the core memory is readily adaptable to parallel operation and has a low access time. Read-out from magnetic core memory is, however, totally destructive; consequently, means are provided for automatically writing back into magnetic core memory any word which has been read but is not to be erased. The disadvantage of destructive read-out is more apparent than real, since the re-writing equipment used is the same as for the execution of a store class instruction. An overall view of one of the two memory array units is shown in figure 6–1. A simplified block diagram of a magnetic core memory element is shown in figure 6–2 and is discussed in 1.3 of this Chapter.

1.2 BASIC THEORY OF MAGNETIC MEMORY

Certain ceramic materials, known as ferrites, have magnetic properties. Some of these materials have high values of remanent flux. Such materials can be used, in the form of magnetic cores, to store binary information by denoting one state of remanence as 1 and the other as 0. The ferrites are used in the form of toroidal (ring-shaped) cores which are set to their states of remanence by single-turn windings linking the cores. (See fig. 6–3.) Figure 6–4, part A, is a typical B–H curve, or hysteresis loop, for a ferrite core. The application of a magnetomotive force of I ampere-turns to the ferrite core (which is initially in a flux state represented by point A) causes the core to change to a flux state represented by point C. With the single-turn windings used in the core memory, the magnetomotive force is numerically equal to the current; therefore it is simpler to state that a current I is applied to the core. This simplification is made in the following discussion. The removal of this current I will cause the core to be left in a flux state represented by point E. If a current, −I, is applied to the core which is in the E state, the core flux will be switched to point G. When the current −I is removed, the flux state of the core will again be represented by point A. The transition from one state of remanence to the other is called switching the core.

When a core is in a flux state represented by point E, it is said to be in the 1 state; when the flux state is represented by point A, the core is said to be in the 0 state.

A portion of a memory plane similar to those used in AN/FSQ–7 (XD–1, –2) Combat Direction Centrals internal magnetic core memory is shown in figure 6–5, which also shows the method of winding the cores. Figure 6–6 shows a memory plane before installation and indicates the relative

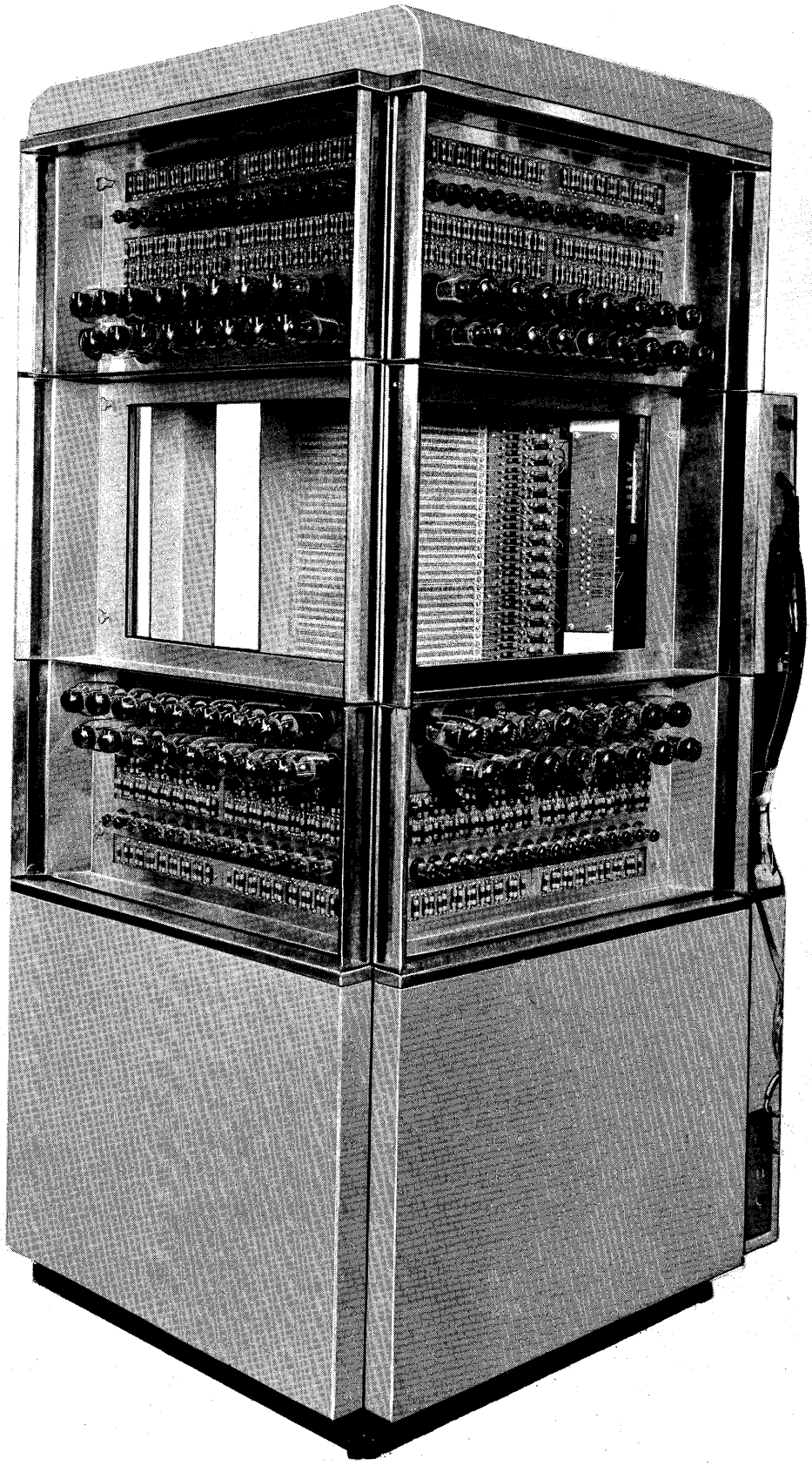


Figure 6–1. The Memory Array Unit, Overall View

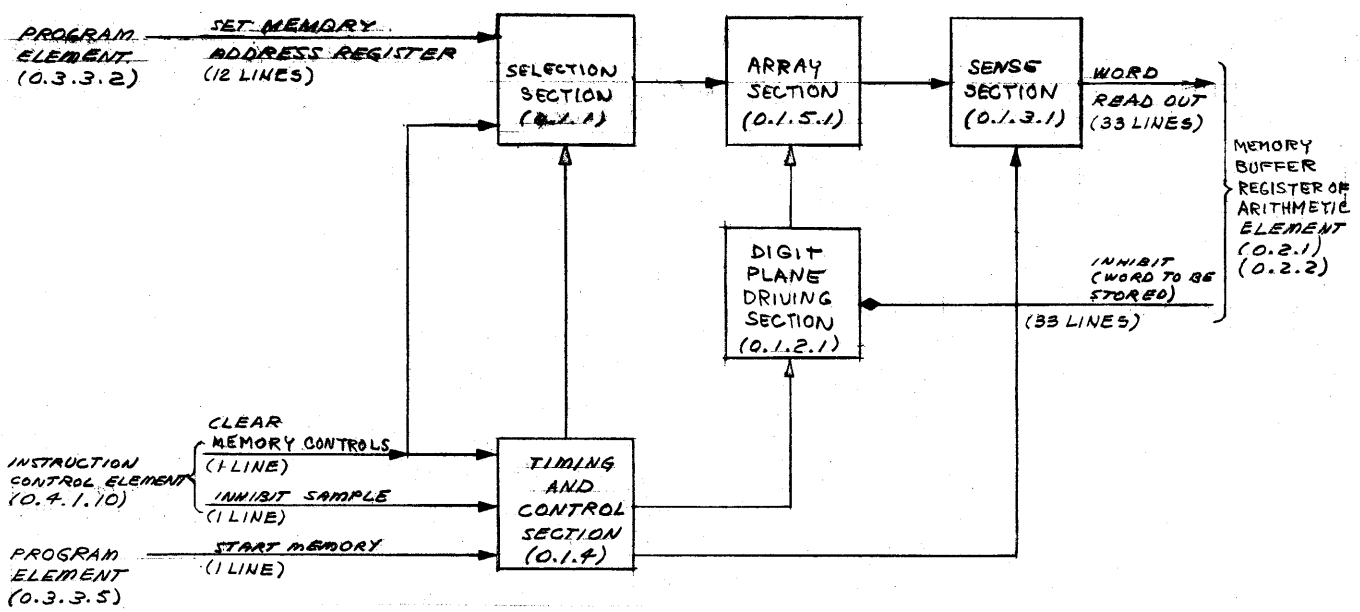


Figure 6–2. Magnetic Core Memory, Simplified Block Diagram

sizes of the ferrite cores and complete plane. The cores are arranged in square arrays so that each core in the array is situated at the junction of two mutually perpendicular, single-turn windings called the selection plane windings. Those in one direction are referred to as X windings and the other group as Y windings. Suppose that a current of I amperes is the minimum current necessary to change a core from the 0 state to the 1 state. Then, if current I/2 is supplied to a single X and a single Y winding, the only core which can change state will be at the intersection of these two windings. This type of memory is known as a coincident current memory and its effectiveness depends primarily upon the rectangularity of the core hysteresis loop. Four windings, called the X, Y, digit-plane, and sense windings, actually pass through each core. (See fig. 6–3.)

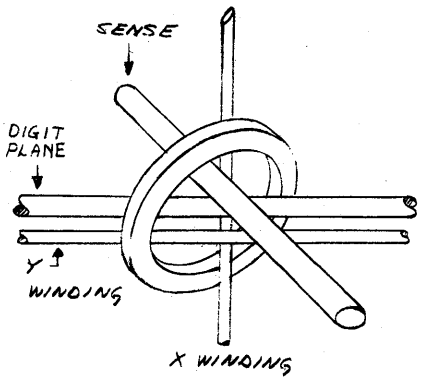
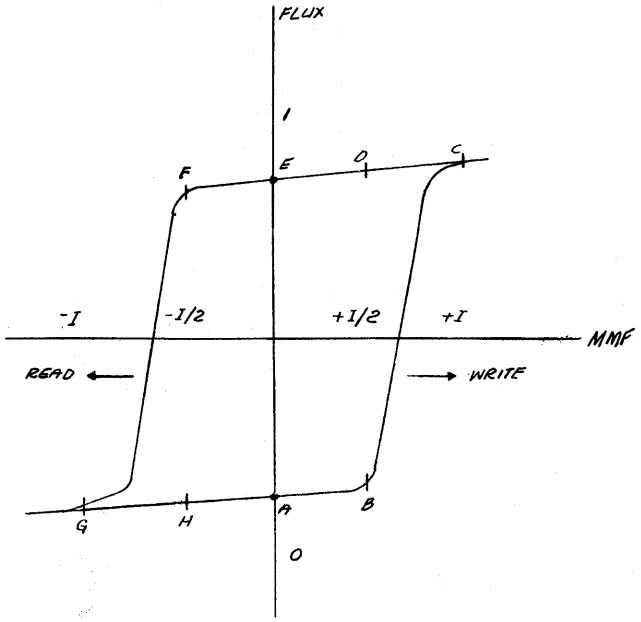


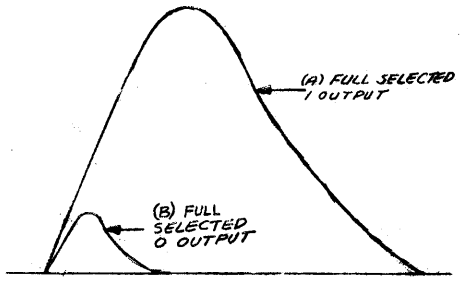
Figure 6–3. Typical Core and Windings

As previously stated, the memory planes (called digit planes) are stacked vertically to form a three-dimensional array. Each Y or X winding is connected in series with the corresponding X or Y windings in the planes above and below it. Digit and sense windings are not connected between the various planes. Figure 6–7 is a simplified diagram of the magnetic memory array. To simplify the illustration, the sense winding is not shown at all, and only one each of the 64 series-connected X and Y windings is shown in part. For the purposes of this basic discussion, the sense and digit-plane windings may be considered to be the same, although it should be remembered that this is not actually the case. As shown in figure 6–7, if read currents of $-I/2$ amperes are applied to one X and one Y winding in the array, a vertical line of cores is selected at the intersection of the planes in which the X and Y windings lie. As a result of these coincident read current pulses, the cores that were previously in the 1 state will be switched to the 0 state, so that at the end of the read current pulses, all of the cores in the selected vertical line will be in the 0 state.

If the selected core in one of the planes had been in the 1 state, point E, figure 6–4, part A, prior to receiving the read current pulses, the core would be switched to the 0 state, traversing the path E–G–A. This represents a relatively large flux change and, consequently, a relatively large voltage will be induced in its sense winding. (See curve A, fig. 6–4, part B.) If the selected core



A. HYSTERESIS LOOP OF A BISTABLE CORE



B. TYPICAL CORE RESPONSE

Figure 6–4. Hysteresis Loop of Bi-Stable Core and Typical Core Response

had been in the 0 state, shown at point A, figure 6–4, part A, the core would be switched but would traverse the path A–G–A. This represents a relatively small flux change, therefore a relatively small voltage would be induced in the sense winding, as shown by curve B, figure 6–4, part A. These differences in magnitude and peak time are large enough to allow discrimination between a core in the 1 state and a core in the 0 state when information is read out of core memory. Further details of the reading and writing processes will be discussed in 1.3 of this Chapter.

1.3 OVERALL OPERATION

A simplified block diagram of the magnetic core memory is shown in figure 6–2, and a more detailed block diagram in figure 6–8. These diagrams should be referred to whenever necessary during this and following paragraphs. Table 6–1 provides a description of the input and output signals to and from the memory element of the Central Computer System.

TABLE 6–1. MEMORY ELEMENT—INPUTS AND OUTPUTS

INPUTS AND OUTPUTS	FROM OR TO
Start memory (pulse to memory pulse distributor)	Program element (0.3.3.5 address register, program counter, or IO address counter)
Inhibit sample (pulse to sample gate generator)	Instruction control command generator 0.4.1.10 (see above)
Set memory address register (pulse to memory address register, 6 channels to left and 6 channels to right)	Program element (0.3.3.2, address register)
Clear memory control (pulses to left and right memory address registers; sample, read, write, and inhibit gate generators; and to memory time pulse distributor)	Instruction control element (memory command generator 0.4.1.10)
Output (pulses from sense section, 33 channels)	Left and right arithmetic elements (memory buffer register 0.2.1.1, 0.2.1.2, 0.2.2.1, 0.2.2.2)
Inhibit (d-c levels to digit-plane driving section 33 channels)	Left and right arithmetic elements (memory buffer register 0.2.1.1, 0.2.1.2, 0.2.2.1, 0.2.2.2)

Operation of the magnetic core memory array involves selection of an address, reading the stored word, and writing the same word (or a different word) into core memory. The reading of a stored word always erases the word from magnetic core memory; consequently, a read operation is always accompanied by the temporary storage of the word in the memory buffer registers (which are physically part of the arithmetic element) and by the rewriting of the word into magnetic core memory. The necessity for always rewriting a word into magnetic core memory, and the fact that all information read from or written into magnetic

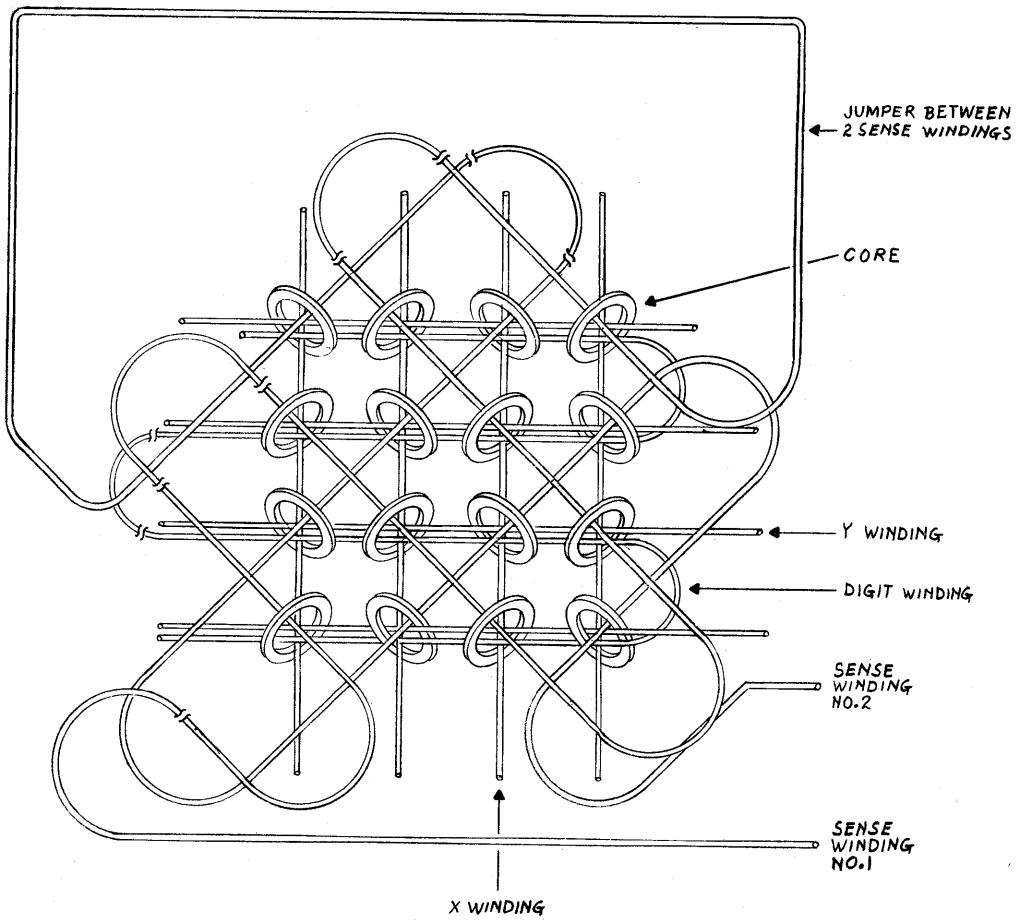


Figure 6–5. Portion of Memory Plane

core memory must pass through the memory buffer registers, has resulted in store and read-out cycles which are very nearly the same. These fundamental operations of the memory element (selection of address, reading, and writing) are under control of the program and instruction control elements which supply the input signals to magnetic core memory. (See fig. 6–2 and table 6–1.) In particular, the program element supplies the start memory pulse to the timing and control section shown in figure 6–2 which, through delay elements and amplifiers, supplies control pulses to the various memory sections at the appropriate times. This section functions, in short, as a memory clock. A typical timing chart is shown in figure 6–9.

Selection of the address is performed by the selection section. The selection section contains the memory address register which is set from either the address register, the program counter, or the IO address counter, all contained in the program element.

The memory address register outputs are fed into two diode-matrix decoders and to the memory gate generators shown in figure 6–8. These decoders and the X and Y selection gates select one out of 64 X drivers and one out of 64 Y drivers. The drivers supply the necessary current to a single X and Y co-ordinate in the three-dimensional array of magnetic cores, as described in 1.2 of this Chapter. The X and Y drivers thus provide the coincident selection currents necessary for performance of the read and write functions. During the read operation, the output signal from the selected core in each plane is picked up by the sense winding and amplified using a four-stage balanced differential amplifier. The amplified signal is fed into a rectifying cathode follower which produces a positive signal output, regardless of the polarity of the signal on the sense winding. The output of the cathode follower is then applied to the suppressor grid of a model A gate tube. A standard pulse from the clear and sample gate generator of the timing and control section shown

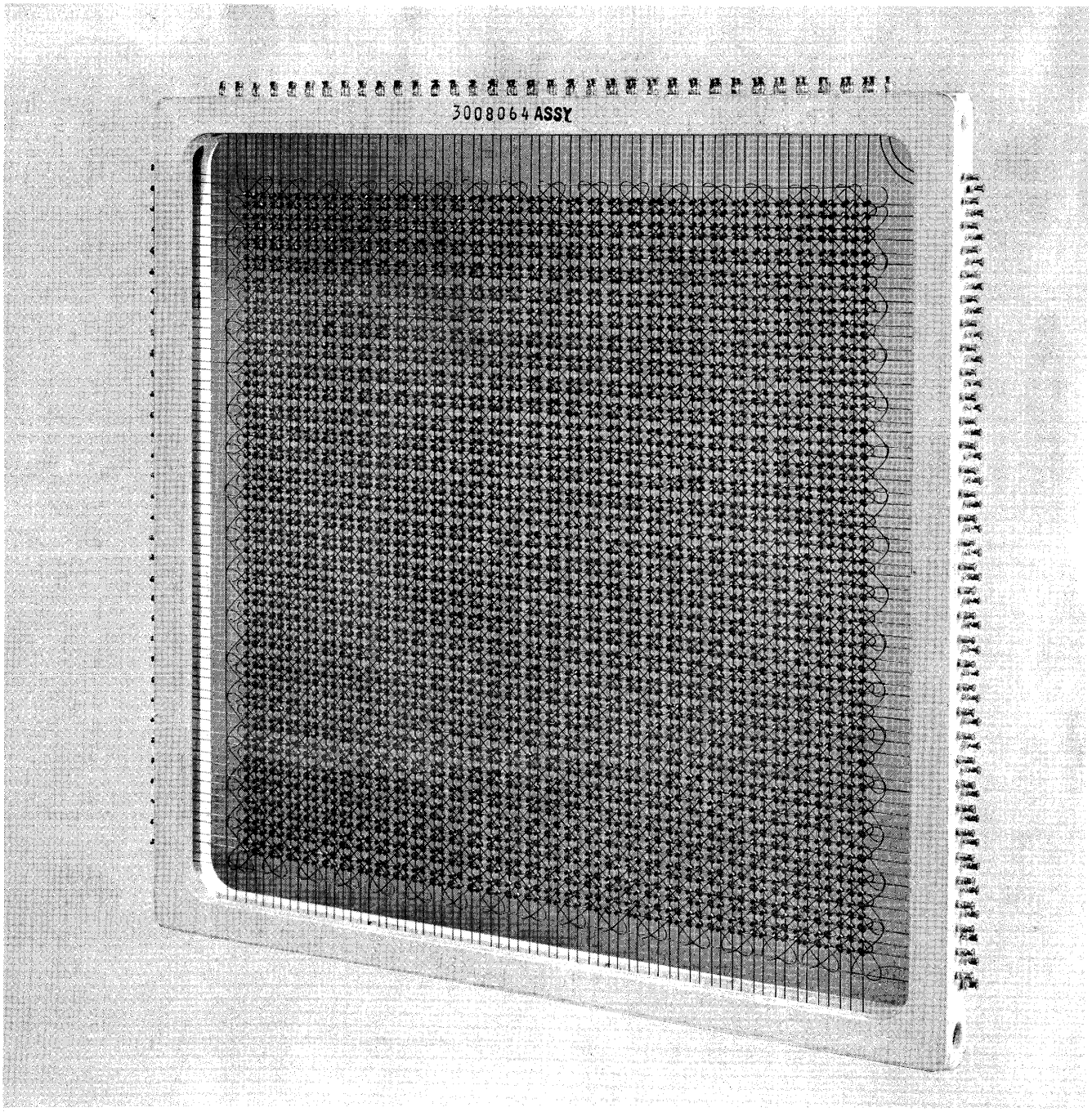


Figure 6–6. Ferrite Core Memory Plane

in figure 6–8 is applied to the control grid of the gate tube and is used to sample the amplified core output signal. If the selected core had contained a 1, the gate tube would generate a pulse which would set the memory buffer register to the 1 state. If the selected core had contained a 0, the gate tube would not generate a standard pulse and the memory buffer register would remain in the cleared or 0 state.

The process of reading information from a selected vertical line of cores leaves that line of

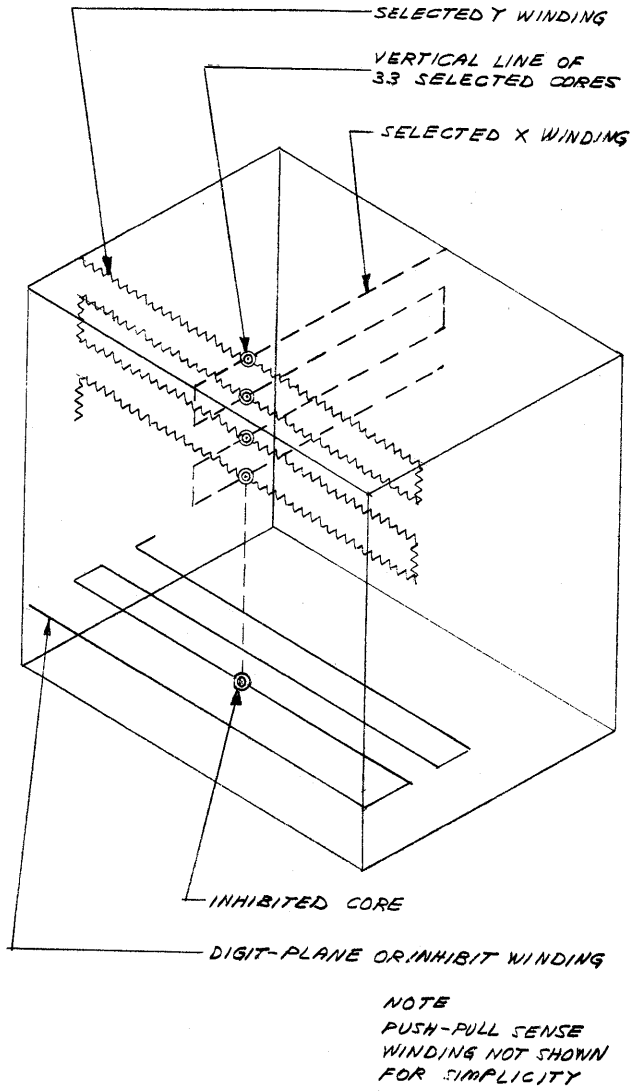


Figure 6–7. Magnetic Memory Array, Simplified Diagram

cores in the 0 state. With the exception of the store cycle, the information that is read out of magnetic core memory is immediately written back into the same location. This is accomplished by the use of the X and Y co-ordinate lines and the digit-plane winding. The digit-plane driver of each plane is controlled by the d-c level from the 0 side of its corresponding memory buffer register. If the memory buffer register is in the 0 state, the d-c level from the 0 side will cause the digit-plane driver to generate a current pulse of −I/2 amperes in the digit-plane winding when it is gated with the inhibit gate generator pulse from the timing and control section. (See fig. 6–8.) If the memory buffer register is in the 1 state, the digit-plane driver will not generate a current pulse. In reading out a 0 in the selected core of a given plane, the memory buffer register remains in the 0 or cleared state and conditions the digit-plane driver so that it can generate a current pulse. During the writing operation, three currents are simultaneously generated, +I/2 ampere pulses on the co-ordinate X and Y lines, and a −I/2 ampere pulse on the digit-plane winding. The sum of these three currents is +I/2 amperes, which will cause the core to traverse the flux path A–B–A shown in figure 6–4, part A, and therefore the core will remain in the 0 state. Thus a 0 is written into the selected core. In reading out a 1 in the selected core of a given plane, the corresponding memory buffer register is set to the 1 state, conditioning the digit-plane driver so that it will not generate a current pulse when it is gated with the inhibit gate generator pulse. As a result, during the writing process only the co-ordinate X and Y lines are pulsed with coincident currents. The sum of these current pulses is +I amperes, which will cause the core to be switched, traversing the flux path A–C–E. (See fig. 6–4, part A.) Thus, a 1 is written into the selected core.

When a word is to be written into core memory during a store class instruction, the operation proceeds as previously described except that an inhibit sample pulse is supplied from the instruction control element to the timing and control section of the core memory at the same time as the start memory pulse. (See fig. 6–9.) The inhibit sample pulse is used to set the sample gate generator only when a new word is to be stored in memory. The zero side of the sample gate generator conditions a gate tube which, when deconditioned, prevents the application of the sample pulse to the sense amplifier gate tubes. Information read out from core memory is effectively erased since the contents of memory are not sampled and therefore are not transferred to the memory buffer registers. The new word to be stored is transferred into the memory buffer register at the same time as the start memory pulse is generated. As in a normal write process, the memory buffer register conditions the digit-plane drivers so that the new word, which is temporarily stored in the memory buffer register, is written into core memory.

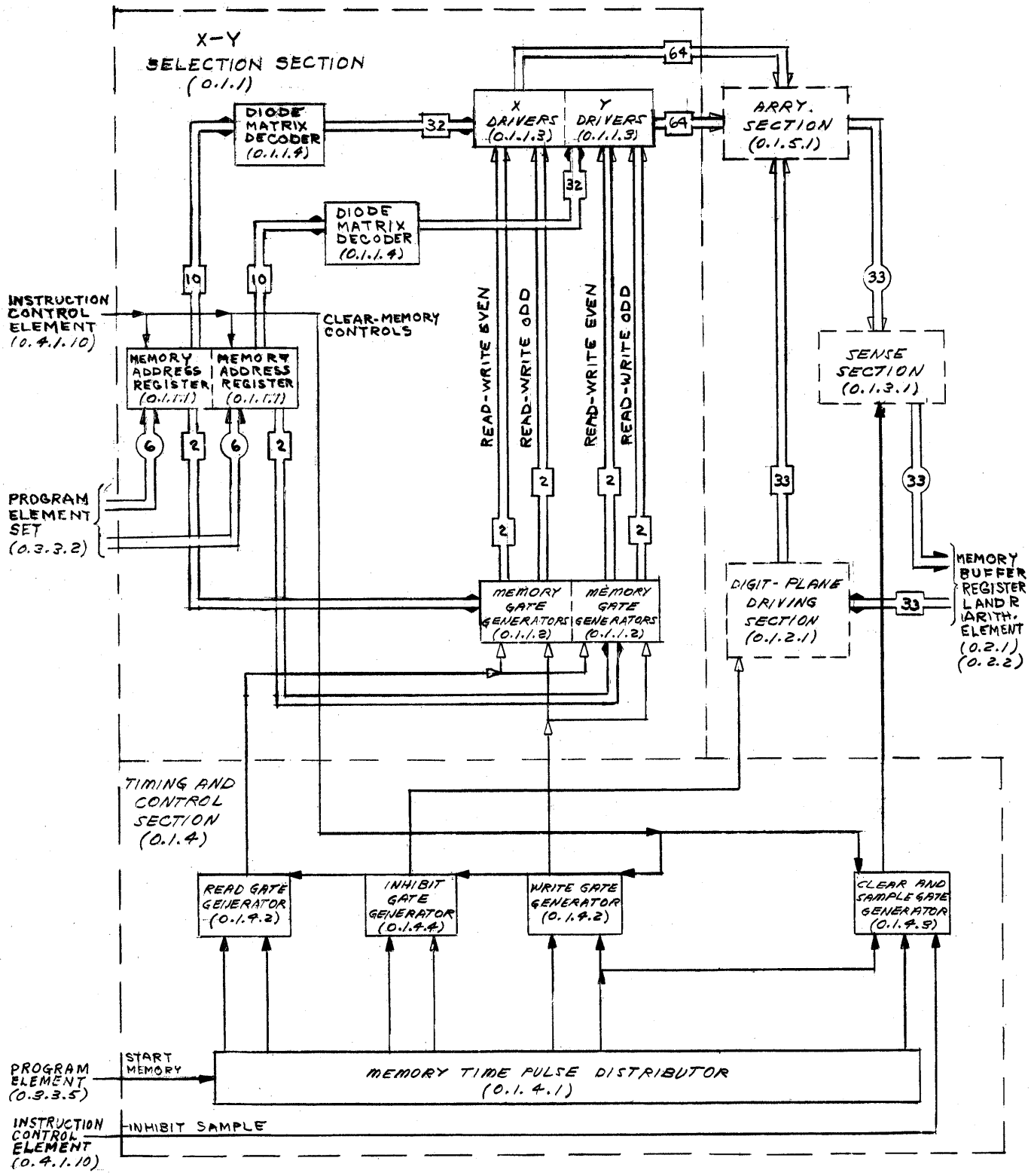


Figure 6–8. Magnetic Core Memory, Block Diagram

1.4 HALF- AND FULLY-SELECTED CORES

In each digit plane, there are 126 other cores in the same row and column with the selected core. These cores are termed half-selected cores. The pulses producing the half-selected cores are termed half-read or half-write, depending on their direction. During the read process, if a half-selected core contains a 0, the core state will approximately follow path A–H–A shown in figure 6–4, part A; if a half-selected core contains a 1, the path followed will be E–F–E. (As discussed in 1.5 of this Chapter, this is true only to a first approximation.) In either case, only a small noise output results, and the information is not destroyed. The noise developed by the read half-select pulse is slightly larger when a core is in the 1 state than when a core is in the 0 state. The remaining 3969 cores in the plane are termed non-selected cores, and are unaffected by the read process.

Since a write process is always immediately preceded by a read process during any selection of a memory address, the selected core will always be in state A when writing starts. To write a 0 into a core, the X and Y windings are pulsed simultaneously with currents of $+I/2$ and the digit-plane winding receives an inhibit pulse of $-I/2$ amperes. The resulting paths of magnetic states on figure 6–4, part A, are then as shown in table 6–2:

TABLE 6–2. WRITING A–0

CORES	PATHS
1 selected	A–B–A
126 half-selected	unaffected
3969 non-selected	A–H–A or E–F–E

Thus, no resulting change of state occurs in any of the 4096 cores.

To write a 1 into a core, the X and Y windings are pulsed simultaneously with currents of $+I/2$, and the digit-plane winding is not pulsed. The resulting paths of magnetic states as shown in figure 6–4, part A, are then as shown in table 6–3:

TABLE 6–3. WRITING A–1

CORES	PATHS
1 selected	A–B–C–D–E
126 half-selected	A–B–A or E–D–E
3969 non-selected	unaffected

Thus, only the selected core changes state.

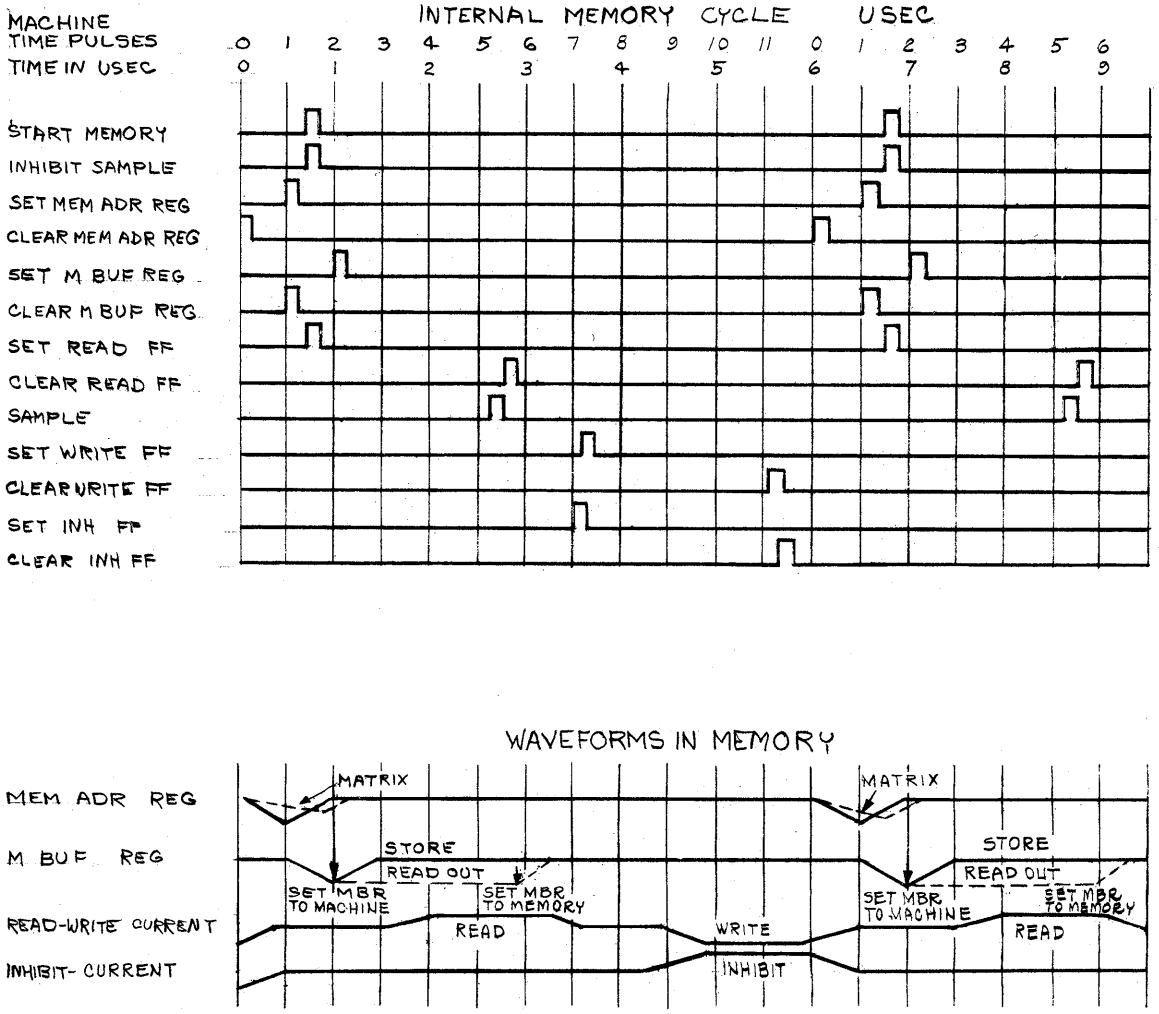


Figure 6–9. Magnetic Core Memory Cycle

CHAPTER 2
SELECTION SECTION

2.1 GENERAL

The selection section consists of two identical circuit groups, one each for the X and Y selection co-ordinates of the memory array. One of these groups (either the X or Y) is shown in simplified form in figure 6–10. Since the two circuit groups are identical, the following discussion will apply to either.

