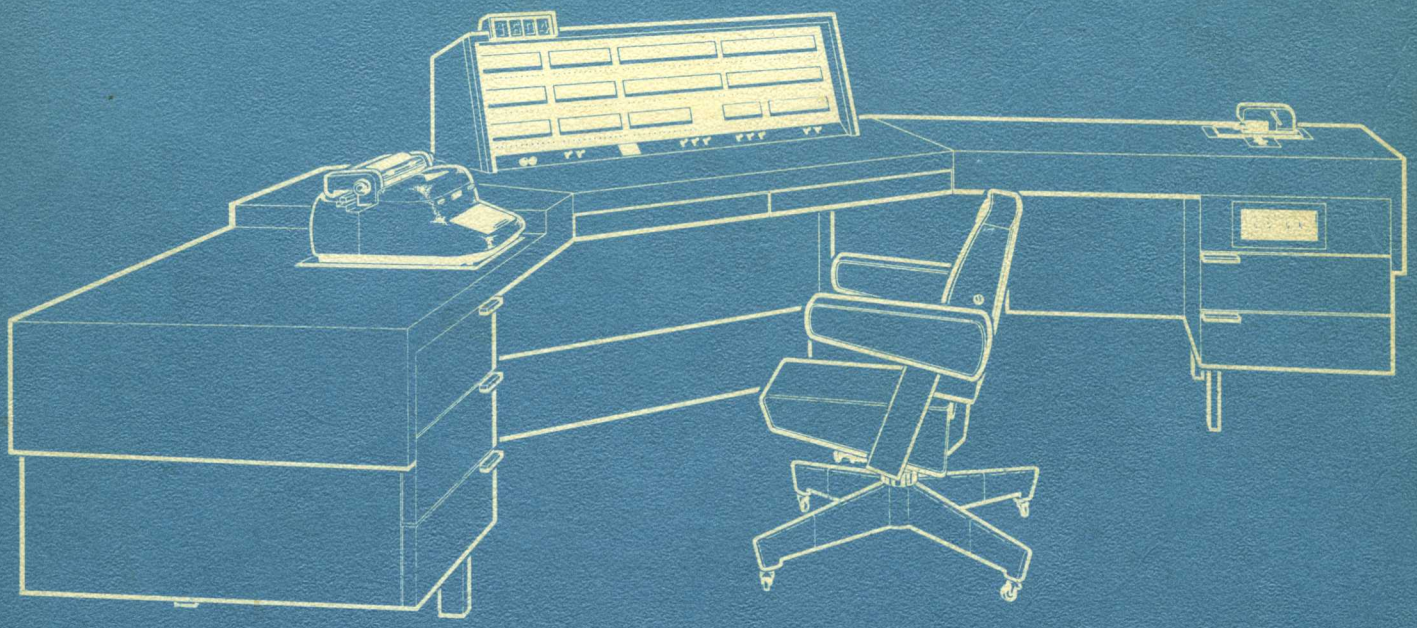
RH Bruce

# 1604 COMPUTER

## Volume 2:
## PRINCIPLES OF OPERATION



# INSTRUCTION BOOK

CONTROL DATA CORPORATION
MINNEAPOLIS, MINNESOTA

# 1604 COMPUTER

## Volume 2:
## PRINCIPLES OF OPERATION

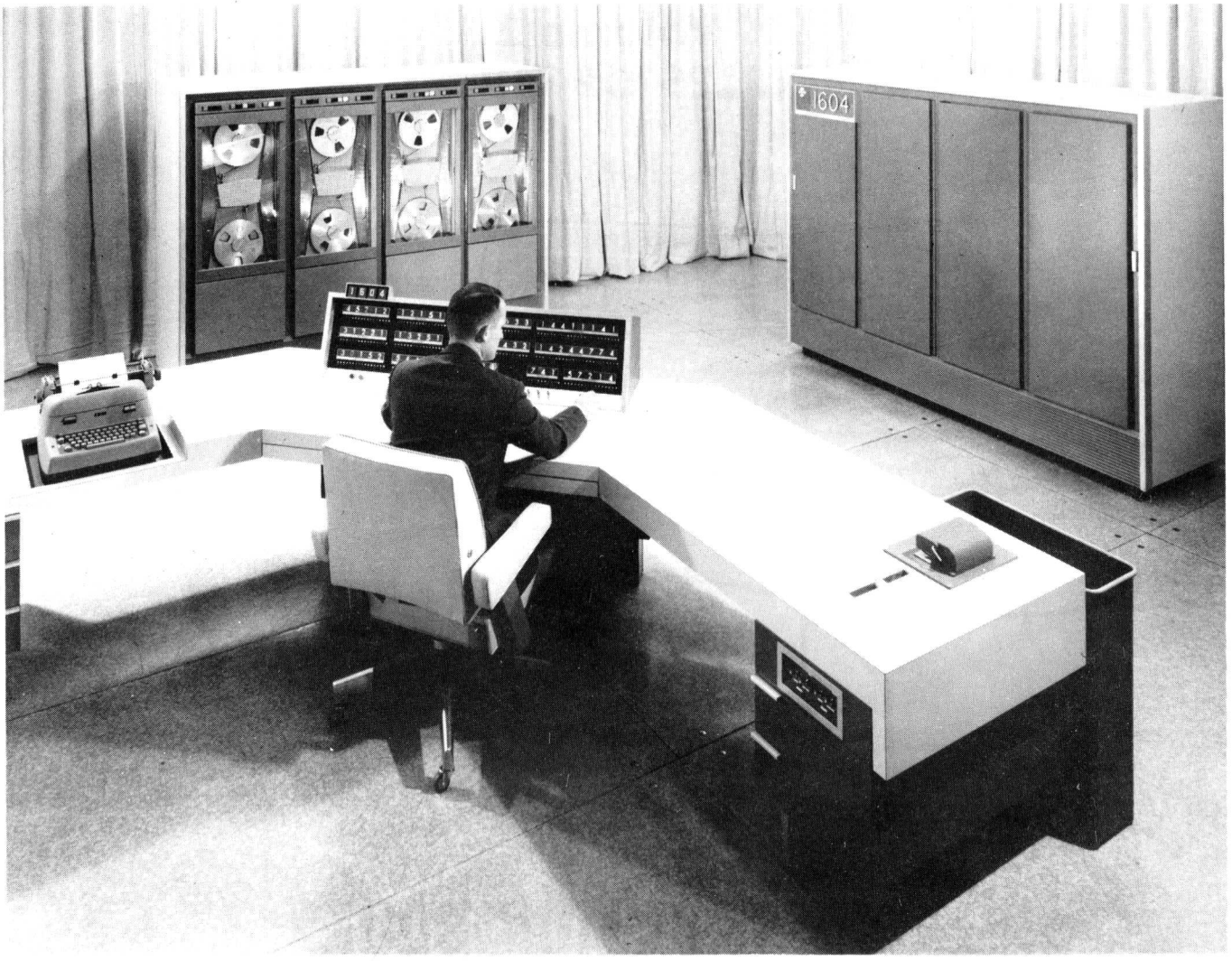
# INSTRUCTION BOOK

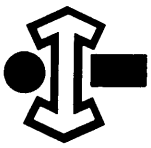
PUBLICATION 032a

CONTROL DATA CORPORATION
MINNEAPOLIS, MINNESOTA



Revised 11/60


1604
1604



# FOREWORD

The instruction book treats only the basic units in a 1604 system (main computer and console). This volume takes up the logical principles of operation for the basic units. Other volumes in the instruction book are:

Vol. 1 Description and Operation

Vol. 3 Maintenance

Vol. 4 File of Equations

Vol. 5 Diagrams

Instruction books for other equipments in a 1604 system are:

1607 Magnetic Tape System (two volume)

1605 Adaptor

Information included herein is subject to correction and change.