The selection section functions to select the address (or particular vertical line of cores in the array of magnetic cores) from which information is to be read out, or in which information is to be stored. The selection section also functions to supply the two simultaneous currents to the memory array that are necessary to effect the coincident current selection described in 1.2 of this Part. The principal components of each half of the selection section, shown in figure 6–10, are the memory address register, the diode-matrix decoder, the selection gates, and the drivers. Each of these components is discussed briefly below.

2.2 MEMORY ADDRESS REGISTER

Each half (either X or Y) of the memory address register consists of six flip-flops. At the beginning of each memory cycle these flip-flops are cleared by a pulse from the instruction control element. The flip-flops are then set from the program element with the desired address in binary form. The memory address outputs are decoded in two steps. Outputs from both the 1 and 0 sides of five of the flip-flops are supplied to the diode-matrix decoder and the two outputs of the sixth flip-flop are supplied to the selection gate circuits.

The diode-matrix decoder decodes five of the binary digits to select one of 32 pairs of core memory drivers (CMD's). The output of the sixth flip-flop is used to condition one of the two groups of read and write memory gate generators (MGG's) of the X or Y selection gate. Thus, both the CMD's and the memory gate generators are used as logical AND circuits as well as power amplifiers.

2.3 DIODE-MATRIX DECODER

The diode-matrix decoder, shown in figure 6–11, accepts information from both the 1 and 0 sides of five flip-flops in the memory address register. This information is decoded in a diode negative AND circuit with the result that one out of the 32 output lines is selected. The selected line conditions two core memory drivers, one corresponding to an even, and the other to an odd address. One of the two conditioned drivers is selected by the selection gate described in 2.4 of this Chapter. Each X or Y diode-matrix decoder is, in actual construction, located on four separate panels of the array frame of the memory element. This arrangement permits location of the diodes and associated tubes near each other, thus avoiding long leads. The use of the 32-output diode matrix and selection gates results in 224 fewer diodes in the matrix than would normally be used in the conventional 64-output matrix used for the same decoding function.

2.4 SELECTION GATES

The X or Y selection gate, shown in figure 6–10, consists of four memory gate generator (MGG) circuits. The memory gate generator circuits are logical AND circuits, each requiring both a d-c level from the sixth bit (flip-flop) of the memory address register and a read or write gate from the timing control section. Two of these four MGG's are used for writing, and two are used for reading.

If the sixth memory address register flip-flop is cleared, the d-c level from the 0 side will condition the MGG's whose outputs are marked read even and write even on figure 6–10. These MGG's respectively activate the read or write lines of the 32 even core memory drivers when pulsed by a read or write gate from the timing and control section. Similarly, if the memory address register flip-flop is set, the read odd and write odd outputs will be activated upon receipt of a read or

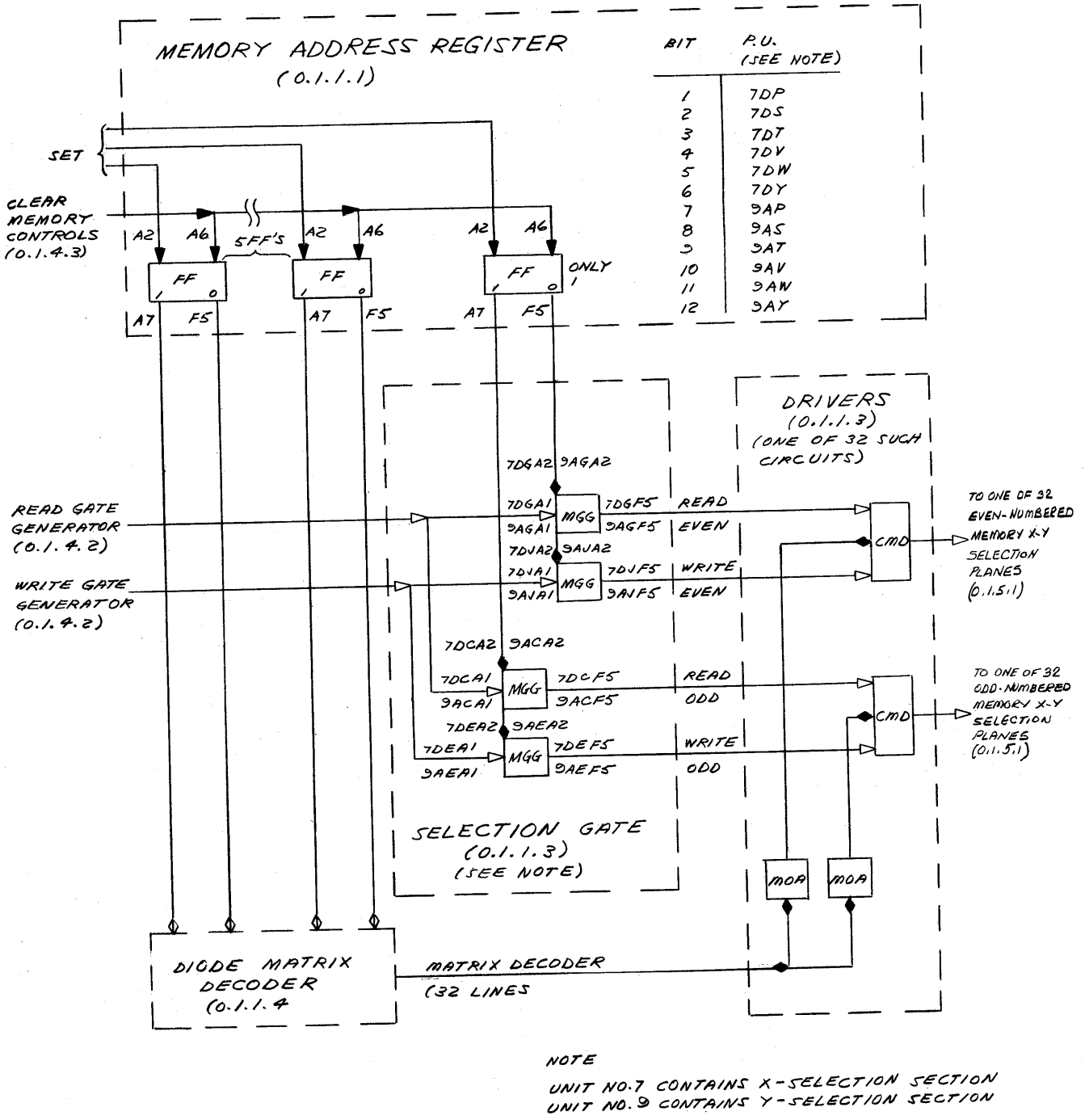


Figure 6–10. X and Y Selection Section

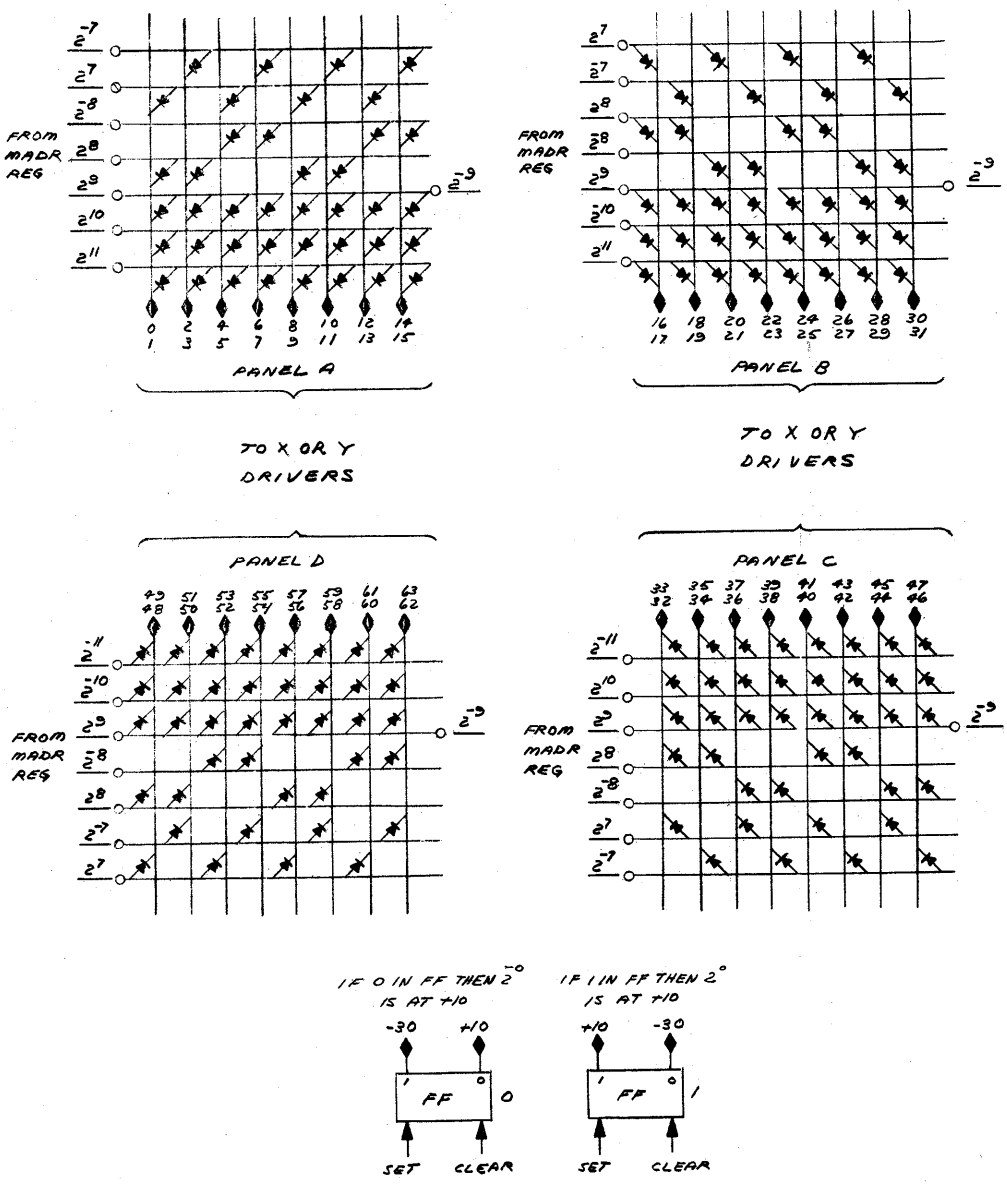


Figure 6–11. X or Y Diode Matrix Decoder, Block Diagram

write gate from the timing and control section. These nonstandard pulse outputs are supplied to the 32 odd core memory drivers.

2.5 DRIVERS

Two of the 64 X or 64 Y core memory drivers (CMD's) are conditioned by the output of the diode-matrix decoder through one of the 32 matrix output amplifiers (MOA's). At the proper time, one of these two CMD's receives a read and a write gate from the selection gate circuits described in 2.3 of this Chapter. The selected CMD in turn supplies a current pulse to an X or a Y co-ordinate plane of the core memory array. This current pulse is sufficient to half-select the cores of the X or Y co-ordinate plane of the array, as described in 1.3 of this Part. In conjunction with the half-selection current in the remaining Y or X co-ordinate plane, a vertical row of cores at the intersection of the two co-ordinate planes is fully-selected. Overall operation of the CMD's is the same during either the read or write function, except that the polarity of the output current in one is reversed with respect to the other. The timing of the read and write gates and their functions are shown in figure 6–9.

CHAPTER 3
DIGIT-PLANE DRIVING SECTION

A block diagram of the digit-plane driving section is shown in figure 6–12. This section consists of 33 functionally identical circuits, one corresponding to each active plane of the array. The digit-plane driving section functions to supply the inhibit current pulses to the memory array digit planes, as described in 1.3 through 1.5 of this Part. Each digit-plane driver receives nonstandard pulses from the inhibit gate generator of the timing and control section, and a d-c level from one of the flip-flops of the memory buffer register of the arithmetic element. (See figs. 6–8 and 6–9.)

The output of a digit-plane driver is a current pulse of —I/2 amperes in amplitude (i.e., a half-selection pulse in the read direction).

The inhibit gate is applied, and an inhibit current pulse is generated only if the memory buffer register flip-flop contains 0 when the inhibit gate is applied. The timing of this function is shown in figure 6–9.

Figure 6–12 shows that each digit-plane driver consists of a gate conditioned by the memory buffer register when the corresponding MBR flip-flop contains a 0. The remaining input to the gate is pulsed by the inhibit gate generator. The output of the gate feeds an OR circuit. The OR circuit output feeds a cathode follower, differential amplifier, and power amplifier, which drive the low-impedance, high-current, digit-plane winding of the magnetic core array.

Since it is desired to make the current in the digit-plane winding proportional to the input voltage, current, or series, feedback is applied around the driver circuit. It may be stated here, without going into circuit details which are treated in another manual (MRD), that this function is accomplished by feeding back the voltage at both ends of a small resistor in series with the digit-plane winding to the differential amplifier. The two voltages thus obtained are subtracted in the differential amplifier, and their difference (which is proportional to digit-plane current) is subtracted from the input pulse. The result is a current waveform that resembles the input voltage waveform.

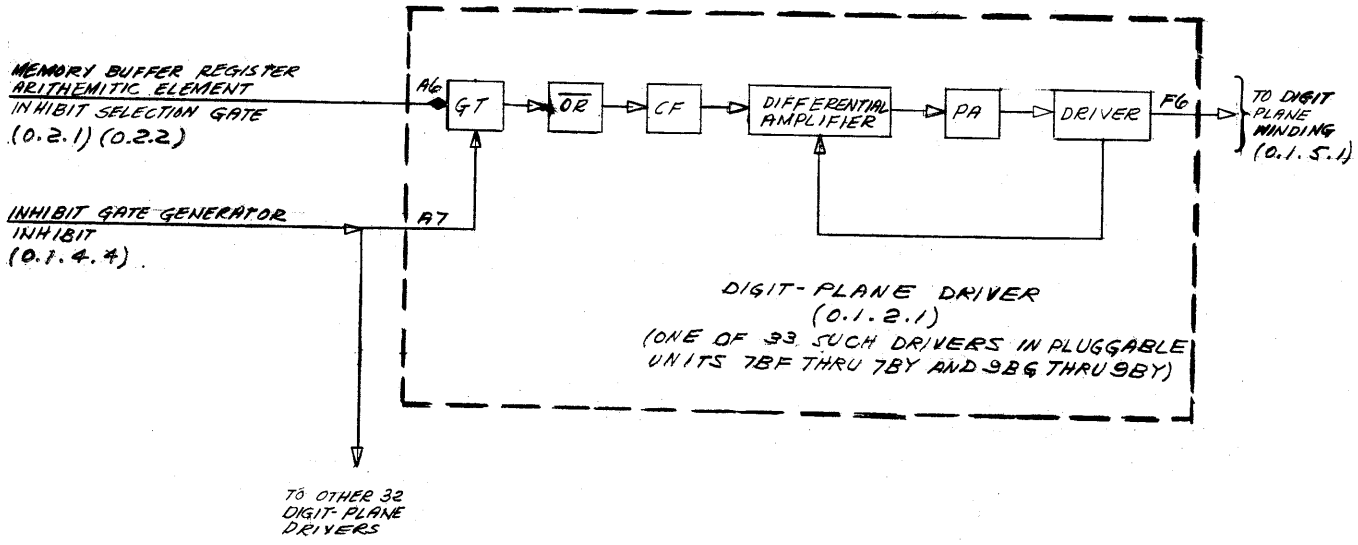


Figure 6–12. Digit Plane Driving Section, Block Diagram

CHAPTER 4
SENSE SECTION

The sense section consists of 33 amplifying and sampling channels, one channel for each digit plane of the core memory array. Figure 6–13 is a block diagram of the sense section. The design of each channel of the section is a result of the necessity for minimizing and canceling noise pulses. These noise pulses arise from half-selected core outputs in the memory array and from capacitive coupling between driven windings (X, Y, or digit-plane), and the sense winding.

When one core in a plane is selected, the remaining 126 cores in the same row and column with the selected core are half-selected. (Refer to 1.4 of this Part.) Although the outputs from these half-selected cores are individually small, they can add algebraically, and may produce a difference signal with an amplitude approaching that of a fully-selected core. At best, the addition of the outputs from a number of half-selected cores will result in a reduced one-to-zero ratio in the signal applied to the sense amplifiers. In order to minimize this effect, the memory element employs a bi-polar sense winding. Half of the cores are of the same polarity with respect to the sense winding, and half are of the opposite polarity. The winding is constructed in two series-connected sections, wound as shown in figure 6–5. The winding is connected to the inputs of a balanced differential sense amplifier. The bi-polar nature of the sense winding tends to cancel the noise from the half-selected cores. It is possible to obtain a condition where all of the half-selected cores of one polarity contain a 1, and all of those of the opposite polarity contain a 0. This would result in a minimum amount of cancellation. Thus the amount of noise cancellation achieved in any actual case depends on the information stored in the cores at that time and, therefore, on the particular program being executed.

Because the sense winding is balanced and ungrounded, noise voltages in the sense winding resulting from capacity coupling to driven windings raise or lower the sense winding potential as a whole, thus producing a common mode signal at the input to the differential amplifier. The differential amplifier is insensitive to such inputs and produces no output.

The final (cathode follower) stage of the amplifier rectifies, in effect, the signals from the differential amplifier, producing a positive output for pulses of either polarity from the sense winding. Because of this action, the polarity of the sense winding connection to the sense amplifier is unimportant. The output of the cathode follower stage is applied to a gate circuit, which improves the one-to-zero discrimination by sampling, and provides the standard pulse required by the memory buffer register. The gate produces an output only when it receives a standard pulse from the clear and sample generator as well as the non-standard sense amplifier output. The time relationship between the sense amplifier output and the sample pulse is adjusted for optimum signal-to-noise ratio. This procedure is effective because the noise pulses and the desired signal pulses are at maximum amplitude at different times.

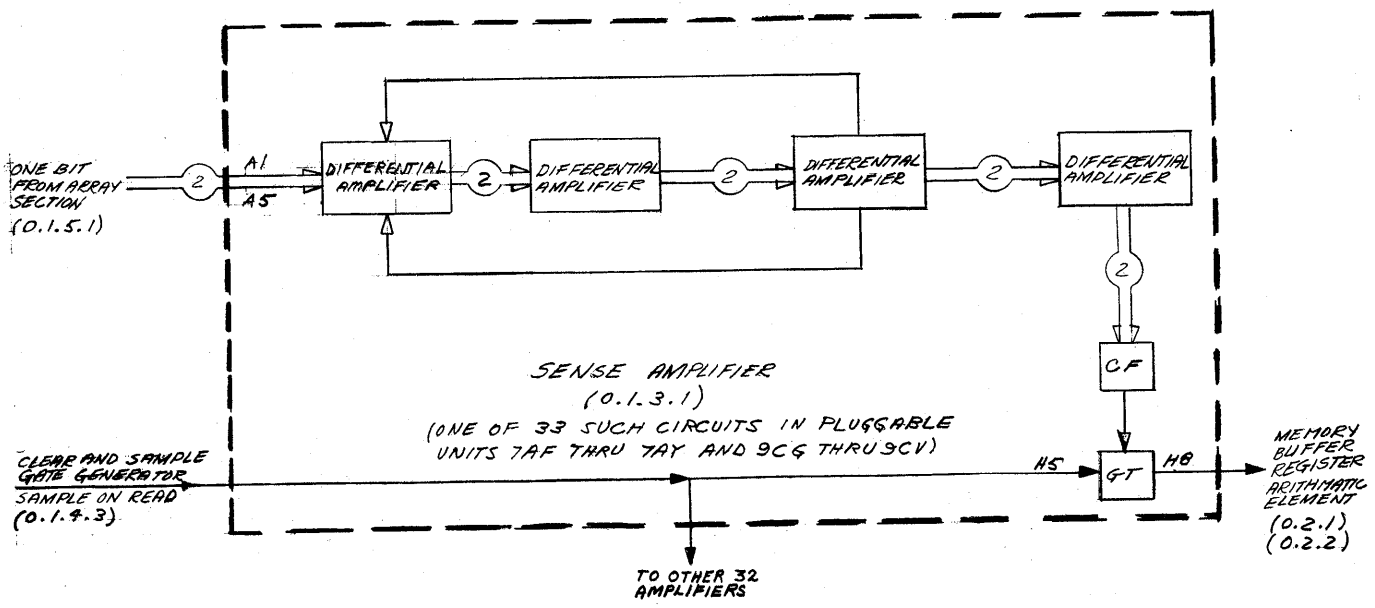


Figure 6–13. Sense Section, Block Diagram

CHAPTER 5
TIMING AND CONTROL SECTION

5.1 GENERAL

The timing and control section receives the start memory and inhibit sample pulses from the program and instruction control elements, respectively, and generates the gates necessary for operation of all parts of magnetic core memory in proper sequence. A simplified block diagram of the timing and control section is contained in the overall block diagram. (See fig. 6–8.) A more detailed block diagram is shown in figure 6–14. As shown in figure 6–14, the start memory pulse is applied to a chain of delay lines. Each of the pulses required in other sections of core memory is generated by a flip-flop pulsed by appropriate outputs of the delay chain. As far as core memory is concerned, only two types of cycles can occur. These are distinguished from each other in that the inhibit sample and set memory buffer pulses, shown in figure 6–9, occur in only one of them. The cycle in which the inhibit sample and set memory buffer pulses occur is used when a store class instruction is executed. The other type of cycle is used when a stored word is to be read out of magnetic core memory into the memory buffer register. In this case, the inhibit sample and set memory buffer pulses do not occur.

5.2 MEMORY PULSE DISTRIBUTOR

The memory pulse distrbutor consists of delay units, shown in figure 6–14, which time the pulses to the gate generators. Power amplifiers make up for losses in the delay units. The relative pulse times of the signals to the various gate circuits may be read from figure 6–14 by noting the value of delay between the various outputs. The time relationships are also given in the timing chart of figure 6–9.

5.3 READ AND WRITE GATE GENERATOR

The read portion of the read and write gate generator receives the set read pulse from the memory pulse distributor at the same time as the start memory pulse; the clear read pulse is received approximately 2 microseconds later. Separate flip-flop gate generators are provided for the X and Y read gates. The nonstandard output pulses from the read gate generators are supplied to the X and Y selection gates respectively. (Refer to 2.4 of this Part.)

The write gate generator flip-flops receive a set pulse approximately 2.8 microseconds after the start memory pulse. The write gate generator receives a clear pulse approximately 2 microseconds later (or 4.8 microseconds after the start memory pulse). The output from the separate X and Y write gate generator flip-flops is supplied to the X and Y selection gates respectively. (Refer to 2.4 of this Part.)

5.4 CLEAR AND SAMPLE GATE GENERATOR

The clear portion of the clear and sample gate generator amplifies the clear memory controls pulse received from the instruction control element without affecting the logic. The distribution of the clear memory controls pulse is shown in the overall block diagram. (See fig. 6–8.) The sample gate generator is shown in figure 6–14.

The sample gate generator differs from the gate generators previously described. As shown in figure 6–14, a circuit gate is provided which is conditioned by the 0 side of a flip-flop. In this condition, the gate passes the sample pulse to the sense section approximately 2.0 microseconds after the start memory pulse. If a word is to be stored in core memory, the inhibit sample pulse from the instruction control element is supplied to the sample gate generator flip-flop. The 0 side of the flip-flop is then down, the gate is not conditioned, and no sample pulse is supplied to the sense section. (Refer to Chapter 4 of this Part.)

5.5 INHIBIT GATE GENERATOR

The inhibit gate generator, shown in figure 6–14, is a flip-flop gate generator similar to the read and write gate generators previously described. (Refer to 5.3 of this Chapter.) The set inhibit pulse is supplied from the memory pulse distributor to this circuit approximately 2.7 microseconds after the start memory pulse. The clear inhibit pulse from the same source arrives approximately 2.2 microseconds later. The nonstandard inhibit pulse output is supplied to the digit-plane drivers. (Refer to Chapter 3 of this Part.)

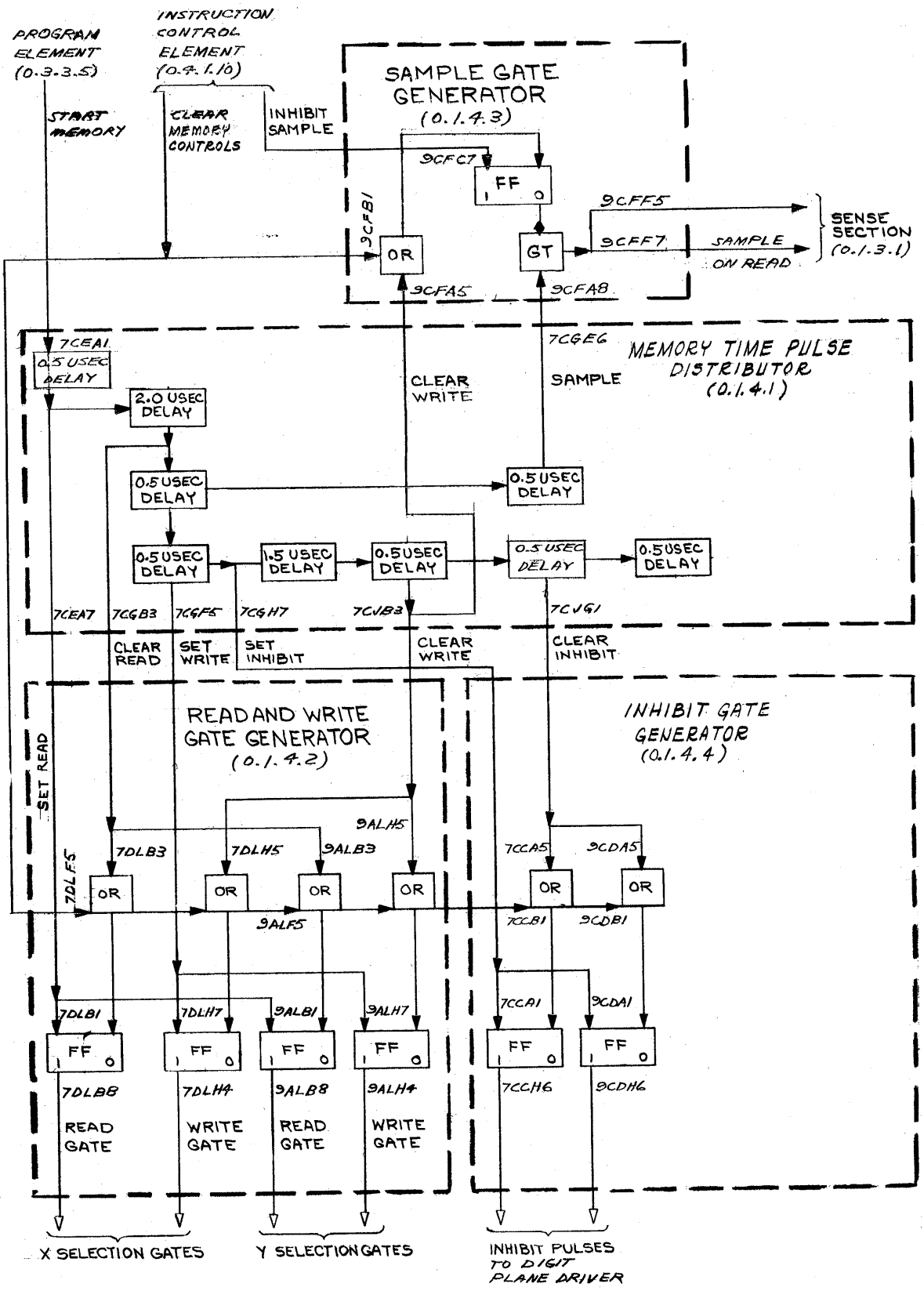


Figure 6–14. Timing and Control Section, Block Diagram

CHAPTER 6
ARRAY SECTION

6.1 GENERAL

The functional description of the magnetic core memory array was discussed in 1.2 and 1.3 of this Part, to which reference should be made. This section deals only with the unusual characteristics of the core memory array.

6.2 CIRCUIT DESCRIPTION

Each of the X and Y selection windings is driven through a fuse, shown in figure 6–15, which protects the array winding against overload. Each selection winding is connected in series with two 10-ohm resistors in parallel which, together with the winding, provides proper loading for the core memory driver. (Refer to 2.5 of Chapter 2.) These components are located on resistor and fuse boards in the array section. Each digit-plane driver is also connected to its digit winding through a fuse. The digit winding is connected in series with two 10-ohm resistors in parallel and in addition, is shunted to ground with a 100-ohm resistor. (See fig. 6–15.) The sense winding does not have any corresponding resistors in the array section. More complete information on the circuit connections for the core memory array is provided in Engineering Data drawing 3008620, sheets 1 through 5.

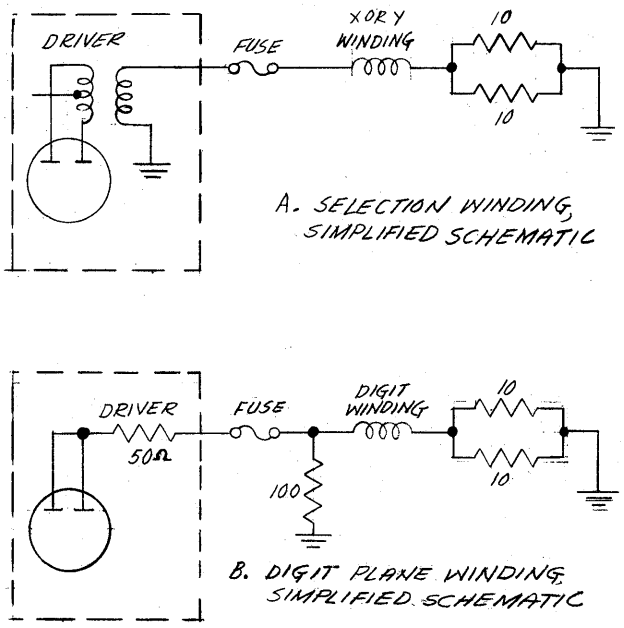


Figure 6–15. Core Memory Array Circuits, Simplified Diagram

6.3 MECHANICAL CONSTRUCTION AND ARRANGEMENT

The nonstandard physical arrangement of the magnetic memory array unit results from two principal causes: the necessity of avoiding temperature extremes, and the necessity of short electrical leads. These considerations have resulted in the memory array housing shown in figure 6–1. The left side of figure 6–1 is actually the front face of the memory array unit.

The 36 memory planes used in this array unit are assembled in a vertical stack, which is located in the horizontal center of the array unit. These planes are wired together so that each of the 64 X or 64 Y co-ordinate lines of each plane is connected in series with the corresponding co-ordinate line of each of the other planes. As a result, each of the 64 X or Y driver lines link 64 x 36, or 2304 cores.

The 64 X drivers are mounted in four driver panels, two of which are shown in the front view of figure 6–1. The remaining two driver panels are located on the rear face of the array unit. The 64 Y drivers are similarly mounted in four driver panels located in the left and right faces of the array unit. The eight driver panels used in this memory array are identical; consequently, they are interchangeable.

Each of the driver panels contain 16 driving circuits (core memory drivers), 16 diode-matrix output amplifiers (matrix output amplifiers), and one-quarter of the diode circuits of one of the diode matrices. Electrical connection between a driver panel and the array unit is accomplished

by the use of three multi-contact connectors. The outputs of any of the driver panels taken in sequence will yield consecutive addresses. To keep the leads from the driver panels to the array as short and even as possible, the consecutive outputs of a given driver panel are connected to every fourth X or Y co-ordinate line winding.

To properly cool the tubes in the array unit, conditioned air is forced into the unit through a duct opening in the bottom of the unit. Tube cooling is accomplished by having the conditioned air escape from the unit through air openings at the base of each tube. The stack of core planes is cooled by blowing room temperature air into the base of the stack. This air escapes through the top of the array unit.

Further information on physical construction can be obtained from Engineering Data drawing 3008620, sheets 1 through 5.

PART 7

MAINTENANCE CONTROL ELEMENT

CHAPTER 1

INTRODUCTION

To prevent malfunction of the AN/FSQ-7 Combat Direction Central during critical periods of operation, the condition of the circuits and components is checked by test programs which indicate the overall status of the system. If equipment failures are anticipated during this check, the test program is used to localize and isolate the faulty components. This test program is controlled by the maintenance console, which also loads these test programs into the Central Computer System.

1.1 FUNCTION AND USE

The primary purpose of the maintenance control element is to minimize system malfunction, first, by detecting faulty operation; second, by indicating the faulty components to be replaced, thereby maintaining the system in proper operating condition. In addition to loading the Central Computer System with test programs, the maintenance control element also serves the indispensable purpose of loading it with the initial operating programs. These programs are loaded into the Central Computer System by pushbuttons on the maintenance console, an integral part of the maintenance control element, which control the source of the input program. The selected source may be either the card reader, test memory, or the auxiliary memory drums.

The techniques involved in detection and correction of faulty machine operation fall into two categories. The first assumes that the machine is running satisfactorily under normal operating conditions. By subjecting these circuits to temporary test conditions more severe than those encountered in normal operation, prediction can be made as to the troubles likely to occur. These test conditions not only show where future troubles may develop, but also indicate how much operating time remains before the components deteriorate to a point where trouble necessarily develops. This first category of testing is in the nature of a preventive maintenance program, performed at regular intervals while the machine is in proper operating condition, to forestall possible breakdown of the Central Computer System during critical operating periods.

If the preventive maintenance program indicates the likelihood of a failure, the second category of maintenance is employed. This is in the nature of a corrective program which localizes and isolates the faulty unit or component and indicates, in a form intelligible to the operator, where the fault is located, permitting this faulty unit or component to be replaced immediately.

1.2 LOADING THE CENTRAL COMPUTER SYSTEM WITH INITIAL PROGRAMS

Two types of programs may be loaded into the Central Computer System by means of pushbuttons on the maintenance console, one being the test program used for either preventive or corrective maintenance and the other, the initial operational program by which the Central Computer System solves the problems presented to it. During the early stages of the computer test, the instructions for testing the computer are held in test memory and are then fed into the computer for execution. Once core memory has been tested and found to be reliable, test programs are usually fed into the internal core memory and stored there.

These complex test programs may be initially stored on cards or on the auxiliary memory drums, from where they are loaded into core memory at the appropriate time.

The initial loading program (considered to be part of the larger type programs mentioned above) for Central Computer System operations may be stored either on cards or on auxiliary memory drums before it is loaded into the Central Computer System. This initial program is transferred into core memory and thereafter specifies the sequence of actions to be followed by the Central Computer System. For both test and operational programs, the source of the program is selected by depressing either the LOAD FROM CARD READER, the START FROM TEST MEMORY, or the LOAD FROM AUXILIARY MEMORY DRUMS pushbuttons.

1.3 PREVENTIVE MAINTENANCE

The preventive maintenance reliability program checks the AN/FSQ–7 Combat Direction Central under operating conditions more severe than those normally encountered. These severe test conditions are set up using the marginal checking system and certain appropriate reliability programs. The AN/FSQ–7 Combat Direction Central is divided into eight equipment groups, four of which are contained in the Central Computer System, and four in the associated IO systems. However, only one of these groups may be tested at any given time. Five of the operating voltages present in the AN/FSQ–7 Combat Direction Central are used for marginal checking purposes but only one may be used at any given time for these purposes. The operating voltage being used can be varied to some fixed limit above or below the normal operating potential. The amount of the variation is termed the excursion. The excursion is applied to the selected group of circuits while a program, called a reliability program, is run through these circuits. This particular operating voltage, as well as the magnitude of the excursion, is chosen to accentuate the effect of any undesirable component change in the tested circuits. This check is then repeated on other groups, until all desired parts of the system have undergone checking. The various circuits undergoing tests are checked in groups called margin groups. By definition, margin groups contain circuits which have the same operating potentials applied to their circuit elements and have approximately the same margin. By definition, a margin is the excursion magnitude at the time a circuit fails to operate during the reliability program. When unreliable or faulty operation is detected by means of this check, the selected margin group may be broken down into smaller groups, termed logic groups, for easier localization of the particular trouble. Further localization and isolation of the faulty components is accomplished by diagnostic programs.

Checking for low operating margins (low pulse output, faulty timing, etc.) is a daily preventive maintenance procedure. For the complete routine, several different programs are used, each designed to exercise a different portion of the computer. When one portion has passed the test satisfactorily, it is safe to use that portion in testing some other part of the system. The programs for these tests are designed with the greatest possible number of check commands so that, after any error occurs, no more than a few commands will be executed before the Central Computer System stops. The initial preventive maintenance program steps; i.e., the first instructions of such a program, check only a few of the components or units in an equipment group. These initially tested components or units are the ones which are to be used continuously in subsequently carrying out more complex and comprehensive test programs.