For additional copies or other information write to:

Control Data Corporation
Minneapolis 15, Minnesota

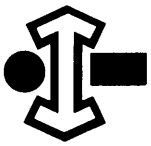


TABLE OF CONTENTS

Chapter 1 Introduction 1-1

Physical Divisions of the Computer 1-2
Over-all Analysis of Computer 1-2
Number Systems 1-9
Building Block 1-13
Logical Equations 1-22
Logic Diagram Symbols 1-24
Typical Uses of Building Block 1-27
Boolean Algebra 1-37

Chapter 2 Control Section 2-1

Program Control Register 2-3
Index Registers 2-11
Address Buffer Register 2-12
Program Address Register 2-17
Control Sequences 2-20
Static Control 2-72
Main Computer Operating Controls 2-83
Master Clock 2-90

Chapter 3 Arithmetic Section 3-1

Arithmetic Registers 3-1
Basic Operations 3-6
Addition 3-9
Subtraction 3-13
Shifting 3-15
Multiplication 3-19
Division 3-26
Floating-Point 3-31
Logical Product 3-41
Register Sensing Pyramids 3-42
Arithmetic Faults 3-45

Chapter 4 Storage Section 4-1

Principles of Magnetic Core Storage 4-3
Address Selection 4-13
Bit Plane Circuits 4-23
Storage Sequence Control 4-30
Electronic Theory of Memory Circuits 4-44

TABLE OF CONTENTS (CONT'D.)

Chapter 5 Input-Output Section 5-1

Communication Cable Groups 5-3
Equipment Selection 5-7
Computer Input 5-9
Computer Output 5-10
Input-Output Control 5-13
Input-Output Instructions 5-25
External Function Sequence 5-30
Auxiliary Sequence 5-35
Search and Transfer Sequence 5-49

Chapter 6 Console Input-Output Equipment 6-1

Modes for Handling Data 6-2
External Functions 6-3
Input Distributor 6-8
Output Distributor 6-10
Paper Tape Reader 6-12
Paper Tape Punch 6-17
Typewriter 6-20

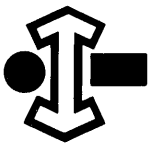
Chapter 7 Power and Cooling 7-1

Motor Generator 7-1
Main Power Distribution 7-6
Main Cabinet Distribution 7-6
Console Power Distribution 7-9
Cooling System 7-11
Protective Interlock System 7-12

Appendices

A. Borrow Pyramid A-1
B. Glossary of Terms B-1
C. List of Instructions C-1

List of Figures 1
List of Tables 4



# CHAPTER 1

# INTRODUCTION

The Principles of Operation volume of the 1604 instruction book describes the operation of the central computer and the external equipment at the console. Chapter 1 presents background material and acquaints the reader with the overall logical operation and structure of the computer. The next four chapters give a detailed analysis of the major sections of the computer: control, arithmetic, storage and input-output. The final two chapters deal with the console-mounted external equipment and the power system. Principles of operation of the 1605 Adaptor and the 1607 Magnetic Tape System are presented in separate volumes.

Principles of Operation emphasizes what circuits do logically rather than how they operate electrically in performing a function. The logical function of a type of circuit varies from instance to instance, but since the computer is constructed from a great number of electrically similar circuits and stages, a single examination of the electronics of the basic circuit will be sufficient. The small number of unique circuits are treated at the point where their logical use is discussed.

Reference material supplementing descriptions of computer operations is located in several sections of the instruction book:

1) File of Equations (volume 4) - the complete and ultimate source of all information concerning the logic of the computer
2) Logic diagrams (volume 5) - a graphic representation of the logical relationship given in the File of Equations; may be used as a detailed supplement to the logic discussed in this volume

3) Command timing charts (volume 3) - for each instruction, a sequential list of the commands performed in the execution of the instruction

4) Octal operation code(appendix C) - octal designation for each instruction

In addition, a glossary of abbreviations and terms is located at the back of this volume. Abbreviations and terms that are employed with a special or unusual significance are defined there.