The four equipment group divisions of the Central Computer System are shown in figure 7–1. The test program starts by testing internal core memory from test memory. This is done so that core memory can be loaded as soon as possible with the more complex programs necessary to test all remaining portions of the AN/FSQ–7 Combat Direction Central equipment, since the capacity of test memory is limited. If core memory is found to be reliable, then the memory buffer register, the operation register in the instruction control unit, and the address register in the program element are tested. If these are found to be reliable, the next test uses these registers in testing another register, such as the index register, by executing a *Reset Index Indicator (XIN)* instruction. If these registers are reliable, an *Add Index Register (ADX)* and a *Store Address (STA)* instruction might be executed which would transfer the information from the index registers to the address register; then to the right A register, thus testing this new register; and from there to core memory via the memory buffer register. Each subsequent instruction in the reliability program is chosen so that the registers previously tested are used again while one additional register is tested during the execution of the new instruction. Therefore, the reliability program starts by testing some of the basic

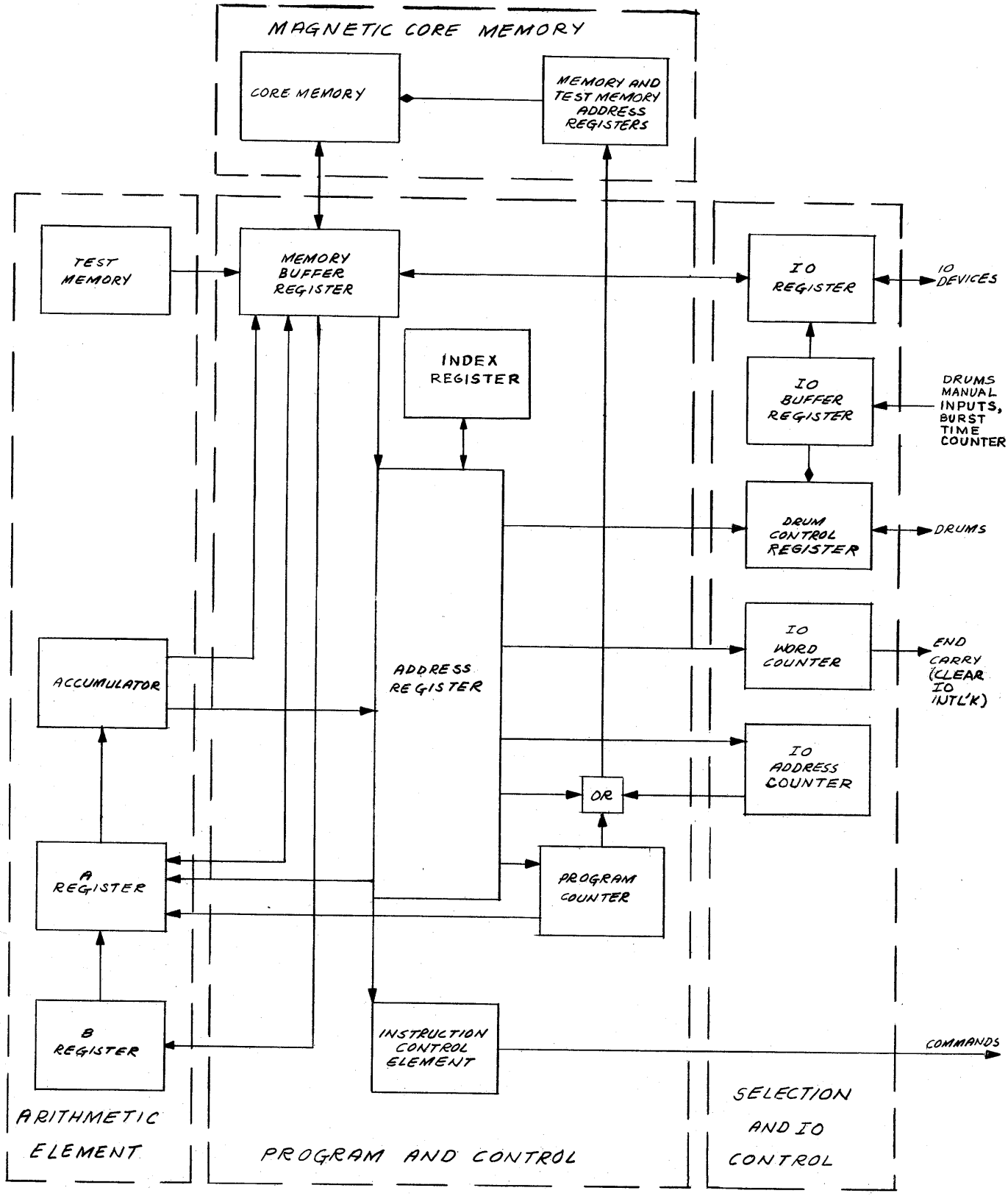


Figure 7-1. Marginal Checking Breakdown of Central Computer System

registers, later using these tested registers in subsequent program steps to test the reliability of additional registers. The instructions used range from the simplest, such as the *Program Stop (HLT)* and *Branch and Index (BPX)*, which use and test a minimum of the basic registers, to the most difficult, such as a *Multiply (MUL)* or *Divide (DVD)* instruction, both of which use a large portion of the arithmetic element registers. The programmed instructions are arranged in a sequence such that no new registers, except the initial ones, are tested without first testing those registers which transfer information to the new registers. In this system, each register tested is checked and rechecked a number of times, making sure that its operation is satisfactory. When the reliability program is run in conjunction with the voltage variations imposed by the marginal checking system, circuit aging and deterioration, which may later cause faulty operation, are detected. From this information, charts may be drawn as a schedule for timely component replacement.

In addition to the reliability program, a COMPLEMENT pushbutton complements the flip-flops, reversing their present status. This test may be run repeatedly at the high speed of the Central Computer System to make sure that the flip-flops are operating properly. A loudspeaker and amplifier are also provided and may be connected to bit 1 or bit 2 of either the left or right accumulator register.

1.4 CORRECTIVE MAINTENANCE

Two types of failures that occur during machine operation, which cannot be foreseen or detected by the deterioration tests of the marginal checking and reliability programs, are sudden or catastrophic failures and certain intermittent breakdowns. Sudden failures include open- or short-circuited tube elements, faulty capacitors, etc., while intermittent failures include such faults as momentarily open- or short-circuited tube elements, defective capacitors, or even rosin solder joints which may break momentarily and then re-establish contact. A poor solder joint with a long time-duration break can be located by the reliability program.

The diagnostic program is written specifically for the testing of separate portions of the equipment only, rather than for the entire equipment, and is used in conjunction with special, built-in facilities for rapid analysis and localization of failures. The initial writing of the diagnostic program is accomplished in a manner similar to that of the reliability program. When faulty operation is detected, the portion of the program where errors occur can be run again and repeated continuously by means of the cyclic program counter and control. The cyclic program control tests specific, faulty circuits at the normal high-operating speeds of the Central Computer System. This control may be used to start and stop the Central Computer System at any steps in the program, to repeat a group of steps continuously, to repeat one step a number of times, or to pause so that the various indicators may be examined. Then, when the CONTINUE pushbutton is depressed, the regular program continues from the point at which it was interrupted. In general, the cyclic program control permits an operator to set up complicated conditions identical with or equivalent to those of normal operations and, at the same time, obtains an outward simplicity which makes for relatively simple analysis. As an auxiliary aid to the cyclic program counter, various pushbuttons on the maintenance console can be used for diagnosing trouble in specific circuits. These pushbuttons are listed in table 7–1.

TABLE 7–1. AUXILIARY PUSHBUTTONS USED FOR SERVICING

PUSHBUTTON	FUNCTION
PROGRAM CONTINUE	Starts the program at the point of interruption
COMPLEMENT	Reverses the state of certain selected flip-flops
MEMORY CYCLE	Provides for running one memory cycle at a time
INSTRUCTION STEP	Provides for running one instruction at a time
START FROM TEST MEMORY	Allows the program to start from test memory and then either continue from there or branch to some other storage device

Certain portions of the Central Computer System, such as control, will be very difficult (if not impossible) to reach by diagnostic programs. In such cases, the program only attempts to localize the trouble to two or three of the control flip-flops. The trouble is then isolated by an oscilloscope and a meter. Neon lights, connected to one side of the flip-flops in all registers, are mounted on the front panel of the maintenance console to indicate the existing status of the flip-flops.

CHAPTER 2
OPERATIONAL ANALYSIS

The maintenance console comprises a number of separate and distinct panels each of which is devoted to a maintenance function such as operation, service, test switches, indicator lights, and alarms. Even though the specific functions performed by the individual controls, lights, and alarms are not related to each other, their integrated action is such that an overall, comprehensive test of the system circuits can be performed. The maintenance console comprises three distinct panels. (See fig. 7–2.) The left-wing panel is the marginal-checking main control panel. The center-wing panel is the Central Computer System control panel which contains all the servicing and operating controls as well as the control indicator lights for the Central Computer System. The right-wing panel consists of the indicator neons. These are connected to one side of the flip-flops of the Central Computer System registers and counters, except for those of the time pulse distributor, which are located in the center-wing control panel.

The circuit analysis in this Chapter groups the functions of the maintenance console into three divisions: the marginal checking control system, the manual operations control system, and the indicator lights and alarms. In addition, the auxiliary equipment used in conjunction with the maintenance console is described in the last paragraph of this Section.

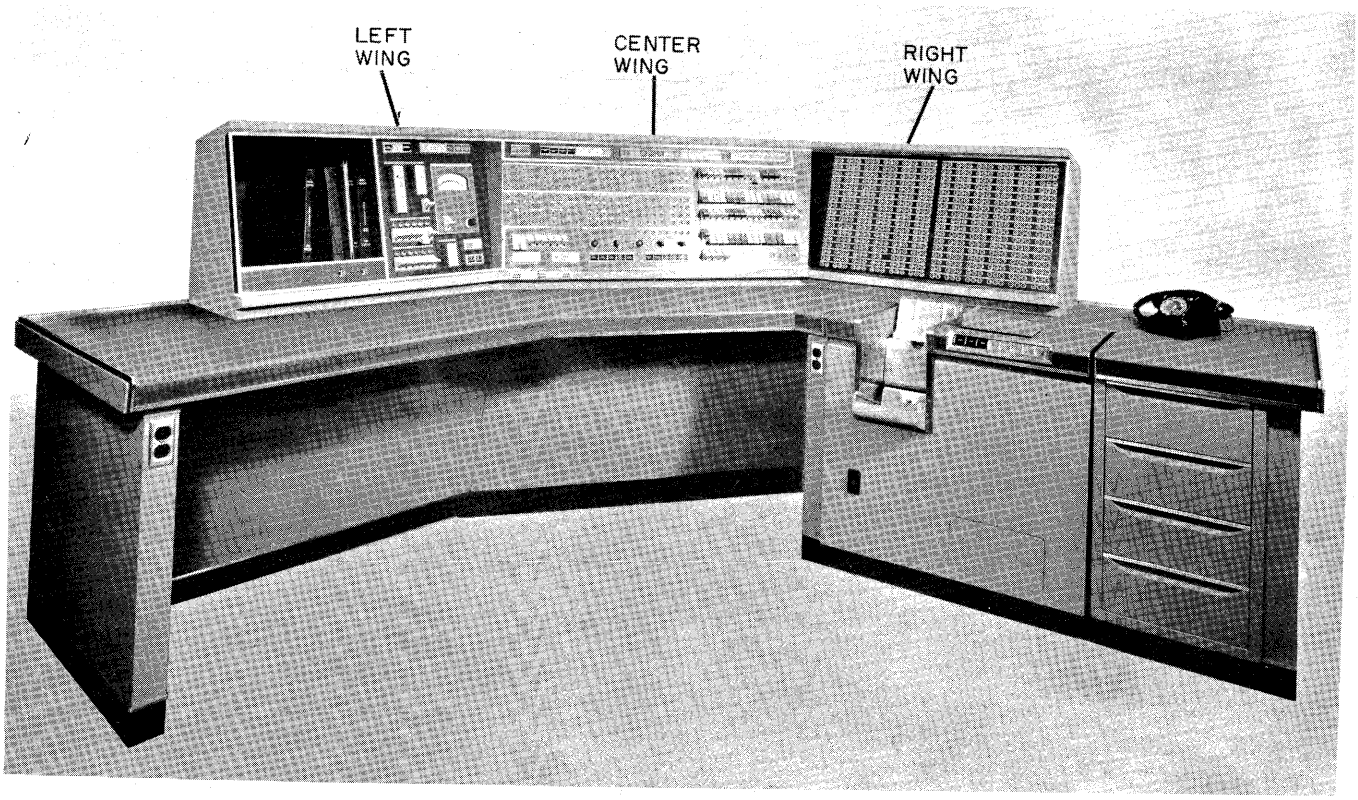


Figure 7–2. Maintenance Console

SECTION 1

MARGINAL CHECK CONTROLS

Marginal checking of the Central Computer System is a method of testing which simulates the aging of components by varying the voltages applied to the circuit elements. To localize trouble sources rapidly and to prevent confusion in interpreting the results of these tests, the AN/FSQ-7 (XD-1,-2) Combat Direction Central has been divided into blocks called equipment groups, voltage groups, margin groups, and logic groups.

1.1 DEFINITION OF TERMS USED

Many names used in the marginal checking system may sound much alike but describe different units or functions and these may erroneously be interchanged. The terms used in the subsequent text are defined, therefore, in order to avoid any duality of meaning.

a. A voltage group consists of all those circuits within the entire system which are to be marginal checked by one voltage.

b. An equipment group consists of a number of different units having a common location and function.

c. A margin group consists of all those circuits having approximately the same margin for one specific marginal checking voltage.

d. A logic group is composed of circuits having the same function and the same approximate margin for one specific marginal checking voltage.

e. Service voltage consists of the normal operating potentials applied from the regular power supply to the circuit elements during normal operating condition.

f. Marginal checking service voltage is the specific voltage selected (viz, one of the following: +250, +150, +90, −150, −300) as the input voltage to the marginal checking system.

g. Excursion is the magnitude of the voltage variation used for marginal checking. This variation may be in either a positive or negative direction.

h. Marginal checking test voltage is the sum of the voltage produced by the addition or subtraction of a voltage variation (excursion) to or from the service voltage used for marginal checking.

i. Margin is the operating limit of a circuit and will continually and gradually change because of variations due to the aging of components.

1.2 BLOCK DIAGRAM ANALYSIS

The five voltages used for marginal checking purposes are shown in figure 7–3. The specific voltage to be used in testing is fed through an amplidyne to one of the eight equipment groups. It will be noted (figure 7–3) that only one voltage and one equipment group may be specified for tests at any one time. Each equipment group is subdivided into six margin groups, and each margin group is divided into a maximum of six logic groups. The marginal checking test voltage; i.e., selected voltage plus excursion, is applied to one or more logic groups within the selected margin group. Note from the previous discussion of the marginal checking system that the marginal checking test voltage applied to the logic groups performs no test function unless a program is run through the logic groups concurrently with the application of the test voltage.

The marginal checking breakdown just described is necessary to facilitate the running of routine tests, such as the reliability or diagnostic program, without propagating or pyramiding any errors which may occur. The various margin groups are provided because of the great difference in margins between the circuits within an equipment group and a voltage group. For instance, an A type flip-flop (checked by the −250-volt line) has destructive excursion limits of +100 and −125 volts, whereas a B type flip-flop (checked by the −150-volt line) has a destructive excursion limit of +75 and −100 volts. The difference in circuit type as well as circuit application dictates the assignment of these flip-flops to different margin groups. In addition, various gates associated with command generators or flip-flops in one register are separated into different margin groups to prevent the propagation of errors. For example, the add gates associated with the accumulator command generators are in one margin group while the even gates are in another.

Logic groups within a margin group are split into functional groups. For example, gates in the instruction control element which generate the commands executing the *Branch and Index (BPX)*

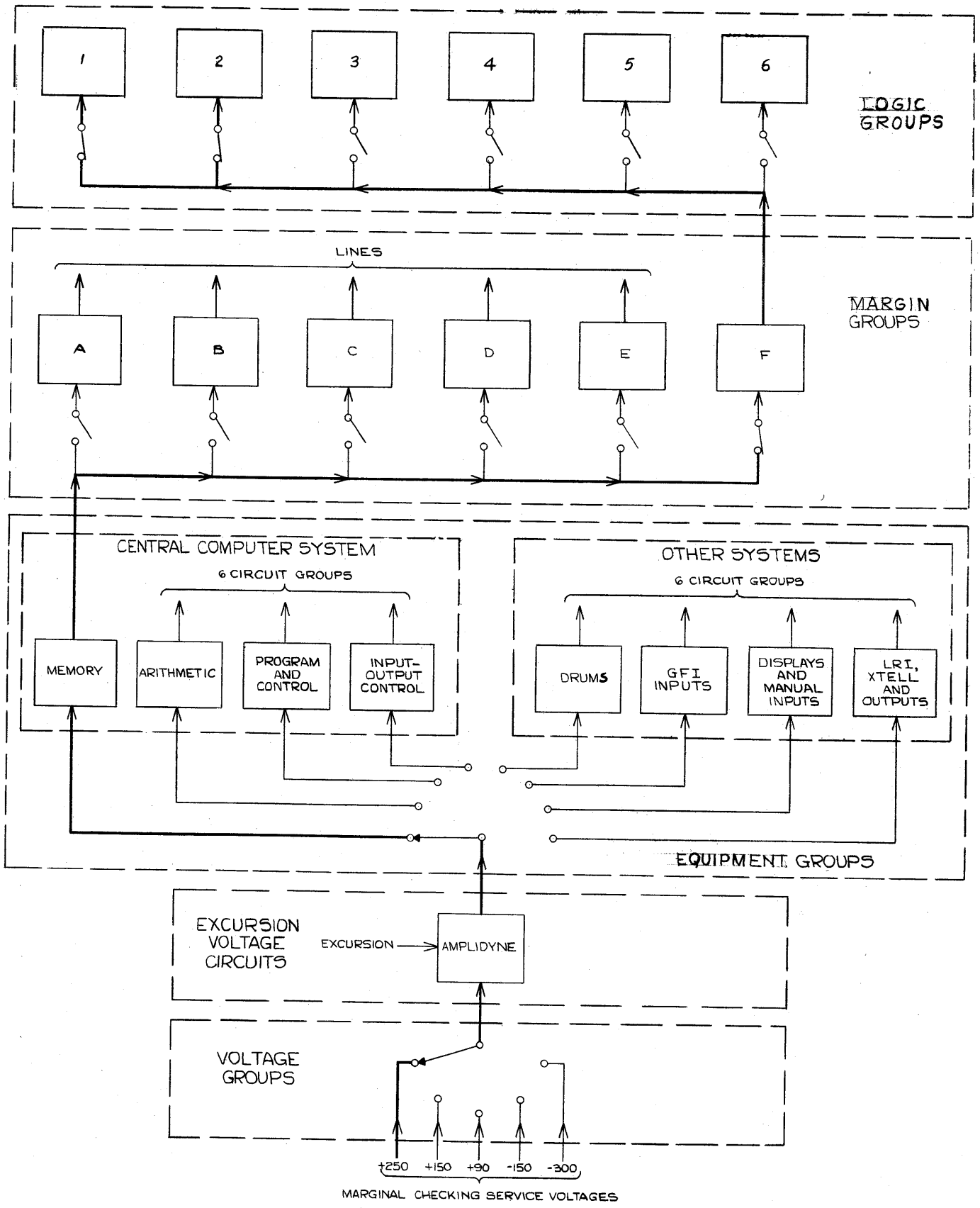


Figure 7-3. Marginal Checking System, Block Diagram

instruction are in one logic group while gates which generate the commands executing a *Store* or *Reset* instruction are in another. Transfer gates and counter gates are separated into different logic groups and even different margin groups. In addition, odd order and even order gates are separated so that a carry-over from one gate to another is prevented if an error occurs.

1.3 MARGINAL CHECKING MAIN CONTROL PANEL

The marginal checking control panel on the maintenance console comprises a number of switches, controls, and lights which are listed in table 7–2 and illustrated in figure 7–4. The switches and controls provide either manual or automatic (programmed) control of marginal checking. Further, two manual control modes are provided. By means of the MANUAL status pushbutton, all selection (of equipment groups, voltage groups, margin groups, logic groups and voltage excursion groups) is performed by the operator at the maintenance console. By means of the SATELLITE status pushbutton, control is exerted by an operator who is troubleshooting at some remote location. Programmed automatic control of marginal checking is provided by the CALCULATOR pushbutton on the maintenance console.

TABLE 7–2. MARGINAL CHECKING MAIN CONTROL PANEL

SWITCHES AND CONTROLS	INDICATOR LIGHTS
START AMPLIDYNE (microswitch)	AMPLIDYNE READY and READY lights are illuminated after a 50 second time delay.
EXCURSION CONTROL (pushbuttons)	
MANUAL	MANUAL
SATELLITE	SATELLITE
CALCULATOR	CALCULATOR
PRESET	PRESET
EQUIPMENT GROUP SELECTOR (pushbuttons)	
MEMORY	MEMORY
ARITHMETIC	ARITHMETIC
PROGRAM AND CONTROL	PROGRAM AND CONTROL
IO CONTROL	IO CONTROL
DRUMS	DRUMS
DISPLAY AND MAN INPUT	DISPLAY AND MAN INPUT
SDV RADAR INPUT	SDV RADAR INPUT
OUTPUT, XT AND MRI	OUTPUT, XT AND MRI
VOLTAGE SELECTOR (pushbuttons)	
+250	+250
+150	+150
+90	+90
−150	−150
−300	−300
MARGIN GROUP SELECTOR (toggle switches)	
A	A
B	B
C	C
D	D
E	E
F	F
LOGIC GROUP SELECTOR (toggle switches)	
1	1
2	2
3	3
4	4
5	5
6	6
MANUAL EXCURSION (potentiometer)	
VOLTMETER SCALE (toggle switch) Range: ±120V, ±30V	
START EXCURSION (microswitch)	EXCURSION ON
STOP EXCURSION (microswitch)	EXCURSION ON light goes off
STOP AMPLIDYNE (microswitch)	All lights go off
	SEQUENCE FAULT
	AMPLIDYNE CIRCUIT BREAKER
	72 V or 48 V CIRCUIT BREAKER

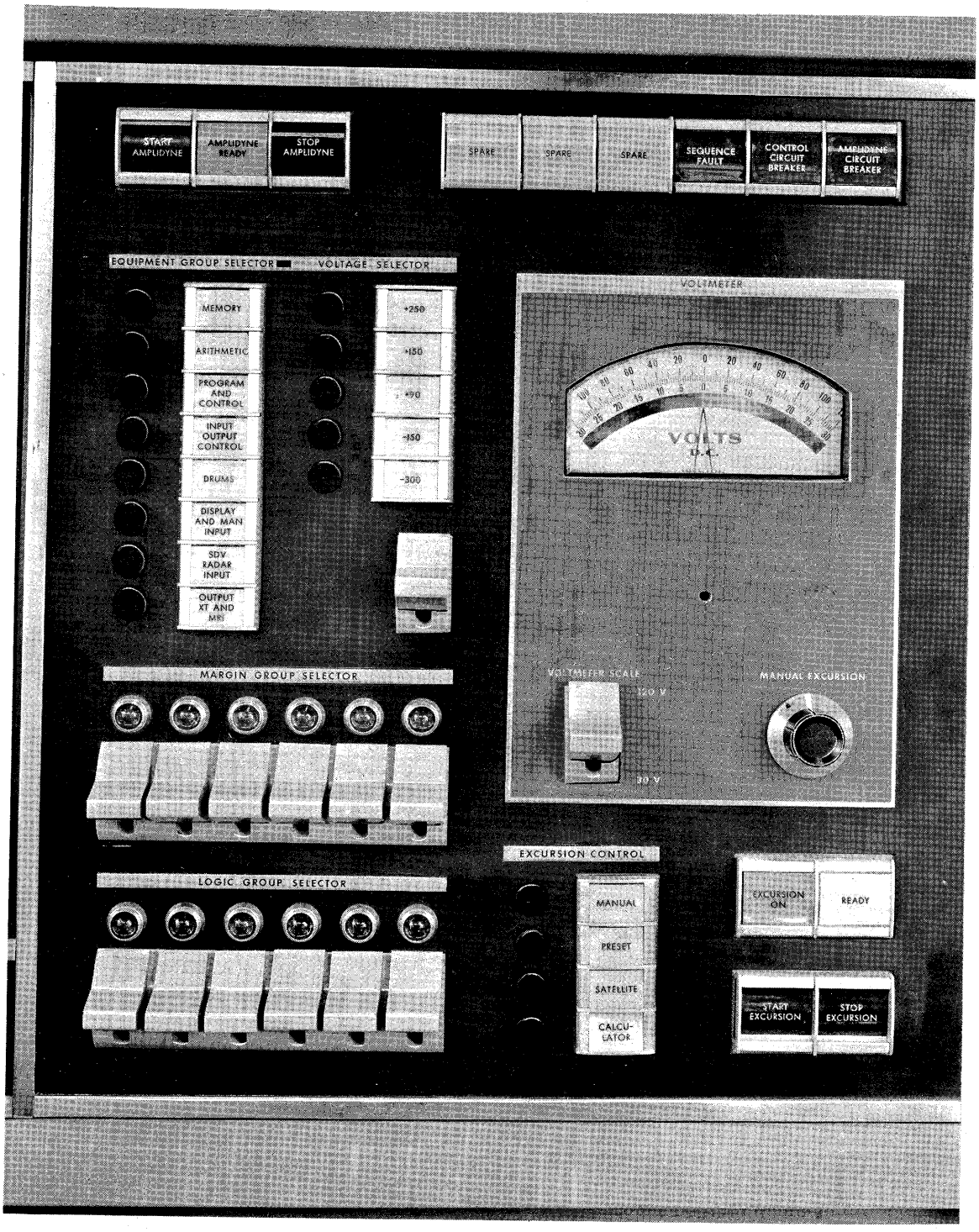


Figure 7-4. Marginal Checking Main Control Panel

The sequence in which the switches and controls on the maintenance console are used is not rigid though the approximate sequence of operation is as follows:

A. Depress the START AMPLIDYNE microswitch. A signal is applied to the amplidyne relay which starts the amplidyne motor. The reference voltage is switched on and the cam timing motor is energized. This reference voltage is in series with the amplidyne and is varied by the MANUAL EXCURSION potentiometer. Thus, it adds to or bucks (depending on the polarity of the excursion) the amplidyne circuit voltage, thus placing an excursion voltage on the circuits to be tested. After a time delay of approximately 50 seconds, certain relays in the amplidyne circuits complete a path which allows the cam units to start timing. Note, however, that the actual timing begins only when the START EXCURSION switch is depressed. The AMPLIDYNE READY light and the READY light are now illuminated.

B. During the 50-second time delay just described, the desired EXCURSION CONTROL, EQUIPMENT GROUP SELECTOR, and VOLTAGE SELECTOR pushbuttons are depressed. The lights associated with the selected pushbuttons are illuminated.

C. The desired MARGIN GROUP SELECTOR and LOGIC GROUP SELECTOR toggle switches are operated, illuminating the lights associated with the specifically selected groups.

D. The MANUAL EXCURSION potentiometer is rotated to give the desired excursion, which is indicated on the voltmeter.

E. Depressing the START EXCURSION microswitch sets the marginal checking operation into motion.

The sequence of operations illustrated in figure 7-5 which occurs when the START EXCURSION switch initiates marginal checking is as follows:

A. The status relays (manual or calculator) are selected and held.

B. The equipment group and voltage group relays are selected and held.

C. The intermediate relays are selected and held.

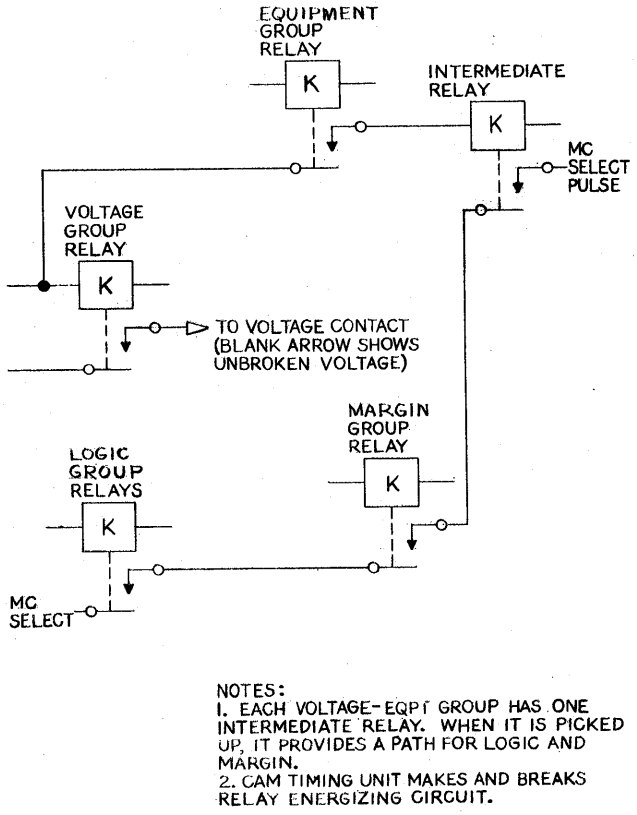


Figure 7-5. Marginal Checking Selection

D. Shortly thereafter, the margin group and logic group relays are selected and held.

E. The excursion voltage is now applied to the circuits under test.

F. After one cycle of rotation (of 400-millisecond duration), the cam unit stops timing and either a new excursion may be applied to the circuits under test or the same or a different excursion may be applied to another margin group or logic group.

Some of the devices used in the marginal checking system, such as the amplidyne, the reference voltage supplies, and the cam timing units, are discussed in detail in Manual 6, Part 5, Marginal Checking Control Element. These will be described briefly in this Section, however, to supply the information necessary to round out the subsequent discussion. The amplidyne is a special d-c generator with several field and control windings. The sum of the two reference voltage supplies, one of +150 and the other of −150 volts, is the input to the field windings of the amplidyne. When these voltage inputs to the field are equal, their net effect is to cancel each other since they are of opposite polarity. By means of the amplidyne, the net voltage determined by the setting of

the MANUAL EXCURSION potentiometer is introduced in series with the marginal checking service voltage. If the net voltage is positive, it adds to the marginal checking service voltage, whereas if the net voltage is negative, it bucks the marginal checking service voltage. This net voltage from the reference voltage supply plus the voltage used for marginal checking thus becomes the marginal checking test voltage which is applied to the margin groups and logic groups to perform marginal tests.

The cam timing unit initiates the proper sequence of operations in the marginal checking system. As the shaft of the cam contactor unit rotates, the various cams lock in and energize the marginal checking relays in a specific time sequence, as explained in detail in Part 5 of PH 42–00001 and briefly in the following paragraphs. Two cam timing units are provided, one for manual operations (MANUAL and SATELLITE) and one for automatic operations (CALCULATOR).

The general manner in which marginal checking selection operates is partially depicted in figure 7–5. The cam contactor timing unit makes and breaks the various relay energizing circuits (equipment group, voltage group, margin group, and logic group relays). The sequence is such that the equipment group and voltage group relays are energized first. Each voltage-equipment group has one intermediate relay. When this relay is energized through the contacts of the equipment group relay, a path is provided through the contacts of the margin group and logic group relays for the equipment selection voltage. Shortly after the previous two group relays are energized, the cam timing unit energizes the margin group and logic group relays. When this is done, a complete path is provided for selecting and applying the proper voltage and excursion to the desired logic groups and margin groups.

1.4 SELECTION FOR MARGINAL CHECKING

A simplified diagram showing how selection is accomplished for marginal checking purposes is given in figure 7–5. All figure references in subsequent subparagraphs in this Section will apply to this figure. For greater circuit detail and a complete discussion of the various components which are not contained in the maintenance control element, refer to Part 5 of PH 42–00001. It must be recalled at this point that the sole function of selection in the marginal checking system is to apply a desired voltage plus a desired excursion to the selected margin groups and logic groups through which the reliability program is run.

1.4.1 Equipment Group Selection

As the equipment group cam on the cam timing unit rotates (figure 7–5), the contactor feeds —48 volts through the status (calculator) relay contacts to one of the eight equipment group switches on the maintenance console panel. When one of these switches has been closed by the operator at the maintenance console, the selected equipment group relay is energized. This relay corresponds to the particular equipment switch which was activated by the operator at the maintenance console.

1.4.2 Voltage Group Selection

As the cam timing unit shaft continues rotating, the voltage group cam contactor closes. (See fig. 7–5.) The —48 volts is fed through the status relay contacts to that one of the five voltage group switches on the maintenance console which has been closed by the operator, thus energizing both the voltage group and intermediate relays. When the voltage group relay is energized, —48 volts is fed through the contacts to energize the voltage contactor relay, through whose contacts the voltage used for marginal checking is fed to the open contact of the marginal checking relay.

1.4.3 Margin Group Selection

Shortly after the voltage group relay is energized, the rotation of the cam timing unit shaft closes the margin group cam contactor. This action energizes the margin group relay by applying —48 volts to it through one or more of the six margin group switches selected by the operator at the maintenance console. The contacts of the margin group relay are closed, thus providing a path for the marginal checking select pulse when the logic group relay is energized.

1.4.4 Logic Group Selection

At approximately the same time that the margin group relay is energized, the logic group cam contactor closes and energizes the logic group relay through one or more (depending on the number activated) of the logic group switches on the maintenance console. The marginal checking select pulse (—48 volts) is then fed through the contacts of the logic group relay, the margin group relay, and the intermediate relay to energize the marginal checking relay. Closure of the marginal checking relay contacts applies the voltage used for marginal checking to the margin groups and logic groups to be tested in conjunction with the reliability program.

1.5 MARGINAL CHECKING STATUS OR MODE OF OPERATION

The marginal checking system of AN/FSQ–7 Combat Direction Central may be operated in the manual, satellite, or calculator control mode. Each one of these control modes differs in application but not in function.

1.5.1 Manual Control

In manual control, marginal checking selection is performed by pushbuttons and toggle switches on the marginal checking main control panel located in the maintenance console. The magnitude and polarity of the excursion voltage are controlled by a potentiometer and voltmeter on the main control panel. These have been discussed previously in 1.4 of this Section.

1.5.2 Satellite Control

Satellite control is a manual control mode wherein the excursion voltage is controlled by a potentiometer remote from the maintenance console. All other selection and control operations are performed by the operator stationed at the maintenance console. Satellite control has been incorporated in the marginal checking system to enable maintenance personnel to control the excursion voltage while using electronic test equipment at any part of the Combat Direction Centrals. Each marginal checking and distribution (MCD) unit is provided with a satellite test unit consisting of a potentiometer and voltmeter so that the voltage excursion may be controlled individually at each MCD unit. The operator or troubleshooter at the remote unit controls the excursion when the SATELLITE pushbutton is depressed. This action supplies the remote operator with an interlocked ground connection which completes the ground circuit of the potentiometer and voltmeter at the remote unit.

1.5.3 Calculator Control

In calculator control, all marginal checking operations are controlled by the reliability program through the circuits of the Central Computer System. No human intervention is required and each marginal checking sequence will be automatically initiated as the program requires it. When the program requires that a certain group of circuits be marginally checked, it will cause the transfer of a binary coded word from internal core memory to the test register in the arithmetic element. The flip-flops of the test register energize relays by means of thyratrons located in the marginal checking system. These relays will direct the execution of the marginal checking routine. The various bit assignments of the marginal checking thyratron register are shown in table 7–3 and in figure 7–6.