## PHYSICAL DIVISIONS OF THE COMPUTER

The 1604 Computer consists of the main computer cabinet, the console and the 1607 magnetic tape cabinet. A 1604 system (figure 1-1) may also have the 1605 Adaptor cabinet and the several IBM external equipments which are connected to the computer by means of the 1605. The latter provides data buffers and control circuitry for the IBM card reader and punch, line printer and tape units.

The main computer cabinet contains the computer proper and the control circuitry for the external equipment at the console. The console holds the operator's panel which provides indicators and operating controls; it also contains the monitoring typewriter, paper tape punch and paper tape reader. The 1607 magnetic tape cabinet houses four tape units and the associated control circuitry.

## OVERALL ANALYSIS OF COMPUTER

The computer can be divided functionally into four major sections: (1) input-output, which provides the means of communication between the computer and the various external equipments; (2) arithmetic, which performs the arithmetic and logical operations required for the execution of instructions; (3) storage, which provides internal storage for both data and instructions; and (4) control, which coordinates and sequences all the operations which

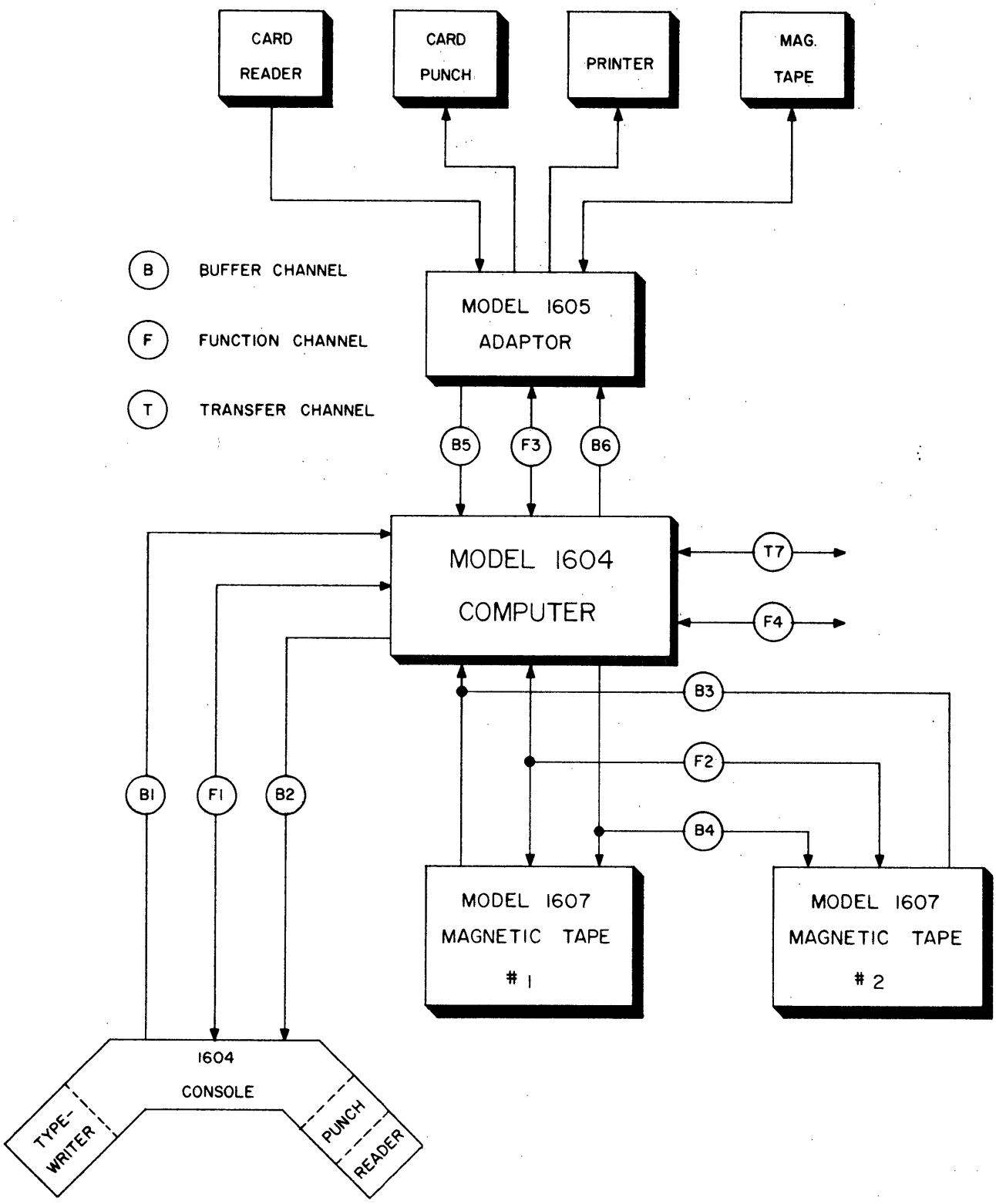



Figure 1-1. Typical 1604 System.

carry out the execution of an instruction.

Figure 1-2 is a simplified diagram of the main computer. All the registers of the computer appear in this diagram; however, the control circuits such as sequences and individual control flip-flops are not shown. The following paragraphs describe each register briefly; next, the basic operations performed in the execution of a pair of typical instructions are discussed.

## INPUT-OUTPUT SECTION

There are four input channels (1, 3, 5 and 7) which bring information into the computer. The X register receives the information from these channels. Channels 1, 3 and 5 are used for buffer communications; channel 7 is used in transfer type communication, a very high-speed method of exchanging data. Information from the input equipment on the 1604 console is always received via channel 1. Typically, channel 3 is connected to a 1607 Magnetic Tape System. Channel 5 provides another means of input which may be used by the 1605 Adaptor, a second 1607 or another equipment.

Output registers $0^1$, $0^2$, $0^3$ and $0^4$ are used to send information from the computer via output channels 2, 4, 6 and 7, respectively. Function inverters $0^0$ are used to transmit control information to the various external equipments.

## ARITHMETIC SECTION

The accumulator or A register is the principal arithmetic register since nearly all arithmetic and logical operations make use of A. This register has provisions for the parallel addition of X to its content, and can be shifted either separately or in conjunction with the Q register.