TABLE 7–3. BIT ASSIGNMENTS OF CALCULATOR CONTROL THYRATRON REGISTER

BIT	DESCRIPTION
LS	(−) Start or restart marginal checking excursion operation and program
	(+) Restart with program and change excursion polarity and magnitude
L1–L3	Equipment group selector bits
L4–L9	Margin group selector bits
L10–L15	Logic group selector bits
RS–R1	Available for future expansion
R2–R3	Restart program after the excursion has been applied
R4–R5	Restart program after the excursion has been removed
R6–R7	Excursion duration bits
R8–R10	Voltage selector bits
R11	Excursion polarity
R12–R15	Excursion magnitude

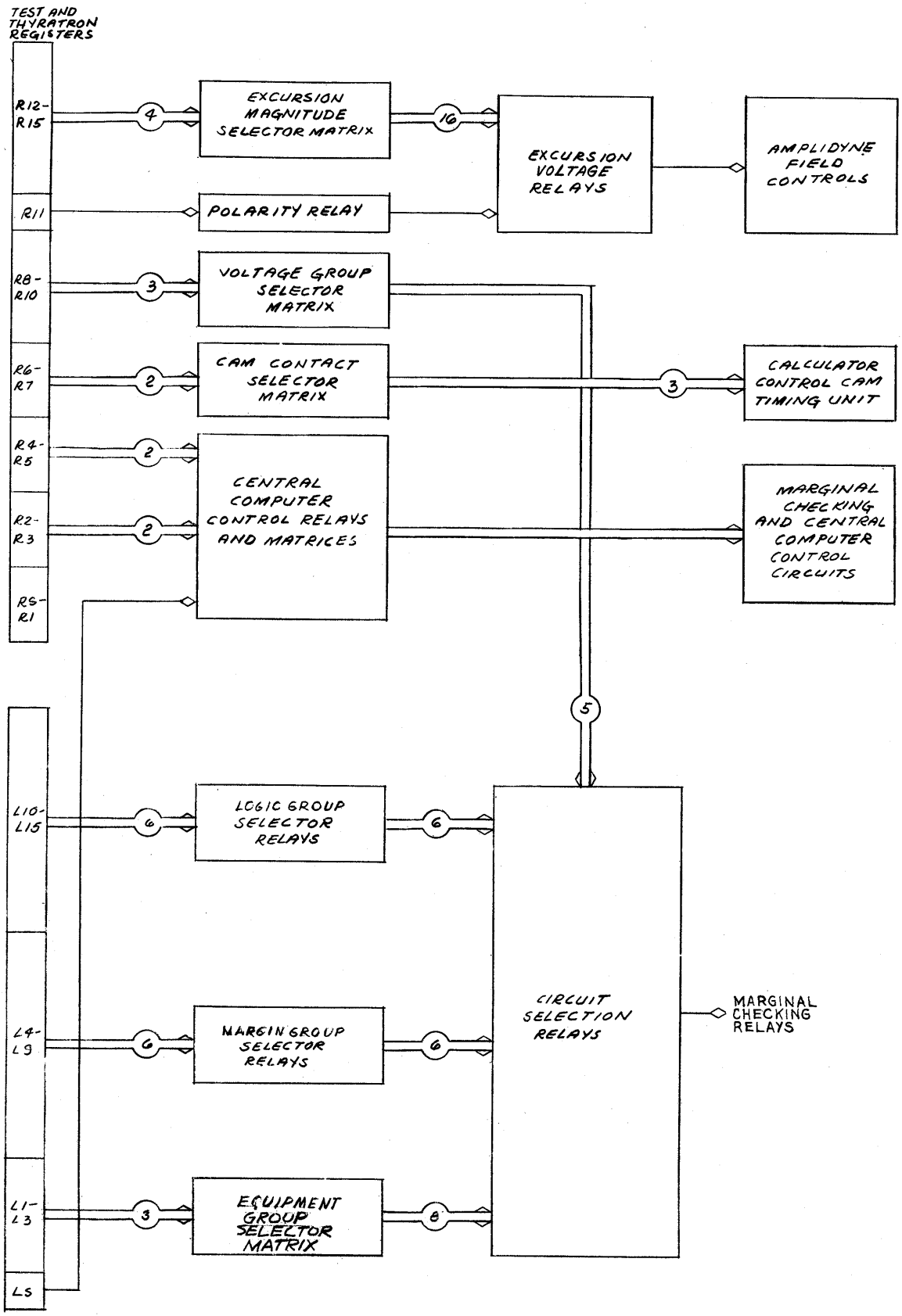


Figure 7-6. Calculator Control, Marginal Checking

SECTION 2

MANUAL OPERATIONS CONTROL

The manual operations control causes the Central Computer System to operate in accordance with the commands received from the pushbuttons and switches on the maintenance console. Circuits in the maintenance console interlock the operation of the pushbuttons so that manual operations will not interfere with the performance of regular Central Computer System programs. When a pushbutton is operated, the associated relay contacts actuate thyratron pulse generators which produce 0.1-microsecond pulses. Since these thyratron pulses may or may not be generated coincidentally with the command pulses generated in the instruction control element, circuits are provided which synchronize the thyratron pulses with the machine timing pulses.

Some of these pushbuttons are used to load either initial operational programs or test programs into the Central Computer System. Three of the pushbuttons located on the center-wing control panel of the maintenance console (figure 7–7) are used to load programs into the magnetic core memory of the Central Computer System. These are the START FROM TEST MEMORY, LOAD FROM CARD READER, and LOAD FROM A.M. DRUMS pushbuttons. When the latter two pushbuttons are operated, an automatic sequence of events occurs which, in effect, executes the following instruction sequence:

a. Loads the address counter with the first magnetic core memory address to be used in the subsequent word transfer

b. Selects the desired program source (drums, cards)

c. Reads at most 24 words from the selected program source

d. Delays the execution of any instruction until at most 24 words are transferred into magnetic core memory

e. Carries out the instructions dictated by the words just transferred into magnetic core memory

The controls on the center-wing control panel are divided into operating controls and servicing controls. The operating controls listed in table 7–4 are used largely in conjunction with the servicing controls, whereas the servicing controls themselves may never be used during normal machine operations. The servicing controls are used to simulate various subprograms, to initiate program steps, or to test basic circuit operations such as clear or complement.

It should be noted, and will be explained in following paragraphs, that the three program-loading pushbuttons are operative only if the Central Computer System has been halted by either a *Program Stop (HLT)* instruction or by depressing the PROGRAM STOP pushbutton.

Before the execution of a program is initiated, certain controls shown on the center-wing panel of the maintenance console, which are not used during normal Central Computer System operations, must nevertheless be preset to a desired position. Most of these controls determine the action of the computer when certain conditions prevail; i.e., these controls determine whether the program steps are executed sequentially or whether the program branches to a non-sequential program step under certain conditions. Table 7–5 lists these controls, specifies their settings, and describes the functions performed for the various settings.

The following controls are used only for servicing the AN/FSQ–7 (XD–1,–2) Combat Direction Centrals:

a. COMPLEMENT pushbutton

b. INSTRUCTION STEP pushbutton

c. CYCLE PROGRAM COUNTER and CYCLIC PROGRAM switches

d. MASTER RESET pushbutton

e. RESET FLIP-FLOP pushbutton

f. MEMORY CYCLE pushbutton

In addition to these pushbuttons and switches, several of the operating controls are also used in conjunction with servicing and testing. These are:

a. PROGRAM STOP pushbutton

b. PROGRAM CONTINUE pushbutton

c. CLEAR MEMORY pushbutton

d. CLEAR ALARMS pushbutton

All of the pushbuttons and switches just referred to, in addition to the power controls and indicators, will be discussed in detail in subsequent paragraphs.

EMERGENCY OFF
POWER OFF
AC ONLY
POWER ON
ACKNOWL EDGE
MOTOR OFF
POWER OFF
AC ONLY
POWER ON
NOT READY
READER
PUNCH
PRINTER
DRUMS
TAPE MACHINE
DRUM SELECTION
DRUM SELECTION REGISTER
FIELD SELECTION
ADDRESS
IDENTIFY 2-10
IDENTIFY 11-13
IDENTIFY 14-15
INTERLEAVE MODE
DRUM PARITY CHECK
LOOP ADDRESS COUNTER
READ
WRITE
DRUMS
WRITE ENABLED
DRUM SELECTED
DRUM FULL ALARM
DRUM ADDR REG REQUEST
I/O BUFFER FILLING SYNC
I/O BUFFER FILLING
INSTRUCTION
INSTRUCTION STEP
MEMORY CYCLE
INPUT OUTPUT INTERLOCK
LOAD
PAUSE DELAY CLEAR SYNC
CONTINUE SET SYNC
BRANCH
I/O CONTROL SET
I/O CONTROL CLEAR
2 MEGACYCLE SET SYNC
CLEAR PAUSE SYNC
WORD COUNTER TRANSFER RECORD
WARNING LIGHT
BOX WARNING LIGHT
IMMEDIATE CONT MEM
MAIN LIGHT CONT MEM
MAIN LIGHT CONT MEM
ARITHMETIC
REAL TIME CLOCK FREQUENCY DIVISION
CLOCK REG CALC SYNC
CLOCK REG GATE
CONNECT FOR DIVISION
STORAGE CARRY
SEFE
RIGHT
AUXILIARY OVERFLOW
SEFE
TRY COND COLS 49-80
PARITY COUNT BIT
SELECTION
MECH CLOCK SEL SYNC
MANUAL INPUT SWITCH SEL
INPUT OUTPUT REGISTER SEL
ARRAY REQUEST
BLANK
SINGLE WORD COUNTER
MECH CLOCK SEL
BOX REQUEST SYNC
READ
WRITE
READ OR WRITE COMP
OUTPUT ALARM
READER NOT READY
PUNCH NOT READY
PRINTER NOT READY
CARD HATCH DIRECTION
OUTPUT ALARM SYNC
OPERATE
ASSIGNED
CONTINUE
CONTINUE
CONTINUE
CONTINUE
OPERATE
TEST MEMORY
DRUM PARITY
MEMORY PARITY
TAPE PARITY
OVER FLOW
TAPE TEST
STOP
STOP
STOP
STOP
TEST
TEST DRUMS
TEST OUTPUTS
TEST COMPUTER
MASTER READY
COMPUTE
SPARE
COMPLEMENT
CYCLIC PROGRAM
DELAY
OFF
NO DELAY
CLEAR MEMORY
RESET FLIP-FLOPS
CLEAR ALARMS
READY INPUT OUTPUT UNITS
MEMORY CYCLE
INST STEP
COMPLEMENT

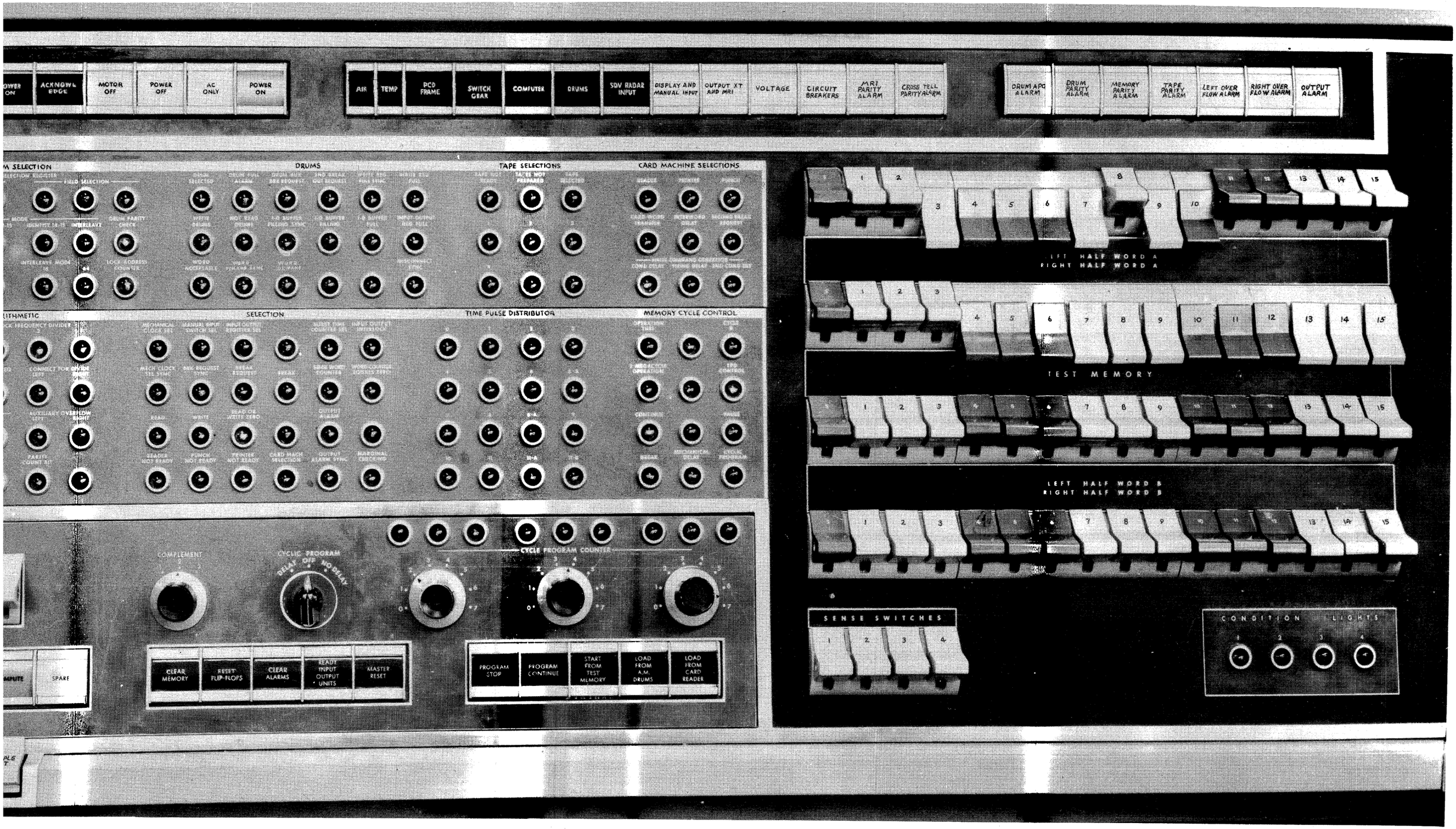


Figure 7–7. Central Computer System Main Control Panel

TABLE 7-4. PROGRAM OPERATING CONTROLS

CONTROL	FUNCTION
LOAD FROM CARD READER	This pushbutton performs a control clear function which clears all flip-flops in the instruction control element, selection and IO control element, and program element (including both index registers), but not the left and right arithmetic elements and, in addition, clears all computer alarms. The 24 (or less) instruction words contained on one card are then loaded into the first 24 internal magnetic core memory locations and the Central Computer System executes the program designated by these instructions.
LOAD FROM A.M. DRUMS	The function performed by this pushbutton is identical to that of the LOAD FROM CARD READER pushbutton except that the program is transferred from A.M. drums.
START FROM TEST MEMORY	This pushbutton also performs the above control clear function, and causes the Central Computer System to execute the program stored in the 16 addresses of test memory.
PROGRAM STOP	The Central Computer System is halted after it has completed the instruction being executed. If a break-in or break-out cycle is being executed, all the input-output breaks are completed before the Central Computer System stops, but no further instructions will be executed. However, if it is necessary to stop the Central Computer System before all the input-output break cycles are completed, depress the RESET FLIP-FLOP pushbutton to stop the Central Computer System immediately.
PROGRAM CONTINUE	This pushbutton clears the parity and left and right overflow alarms, output alarm, and drum APC alarm.
CLEAR ALARMS	This pushbutton clears the parity and left and right overflow alarms, output alarm, and drum APC alarm.
CLEAR MEMORY	All core memory positions are reset to positive zero, but the flip-flops are not cleared.
READY IO UNITS	Places the card machines and tapes in a state of readiness.

TABLE 7-5. PRELIMINARY CONTROL ADJUSTMENT

CONTROL SWITCHES	DESCRIPTION
OPERATE-TEST	Set to OPERATE. This interlocks the servicing controls so that the Central Computer System program is not interrupted by inadvertent depression of one of the loading or servicing pushbuttons. In the TEST position, this switch supplies the voltages to energize the servicing circuits.
TEST MEMORY ASSIGNED-UNASSIGNED	In the ASSIGNED position, the TEST MEMORY toggle-switch registers A and B are assigned addresses 20,000 and 20,001 respectively, the live register is assigned address 20,017, and the plugboard registers are assigned addresses between the two. In the UNASSIGNED position, all TEST MEMORY registers are assigned as plugged.
TEST MEMORY toggle registers A and B	These switches may be assigned to any of the first 15 addresses of test memory by inserting a plug in the A or B assignment hubs of the desired address on the test memory plugboard.
PARITY CONTINUE-STOP and OVERFLOW CONTINUE-STOP	These are set to the desired position. In the STOP position, the Central Computer System will stop on a parity error or overflow. In the CONTINUE position, the Central Computer System will turn on the respective parity alarm or overflow flip-flop and its associated alarm light and will continue with the program. In order to allow the calculator to take any specific action in the event of a parity error or overflow alarm, the program must sense the alarm flip-flop for such an error.
SENSE	Unless specific settings are desired, these should be off.
CYCLE PROGRAM COUNTER and CYCLIC PROGRAM	These controls are used only for servicing and should be OFF during normal operations.

2.1 FUNCTION AND USE OF THE TIME PULSE DISTRIBUTOR

The time pulse distributor (TPD), described in detail in Part 2, is responsible for the generation of time pulses (TP) and instruction pulses (IP), as shown in figure 7–8, which is a simplified logical block schematic diagram of the TPD and the lines controlling it. The TPD converts 2-megacycle pulses (from the 2-megacycle oscillator) into timed pulses (TP–0, TP–1, IP–1, TP–2, IP–2, etc.) which generate the commands used to execute all the instructions of the Central Computer System. The manner of operation of the TDP is dictated by four flip-flops in the time pulse distributor control, TPDC); namely, the TPD control, continue, break, and pause flip-flops, which have also been described in Part 2. At the direction of these flip-flops in the TPDC, the TPD may generate both TP and IP pulses, TP pulses only, neither TP nor IP pulses (during a pause operation), or the TPD may be shut off entirely.

During automatic operations of the Central Computer System, the TPD is never completely shut off as occurs when both the TPD and continue flip-flops are cleared. For manual operations control, however, the PROGRAM STOP pushbutton must be depressed before pushbutton operation is initiated, thus halting the TPD and stopping the Central Computer System. The next step is to depress the desired pushbutton, which now sets in all the conditions necessary to execute the operations designated by this pushbutton. The last operation performed by the depressed pushbutton, after a time delay dependent on the electromechanical inertia of the various relays in the activated circuits, is that of restarting the TPD. When the TPD is restarted, the TP's and IP's now generated will execute the operations set in by the pushbutton. The TPD, before being restarted, may sometimes be preset to time level 8 (TL–8). Since a new instruction is transferred into the operation and address registers at program time 7 (PT–7), presetting the TPD to TL–8 automatically skips this step, thus inhibiting the bringing down and decoding of a new instruction until PT–7 of the next program cycle. If the TPD is not preset, however, the first pulses generated will automatically be TP–7 and IP–7.

When the PROGRAM STOP pushbutton is activated, as a prelude to manual operations control, the TPD control and continue flip-flops are cleared at TP–6. (See fig. 7–8.) Clearing the continue flip-flop removes the conditioning level on the AND circuit associated with TPD–7 so that TP–7 and IP–7 are not generated; i.e., TP–6 and IP–6 are the last pulses generated. Clearing the TPD control flip-flop removes the conditioning level from the gate associated with the TPD control flip-flop. The TP and IP driver lines are thus deactivated, since 2-megacycle pulses are not gated through to drive these lines. This action effectively shuts off the TPD so that no pulses are generated. Restarting the TPD now consists of setting the TPD control and continue flip-flops, so that 2-megacycle pulses may be gated through to drive the TP and IP driver lines, thus generating pulses starting from PT–7 and IP–7. As an additional condition for generating IP pulses, the IP driver gate must be conditioned by having the break flip-flop in a cleared condition. In order to keep generating pulses, the AND circuit associated with TPD–0 must be fully conditioned. This AND is conditioned through an OR circuit by either one of two levels, clearing of the pause flip-flop (no pause) or setting of the break flip-flop (during a break-in or break-out cycle). When no pause or break occurs, this AND is conditioned by the no pause; i.e., by the 0 side of the pause flip-flop. In this mode of operation, both TP's and IP's are generated. However, if a pause type of operation occurs, as in the execution of instructions in the multiply, divide, or shift class, the pause flip-flop is set, thus deconditioning this AND circuit. Even though 2-megacycle pulses are gated through to drive the TP and IP driver lines, no pulses are generated, except the initial pulses from TP–7 to TP–11 and from IP–7 to IP–11. TP–0 will not be generated, and consequently the subsequent TPD flip-flops will not be set and will not generate TP or IP pulses. If the pause was introduced by a shift class instruction, the number of shifts is counted off by 2-megacycle pulses applied when the 2-megacycle operate flip-flop is set for this type of operation. At the end of the pause, the pause flip-flop is cleared and the TPD is restarted at TP–0. When a break cycle occurs, the break flip-flop is set. This action conditions the AND on TPD–0 so that TP pulses are generated. The IP driver gate, however, is deconditioned when the break flip-flop goes from 0 to 1 so that IP pulses are inhibited and the break cycle is executed by the TP pulses only. At the end of the break-in or break-out cycle, the break flip-flop is cleared, conditioning the IP driver gate so that the TPD now generates both TP and IP pulses.

Three additional and previously unmentioned flip-flops in the TPDC, the load, the instruction

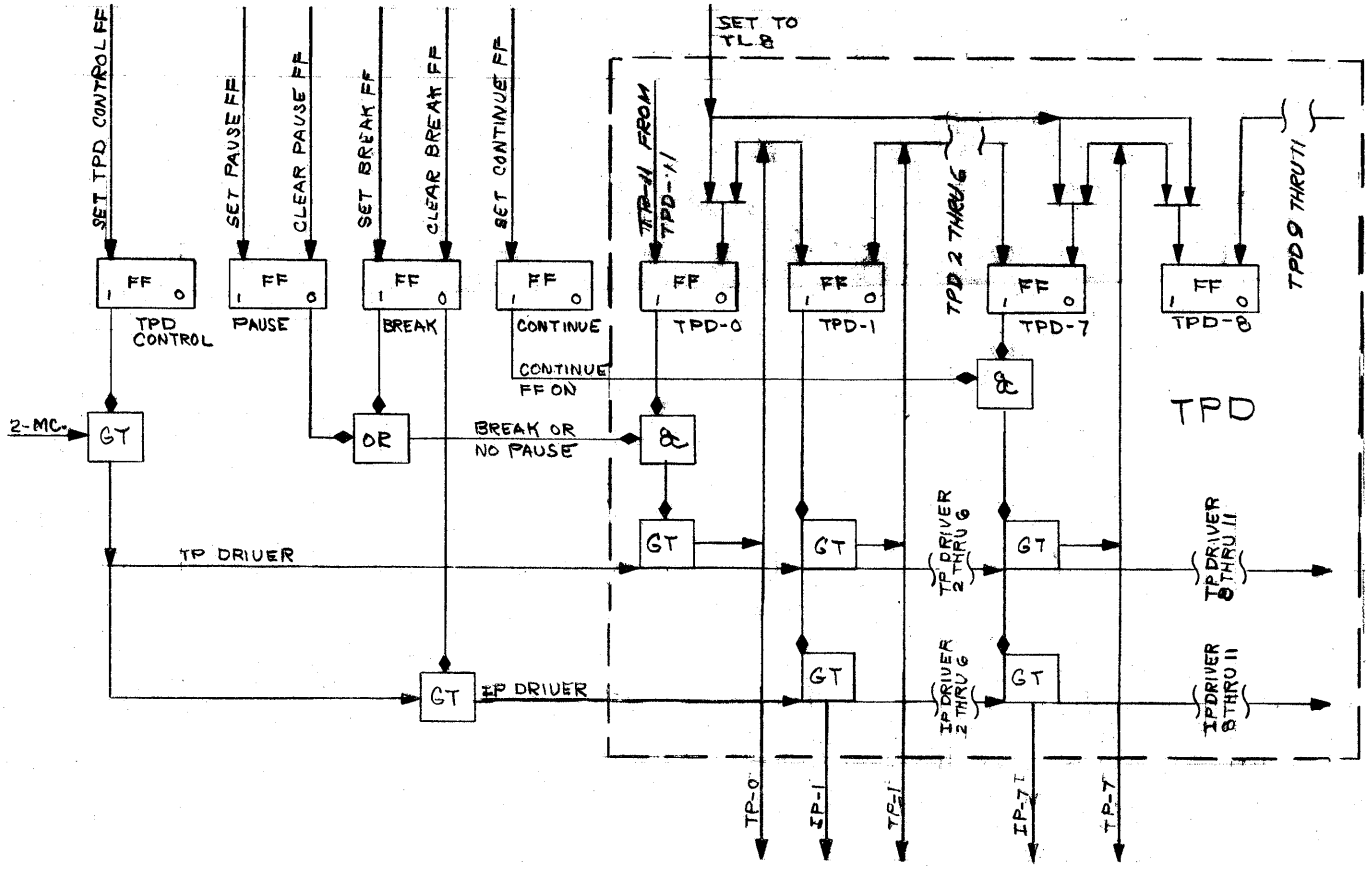


Figure 7–8. TPD Control of Machine Pulse Generation

step, and the memory cycle flip-flops, exert a secondary control over the TPD. These flip-flops, in reality, control either the TPD control or the continue flip-flops and will be discussed subsequently.

2.2 SYNCHRONIZING CIRCUITS

Synchronizing circuits are essential for pulses which pass through a number of circuits (such as gates, power amplifiers, and especially delay units) before setting any flip-flop which conditions a gate tube. The total delay produced by these circuits may shift the pulse timing so that it is no longer coincidental with machine timing. Synchronizing circuits must be used in such instances to insure full pulse amplitude for further use in the Central Computer System. Delay circuits which need synchronizing circuits are contained in the cyclic program counter instruction step, load, parity check, etc., circuits.

Figure 7–9 illustrates the unreliability of pulses which are delayed and are not coincidental with machine timing pulses. In part A of this figure, machine timing pulses are labeled TP–1, TP–2, etc., and the delayed pulses are labeled 1, 2, 3, and 4. Between the two is shown the status of the flip-flop, which is set by the delayed (time shifted) pulses. When pulse 1 sets the flip-flop used to condition the gate, the flip-flop rises and settles before TP–2 occurs, fully conditioning its associated gate tube, as shown in part B of this figure. When pulse 2, 3, or 4 sets the flip-flop, however, the latter will not have settled when TP–2 occurs. As a consequence, the gate tube associated with the flip-flop is not fully conditioned and the output pulse will be of a lower amplitude than that produced by pulse 1. Pulse 2 may or may not be of an amplitude useful in the Central Computer System, but pulses 3 and 4 are definitely not usable and therefore are unreliable pulses. If the pulse is generated later than pulse 1, at any time between TP–1 and TP–2, the usable pulse amplitude is not maximum and is therefore unreliable.

Figure 7–10 illustrates the synchronizing circuit used to obtain the maximum amplitude pulse output needed to insure pulse reliability. The delayed pulse (from any cascaded circuits or delay unit) sets the synchronizing flip-flop which, in turn, conditions its associated gate tube. When the next machine timing pulse (IP or 2-megacycle pulse) senses this gate, the pulse is gated through to set the control flip-flop, which may be the TPD control, the continue, etc., flip-flops. The sole function of the first machine timing pulse is to set the

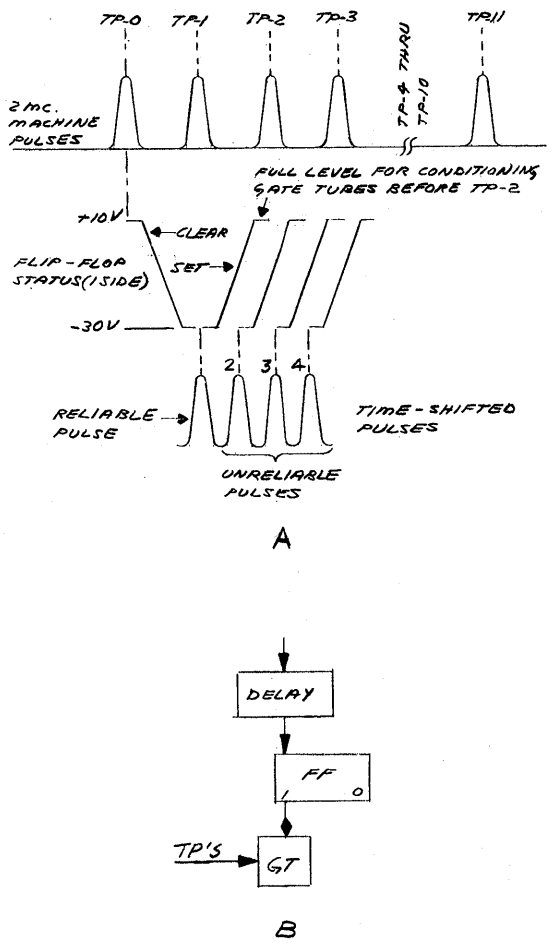


Figure 7–9. Unreliability of Time-Shifted Pulses without Synchronization

control flip-flop, so that no output is obtained as a result of the first timing pulse. The control flip-flop now has ample time to rise and settle before the second machine timing pulse is applied to the gate conditioned by the control flip-flop. This action guarantees that the second and all succeeding pulses are passed to the output as full-amplitude pulses and are reliable for further Central Computer System operations. The first pulse appearing on the output line will, in addition, clear the synchronizing flip-flop, thus disabling its circuit until such time as it is again required. Even though the example just described is used to set a control flip-flop, it can be used to clear a control flip-flop as well. The pause flip-flop circuit shown in figure 7–15 is a type which uses a synchronizing circuit to clear the control flip-flop.

In the subsequent logical block diagrams illustrating the pushbuttons used for manual operations control, the output of the activated pulse generators is fed through synchronizing circuits to set or clear the various control flip-flops. Since most of the pushbuttons start the time pulse distributor at the very last step in operations, synchronization of the pushbutton pulses to machine timing is not highly essential. Suppose, for example, that one of the 2-megacycle oscillator pulses occurring at PT–1, PT–2, PT–3, etc., senses the gate conditioned by the TPD control flip-flop, which is not fully conditioned because of the lack of a synchronizing circuit. Since the gate is not conditioned, no output pulse is generated and the TPD remains off so that no IP's are generated and the instruction remains unexecuted. When the next oscillator pulse senses the gate, it is fully conditioned, thus generating a pulse which starts the TPD so that the instruction is executed. The consequence of not synchronizing the pushbutton pulse is to start the TPD 0.5 microsecond later than would occur if the first pulse were reliable.

2.3 INTERLOCKING CIRCUITS

As stated in the introductory paragraph of Section 2, pushbutton operations are interlocked so that manual operations will not interfere with the performance of regular Central Computer System programs. Two interlocking circuits are used. One provides what are termed non-calculating (NC) voltages for the program loading buttons and their associated relays (NC ground and NC −48 volts); the other provides test voltages for the various servicing and operating pushbuttons and their associated relays (T ground and T −48 volts). As will be shown in subsequent diagrams, non-calculating voltages are derived when the continue flip-flop is cleared and test voltages

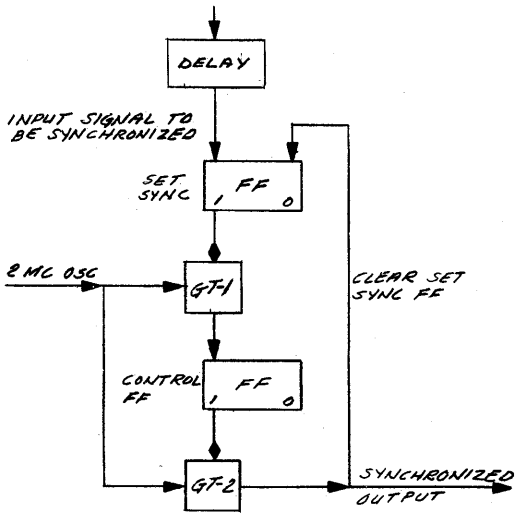


Figure 7–10. Synchronizing Circuit, Simplified Schematic

are derived when the OPERATE-TEST switch is in the TEST position. The term non-calculating means that the Central Computer System is not performing any calculations; i.e., the TPD and the Central Computer System are halted before these non-calculating voltages can be applied to the respective pushbuttons and their associated relays. The same condition applies for the test voltages.

2.3.1 Derivation of Non-Calculating Voltages

The sequence of actions which takes place in the maintenance console when power is applied to the relays will be considered in two steps, the first of which will show relay action prior to the derivation of non-calculating voltages and the second will show the various relays in a settled condition. Figure 7–11 illustrates the relay action which takes place in the first step. When all power is on, a relay contact closes and applies −48 volts to relay K–A10, energizing it. The single contact of relay K–A10 closes, thus applying 120-volt ac to the Haydon Timer through contact 4 of relay K–B2. The single contact of the Haydon Timer closes, after a preset time delay, energizing relay K–B1 by applying −48 volts through contact 3 of relay K–B2. When the contacts of relay K–B1 close, relay K–B3 (the power on, clear, and reset relay) is energized by the −48 volts applied through contact 2 of relay K–B1, and relay K–B2 is energized by −48 volts applied through contact 1 of relay K–B1.

Figure 7–12 illustrates the action which follows when the relays have settled. When the contacts of relay K–B2 close, three actions occur:

a. Contact 4 of relay K–B2 breaks the 120-volt a-c circuit, thus de-energizing the Haydon Timer.

b. Contact 3 breaks the −48-volt circuit to relay K–B1, de-energizing it and contact 3 now becomes the holding contact for relay K–B2 until all power is shut off.

c. Closure of contact 2 on relay K–B2 illuminates the MASTER READY light by applying −48 volts through the contacts of relay K–B9. Non-calculating −48 volts and non-calculating ground are provided by contacts of relay K–B9 as long as this relay remains de-energized. This condition is indicated by the illumination of the MASTER READY light. Relay K–B9 will remain de-energized as long as the continue flip-flop is in a cleared condition and will thus provide non-calculating voltages.

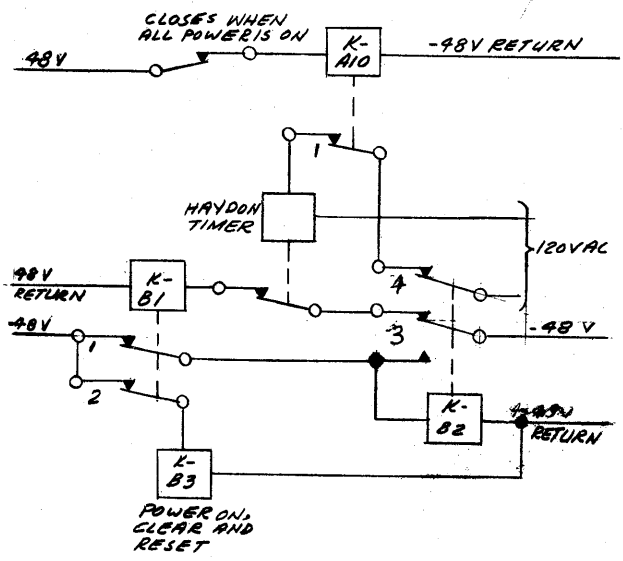


Figure 7–11. Relay Action prior to Derivation of Non-Calculating Voltages

When the continue flip-flop is set, however, compute relay KD2 is energized. The single contact of relay KD2 closes, energizing relay K–B9. Closure of the contacts of relay K–B9 breaks the non-calculating circuit, supplying a calculating ground for the PROGRAM STOP pushbutton and illuminating the COMPUTE light rather than the MASTER READY light. In this state, the TPD supplies TP and IP pulses to the Central Computer System so that calculations may take place. Table 7–6 shows the pushbuttons and relays in the maintenance console which use the non-calculating voltages.

TABLE 7–6. PUSHBUTTONS AND RELAYS USING NON-CACULATING VOLTAGES

PUSHBUTTON USED	RELAY ENERGIZED
LOAD FROM CARD READER	K–A1
LOAD FROM A.M. DRUMS	K–A2
START FROM TEST MEMORY	K–A3
	K–A4 (If either relay K–A1, K–A2, or K–A3 is energized)

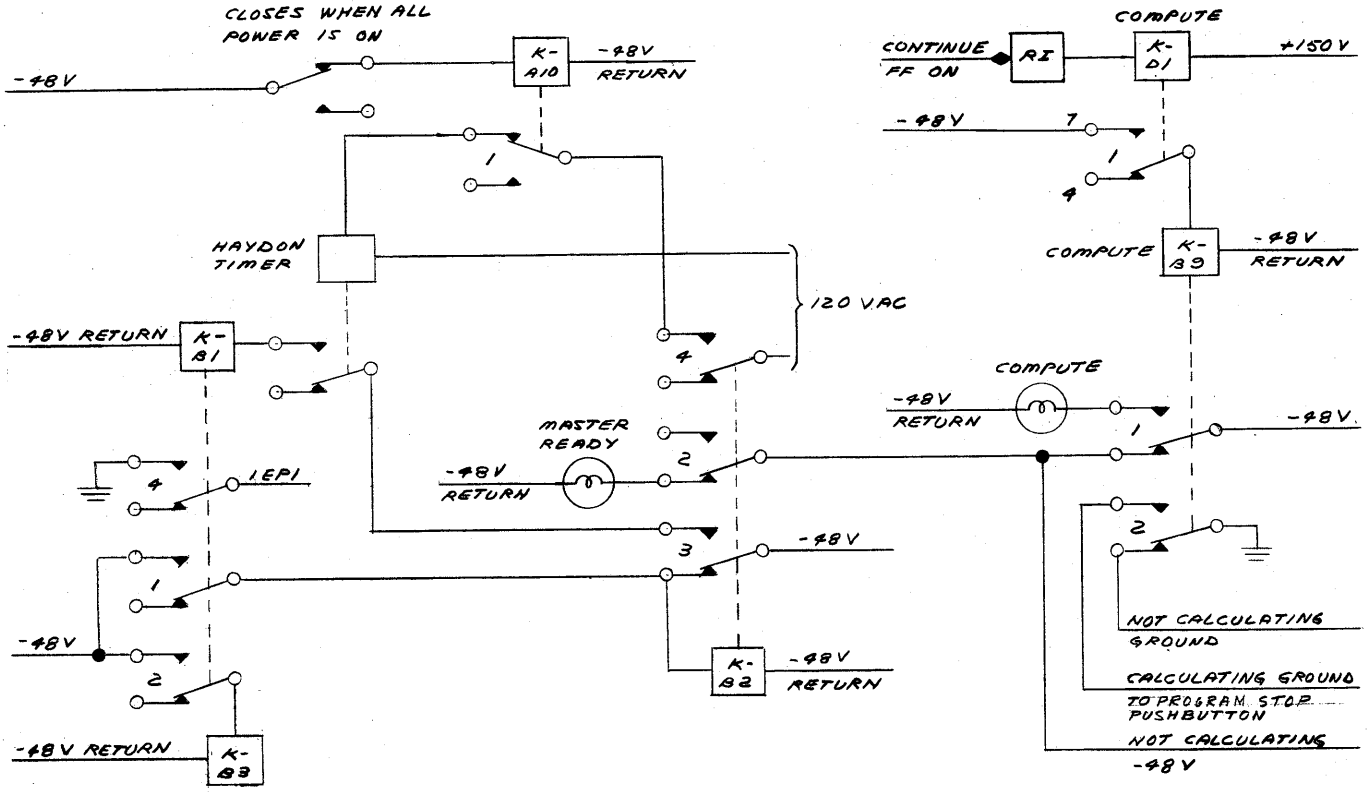


Figure 7–12. Derivation of Non-Calculating Voltages

2.3.2 Derivation of Test Voltages

Figure 7–13 illustrates the manner in which test voltages are provided for manual operations. Operation of the OPERATE-TEST switch to the TEST position provided a test ground (T GD) and a test —48 volts (T —48V) for the appropriate pushbuttons. The pushbuttons which utilize the test voltages and the relays which are energized are shown in table 7–7. At the same time that the switch is operated, the TEST light is illuminated, thus indicating to the operator the mode of operation of the Central Computer System.

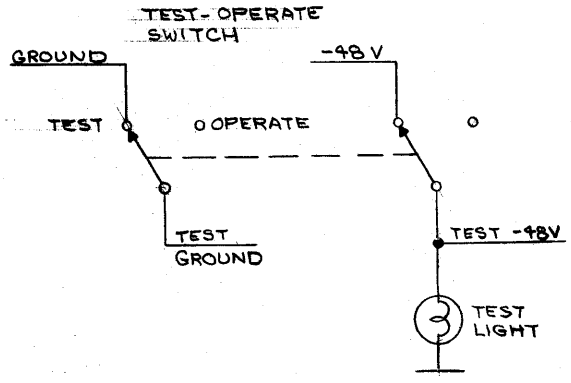


Figure 7–13. Switch Providing Test Voltages

TABLE 7–7. PUSHBUTTONS AND RELAYS USING TEST VOLTAGES

PUSHBUTTON USED	RELAY ENERGIZED
MASTER RESET	K–A5
RESET FLIP-FLOPS ..	K–A7
CLEAR MEMORY	K–A8
MEMORY CYCYE & INSTRUCTION STEP.	K–A4
	K–A6 (If relay K–A5 is energized)
	K–A9 (If relay K–A8 is energized)

2.4 PROGRAM LOADING PUSHBUTTONS

Three pushbuttons on the maintenance console provide facilities for introducing either test programs, initial operating programs, or overall control programs into the Central Computer System. These are the START FROM TEST MEMORY, LOAD FROM CARD READER, and LOAD FROM A.M. DRUMS pushbuttons. Each of these pushbuttons performs a special function in the

execution of the overall program used by the Central Computer System. For example, test memory may hold the initial or simple instructions which are used at the start of a test program. Test memory also may contain instructions to branch the program at the proper time to either the card reader, A.M. drums, tapes, etc., in order to bring in necessary additional information or to initiate the loading of utility programs. The card reader, on the other hand, inserts the initial program which starts Central Comptuer System operations. In addition, the card reader is used to introduce the complex diagnostic test programs into the Central Computer System. The auxiliary memory provides the overall master control and the timing necessary to integrate the whole program. In addition, various subroutines, data, and data tables necessary for computations are stored in auxiliary memory. These are introduced into the Central Computer System whenever the master control and timing dictate and are employed automatically by programming.

2.4.1 START FROM TEST MEMORY Pushbutton

Test memory comprises a number of plugboard registers and one flip-flop register located in the left arithmetic element and two toggle switch registers located on the maintenance console. The flip-flop register is called a live or test register while the plugboards and toggle switches are termed static registers. The live register is the only register which may have information transferred into it by the Central Computer System. The static registers contain information inserted by the operator at the maintenance console or at the left arithmetic element which may be read out of these registers by the Central Computer System.

Test memory starts with address 20,000 octal or 8192 decimal and continues sequentially from there through the 16 addresses assigned to test memory. With the test memory assignment switch on the maintenance console in the UNASSIGNED position, the 16 test memory addresses are assigned according to the plugs in the assignment portion of the test memory plugboard. Any one of the addresses may be assigned to toggle switch register A or B, to the test (flip-flop) register, or to the plugboard registers by a plug in the appropriate location. If the assignment switch is in the ASSIGNED position, the first address read $(20{,}000)_8$ is toggle switch register A, the second (20,001) toggle switch register B; octal addresses 20,002 through 20,016 are then read from the plugboard, and finally the test register $(20{,}017)_8$ is read.

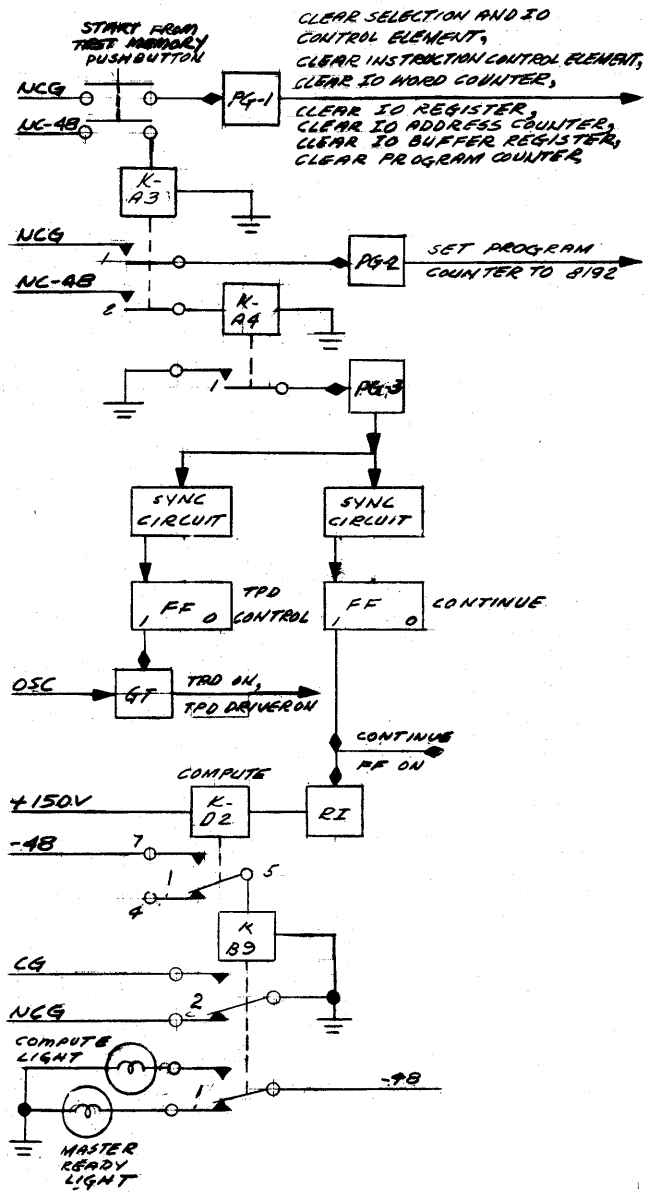


Figure 7–14. START FROM TEST MEMORY, Simplified Schematic

When the START FROM TEST MEMORY pushbutton is depressed, a non-calculating ground and non-caluculating —48 volts are simultaneously applied through the switch contacts to the pulse generator (PG–1) and relay K–A3 respectively, as shown on figure 7–14. The application of ground fires the pulse generator (PG) which then generates an output pulse used to clear the selection and IO control flip-flops, the instruction control flip-flops, the IO word counter, the IO register, the IO address counter, the IO buffer register, and the program counter. With the exception of the

program counter, all the other elements and registers are cleared in anticipation either of bringing a new instruction into the computer or of providing for a transfer between the Central Computer System and some specified IO device. The program counter is cleared so that it may be reset to the first test memory address.

The —48 volts applied to relay K–A3 energizes it, and the contacts are closed after the normal electromechanical delay. The —48 volts fed through the first contact of K–A3 is applied to and energizes K–A4. Ground is applied through the first contact of K–A3 to PG–2, firing it. The output pulse generated by PG–2 is fed to the program element and is used to set the program counter to 8192 decimal or 20,000 octal, which is the first address in test memory. At PT–1 of the ensuing instruction cycle, this address will be sent to the address register in the program element so that the information contained in this address will be transferred to the memory buffer register before PT–7 of the instruction cycle.

When K–A4 is energized and the contact is picked up, ground is applied to PG–3, firing it. The output pulse generated by PG–3 is fed through the synchronizing circuits and is used to set the TPD control and continue flip-flops. Setting these flip-flops permits the time pulse distributor to generate TP's and IP's to continue Central Computer System operations. This will cause the Central Computer System to start from the first register of test memory and branch to any point in the program. The d-c level from the continue flip-flop conditions the relay inverter (RI) which in turn energizes compute relay K–D2. When the contacts of relay K–D2 are picked up, relay K–B9 is energized by the —48 volts applied through the relay contacts and the COMPUTE light is illuminated.

2.4.2 LOAD FROM CARD READER Pushbutton

The LOAD FROM CARD READER pushbutton is used to introduce either an initial operating program or a test program into the magnetic core memory of the Central Computer System. Depressing this pushbutton causes the 24 words contained on one punch card to be loaded into the first 24 memory locations of the memory element and, in addition, causes the Central Computer System to sequentially execute the instructions contained in these words. Since these instructions may consist of a loading program directing the reading of additional cards from the card reader, depression of the LOAD FROM CARD READER pushbutton provides a convenient means of introducing any punched card information into the Central Computer System.

The execution of the various operations dictated by the LOAD FROM CARD READER pushbutton is contingent on the Central Computer System being in the non-calculating mode of operation, since the levels applied through the pushbuttons to the circuits illustrated in figure 7–14 are a non-calculating ground and a non-calculating —48 volts. These non-calculating voltage levels are derived when relay K–B9 (figure 7–14) is not energized. As shown in this figure, relay K–B9 is de-energized when the continue flip-flop is cleared, which occurs when either the PROGRAM STOP pushbutton is depressed or when the *Program Stop (HLT)* instruction is executed. This condition prevails when the COMPUTE light is off and the MASTER READY light is on.

When the above conditions are fulfilled, operating the LOAD FROM CARD READER pushbutton simultaneously applies a non-calculating ground to PG–1 and a non-calculating —48 volts to relay K–A1. (See fig. 7–15.) The applied ground fires PG–1, generating an output pulse which clears the instruction and selection and IO control element flip-flops, the IO word counter, the IO address counter, the IO buffer register, and the program counter. These are cleared in preparation for the IO word transfer which will follow.

The non-caluculating —48 volts energizes relay K–A1. Because of the mechanical inertia of K–A1, the clearing operation is completed before the contacts of K–A1 close. Closure of the first contact of K–A1 applies ground to the pulse generator in the selection and IO control element, firing it. The output pulse of the PG sets the card reader, card machine, read, and IO interlock flip-flops and sets the IO word counter to the complement of 24. The second contact of K–A1 applies ground to PG–2, generating a pulse which sets the load flip-flop and the IO interlock flip-flop in the instruction control element and sets the time pulse distributor to time level 8 (TL–8). The non-calculating —48 volts is applied through the third contact of relay K–A1 and energizes relay K–A4. When the single contact of relay 4 is picked up, ground is applied to and consequently fires PG–3. The output pulse from PG–3 is fed through the sync circuits and sets the TPD and continue flip-flops, which were previously cleared, thus permitting the TPD to generate TP and IP pulses starting with TP–8 and IP–8. (See fig. 7–8.)

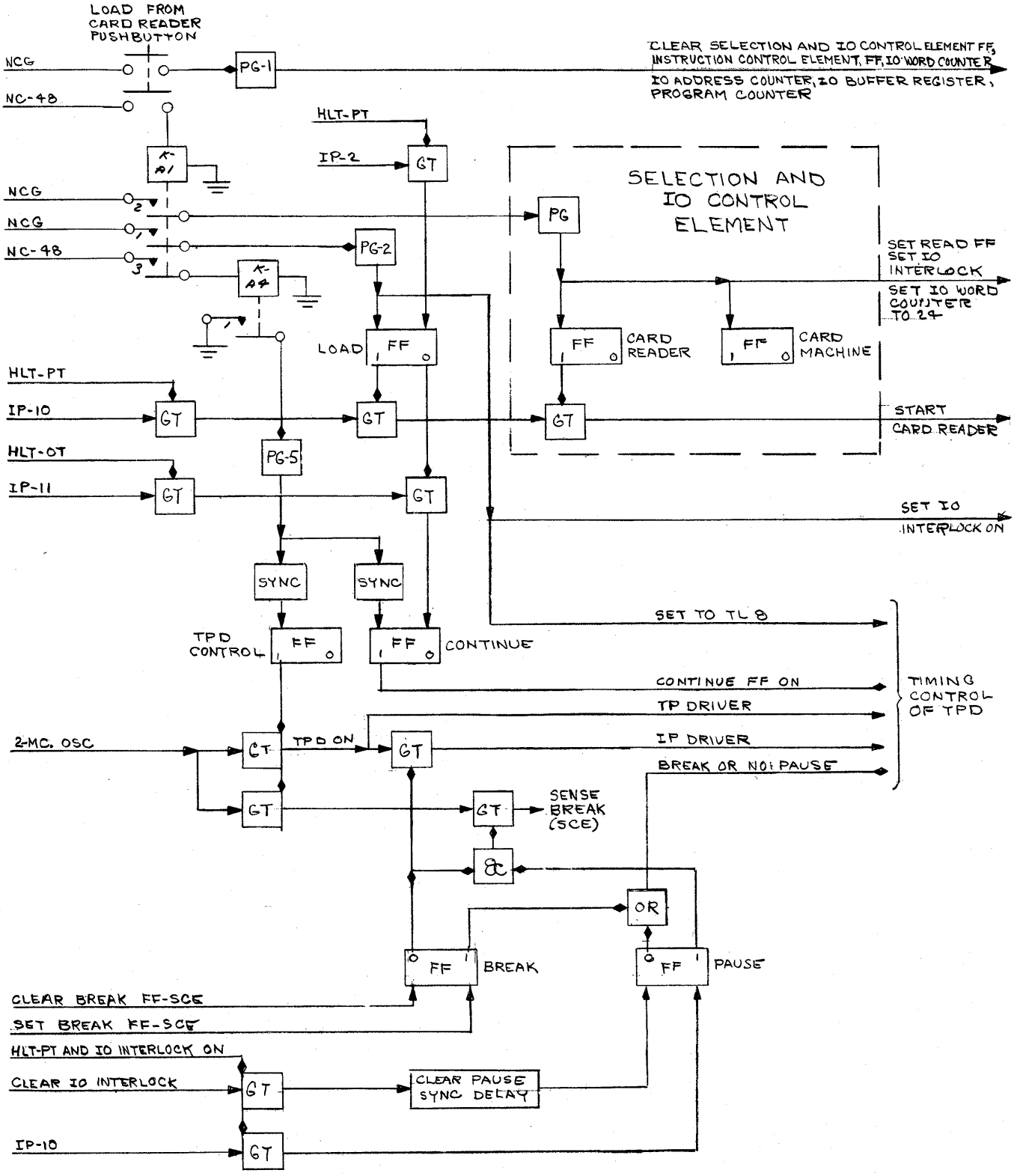


Figure 7-15. LOAD FROM CARD READER, Simplified Schematic

Since the TPD is now running, the Central Computer System will proceed to execute the instruction contained in the operation register. The operation register was previously cleared by PG–1 so that the instruction in the register consists wholly of zeros, which corresponds to a *Program Stop (HLT)* instruction. The first command pulse from the instruction control element which serves to execute the *HLT* instruction occurs at PT–10, as illustrated on the timing chart of figure 7–16. IP–10 senses two gates (figure 7–15) which are conditioned by the *HLT* instruction during program time (HLT–PT). One of these gates generates an output pulse which senses the gate conditioned by the 1 side of the load flip-flop. The output pulse from this gate, in turn, senses the gate conditioned by the 1 side of the card reader flip-flop in the selection and IO control element. The pulse is therefore gated out to the card machines as the start card reader pulse.

The second of these gates generates command pulse 134 in the instruction control element which sets the pause flip-flop. When the pause flip-flop is set, the AND circuit on TL–1 of the TPD is deconditioned so that the TPD does not cycle beyond TP–0. The Central Computer System therefore proceeds from PT–10 to OT–0 where its operations cease because no further TP or IP pulses are generated. The Central Computer System then remains in a pause condition until a break request pulse is received from the card reader, which was activated by the start card reader command at PT–10. Both the 1 side (set) of the pause flip-flop, which is set, and the 0 side of the break flip-flop, which is cleared, are combined in an AND circuit to condition a gate, as shown in figure 7–15. This gate is sensed by 2-megacycle pulses from the oscillator which are gated to sense for a break request from the card reader. (Refer to 1.1 of P–5, C–4.) When the card reader has transferred two words to the Central Computer System, this break request is generated and the break flip-flop is set. Since two words are transferred simultaneously from the card reader to the IO buffer register and the IO register in the Central Computer System, two break-in cycles must be instituted in order to store these words in magnetic core memory. These break-in cycles are initiated by the break-request pulses, which set the break flip-flop each time a break is requested. The TPD thus generates TP's but no IP's, as illustrated in figure 7–8. As shown on this figure, the 1 side of the break flip-flop conditions the AND circuit of TL–0, located in the TPD, allowing the TPD to start. However, the 0 side of the break flip-flop (which is now down) deconditions the IP driver gate. TP pulses will be generated but the generation of IP pulses will be inhibited so that no instruction cycles will be performed by the Central Computer System.

The time pulses generated by the TPD when the break flip-flop is set are sent to the selection and IO control element. This element utilizes these TP pulses to perform all the operations required to transfer the two words from their respective registers into core memory. As each word is stored in core memory, a 1 is added to the IO address counter and the IO word counter so that the former always contains the address in core memory where the next word is to be stored and the latter contains the complement of the number of words which remain to be transferred. In this manner, a total of 24 words can be read from the card reader and stored in core memory locations 0000 to 0023, and a count kept of the number of words to be read. When the first two words have been stored in core memory, the break flip-flop is cleared, thus deconditioning the AND circuit in TL–0 of the TPD. This action stops the TPD before TP–0 is generated and the Central Computer System awaits another break request from the card reader, at which time the next two words are transferred into core memory by two more break-in cycles. The sequence described (two break-in cycles followed by a pause during which time the Central Computer System awaits the receipt of the next break request) is repeated 12 times in all, so that all of the words contained on one punched card will be stored in core memory. When the last word has been transferred to core memory, the IO word counter is reduced to zero and an end-carry pulse is generated in the IO word counter. When the card reaches 13 time (trailing edge of card passing out from under the read brushes), this end-carry pulse causes the IO interlock to be cleared, a condition which in turn clears the pause flip-flop via the pause sync-delay circuits. The AND circuit on TL–0 is now fully conditioned so that TP's are generated, starting with TP–0. Since the break flip-flop is cleared, the IP driver gate is conditioned so that IP's are also generated, starting with IP–1. Commands are now generated to execute the remainder of the *HLT* instruction, starting from OT–1, as shown in the timing chart of figure 7–16.

As shown in figure 7–15, an *HLT* OT–11 pulse (command 270) usually clears the continue flip-flop during the normal execution of the *HLT* instruction, thus stopping the TPD and the Central

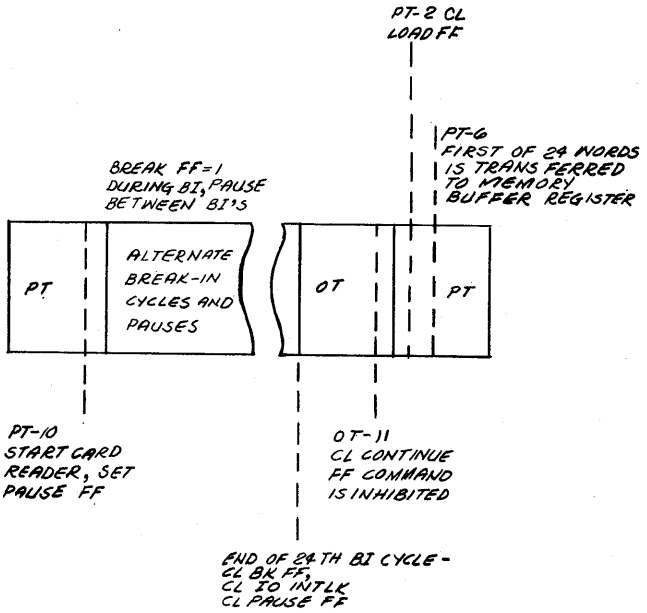


Figure 7–16. Program Stop Instruction on LOAD FROM CARD READER, Timing Chart

Computer System after TP–6 of the next machine cycle. This command, however, is gated through to clear the continue flip-flop only when the load flip-flop is cleared. Since the load flip-flop was set by PG–2, the associated gate is deconditioned and the continue flip-flop remains set. This condition negates the function of command 270 and thus inhibits the execution of the *HLT* instruction. The Central Computer System will therefore not stop at PT–6 as directed by the *HLT* instruction. The contents of the program counter at PT–1, which specify the memory location from which the next instruction is to be taken, are transferred to the memory address register. Since the contents consist wholly of zeros because the program counter was cleared by PG–1, memory location 0000 is specified as the address of the next instruction. At PT–2, the instruction control element command generators generate an *HLT* PT–2 command, clearing the load flip-flop so that any subsequent *HLT* instruction may be executed. The instruction in memory location 0000, which is the first which was written into core memory, is transferred to the memory buffer register during the read portion of the memory cycle before PT–6. At PT–7 of the next instruction cycle, the instruction in the memory buffer register is transferred into the operation and address registers for execution by the Central Computer System. At the same time, the program counter is stepped, thus specifying memory location 0001 as the next address from which an instruction is to be read and executed by the Central Computer System. Thus, after all words have been read from the card reader into core memory, the Central Computer System will proceed to sequentially execute the instructions contained in these words.

2.4.3 LOAD FROM A.M. DRUMS Pushbutton

The LOAD FROM A.M. DRUMS pushbutton enables the operator to conveniently load 24 words located on the first field of the auxiliary memory drums into the Central Computer System. These words are read into the first 24 magnetic core memory locations in sequential order. When the memory element contains the complete 24 words, the Central Computer System sequentially executes the instructions contained in these words, starting with the instruction present in memory location 0000 and proceeding through the instruction in memory location 0023. Since these instructions may introduce a loading program directing the reading of additional drum information into the Central Computer System, the LOAD FROM A.M. DRUMS pushbutton provides a flexible means of introducing a sequence of information into the Central Computer System.

Depression of the LOAD FROM A.M. DRUMS pushbutton (figure 7–17) simultaneously applies non-calculating ground to PG–1 and non-calculating —48 volts to relay K–A2. PG–1, which fires when ground is applied, generates an output pulse which clears the instruction control element and selection and IO control element flip-flops, the IO word counter, the IO address counter, the IO buffer register, and the program counter. This master clear action clears the related registers to prevent the carry-over of information from previous operations and to ready the IO units for input-output word transfers.

Relay K–A2 is energized by the applied —48 volts. After the above clearing action is completed, all three contacts of relay K–A2 are closed. Non-calculating ground is applied through the first two contacts and non-calculating —48 volts is applied through the third contact. Since contacts 1 and 3 of relay K–A1 (figure 7–15) connect directly to the same contacts on relay K–A2 (figure 7–17), the action caused by the closing of these contacts will be identical with that described in 2.4.2 of this Section and therefore will not be repeated in detail.

The ground voltage applied to the first contact fires PG–2, which generates an output pulse that sets the IO interlock and load flip-flops and sets the TPD to TL–8. Contact 2 of relay K–A2

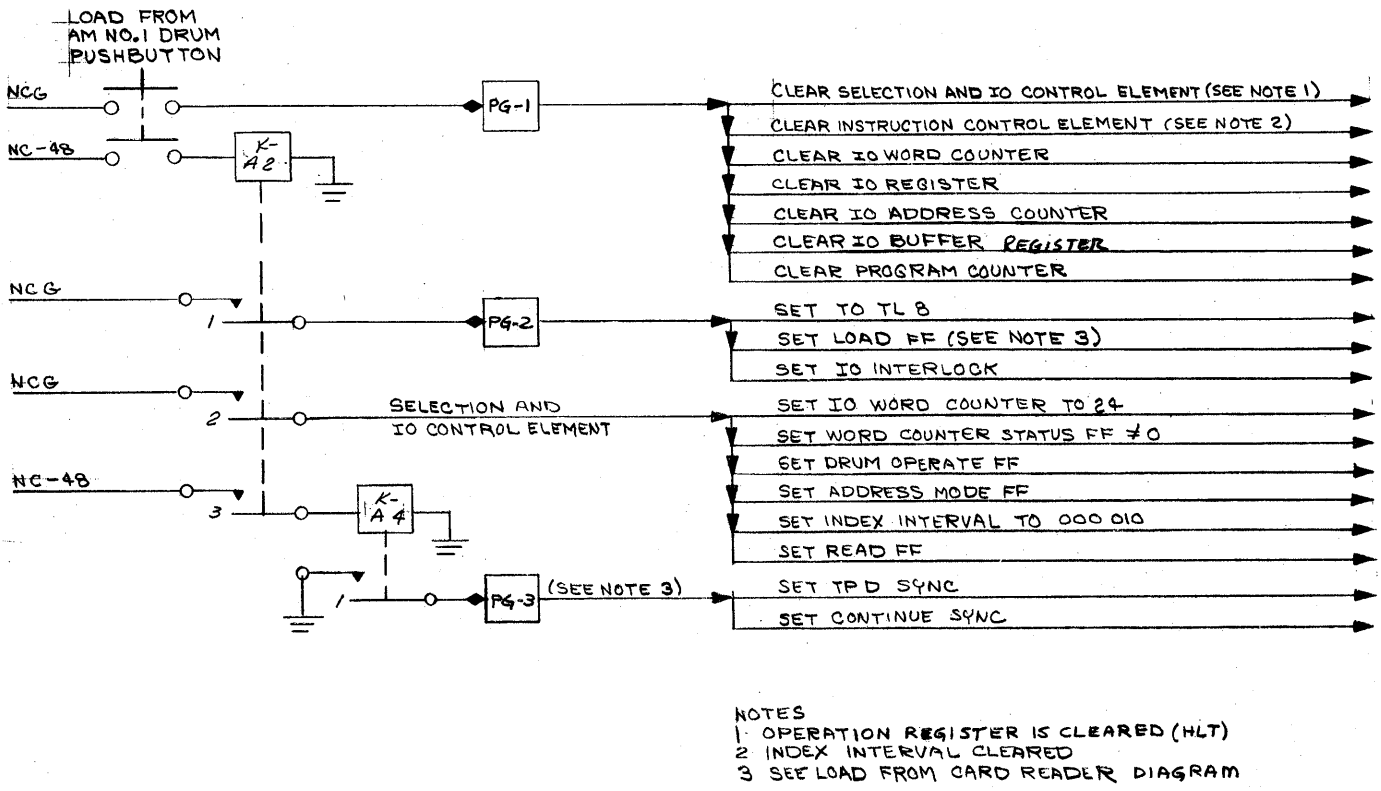


Figure 7–17. LOAD FROM A.M. DRUMS, Simplified Schematic

applies a non-calculating ground level to fire a pulse generator in the selection and IO control element. The output pulse of this generator sets the word counter status flip-flop (indicating that the IO counter contents are not 0), the IO word counter to the 1's complement of 24, and sets the drum operation and the address mode flip-flops. This pulse also sets the index interval register to 2, and sets the read and IO interlock flip-flops in the selection and IO control element. When the previous operations are completed, the TPD control and continue flip-flops are set via the action of contact 3 of relay K–A2. This starts up the TPD, which generates timing pulses beginning with TP–8 and IP–8.

Since the operation register contains all zeros, signifying a *Program Stop (HLT)* instruction a command pulse is generated by the instruction control element at PT–10 to execute the *HLT* instruction. The command pulse senses two gates. (See fig. 7–15.) One of these gates is conditioned by the *HLT* instruction; the other is conditioned by the combination of the *HLT* instruction and the presence of the IO interlock. The IP–10 pulse is gated through to set the pause flip-flop and to sense the gate conditioned by the drum operation flip-flop. The latter flip-flop, however, was set previously; hence the conditioning level is present at the gate and the sensing pulse is gated through as a start read drums pulse.

When the pause flip-flop is set, the Central Computer System executes a pause and remains in that state until the break flip-flop is set. The levels applied by the 0 side of the break flip-flop and the 1 side of the pause flip-flop are combined through an AND circuit to condition a gate tube. The 2-megacycle sensing pulses delivered by the oscillator are gated through this gate tube to sense for a break request. Meanwhile, the start read drums pulse initiates the drum read process, which begins with word transfers from the drum system to the IO buffer register. The break-request pulse sets the break flip-flop via the break-request flip-flop, concurrent with the drum word transfer from the IO buffer registers to the IO register. This activates a series of break-in cycles, during which time the drum words are sequentially transferred into their proper memory locations. When the loading operations are completed, the instructions contained in the stored words are automatically executed by the Central Computer System. A detailed explanation of an identical operation is given in 2.4.2 of this Section except that the drum operation, address mode, and index interval flip-flops described herein replace the card machine and card reader flip-flops.

2.5 PROGRAM CONTROL AND TEST

The PROGRAM STOP and PROGRAM CONTINUE pushbuttons are provided to aid program and test operations. Whenever the operator wishes to stop the program for the purpose of observation, to load operational or additional utility programs, to start test programs, etc., he merely depresses the PROGRAM STOP pushbutton. If the operator wishes to continue with the previous program, he depresses the PROGRAM CONTINUE pushbutton.

The remainder of the pushbuttons and switches discussed are used in testing operations. These test pushbuttons and switches aid the diagnostic test program by successively narrowing down the portions to be tested so that finally a specific unit is found to be at fault. The cyclic program counter, for example, tests a specific portion of the computer program. When the portion of the program which is producing a fault is found, this portion may be run through tests at the rate of one instruction at a time by using the INSTRUCTION STEP pushbutton. If a finer analysis is desired, the instruction may be broken into its composite memory cycles by depressing the MEMORY CYCLE pushbutton, etc.

2.5.1 Cyclic Program Counter

The cyclic program counter is primarily used in conjunction with diagnostic program testing. The cyclic program counter enables the Central Computer System to continuously repeat a selected portion of the program at full computer speed. Since the length of the selected portion of the program to be examined may be varied by the CYCLIC PROGRAM COUNTER octal switches, it is possible to narrow down and isolate the fault to a specific section. A cyclic program may be operated in either the DELAY or NO DELAY mode as determined by the setting of the CYCLIC PROGRAM switch. When the NO DELAY mode is selected, there is a delay of a few microseconds before the program is repeated. When the DELAY mode is selected, an additional delay due to the mechanical response time of the relay in the delay circuit is incorporated between steps. Hence the DELAY mode provides additional time to photograph and observe the flip-flop neon indicators on the maintenance console and thus determine their status before the cyclic program is repeated.

The length of program and mode of operation are determined by the manually operated switches located on the maintenance console. These switches are the CYCLE PROGRAM COUNTER octal switches and the CYCLIC PROGRAM switch. The CYCLE PROGRAM COUNTER octal switches contain a nine digit binary number which is read into the cyclic program counter located in the instruction control element. Essentially, the counter serially subtracts the number of 2-megacycle or instruction pulses used in the program from the contents of the octal switches. When the remainder is zero, an end-carry pulse is generated which halts the Central Computer System momentarily, clears its registers, sets the program counter to $8192|_{10}$ which is the first address in test memory, and transfers the initial number back into the cyclic program counter. The Central Computer System then recycles through the original program, starting from the first instruction stored in test memory.

The cyclic program counter features three 3-pole, 8-position, ganged switches. A potential of either +10 or −30 volts is applied to each octal switch contact so that, for a given position, a specific combination of nine voltages is selected and fed into the nine read-in gates shown in figure 7–18. The gates which receive +10 volts will be conditioned (up) and will therefore produce an output pulse when sensed; those gates which receive −30 volts will be deconditioned and will fail to produce an output pulse. The read-in sensing pulse is applied to the nine gates simultaneously so that each conditioned gate passes an output pulse which sets its associated flip-flop. In the set status, the counting flip-flop represents a binary digit equal to 1. Those counter flip-flops which are connected to the deconditioned gates receive no pulse but remain in the clear (off) status and represent a binary digit equal to 0. The initial status of the nine counting flip-flops before the subtraction process takes place represents the nine-digit binary number which determines the length of the program to be examined. For example, if the CYCLE PROGRAM COUNTER octal switches are in the position which conditions all nine read-in gates, then a read-in sensing pulse will set all nine counting flip-flops. (See fig. 7–18.) This setting denotes the binary number 111 111 111 which corresponds to the decimal number 511. The length of the program examined will therefore consist of approximately 511 instruction pulses.

Figure 7–19 shows three typical stages of the nine-stage counter. Figure 7–20 is a chart illustrating the sequence of counting action for these three stages when the counting flip-flops are set (each flip-flop contains a 1). It should be noted that the carry gates (figure 7–19) are connected

to the 0 side of the counting flip-flops so that these gates are deconditioned when the flip-flops are set. Since the applied instruction pulses sense the carry gate and complement the counting flip-flop simultaneously, the first stage alternately changes status with each input pulse. Thus, if the first input pulse finds carry gate 1 down, there is no output pulse but flip-flop 1 is complemented. When flip-flop 1 has settled (in less than half a microsecond), it is cleared to zero, thereby conditioning gate 1. The second instruction pulse applied, therefore, is gated through to the second stage and also complements flip-flop 1, setting that flip-flop to 1. As indicated on figure 7–20, every second pulse is gated through from the first to the second stage. Since carry gate 2 is deconditioned, the gated input pulse cannot pass through this gate. However, this pulse complements flip-flop 2, conditioning carry gate 2 within half a microsecond. The third instruction pulse complements flip-flop 1, thus clearing it and conditioning carry gate 1. The fourth instruction pulse is gated through conditioned gates 1 and 2 and is applied to deconditioned gate 3, preventing its further progress. In addition, counting flip-flops 1, 2, and 3 are complemented by this pulse so that flip-flops 1 and 2 are set and flip-flop 3 is cleared. Referring again to figure 7–20, it should be noted that every second instruction pulse changes the status of the second stage and that every fourth input pulse produces an output pulse which is fed to the third stage. Instruction pulses five, six, and seven arrange the status of the counting flip-flops so that carry gates 1, 2, and 3 are up. For the condition that was assumed in the three-stage counter shown in figure 7–19, the eighth 2-megacycle instruction pulse ripples through carry gates 1, 2, and 3 to produce an end-carry pulse. This action is typical for any counter; the instruction pulses continue until all the carry gates are conditioned and the next stepping pulse propagates through all the conditioned gates to generate the end-carry pulse. Any number between 0 and 2^N-1 may be read into the cyclic program counter, where N is the number of stages. Actually, the total number of instruction pulses used to execute the program is greater by one or two pulses than the number read into the binary counter because there is an inherent delay in the cyclic program counter control circuits before the counting action starts.

A cycle program may be operated in either of two modes as determined by the setting of the CYCLIC PROGRAM control (CPC) switch shown on figure 7–19. If this switch is initially in the OFF position, there will be no cycling action since relay 18 wll not be energized and the continue and CPC control flip-flops will remain cleared. However, when the CYCLIC PROGRAM control switch is rotated to the DELAY or NO DELAY positions, —48 volts is applied to relay 18, energizing the relay. After the relay contact has picked up, ground is applied to PG–6, firing it. The pulse generated by PG–6 clears the IO buffer registers, the IO word counter, the IO register, the IO address counter, the program counter, the flip-flops in the selection and IO element, and the flip-flops in the instruction control element. After a 1-microsecond delay, the read-in gates are sensed; the program counter is set to 8192_{10} (first address in test memory) and TPD control, CPC control, and continue flip-flops are set. When the CPC control flip-flop is set, it conditions a gate which is sensed at a 2-megacycle rate, and the gated pulse provides the 2-megacycle or IP timing pulses for the counting action. The Central Computer System then starts from test memory and branches to the selected portion of the program at full computer speed, until an end-carry pulse is generated. This end-carry pulse clears the TPD control flip-flop, thereby momentarily halting the production of TP and IP pulses, simultaneously sensing the off, delay, and no delay gates shown on figure 7–19. In the DELAY position of the CYCLIC PROGRAM control switch, the delay gate is continuously conditioned and the end-carry pulse is gated to the mechanical delay flip-flop, to set that flip-flop and thus activate mechanical delay relay K–D1. When the relay contact is picked up, PG–8 is activated and the generated pulse then clears the mechanical delay flip-flop, removing the energizing source from relay K–D1. When the relay contact drops back, ground is applied to PG–9, firing it and producing an output pulse which performs two functions. First, it performs the same master clear of the registers and flip-flops previously mentioned. Second, it passes through a 1-microsecond delay circuit, after which the pulse sets the TPD control, CPC control, and continue flip-flops; sets the program counter to 8192_{10}, and senses the read-in gates. This allows the Central Computer System to start counting and to recycle through the original program contained in the octal switches. When the NO DELAY mode is selected, the no delay gate is continuously conditioned, and an end-carry pulse is gated through a 1-microsecond delay circuit. The same clearing and setting mentioned above takes place when the switch is in the DELAY position except that the mechanical delay

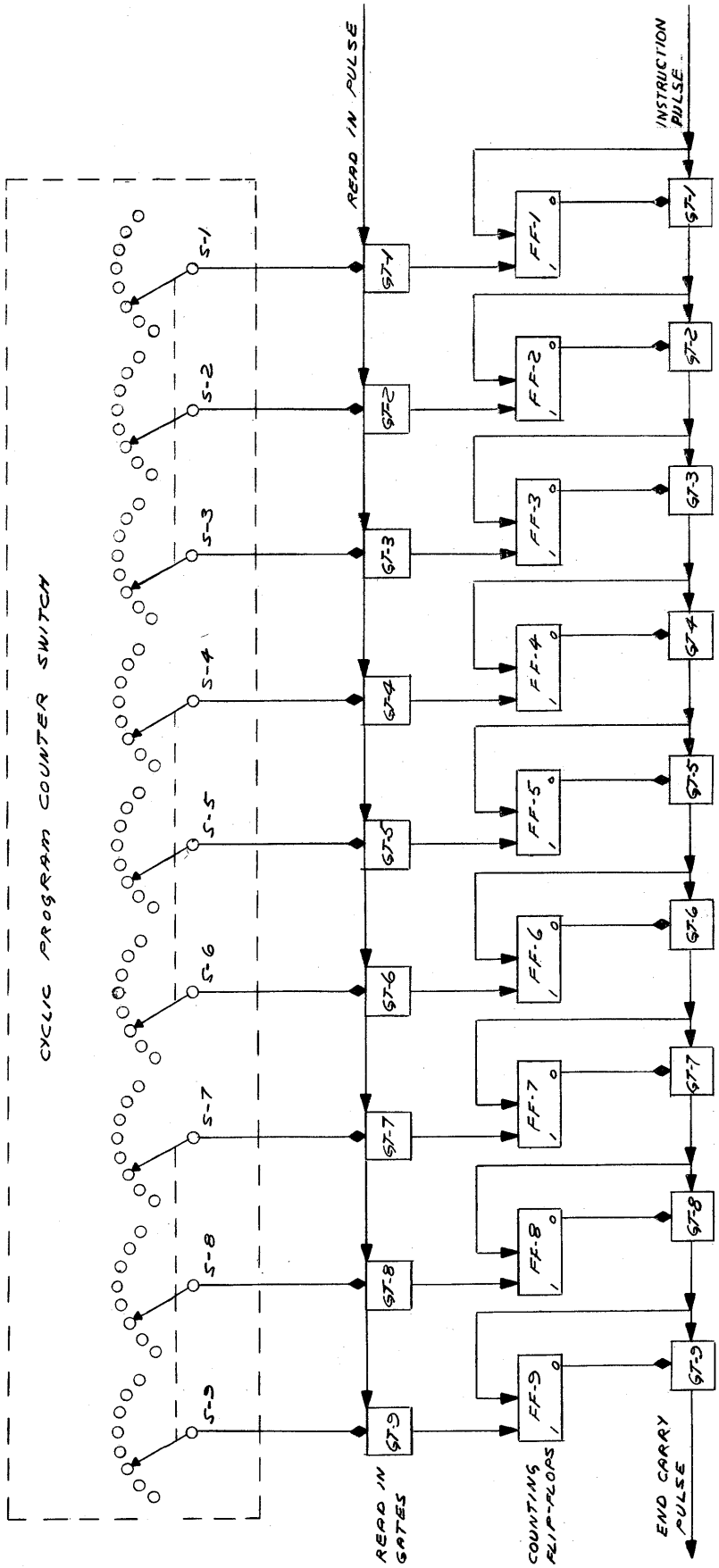


Figure 7–18. Cyclic Program Counter, Counting Circuit, Block Diagram

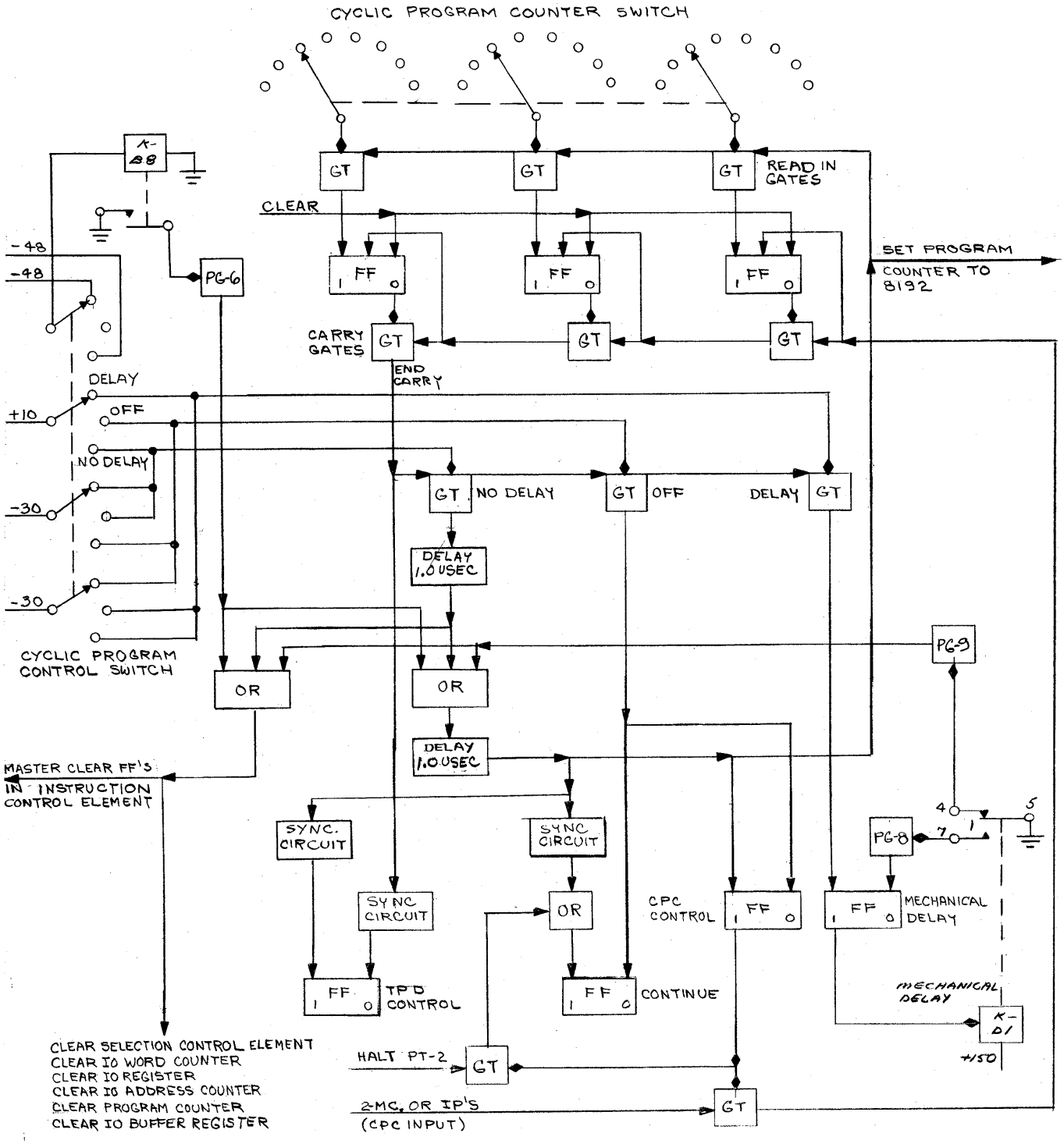


Figure 7–19. Cyclic Program Counter and Control, Simplified Schematic

flip-flop and the relay are not involved. Since the total time involved when the relay and flip-flop are used is a function of the relay response time, the DELAY mode provides an additional time interval during which it is possible to photograph the status of the flip-flop neon indicators. The cyclic program counting action may be halted at the end of any program by manually turning the CPC switch to the OFF position. The end-carry pulse will be gated through the off gate to clear the CPC control and continue flip-flops, thereby halting the Central Computer System.

PULSE NO.	1.	2.	3	4	5	6	7	8
BIT 1	FF-1 SET TO 1	FF-1 CLEAR TO 0	FF-1 SET TO 1	FF-1 CLEAR TO 0	FF-1 SET TO 1	FF-1 CLEAR TO 0	FF-1 SET TO 1	FF-1 CLEAR TO 0
	CARRY GATE 1 DOWN	CARRY GATE 1 UP	CARRY GATE 1 DOWN	CARRY GATE 1 UP	CARRY GATE 1 DOWN	CARRY GATE 1 UP	CARRY GATE 1 DOWN	CARRY GATE 1 UP
	NO PULSE OUT	PULSE OUT	NO PULSE OUT	PULSE OUT	NO PULSE OUT	PULSE OUT	NO PULSE OUT	PULSE OUT
	COMPL. FF-1 CLEAR TO 0	COMPL. FF-1 SET TO 1	COMPL. FF-1 CLEAR TO 0	COMPL. FF-1 SET TO 1	COMPL. FF-1 CLEAR TO 0	COMPL. FF-1 SET TO 1	COMPL. FF-1 CLEAR TO 0	COMPL. FF-1 SET TO 1
BIT 2	FF-2 SET TO 1	FF-2 SET TO 1	FF-2 CLEAR TO 0	FF-2 CLEAR TO 0	FF-2 SET TO 1	FF-2 SET TO 1	FF-2 CLEAR TO 0	FF-2 CLEAR TO 0
	CARRY GATE 2 DOWN	CARRY GATE 2 DOWN	CARRY GATE 2 UP	CARRY GATE 2 UP	CARRY GATE 2 DOWN	CARRY GATE 2 DOWN	CARRY GATE 2 UP	CARRY GATE 2 UP
	NO PULSE OUT	NO PULSE OUT	NO PULSE OUT	PULSE OUT	NO PULSE OUT	NO PULSE OUT	NO PULSE OUT	PULSE OUT
		COMPL. FF-2 CLEAR TO 0		COMPL. FF-2 SET TO 1		COMPL. FF-2 CLEAR TO 0		COMPL. FF-2 SET TO 1
BIT 3	FF-3 SET TO 1	FF-3 SET TO 1	FF-3 SET TO 1	FF-3 SET TO 1	FF-3 CLEAR TO 0	FF-3 CLEAR TO 0	FF-3 CLEAR TO 0	FF-3 CLEAR TO 0
	CARRY GATE 3 DOWN	CARRY GATE 3 DOWN	CARRY GATE 3 DOWN	CARRY GATE 3 DOWN	CARRY GATE 3 UP	CARRY GATE 3 UP	CARRY GATE 3 UP	CARRY GATE 3 UP
				NO PULSE OUT				PULSE OUT
				COMPL. FF-3 CLEAR TO 0				COMPL. FF-3 SET TO 1
								END CARRY

Figure 7–20. Cyclic Program Counter, Action Sequence Chart

2.5.2 PROGRAM STOP Pushbutton

The PROGRAM STOP pushbutton provides a means of manually halting the Central Computer System after the instruction it contains is executed. When the PROGRAM STOP pushbutton is depressed, ground is applied to PG–4, firing that generator and producing a pulse which sets the instruction step flip-flop. Thus, gate 1 (figure 7–21) is conditioned so that the PT–5 sensing pulse is gated through to clear the instruction step and continue flip-flops at TL–5 and at sense gate 2 simultaneously and clear the TPD control flip-flop at TL–6. Further operation is dependent on the status of the IO interlock. If the IO interlock is off, the Central Computer System halts at TL–7. However, if input-output operations are in progress, no new instructions will be executed and all the necessary input-output breaks are completed before the Central Computer System stops. Since the IO interlock is on during input-output operations, gate 2 is conditioned. The PT–5 sensing pulse is therefore gated through a 1-microsecond delay circuit to set the instruction step, continue, and TPD control flip-flops and to set the TPD to TL–8. Since the TPD begins transmission of TP and IP pulses at TL–8, the PT–7 pulse is suppressed until the following memory cycle to enable the Central Computer System to complete the contained instruction before stepping the program counter at PT–7, thus specifying the address of the next instruction. The above operation is continuous, occurring every PT–5, until the IO interlock is off. When the IO interlock conditioning level is removed, the instruction step, continue,

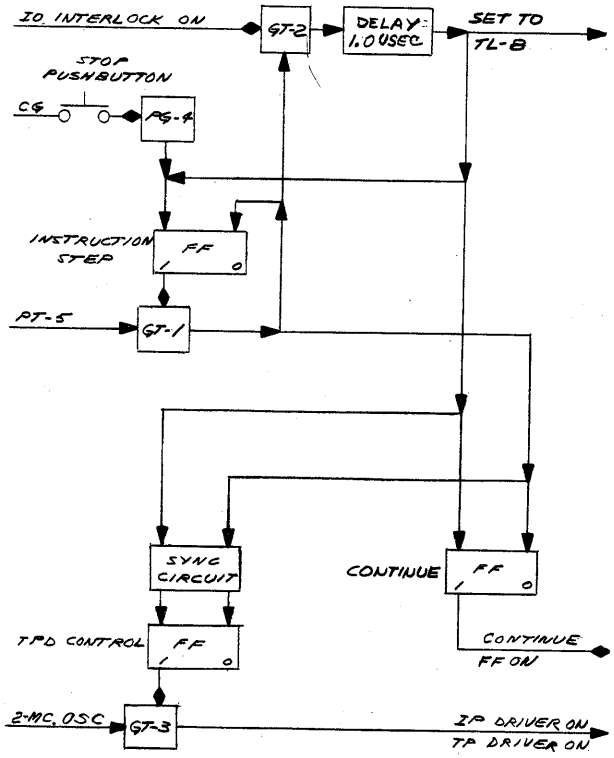


Figure 7–21. PROGRAM STOP Pushbutton Circuit, Simplified Schematic

and TPD control flip-flops are cleared at PT–5 and the Central Computer System halts at TL–7.

2.5.3 PROGRAM CONTINUE Pushbutton

The PROGRAM CONTINUE pushbutton provides a convenient means of restarting the Central Computer System since, when the latter is halted for any reason whatsoever, the operator may depress the PROGRAM CONTINUE pushbutton and thereby resume the execution of the program, starting from the point of interruption. The instruction present in the memory buffer registers is then transferred to the operation register for decoding and execution.

When the PROGRAM CONTINUE pushbutton is depressed, ground is applied to PG–5, firing that generator. (See fig. 7–22.) The output pulse generated by PG–5 is applied through the synchronizing circuits to set the TPD control and continue flip-flops. This starts the generation of TP and IP pulses at TL–7 and the instruction contained in the memory buffer registers is transferred to the operation registers at this time. The automatic execution of the full program is continued, therefore, until a *Program Stop (HLT)* instruction is received by the Central Computer

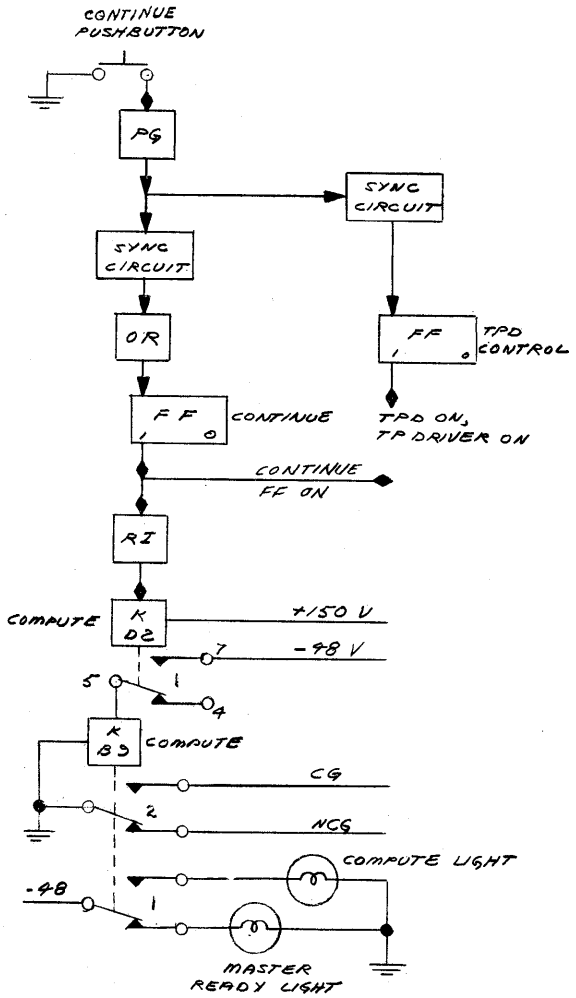


Figure 7–22. PROGRAM CONTINUE Pushbutton Circuit, Simplified Schematic

System or the PROGRAM STOP pushbutton is depressed.

2.5.4 INSTRUCTION STEP Pushbutton

The INSTRUCTION STEP pushbutton allows the operator at the maintenance console to advance the program one instruction at a time after the Central Computer System has been halted, thus testing those circuits which are involved in executing the one particular instruction which has just been stepped. This pushbutton is therefore used primarily in conjunction with diagnostic program testing.

Depression of the INSTRUCTION STEP pushbutton simultaneously applies test ground to PG–4, firing this generator, and test —48 volts to relay 4, energizing it. (See fig. 7–23.) The output pulse generated by PG–4 sets the instruction step flip-flop, thereby conditioning gate tube 1. Since

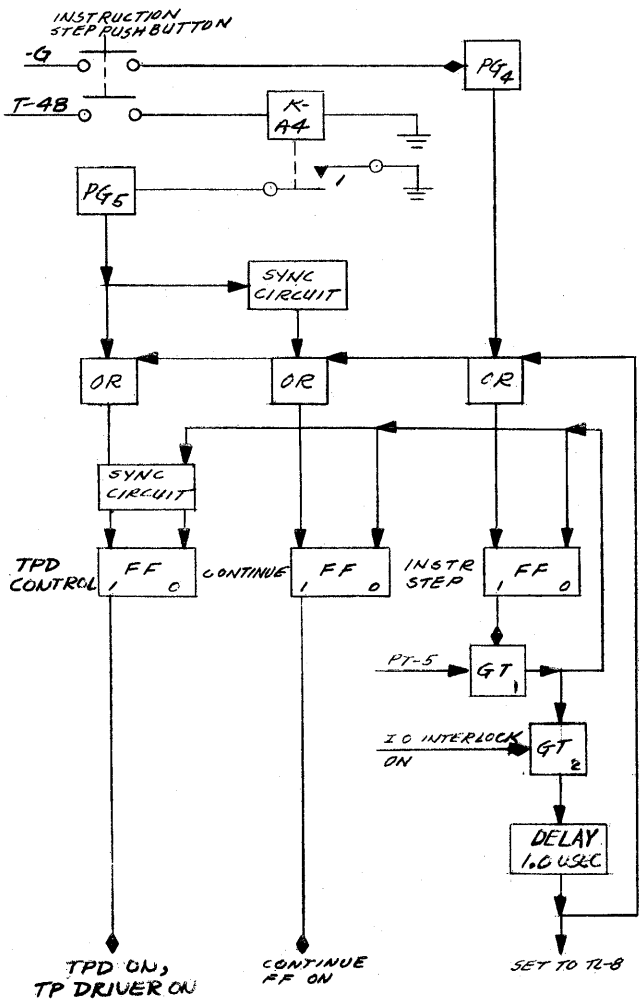


Figure 7–23. INSTRUCTION STEP Pushbutton Circuit, Simplified Schematic

the Central Computer System is halted before the pushbutton is depressed, the TPD control and continue flip-flops are cleared, thus halting the generation of TP and IP pulses and the PT–5 input pulse which normally senses gate tube 1 is not available. After the mechanical delay time, the contacts of relay 4 close. Ground is applied to PG–5, firing it. PG–5 then generates a pulse that sets the TPD control and continue flip-flops. This starts the generation of TP and IP pulses at TL–7 so that the Central Computer System executes a complete instruction cycle. The next PT–5 sensing pulse is gated through to sense gate tube 2 to clear the instruction step, TPD control, and continue flip-flops, thus deconditioning the AND circuit on TPD–6. The Central Computer System therefore halts after TL–6. Further operation is dependent on the status of the IO interlock.

If the IO interlock is off, no further operation takes place until the operator depresses the INSTRUCTION STEP pushbutton and starts the Central Computer System at TL–7. If the IO interlock is on, however, gate tube 2 (figure 7–23) is conditioned, and the PT–5 sensing pulse is gated through. After 1 microsecond, the TPD is set to TL–8, and the instruction step, TPD control, and continue flip-flops are set by the gated pulse. This circuit prevents the Central Computer System from halting during input-output operations and also inhibits the generation of the PT–7 pulse. This pulse transfers the instruction contained in the memory buffer registers into the operation and address registers. Since the operation register is cleared at PT–6, it contains a *Program Stop* instruction which the Central Computer System now attempts to execute. However, because the IO interlock is on, the Central Computer System goes into a pause and will remain in the pause condition for the duration of the input-output operations. Once the input-output operations cease, the IO interlock is turned off, and the next PT–5 sensing pulse unconditionally halts the Central Computer System.

2.5.5 MEMORY CYCLE Pushbutton

The MEMORY CYCLE pushbutton allows the Central Computer System to advance by one memory cycle at a time after the computer has been halted. Each memory cycle is synchronized with a computer machine cycle so that each time a memory cycle is executed the computer program is advanced by one machine cycle. If the Central Computer System is to execute a *Store* instruction containing three machine cycles, PT, OTA, and OTB, then three memory cycles are required to execute this instruction. The first time that the MEMORY CYCLE pushbutton is depressed, a PT cycle is performed and the instruction is brought from memory to the address and operation registers. The second time the MEMORY CYCLE pushbutton is depressed, an OTA cycle is performed and an operand is brought from memory. The third time this pushbutton is depressed, an OTB cycle is performed in which the data is stored in core memory. The MEMORY CYCLE pushbutton is therefore used to test only those Central Computer System circuits involved in the execution of one machine cycle of an instruction, which, in effect, provides a finer breakdown for testing than does the operation of the INSTRUCTION STEP pushbutton.

Depressing the MEMORY CYCLE pushbutton simultaneously applies test ground to PG–11

and test —48 volts to relay 4. (See fig. 7–24.) The pulse generator is fired and produces an output pulse which sets the memory cycle flip-flop, thereby conditioning gate tube 1. The application of test —48 volts energizes relay 4. When the relay contact is picked up, ground is applied to PG–5, firing that generator. The output pulse generated by PG–5 sets the TPD control and continue flip-flops causing the TPD to generate TP and IP pulses starting from TL–7.

The instruction contained in the memory buffer register is transferred at PT–7 to the operation and address registers. The IP pulses generated by the TPD then execute that portion of the instruction comprised of one machine cycle (PT–7 to PT–6 or OT–6). At PT–1 or OT–1 either a new instruction address or the operand address is specified by the program counter or the address register respectively, and this magnetic core memory address is transferred to the memory address register. In addition, the register which specifies the magnetic core memory address also generates a start magnetic core memory No. 1 or No. 2 pulse, so that a memory cycle is initiated in synchronism with the machine cycle.

Gate tube 1, conditioned by the memory cycle flip-flop, is sensed by the IP–5 pulse, which is gated through to clear the memory cycle and the continue flip-flops. Since the AND circuit on TL–7 is deconditioned by the cleared continue flip-flop, the Central Computer System halts after TP–6 and IP–6. The memory cycle which was initiated at PT–1 or OT–1, however, cannot be stopped. Magnetic core memory therefore goes through the read portion of the memory cycle, during which time the data or instruction in the specified memory address is transferred to the memory buffer register before PT–7, then through the write portion of the cycle, so that a complete memory cycle is executed even though the Central Computer System has been halted at TP–7.

During an OTB cycle the computer information held in the A register of the arithmetic element is transferred back into memory. The address register specifies the address in core memory into which the data will be written. Further, this address is the identical address from which the operand was read during the previous OTA cycle. Magnetic core memory is started and the address is designated at OT–1. At OT–2, the data is transferred from the A register to the memory buffer register. When the write portion of the memory cycle is reached, the data held in the

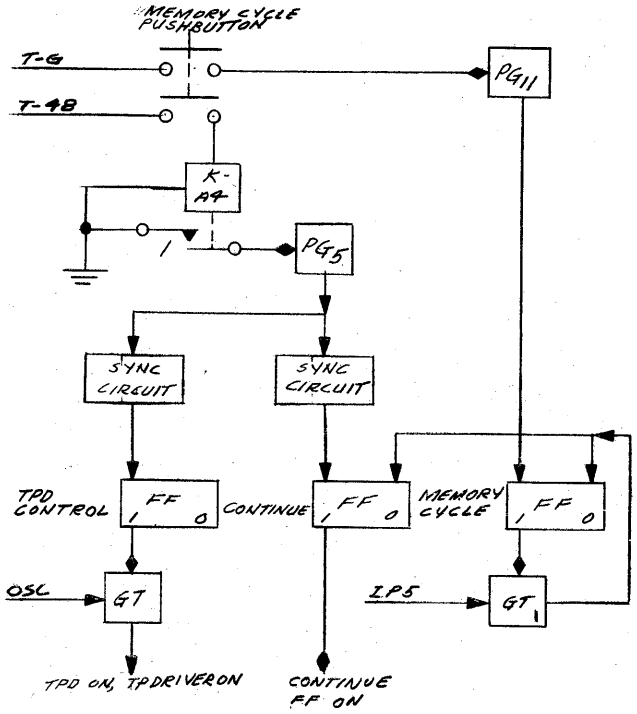


Figure 7–24. MEMORY CYCLE Pushbutton Circuit, Simplified Schematic

memory buffer register will be written into core memory.

2.5.6 CLEAR MEMORY Pushbutton

The CLEAR MEMORY pushbutton enables the operator at the maintenance console to clear all of the 8192_{10} magnetic core memory locations. When the CLEAR MEMORY pushbutton is depressed, test ground is applied to pulse generator 3 (PG–3), firing that generator. In addition, test —48 volts is applied to relay K–A8, activating that relay as illustrated in figure 7–25. The PG–3 output pulse clears the TPD control flip-flop, and thus turns off the TPD so that no TP or IP pulses are generated. When the contacts of relay 8 close, the first contact applies test ground which activates PG–7. An output pulse is therefore generated which clears the instruction control element, the selection and IO control element, the program counter, the IO word counter, the IO address counter, the IO buffer register, and the IO register. Test —48 volts is applied through the second contact of relay K–A8 to energize relay K–A9. After the relay delay time, the contacts of relay K–A9 close, applying test ground to PG–10 and test —48 volts to relay K–A4. Pulse generator 10 fires when ground is applied, thus generating a

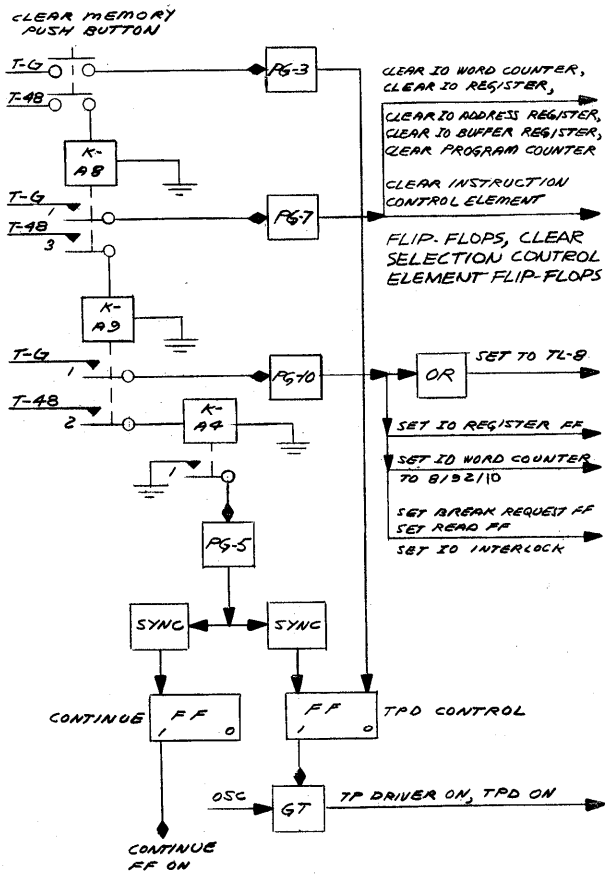


Figure 7–25. CLEAR MEMORY Pushbutton Circuit, Simplified Schematic

pulse which sets the IO word counter to 8192_{10} and sets the IO interlock, the IO register, the break request, and the read flip-flops in the selection and IO control element. The setting of these flip-flops in the selection and IO control element prepares the Central Computer System for the break-in cycles when the next TP–11 pulse occurs. The operation of the selection and IO controls are explained in detail in Part 5.

When the contacts of relay K–A4 are picked up, PG–5 is activated, generating an output pulse which sets the TPD control and the continue flip-flops. This causes the TPD to start up at TL–8. Since the break-request flip-flop is set (refer to 2.4.2 of this Section), the sense-break flip-flop gate is conditioned, so that the TP–11 pulse is gated through to set the break flip-flop. Since the break, IO interlock, and read flip-flops all contain 1, a break-in cycle is initiated. Furthermore, because the IO word counter is set to 8192_{10}, 8192_{10} break-in cycles will be performed. With the IO register in the zero condition due to the previous clearing action, the 8192_{10} break-in cycles will sequentially transfer 8192_{10} words containing all zeros into magnetic core memory and thus effectively clear core memory. The instruction contained in core memory then consists solely of zeros and represents a *Program Stop (HLT)* instruction. However, the operation register already contains all zeros since the instruction control flip-flops were previously cleared by the pulse from PG–7. A *Program Stop (HLT)* instruction therefore exists in the operation register. The Central Computer System attempts to execute this *Program Stop* instruction but is prevented from doing so because the IO interlock is on. After the 8192_{10} break-in cycles are completed, the IO word counter is stepped down to zero and an end-carry pulse is generated which turns off the IO interlock and halts the computer. It can thus be seen that depressing the CLEAR MEMORY pushbutton is analogous to executing the following three instructions:

LDC	0000	Load IO address counter
SEL	$(04)_8$	Select IO register
RDS	8192	Read 8192_{10} words

2.5.7 RESET FLIP-FLOPS Pushbutton

The RESET FLIP-FLOPS pushbutton restores to the cleared condition those control flip-flops and flip-flop registers not ordinarily cleared in normal operation. Since it is essential that these flip-flops shall not carry over unwanted information from the previous program, this clearing action is necessary prior to the initiation of a new program. The operator at the maintenance console can clear these flip-flops by depressing the RESET FLIP-FLOPS pushbutton, thus ensuring that information and data from the previous program will not affect the new program. This action is also performed prior to the use of the COMPLEMENT pushbutton and switch described in 2.2.8 of this Section, so that the status of the control and register flip-flops is known before the complementing action takes place.

Upon depression of the RESET FLIP-FLOPS pushbutton (figure 7–26), test ground is applied to PG–3, firing the generator, and test –48 volts is applied to relay K–A7, energizing the relay. The pulse generated by PG–3 clears the TPD control flip-flop, thus halting the generation of TP and IP pulses. When the three contacts of relay K–A7 close, test ground is applied through the first contact to PG–7 and through the latter two contacts to the left and right arithmetic elements. The test

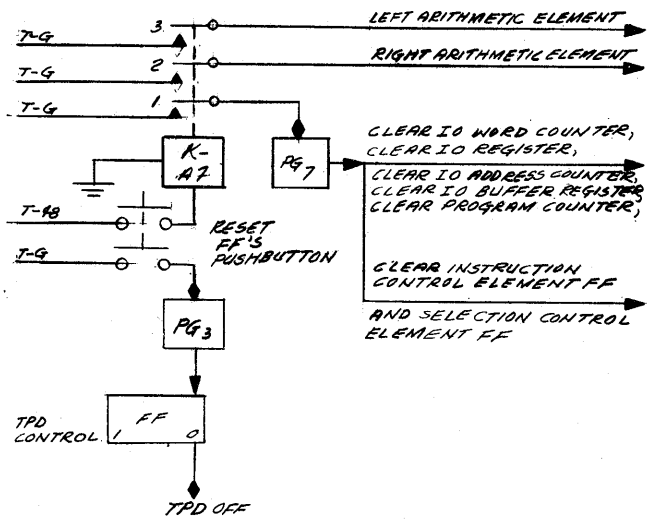


Figure 7–26. RESET FLIP-FLOPS Pushbutton Circuit, Simplified Schematic

ground voltage applied to PG–7 causes the pulse generator to fire, producing an output pulse which clears the IO word counter, the IO register, the IO address counter, the IO buffer register, and the program counter. In addition, this pulse clears selected flip-flops in the selection and IO control and instruction control elements. The test ground voltage applied through the latter two contacts clears all flip-flops in the left and right arithmetic elements except those listed in table 7–8.

TABLE 7–8. ARITHMETIC ELEMENT FLIP-FLOPS NOT CLEARED BY THE RESET FLIP-FLOPS PUSHBUTTON

REGISTER	FLIP-FLOP
Left adder	Bit S—Carry storage flip-flop
Right adder	Bit S—Carry storage flip-flop
Left accumulator	Bit S—Sign control flip-flop
Left IO register	Parity flip-flop Sign bit flip-flop
Left IO register	Bits 1 through 15 flip-flops
Right IO register	Sign bit Card machine thyratron buffer control flip-flop
Right IO register	Bits 1 through 15 flip-flops

2.5.8 MASTER RESET Pushbutton

The MASTER RESET pushbutton simultaneously combines the operations enacted by the depression of the CLEAR MEMORY pushbutton discussed in 2.5.6 of this Section, the READY IO UNITS pushbutton in 2.7.1 of this Section, and the RESET FLIP-FLOPS pushbutton explained in 2.5.7 of this Section. The operator at the maintenance console can therefore conveniently prepare the IO units, the memory element, and the Central Central Computer System for a new program merely by depressing this pushbutton.

When the MASTER RESET pushbutton is depressed, test ground is applied to PG–3, firing the pulse generator, and test –48 volts is applied to relay K–A5, energizing the relay. (See fig. 7–27.) The pulse generated by PG–3 clears the TPD control flip-flop, preventing further generation of TP and IP pulses. This same clearing action is produced by the depression of either the RESET FLIP-FLOPS or CLEAR MEMORY pushbuttons, since the line activating PG–3 is common to all three pushbuttons. After the contacts of relay 5 close, test –48 volts is applied to relays K–A6, –7, and –8, energizing them. Each relay is also energized separately, for it may be seen from figure 7–31 that relay K–A6 is energized by the READY IO UNITS pushbutton; from figure 7–26 that relay K–A7 is energized by the RESET FLIP-FLOPS pushbutton; and from figure 7–25 that relay K–A8 is energized by the CLEAR MEMORY pushbutton. Since the features of the CLEAR MEMORY, READY IO UNITS and RESET FLIP-FLOPS pushbuttons are incorporated into the MASTER RESET pushbutton, the circuit action effected by this pushbutton is identical to the action produced by the combination of these three pushbuttons. Proper understanding of the circuit operation may be obtained by an examination of the detailed explanation of the operation of each of these three pushbuttons.

2.5.9 COMPLEMENT Pushbutton and Switch

The COMPLEMENT pushbutton complements certain control flip-flops and flip-flop registers in the Central Computer System either twice or three times depending on the position of the COMPLEMENT switch. When the COMPLEMENT pushbutton is depressed, all control flip-flops in the selection and IO control and instruction control elements except those listed in table 7–9 are simultaneously complemented. At the same time, all flip-flops in the operation register, the index interval register, and the cycle control are complemented. One-half microsecond later, another complement pulse is applied to these control flip-flops and flip-flop registers.

The operator at the maintenance console may observe the change in status of the flip-flop neon indicators and thus detect improper circuit performance. Since the diagnostic programs cannot

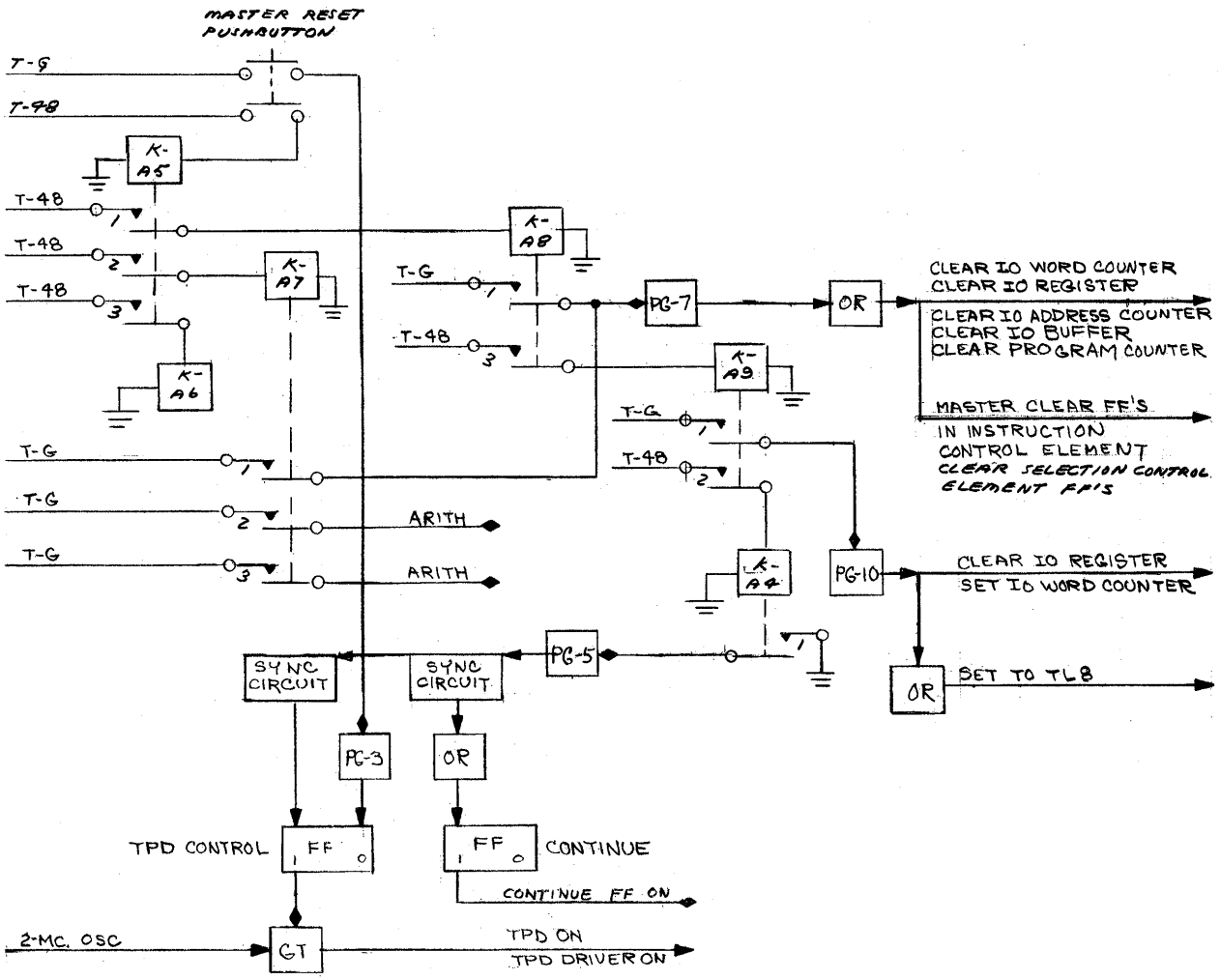


Figure 7–27. MASTER RESET Pushbutton Circuit, Simplified Schematic

adequately test the control and operation register flip-flops of the Central Computer System, the COMPLEMENT pushbutton and switch are incorporated to allow for the testing of these circuits.

The COMPLEMENT pushbutton, when depressed, applies a test ground voltage to PG–12, firing the generator and producing an output pulse. (See fig. 7–28.) This pulse serves two functions, the first of which is to complement selected flip-flops. Secondly, it is shunted through a one-half-microsecond delay circuit to provide a second complement pulse for Central Computer System use and to simultaneously sense the gate tube conditioned by the COMPLEMENT switch. The COMPLEMENT switch is a two-position switch which provides a means of obtaining a third complement pulse. When the switch is in the position designated 3, this gate is conditioned by the +10-volt level so that the sensing pulse is gated through a one-half-microsecond delay circuit to develop a third complement pulse. This pulse is applied to the flip-flops a one-half microsecond after the two pulses generated by the COMPLEMENT pushbutton. When the switch is in the 2 position, this

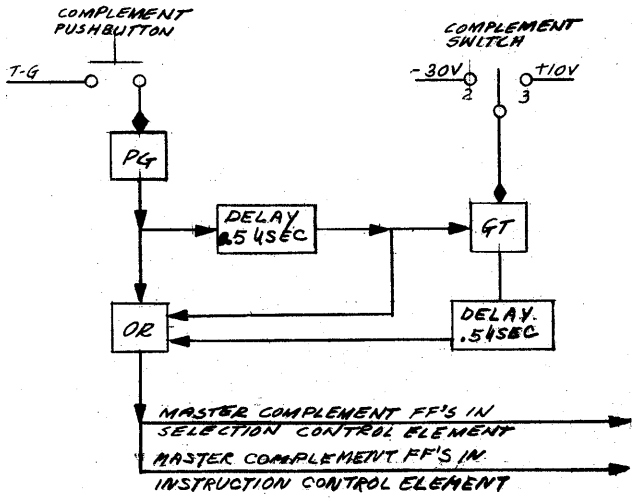


Figure 7–28. COMPLEMENT Pushbutton and Switch Circuit, Simplified Schematic

TABLE 7-9. FLIP-FLOPS NOT COMPLEMENTED BY THE COMPLEMENT PUSHBUTTON

LOCATION	FLIP-FLOP	LOCATION	FLIP-FLOP
Instruction control element	TPD Control		Reader not ready
	TPD control set sync		Printer not ready
	Continue set sync		Punch not ready
	Mechanical delay		Tapes not ready
	Set 2 mc sync		Tapes not prepared
	Clear pause sync		Sense condition lights
Selection and IO control element	Lock address counter		Overflow
	Address mode		Marginal checking (MC)
	Drum parity check		Memory parity
	Identity (3)		Drum parity
	Tape operation		Tape parity
	IO register		Command generation Nos. 1, 2, & 3
	Manual input matrix		Delay
	Burst time counter		Index
	Read-write zero		Punch
	Output alarm		Printer
	Output alarm sync		Card machine
	Spares		Card reader
			Second break request

gate tube is not conditioned. Therefore a third complement pulse is not developed and the COMPLEMENT pushbutton alone provides the complement pulses.

An actual test of the control and register flip-flops consists of observing the state of the flip-flop neon indicators in question, setting the COMPLEMENT switch to either the 2 or 3 position, pressing the COMPLEMENT pushbutton, and observing the status of the flip-flop neon indicators again. The COMPLEMENT switch is then rotated to the other position, the COMPLEMENT pushbutton is depressed, and the flip-flop neon indicators are observed once more. If the switch is in the 3 position and the COMPLEMENT pushbutton is pressed, the flip-flop neon indicators should change status. If the switch is in the 2 position and the COMPLEMENT pushbutton is pressed, the flip-flop neon indicators should not change status. The operator can thus determine the faulty circuits from the reaction of the flip-flop neon indicators.

2.6 NEON LIGHT PANEL

The right-wing panel of the maintenance console consists of the neon indicators. These are connected to the 1 side of the flip-flops of the Central Computer System registers and counters except for those of the time pulse distributor, which are located in the center-wing control panel of the maintenance console. Each neon register of the Central Computer System, located on the right-wing panel (figure 7-29), is arranged according to function in a straight line with octonary spacing. The order and arrangement of the neons are listed in table 7-10.

The contents of certain registers should be changed during the execution of a program. By examination of these registers after completion of a program, the operator should be able to determine whether the program has been executed properly. It is possible to determine where a program has stopped by observing the neon indicators displaying the contents of the program counter.

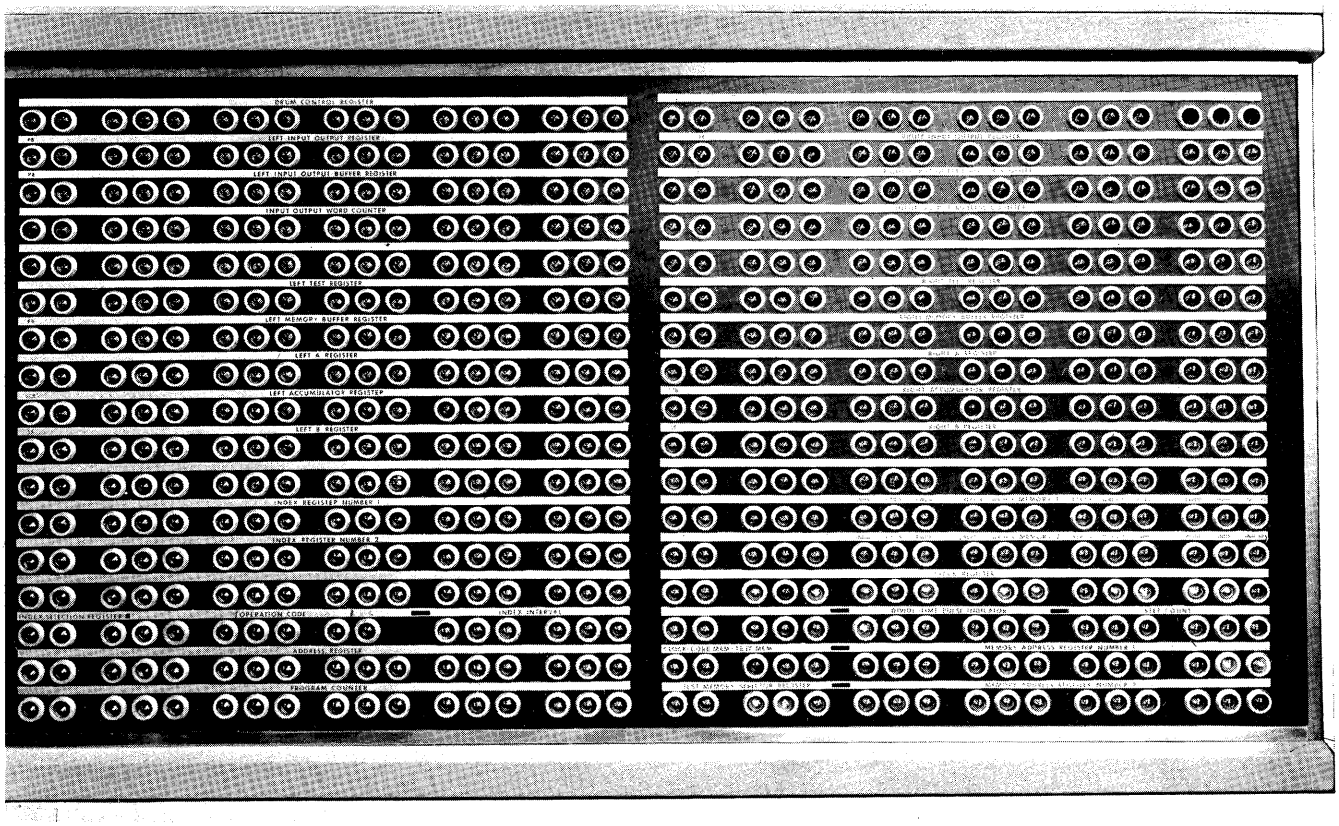


Figure 7–29. Right-Wing Neon Panel

When a stop occurs, the program counter will contain a number equal to one more than the address at which the program stopped. When the program has branched to a halt, the neon indicators of the RIGHT A REGISTER will contain a number equal to one more than the address of the instruction from which the program branched. In addition to the neon indicators the importance of noting other external indications of the progress of a program should be stressed. Such an indication might be

TABLE 7–10. NEON INDICATOR LIGHTS, RIGHT WING

LEFT-HALF	RIGHT-HALF	LEFT HALF	RIGHT HALF
DRUM CONTROL REGISTER		LEFT ACCUMULATOR REGISTER	RIGHT ACCUMULATOR REGISTER
LEFT INPUT OUTPUT REGISTER	RIGHT INPUT OUTPUT REGISTER	LEFT B REGISTER	RIGHT B REGISTER
LEFT INPUT OUTPUT BUFFER REGISTER	RIGHT INPUT OUTPUT BUFFER REGISTER	INDEX REGISTER NUMBER 1	MEMORY 1
INPUT OUTPUT WORD COUNTER	INPUT OUTPUT ADDRESS COUNTER	INDEX REGISTER NUMBER 2	MEMORY 2
LEFT TEST REGISTER	RIGHT TEST REGISTER		CLOCK REGISTER
LEFT MEMORY BUFFER REGISTER	RIGHT MEMORY BUFFER REGISTER	OPERATION CODE	DIVIDE TIME PULSE INDICATOR
LEFT A REGISTER	RIGHT A REGISTER	ADDRESS REGISTER	MEMORY ADDRESS REGISTER NUMBER 1
		PROGRAM COUNTER	MEMORY ADDRESS REGISTER NUMBER 2

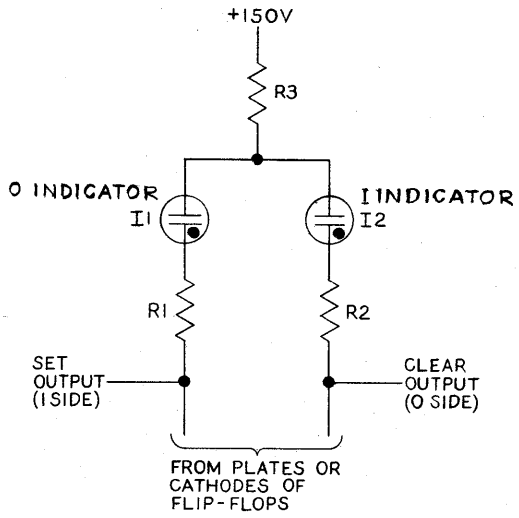


Figure 7–30. Neon Indicator Circuit, Schematic Diagram

the operation of an input-output unit, information being printed or punched, or the status of the CONDITION lights. Information of this kind may give some indication of the correctness of operation of the program.

The manner in which the indicator circuit is connected to the flip-flops is shown in figure 7–30. Neon light I1 and resistor R1 are connected to the set (1) side, while I2 and R2 are connected to the clear (0) side of the flip-flops in the various registers of the Central Computer System. In most instances, the indicating circuit is connected to the 0 side of the flip-flop; i.e., I2 is located in the right wing of the maintenance console while I1, connected to the 1 side, is mounted on the back panel of the particular register to which it is connected. R1 and R2 are connected internally in the pluggable units of the register. Notice that, if the flip-flop is set, the neon connected to the 0 side will light. This is attributable to the fact that the 0 side is at −30 volts and, neglecting the voltage across R2, the common side of I1 and I2 is approximately at +35 volts. This means that the voltage across I1 is about 25 volts. (The set output is +10 volts when the flip-flop is set.) Thus the neon connected to the 0 side of the flip-flop is set and, similarly, the neon connected to the 1 side of the flip-flop is on when the flip-flop is cleared. Certain registers (as the input-output word counter) which receive the complement of the address from the address register, have the 0 side neon indicators on the maintenance console, instead of the 1 side neon indicators.

2.7 IO CONTROL AND TEST

Certain pushbuttons, relays and indicator lights, are provided for readying IO units, supplying them with test voltages, and indicating their status to the operator at the maintenance console. In this manner, the operator at the maintenance console is provided with a means of readying IO units remote from his location.

2.7.1 READY INPUT OUTPUT UNITS Pushbutton

This pushbutton places the card machines and tapes in a state of readiness. The condition for readiness, however, varies for each IO unit. If the program STOP pushbutton is depressed on the card punch, for instance, the card punch is placed in the ready status by depressing the READY INPUT OUTPUT UNITS pushbutton. When the IO unit is ready, a specific light on the maintenance console is illuminated.

When the READY INPUT OUTPUT UNITS pushbutton is depressed, −48 volts is applied to relay K–A6. (See fig. 7–31.) When the contacts of this relay are picked up, the card punch, printer, reader, and tapes are readied by having their circuits completed through the contacts of relay K–A6. If the tapes are selected, the TAPE SELECTED light is illuminated as is light 1, 2, 3, or 4. If a card machine is selected, either the READER, PUNCH, or PRINTER light is illuminated. All these lights are located on the center control panel of the maintenance console.

2.7.2 Drum Test

Drum System testing may be performed either automatically or manually. For automatic testing, the selection of a specific field to be tested and the type test to be performed must be programmed before the tests are initiated. Manual tests are performed with the aid of the various pushbuttons at the drum test panel by the operator at that panel. For either mode of testing, however, the OPERATE-TEST switch at the maintenance console must be placed in the TEST position, and the OPERATE-TEST switch on the Drum System test panel must be placed either in the COMPUTER TEST or MANUAL TEST position. The OPERATE-TEST switch on the maintenance console is discussed in 2.3 of this Section. For a detailed discussion of the type tests performed on the Drum System and the drum circuits involved, refer to Parts 8 and 9 of PH 23–00006.

When the OPERATE-TEST switch at the maintenance console is placed in the TEST position, test ground and test −48 volts are fed from the maintenance console to the test panel of the Drum System. (See fig. 7–32.) With the OPERATE-TEST switch at this panel in the TEST position, the TEST DRUMS light at the maintenance

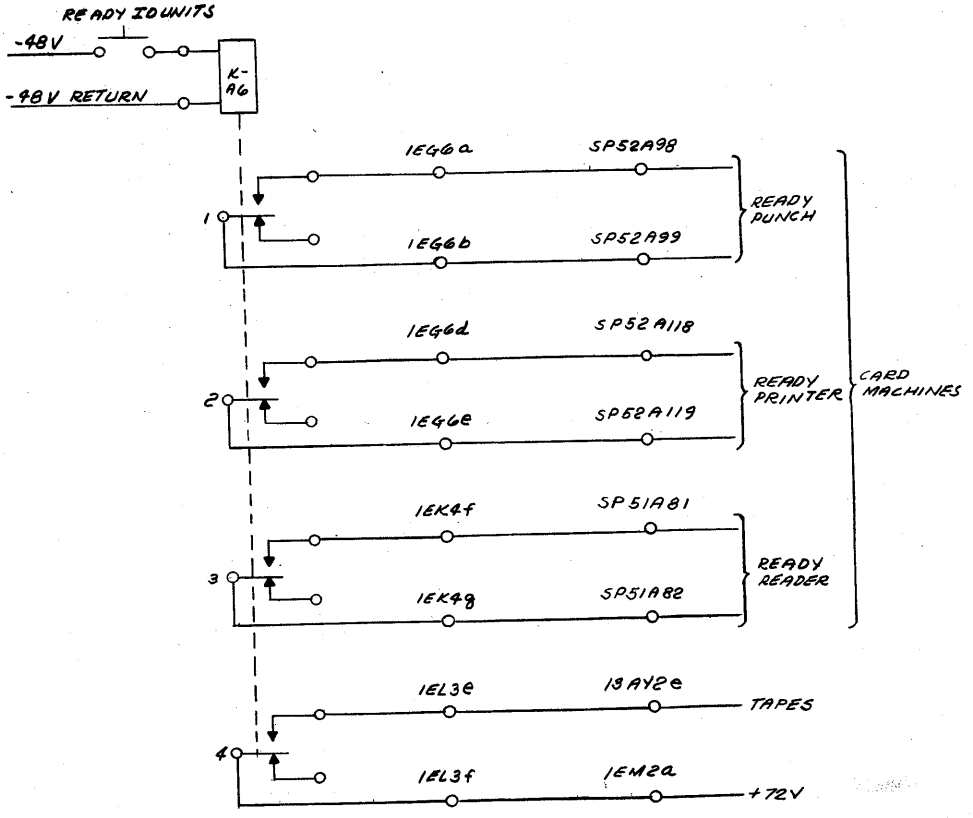


Figure 7–31. READY INPUT OUTPUT UNITS Pushbutton Circuit, Simplified Schematic

console is illuminated. At the same time, the test voltages energize a relay in the Drum System test panel whose contacts supply test voltages and d-c levels to all switches and pushbuttons on the test panel. In this manner, any desired tests may be performed either by programming or by manual operation.

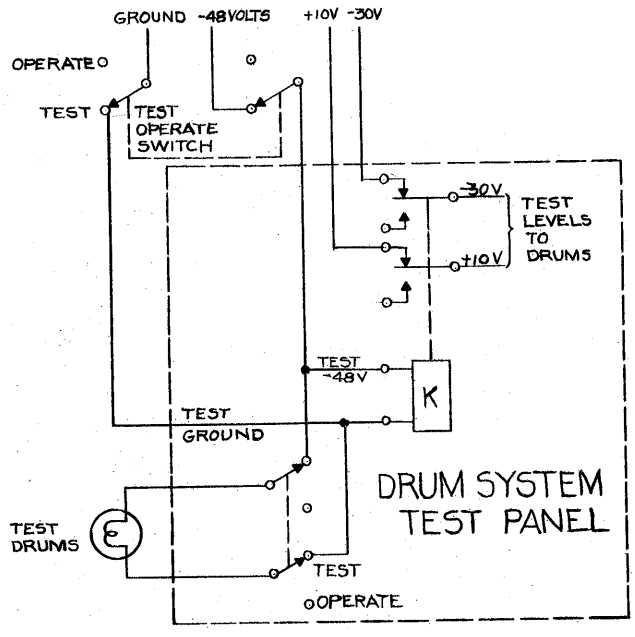


Figure 7–32. Drum Test Circuit, Simplified Schematic

2.7.3 Card Machine Test

There are three card machines used in conjunction with the maintenance console: a type 713 card reader, which is a modified IBM type 711 card reader; a type 723 card punch, which is a modification of the IBM type 721 card punch; and a type 718 printer, which is a modification of the IBM type 716 printer.

When the card machines are used, the designated one is selected during a programmed IO operation. This means that the *Load IO Address Counter (LDC)* and the *Select (SEL)* instructions will be programmed for the desired card machine. Before a *Read (RDS)* or *Write (WRT)* instruction is given, however, the designated machine must be sensed to determine its state of readiness. The *Sense (BSN)* instruction follows the *LDC* and *SEL* instructions to sense the IO not ready unit, so that no hang-up of the Central Computer System may occur. If the selected card machine is not ready, an unconditional branch will take place to the next instruction on the program. At the same time, the NOT READY neon indicator for the designated machine will be illuminated, as shown

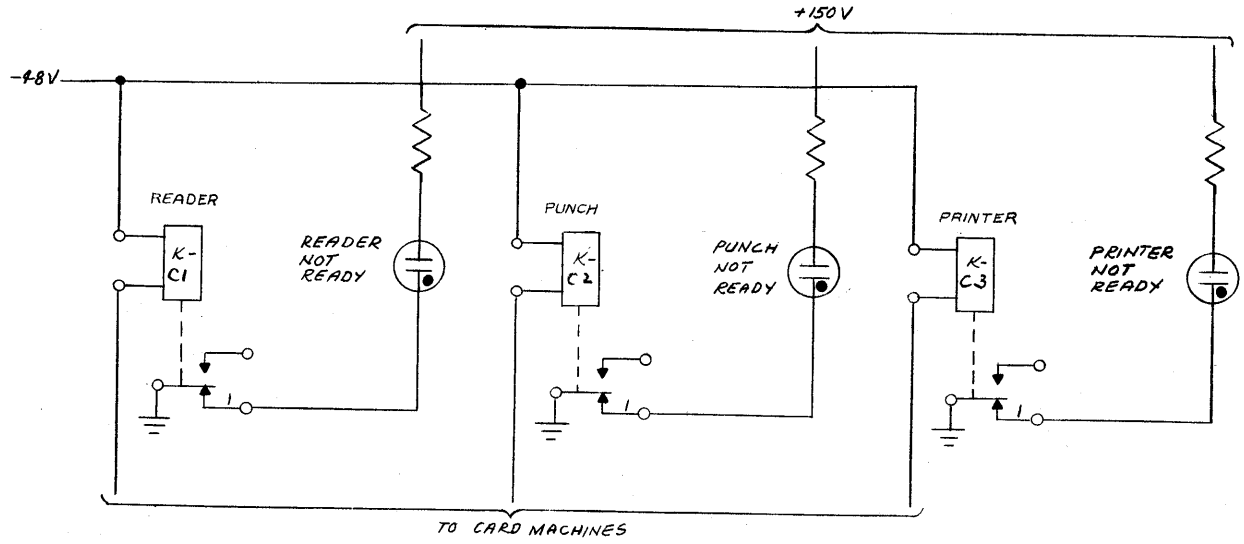


Figure 7–33. Card Machines Not Ready Indicator Circuit, Simplified Schematic

in figure 7–33. If the selected card machine is in a ready status, the appropriate relay (figure 7–33) will be energized so that the neon indicator is not illuminated, since the contacts of the relay open the ground return circuit of the neon light.

If the card reader is to be in the ready status, the following conditions must be fulfilled:

a. It must be under Central Computer System control.
b. There must be a card ahead of the read brushes.
c. The stacker must be full.
d. Power must be on.
e. All fuses must be good.

If the card reader does not meet all of these conditions, it is considered not ready.

The following conditions must be fulfilled if the card punch is to be considered in a ready state:

a. There must be a card ahead of the punching station.
b. The card punch must be under Central Computer System control.
c. The stacker must not be full.
d. The power must be on.
e. All fuses must be good.

If the card punch does not meet all of these conditions, it is considered not ready and the card punch NOT READY light is illuminated to indicate this fact.

Before the card printer can be considered ready, the following conditions must be met:

a. Forms must be contained in the printer and the form stop lever must be closed.
b. The printer must be under Central Computer System control.
c. The test switch must be off.
d. The carriage must not be stopped by depressing the CARRIAGE STOP button.
e. Power must be on.
f. Fuses must be good.
g. The printer control panel must be in place.

If the printer does not meet all of these conditions, it is considered in a not ready state and the printer NOT READY light is illuminated to indicate this condition.

The programmed tests used for the card machines determine whether the machines carry out their designated functions. The machines are selected, sensed for a state of readiness, and then read from (card reader) or written into (card punch or printer). The printer or punch is controlled as to the manner of operation by the *Operate* instruction.

2.7.4 Tape Machine Tests

Tape machine tests are performed manually by the operator at the tape machines. In order to run these tests, however, the TAPE TEST OPERATE-TEST switch at the maintenance console must be placed in the TEST position by the operator at the maintenance console. When this is done, pick relay K1 at the tape machines is energized. (See fig. 7–34.) The relay contacts are closed and are then held by the K1 hold relay coil. With COMPUTER TEST switch 7 at the tape machines in the TEST position, test levels of +10 and −30 volts and a test pulse are supplied to the test circuits of the tape machines. These are provided by

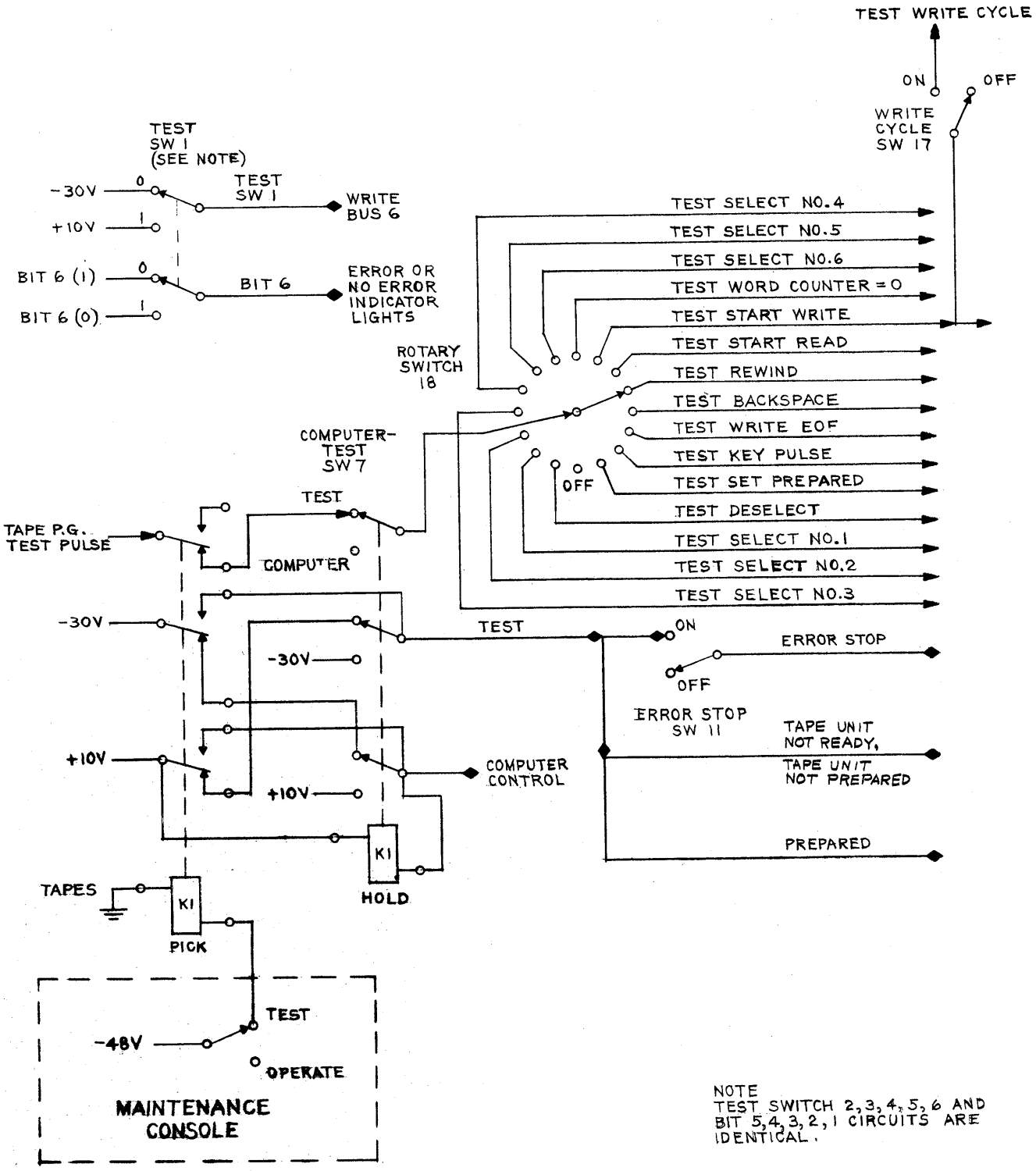


Figure 7–34. Tape Machine Test Circuit, Simplified Schematic

the tape machine circuits to test the functions listed in figure 7–34. By means of ROTARY switch 18, any one of six (with future expansion) tape machines may be selected or de-selected, and a read or write cycle, rewind, backspace, write end-of-file, or set prepared circuits may be tested. In addition, a test key pulse and test word counter equal to zero may be performed.

By means of TEST switches 1, 2, 3, 4, 5, and 6, the write bus lines may be checked while errors in transfer of words are indicated by error indicating lights. The test line shown in the figure will

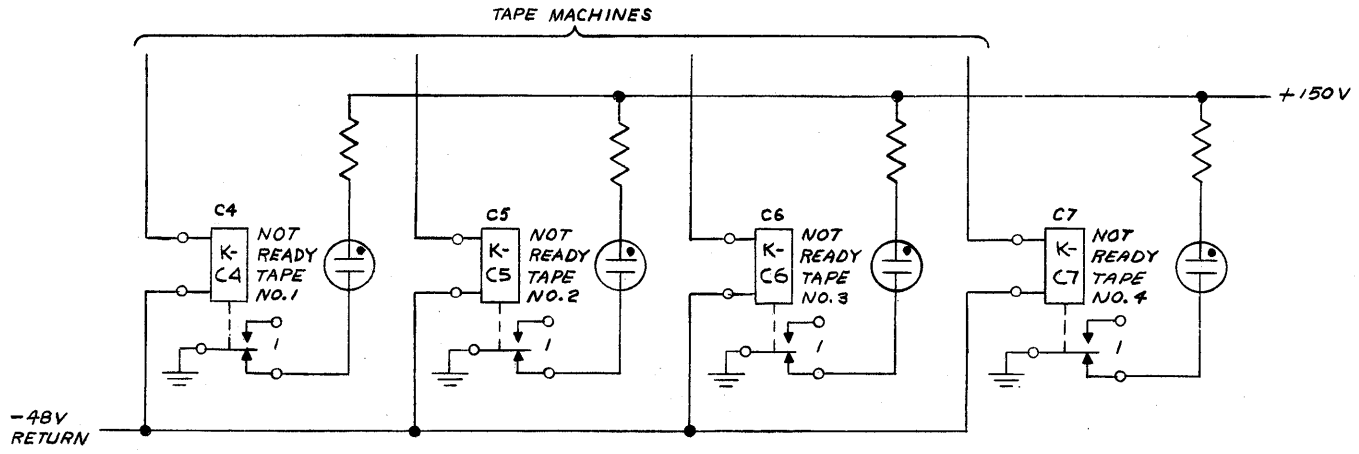


Figure 7–35. Tapes Not Ready Indicator Circuit, Simplified Schematic

test the tape machine for a stop on an error when ERROR STOP switch 11 is ON. The same line is used for conditioning circuits which are used to indicate that the tape unit is not ready, is not prepared, or is prepared. If the tape unit which is selected is not in a state of readiness, the appropriate neon indicator (figure 7–35) is illuminated.

2.7.5 SENSE Switches

Four SENSE switches are provided to enable the Central Computer System to bypass the next instruction in the program and to branch to the address specified by the *Sense* (*BSN*) instructions associated with these switches. The Central Computer System then executes the instruction specified by the *Sense* instruction address. This sense and branch operation will not take place if the switches are in the off position since the proper level will not be applied to the associated AND circuit. If, however, any one of the four SENSE switches is in the position shown in figure 7–36 when the corresponding *Sense* instruction is applied, a d-c level is generated through the associated AND circuit. The AND circuit output level is then applied through the OR circuit to condition a gate tube in the selection and IO control element. At OT–9 of the *BSN* instruction, this gate tube is sensed for proper conditioning level. If the gate is conditioned, the OT–9 pulse is gated through to set the branch flip-flop in the instruction control element. The Central Computer System will branch directly to the address specified by the sense instruction instead of executing the next instruction at the address specified by the program counter.

2.7.6 CONDITION LIGHTS

Four lights, CONDITION LIGHTS 1, 2, 3, and 4, are provided on the center control panel of the maintenance console. These are neon lights connected to flip-flops which are set by four *Operate* (*PER*) instructions. Use is made of one of the four CONDITION LIGHTS to indicate the existence of a specific condition in the Central Computer System which may subsequently be sensed. For example, when a condition light is illuminated, it can denote to the operator at the maintenance console that a designated point in the program has been reached. Further, as a result of arithmetic computations, a condition light may be illuminated to denote some specific result.

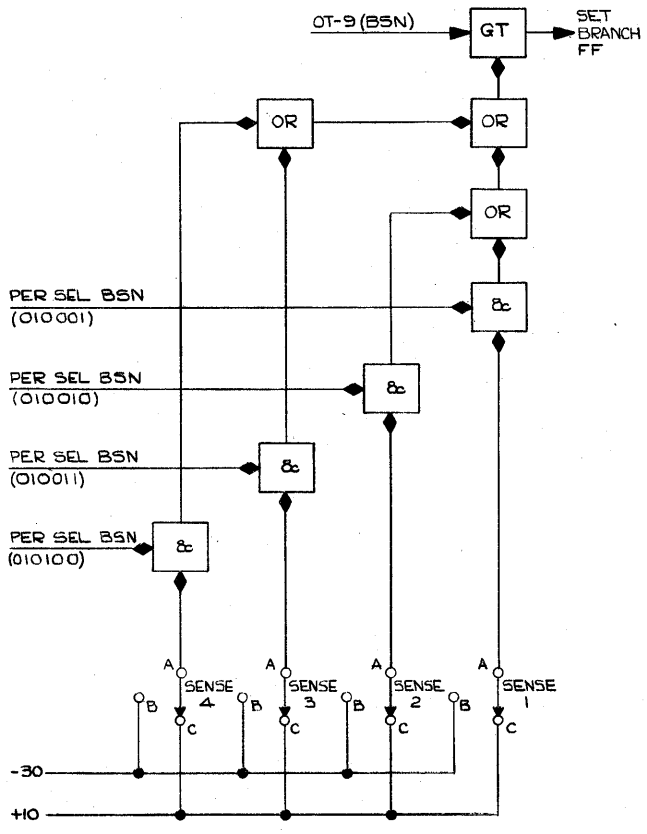


Figure 7–36. Sense Circuit, Simplified Schematic

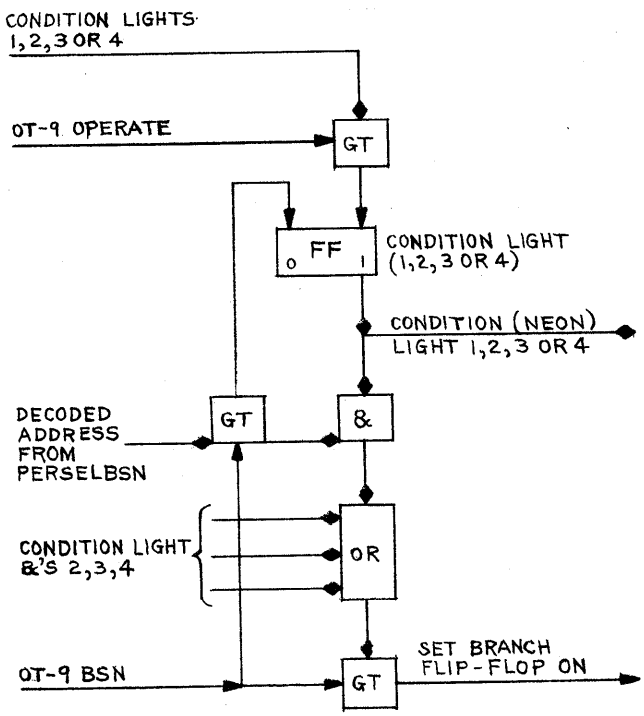


Figure 7–37. CONDITION Light Circuit, Simplified Schematic

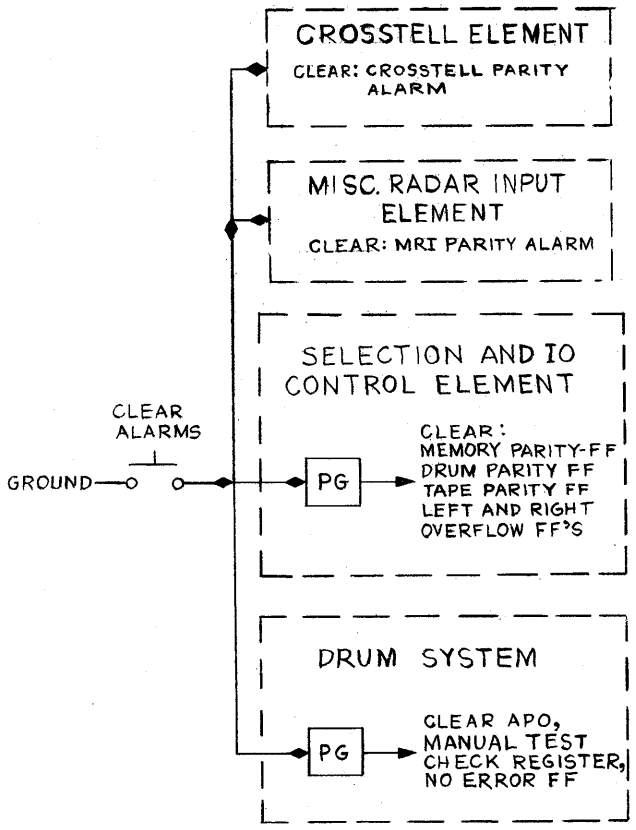


Figure 7–38. CLEAR ALARMS Pushbutton Circuit, Simplified Schematic

This condition light may then by sensed at a later time to determine the condition which existed at the previous time. From this sensing operation, the Central Computer System determines whether or not to branch to another portion of the program.

The existence of the condition which illuminates a specific light is indicated by the status of a flip-flop. When this flip-flop is sensed by one of four *Sense (BSN)* instructions, a branch of control will occur if the corresponding flip-flop is set. If the flip-flop is in the 0 state, the sequential execution of the program instructions will remain unaffected. One of four condition light flip-flops is set at OT–9 during the execution of the *Operate (PER)* instruction. (See fig. 7–37.) The CONDITION LIGHT connected to the corresponding flip-flop is illuminated and one side of the AND circuit is conditioned. When the *Sense (BSN)* instruction is given, the decoded address from the index interval (bits L10–L15) conditions the other side of the AND circuit so that two gate tubes are simultaneously conditioned. At OT–9 of the *Sense* instruction, both gates are sensed and gate through pulses which simultaneously set the branch flip-flop on and clear the condition light flip-flop. When the condition light flip-flop is cleared, the CONDITION LIGHT is turned off and a program branch takes place. Conversely, if a CONDITION LIGHT is not illuminated, a program branch will not occur and the instruction sequence of the program will be uninterrupted.

2.7.7 CLEAR ALARMS Pushbutton

The circuit for the CLEAR ALARMS pushbutton, which is an operating control located on the center-wing control panel of the maintenance console, is shown in figure 7–38. When this pushbutton is depressed, ground is provided which activates a pulse generator in the Drum System. The output pulse from this generator clears the angular position counters, the manual test check register, and the no error flip-flop in the Drum System and turn off the DRUM APC ALARM light on the maintenance console. This same ground line is fed to the selection and IO control element and is used to clear the memory parity flip-flop, drum parity flip-flop, tape parity flip-flop, and the left and right overflow flip-flops. When these flip-flops are cleared, the MEMORY PARITY ALARM, DRUM PARITY ALARM, TAPE PARITY ALARM, and the LEFT and RIGHT OVERFLOW ALARM lights are all turned off. This ground line is also fed to the MRI and Crosstell units, turning off the MRI PARITY ALARM and the CROSSTELL PARITY ALARM.

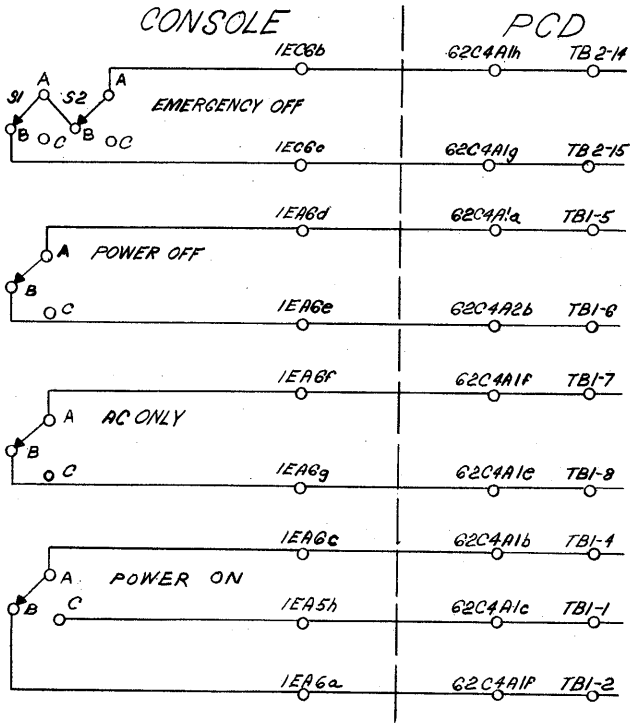


Figure 7–39. Power Controls, Simplified Schematic

2.8 POWER CONTROL

The following power controls (figure 7–39) and associated indicator lights (figure 7–40) are mounted on the center-wing panel of the maintenance console:

a. EMERGENCY OFF (switch)
b. POWER OFF (switch)
c. AC ONLY (switch)
d. POWER ON (switch)
e. MOTORS OFF (indicator light)
f. POWER OFF (indicator light)
g. AC ONLY (indicator light)
h. POWER ON (indicator light)
i. PCD FRAME (indicator light)
j. AC SWITCH GEAR (indicator light)

The POWER ON pushbutton, when depressed, will initiate the power sequencing to the equipment and the POWER ON indicator light will be illuminated. At a later time, the MASTER READY light will be illuminated to indicate that all power is sequenced on properly and that a suitable delay has taken place to allow supply voltage stabilization.

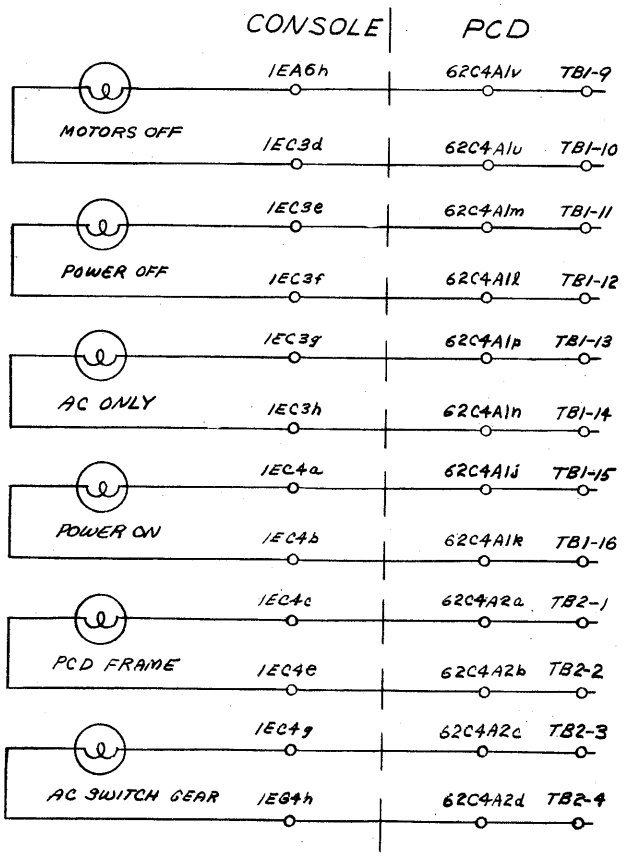


Figure 7–40. Power Status Indicator Lights

The AC ONLY pushbutton is depressed when alternating current only is desired. The motor generator sets are running after this pushbutton is depressed, but no direct current is supplied to the equipment. In addition, the AC ONLY light will be illuminated.

When the POWER OFF pushbutton is depressed, all power is removed from the equipment in the proper sequence but the motor generator sets will continue to run. The POWER OFF light will be illuminated when the power off sequence has been carried out.

Power may be removed completely and immediately from the equipment by activating the EMERGENCY OFF pushbutton. This removes power from all power supplies and all indicating lights will go off, including the MOTORS OFF light.

Operating the POWER ON pushbutton after any of the other power pushbuttons have been depressed automatically restores full a-c and d-c power to the system in the proper sequence.

SECTION 3
INDICATOR LIGHTS AND ALARMS

The various indicator lights and alarms used in conjunction with AN/FSQ-7 (XD-1,-2) Combat Direction Centrals operations (except for the neon lights on the right wing which are connected to the Central Computer System registers) are mounted on the center-wing panel of the maintenance console. The lights listed in table 7-11 serve as warning or alarm lights which, when illuminated, indicate that a fault has occurred in the system. Illumination of a particular light specifies the type of fault and designates to the operator where that fault is located.

3.1 PARITY AND OVERFLOW

The DRUM PARITY, MEMORY PARITY, and TAPE PARITY STOP—CONTINUE switches enable the operator to either halt the Central Computer System or allow the Central Computer System to continue operating after an error occurs in input-output transfers or word transfers out of core memory. When a drum, memory, or tape parity error occurs, an alarm pulse is generated which performs two functions. First, it sets the respective drum parity, memory parity, or tape parity flip-flop. The setting of the parity flip-flop illuminates the associated neon indicator on the maintenance console, thus indicating the presence of a parity error to the operator. Second, the alarm pulse senses the gate tube conditioned by the STOP-CONTINUE switch. Further Central Computer System action is determined by the position of the STOP-CONTINUE switch.

TABLE 7-11. ALARM LIGHT INDICATORS

LIGHT	FUNCTION
PCD FRAME, AC SWITCHGEAR, COMPUTER, DRUMS, SDV RADAR INPUT, DISPLAY AND MANUAL INPUT, OUTPUT XT AND MRI	Illumination of any one of these lights indicates absence of power or a voltage excursion beyond normal. The light which is illuminated designates to the operator the location of the unit at fault.
MRI PARITY ALARM, CROSSTELL PARITY ALARM, DRUM PARITY ALARM, MEMORY PARITY ALARM, TAPE PARITY ALARM	These lights indicate that an error has occurred in transfers of information to or from the unit designated by the light.
LEFT OVERFLOW ALARM RIGHT OVERFLOW ALARM	These lights indicate that the capacity of the respective accumulators has been exceeded. These alarm lights work in conjunction with the OVERFLOW STOP-CONTINUE switch which determines computer action when either excess occurs.
DRUM APC ALARM	This light indicates that any one of the five angular position counters is out of step with drum timing.
OUTPUT ALARM	This light indicates a faulty condition in the output system.
VOLTAGE*	This is used to indicate absence or deviation of power.
CIRCUIT BREAKER*	This light indicates a tripped circuit breaker.
AIR*	This indicates a low air pressure at a location.
TEMP*	This light is used to indicate temperature excursion beyond normal.

** At the same time that these lights are illuminated, an audible warning signal provided by an Edwards buzzer is heard.*

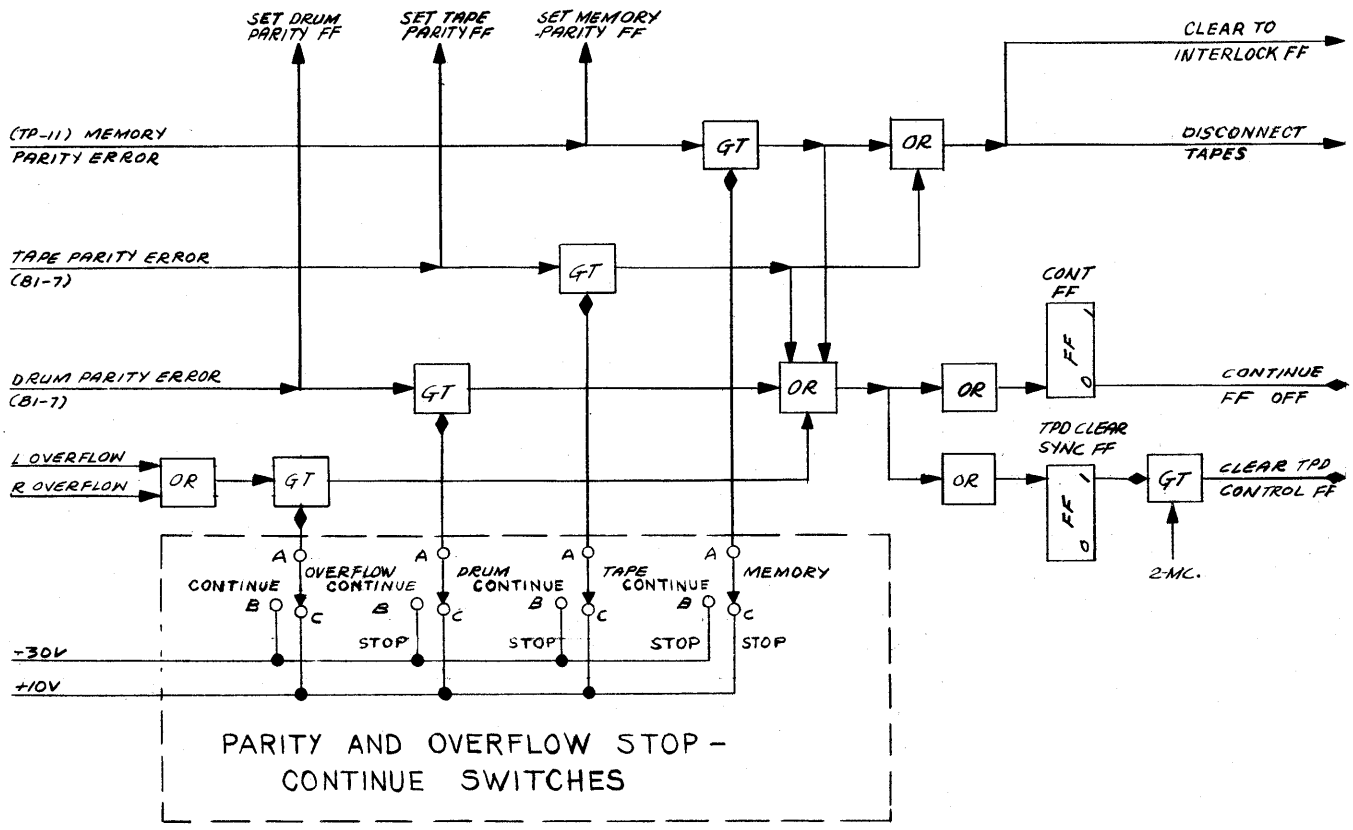


Figure 7–41. Parity and Overflow Circuits, Simplified Schematic

When any one of the parity stop-continue switches is in the STOP position, a +10-volt conditioning level is applied to the associated gate tube. If an error occurs in the transfer of a word out of memory, an alarm pulse is transmitted directly from the arithmetic element to the selection and IO control element at TP–11. This pulse sets the memory parity flip-flop and senses the gate conditioned by the MEMORY PARITY STOP-CONTINUE switch. In the position shown on figure 7–41, the memory parity alarm pulse is gated through to clear the TPD control, continue, and IO interlock flip-flops and to disconnect the tape unit. The same action results when a tape parity error occurs in input-output word transfers except that the tape parity flip-flop is set in this instance, rather than the memory parity flip-flop. Similarly, an input-output error during Drum System word transfer causes a drum parity pulse to be generated. This pulse sets the drum parity flip-flop and senses the gate conditioned by the DRUM PARITY STOP-CONTINUE switch. When the switch is in the STOP position (figure 7–41), the alarm pulse is gated through to clear the TPD control and continue flip-flops and thus halt the Central Computer System. The CONTINUE positions—one apiece for the DRUM PARITY, MEMORY PARITY, and TAPE PARITY STOP-CONTINUE switches—are connected to —30 volts, which prevents the alarm pulse from being gated through to halt the Central Computer System.

The OVERFLOW STOP-CONTINUE switch determines Central Computer System action in the event of an overflow of either accumulator register. If an overflow occurs in the left or right accumulator register, an alarm pulse is generated which senses the gate tube conditioned by the OVERFLOW STOP-CONTINUE switch. When this switch is in the STOP position (figure 7–41), the overflow alarm pulse is gated through to clear the TPD control and continue flip-flops and thus halt the Central Computer System. In the CONTINUE position, however, the gate tube associated with the STOP-CONTINUE switch is not conditioned; hence no alarm pulse is gated through to halt the Central Computer System.

3.2 SYSTEMS STATUS INDICATING LIGHTS

The light indicators shown in figure 7–42, when lit, designate to the maintenance console operator that power is absent or that a voltage excursion beyond normal is present in the system whose corresponding light is illuminated. With

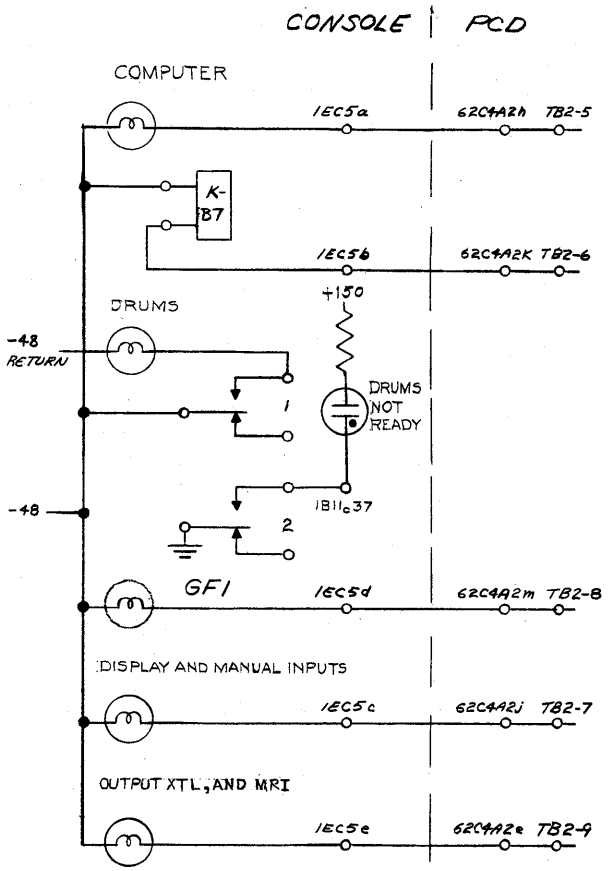


Figure 7–42. Systems Status Indicator Lights, Simplified Schematic

the exception of the DRUMS light and the DRUMS NOT READY neon light, the lights, as well as relay K–B7, are connected to the power control and distribution (PCD) unit. When a trouble of the type just described occurs, the PCD supplies a ground to one of the light or relay circuits so that this particular system malfunction is indicated for the information of the maintenance console operator. For the detailed circuit connections in the PCD, refer to Manual 6, Part 4, Power Control Element.

3.3 AUDIBLE ALARM CIRCUIT

Those alarms or indicators which are not capable of automatically stopping the Central Computer System are provided with an audible alarm. The CIRCUIT BREAKER, VOLTAGE, AIR, and TEMP alarms are not capable of stopping the Central Computer System and the audible alarm provided for these alarm lights is shown in figure 7–43. When a malfunction occurs in one of the power divisions of the Central Computer System, a ground is supplied to the corresponding light alarm and relay. For example, if a circuit breaker trips, the CIRCUIT BREAKER light is illuminated and relay K–B10 is energized so that the 120-volt a-c circuit is completed to the Edwards buzzer, which provides the audible alarm. The VOLTAGE light indicates absence of power or an excursion beyond normal; the AIR light indicates a low air pressure at a particular location; the TEMP light indicates a temperature condition beyond normal at a location.

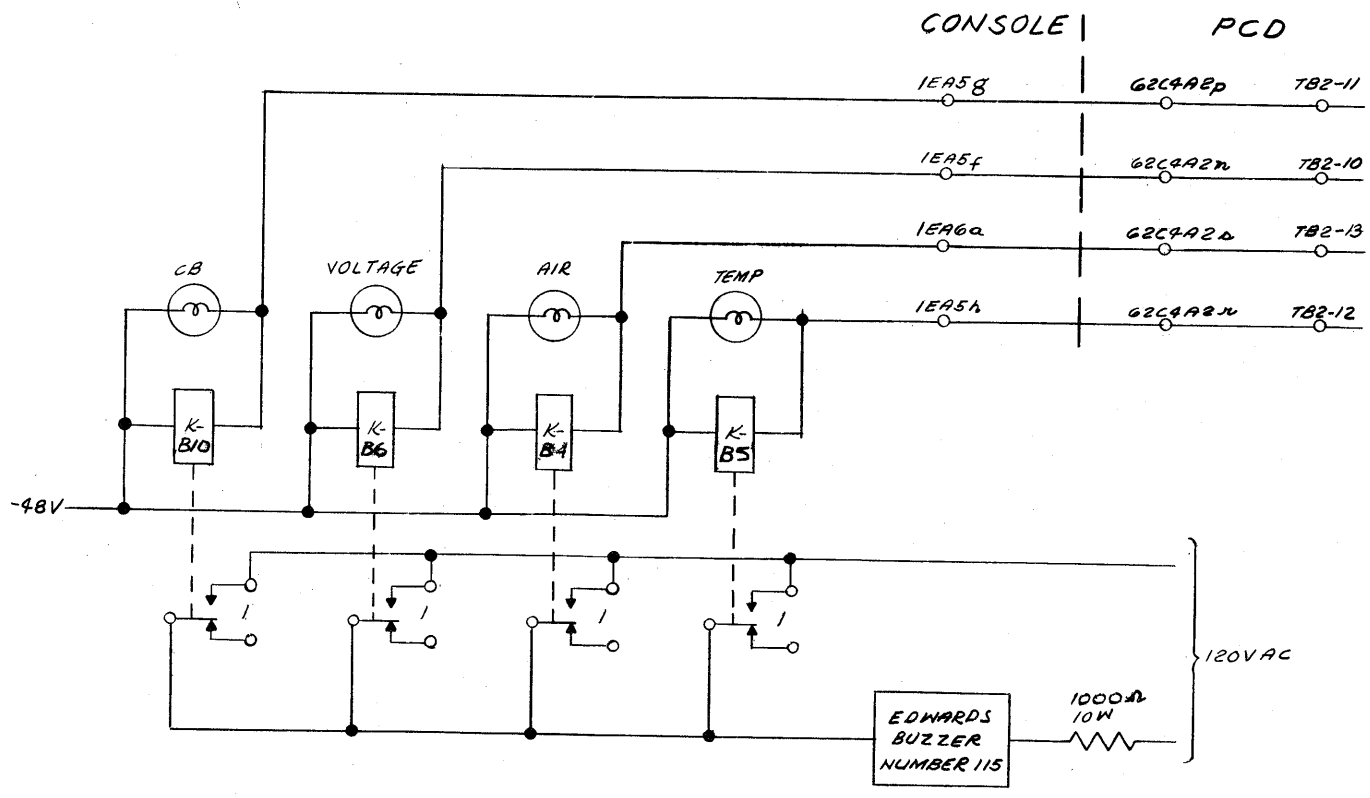


Figure 7–43. Audible Alarm Circuit, Simplified Schematic

SECTION 4

AUXILIARY EQUIPMENT

Two pieces of equipment are provided as auxiliary equipment with the maintenance console. One of these is an amplifier which is permanently connected to the accumulator registers. The other is a 24-station telephone system which is used for communication with all other parts of the AN/FSQ–7 (XD–1,–2) Combat Direction Centrals.

4.1 AUDIO AMPLIFIER

The audio amplifier used in the maintenance console consists of a single-stage amplifier, speaker, volume control, and a five-position selector switch, as illustrated in figure 7–44. The volume control and selector switch are located beneath the bookshelf to the left of the marginal checking main control panel. (See figure 7–2.) In normal operation, the input of the amplifier is connected to bits 1 and 2 of either the left or right accumulator registers. This connection allows the operator at the maintenance console to hear the sounds produced by frequency variations generated during the execution of various programs so that he may monitor the program aurally. Once the operator familiarizes himself with the program sounds, variations due to machine errors become apparent very readily.

To disable the amplifier, the selector switch (figure 7–44) is turned to the OFF position so that the control grid of the type 1782A tube is connected to —30 volts, thus cutting off the tube. In the four other selector switch positions, the input to the amplifier is connected to bits 1 or 2 of either the left or right accumulator register so that the operator at the maintenance console may monitor one of the four circuits. The volume control, which varies the input voltage to the grid of the amplifier tube, is then adjusted by the operator so that a comfortable volume level is obtained at the speaker. The circuit used is that of a normal amplifier with the suppressor grid connected to the plate and the screen grid connected to the plate supply so that the tube acts as a tetrode amplifier.

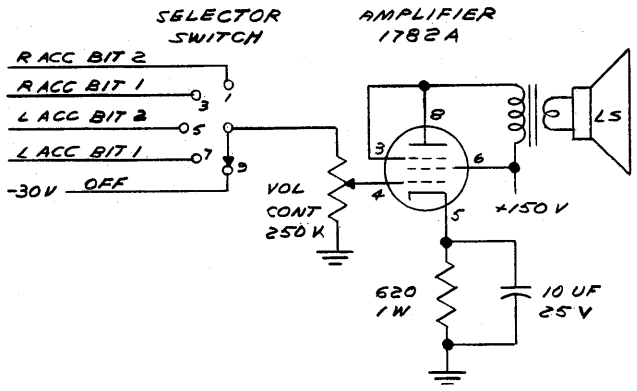


Figure 7–44. Audio Amplifier Simplified Schematic

4.2 TELEPHONE COMMUNICATIONS SYSTEM

Two 10- by 30-inch areas are provided on the maintenance console for telephone equipment. This equipment consists of three six-pushbutton panels, a dial unit, and a handset at each of the three locations in the maintenance control area at the maintenance console. The internal circuit number 21 is assigned to the three stations at the maintenance console. One of these stations, located at the bookshelf to the left of the marginal checking main control panel, is assigned extension number 401; another station, in front of the right-wing neon panel, is assigned extension number 402; and the last station, on the maintenance desk, is assigned extension number 403. The three panels of 18 pushbuttons are assigned as follows for circuit selection:

a. Four dial lines
b. Four conference lines
c. Four direct lines
d. Four public address circuits
e. Two spare pushbuttons

The four dial lines are commoned and are of the number-seeking type for incoming calls; i.e., if the number called is busy, the signal hunts for a line that is not busy. For example, if 401 is called and the number is busy, 402 is called, and so on. Four of the dial numbers are designated as conference numbers. When a conference circuit is in use, an indicator at the maintenance console is illuminated. If the operator wants to listen in, the pushbutton associated with the conference line is depressed.

The direct lines are used for communications with the power area, the repair area which is remote from the maintenance console, or the telephone room. The dial conference lines consist of a single talking pair to which a supervisor and a subordinate have access. Selective signalling from the supervisor to a particular subordinate is accomplished by means of a dial. Operation of a circuit key by a subordinate signals the supervisor only. The conference code number on the dial provides means for a supervisor to summon all subordinates for a conference.

APPENDIX A
INSTRUCTION CONTROL UNIT COMPONENTS

The pluggable unit is the basic physical component of AN/FSQ–7 Combat Direction Central. Pluggable units exist in six- and nine-tube models. There are 18 variations on these two basic pluggable units in the instruction control unit. Figure A–1 shows the pluggable unit arrangement in the instruction control unit and table A–1 gives the pluggable unit type number and quantity required. It should be noted that of the 160 pluggable units in the instruction control unit only 151 are in use. Nine are spare locations.

TABLE A–1. PLUGGABLE UNIT COMPLEMENT FOR INSTRUCTION CONTROL UNIT

TYPE NO.	QUANTITY REQUIRED	TYPE NO.	QUANTITY REQUIRED
6001	26	6012	17
6002	1	6013	4
6004	2	6016	5
6005	8	6017	11
6006	1	6018	2
6007	22	6020	6
6008	2	6024	14
6010	29	6025	1
6011	4	6080	1
		6081	1

Types 6010 and 6017 contain at least one power cathode follower. They are six-tube pluggable units. All others are nine-tube units. Six- and nine-tube units are of the same size and use the same plug base.

The following paragraphs give a more detailed discussion of each functional group of pluggable units in the instruction control unit. These functional groups (components) are listed in table A–2 below.

A.1 OPERATION REGISTER

Each bit of the operation register is contained in one pluggable unit. All bits except the sign bit contain one high-speed flip-flop and two power cathode followers. One cathode follower is connected to each side of the flip-flop. High-speed flip-flops are necessary because the operation register is set to a new instruction 0.5 microsecond after it has been cleared. Power cathode followers are needed to drive the matrices connected to the operation register. The heaviest load that is driven by a power cathode follower in the operation register is 15 milliamperes of AND current. The current required to charge the capacitive load is negligible compared to the d-c load. Since a power cathode follower can drive approximately 40 milliamperes of AND current, only about 40 percent of the power cathode follower's driving power is used under the worst possible conditions.

There is room for one more tube in each pluggable unit of the operation register. There is also an unused gate tube card in each pluggable unit. This card standardizes the operation register pluggable units with other pluggable units in the computer.

A.2 CYCLE CONTROL

The cycle control is contained in five pluggable units. A spare pluggable unit is located near the cycle control. There is an unused flip-flop in

the cycle control. This flip-flop can be used to apply an additional input to the control matrices if the need arises. Another pluggable unit in the cycle control contains pulse amplifiers which are used for clearing and complementing the operation register and the cycle control. All flip-flops in the cycle control are high-speed flip-flops, but only the branch flip-flop has to respond in 0.5 microsecond. The other flip-flops have 1.0 microsecond in which to change their state.

The most heavily loaded d-c output is the PT (program time) output. Fifty percent of its capacity is being used.

A.3 CLASS CYCLE MATRIX

The class cycle matrix is contained in five pluggable units.

Each output line of the class cycle matrix is driven by a power cathode follower. The most heavily loaded output is the shift class line. It must drive 400 micromicroseconds of capacity, 15.5 milliamperes of AND current, and 22.3 milliamperes of OR current.

There are three unused power cathode followers and three unused five-input AND circuits which can be used to provide additional outputs for the class cycle matrix. These AND circuits can also be used to expand the variation matrix.

A.4 VARIATION MATRIX

The variation matrix is contained in five pluggable units.

Each output line of the variation matrix is driven by a power cathode follower. The most heavily loaded line is the 001 line; it must drive approximately 300 micromicrofarads of capacity, 13.5 milliamperes of AND current, and 3 milliamperes of OR current.

There are two unused five-input AND circuits and two unused power cathode followers in the variation matrix. These circuits are adjacent to the class cycle matrix, so that they can be used to expand the class cycle matrix as well as the variation matrix.

A.5 INDEX SELECTION MATRIX

The index selection matrix is contained in a single pluggable unit. An unused pluggable unit location is provided adjacent to the index selection matrix. This space can be used to expand the matrix if more index registers are added to the machine. At present there are two index registers and the right accumulator register can be used as a third.

TABLE A–2. INSTRUCTION CONTROL FRAME COMPONENTS

LOGICAL NUMBER	COMPONENT
	D-C Generation Equipment
0.4.2.1	Class Cycle Matrix
0.4.3.1	Variation Matrix
0.4.4.2	Miscellaneous Class Instruction Matrix
0.4.4.3	Add Class Instruction Matrix
0.4.4.4	Multiply Class Instruction Matrix
0.4.4.5	Store Class Instruction Matrix
0.4.4.6	Shift Class Instruction Matrix
0.4.4.7	Branch Class Instruction Matrix
0.4.4.8	IO Class Instruction Matrix
0.4.4.9	Reset Class Instruction Matrix
0.4.5.2	Cycle Control
0.4.6.1	Operation Register
0.4.9.1	Index Selection Matrix
0.4.13.1	Memory Unit Selection Control
	Pulse Generation and Control Equipment
0.4.7.1	Time Pulse Distributor
0.4.7.2	Time Pulse Distributor Control
0.4.7.3	Instruction Pulse Drivers
0.4.7.4	Oscillator
0.4.12.1	Step Counter
0.4.14.1	Divide Time Pulse Distributor
0.4.1.5	IO Control Command Generators
0.4.1.10	Memory Control Command Generator
0.4.1.21	Memory Buffer Registers Command Generators
0.4.1.23	A Registers Command Generators
0.4.1.25	Adders Command Generators
0.4.1.27	Accumulator Registers Command Generators
0.4.1.29	B Registers Command Generators
0.4.1.33	Program Control Command Generators
0.4.1.36	Index Registers Command Generators
0.4.1.40	Instruction Control Command Generators
0.5.1.51	Selection and IO Control Command Generators
	Manual Operations Control
0.4.7.5	Manual Operation Control

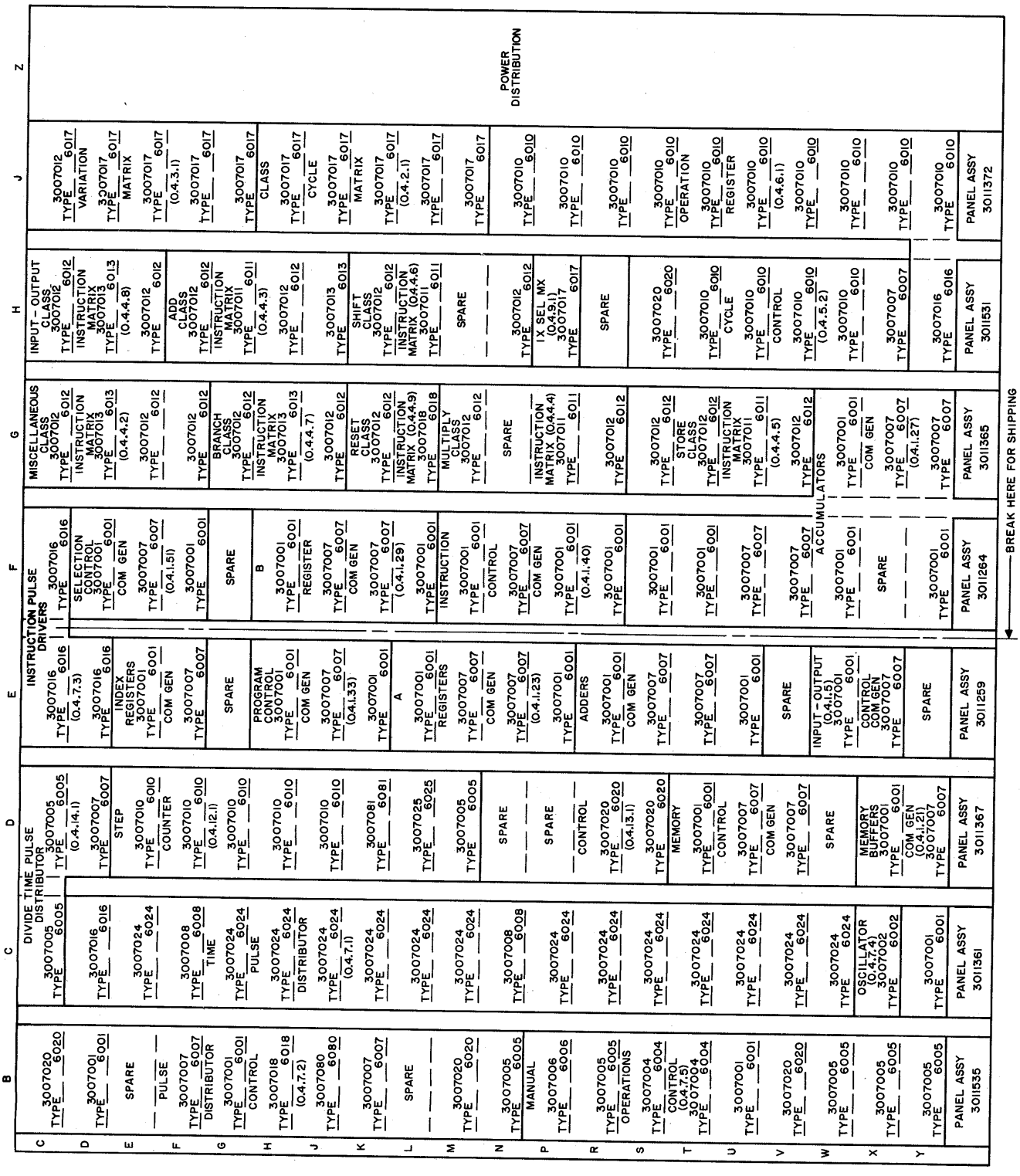


Figure A–1. Instruction Control Frame Panel Layout, Rear View