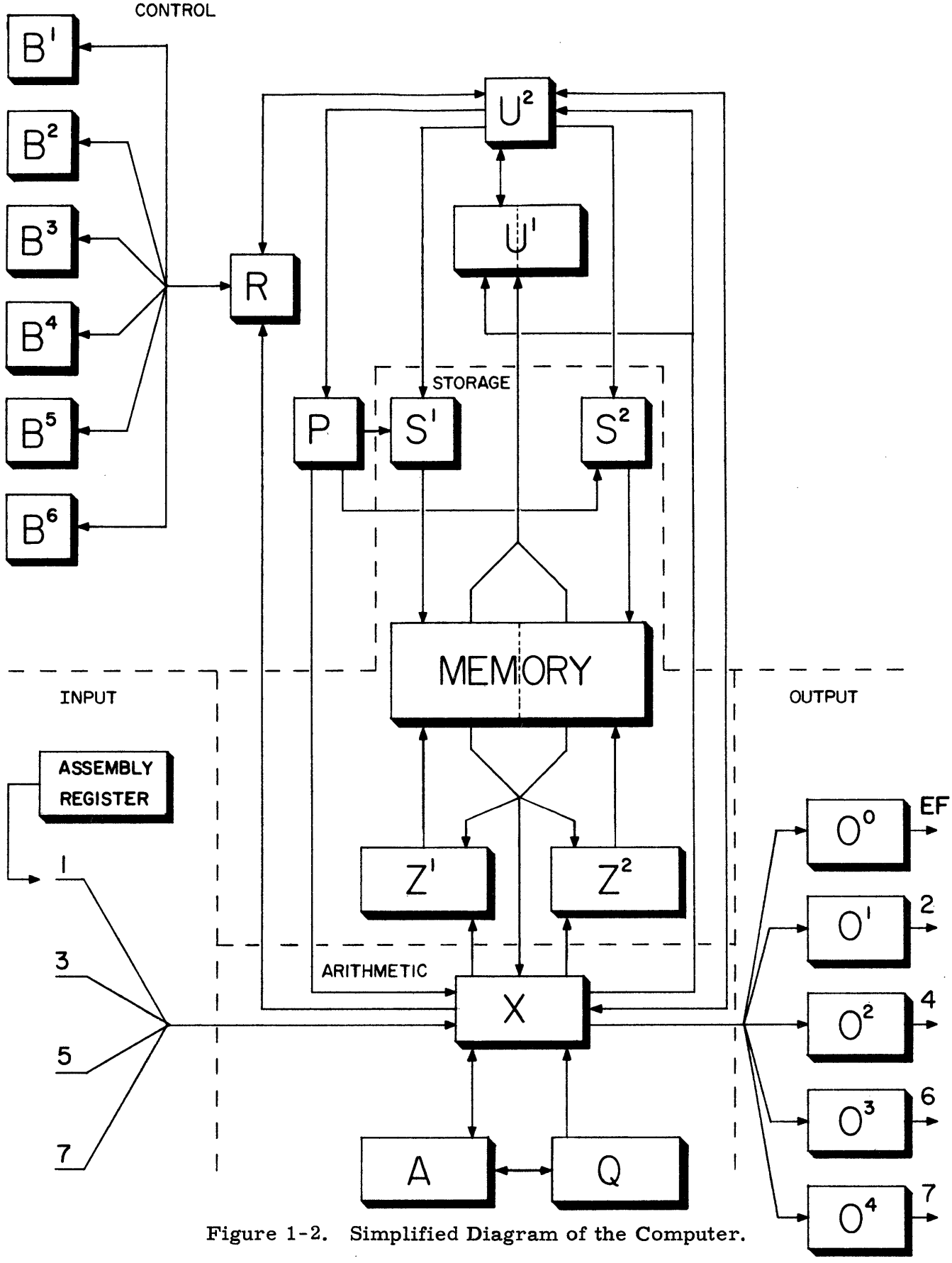



Figure 1-2. Simplified Diagram of the Computer.

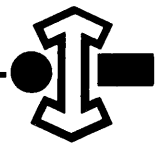
The Q register is an auxiliary arithmetic register; it assists the accumulator in the performance of the more complicated arithmetic operations. It is used in combination with the X register in the formation of logical products and can be shifted either separately or in conjunction with A.

The X, or exchange, register is used in arithmetic operations as well as in most data transmission between various sections of the computer.

## STORAGE SECTION

Two magnetic core storage units form the heart of the storage section. Each unit contains 16, 384 locations or addresses for 48-bit words; thus, the total storage capacity is 32, 768 words. One unit is called even because all locations denoted by even address words are contained in it; the other unit is called odd for similar reasons. Since the two units are independent, references to them can overlap considerably. Each unit has a storage address register into which is entered the address of the location from which a 48-bit word is to be taken or into which it is to be entered. The address register for the even storage unit is $S^1$, while $S^2$ is the address register for the odd storage unit. In addition, each unit has a storage restoration register ($Z^1$ and $Z^2$) which holds the word to be written into a given storage location.

Words to be read out of either storage unit are entered into the X register, from which they are transmitted to the appropriate register. Words to be entered or written into a storage unit are transmitted from X to the appropriate Z register, which is then sampled to determine what is to be entered in the location.



CONTROL SECTION

The control section acquires an instruction from storage, then interprets it and sends the required commands to other sections. A 24-bit instruction is composed of three parts or codes, designated by the letters f, b and m. The upper six bits are the operation code (f), which specifies the operation to be performed. The operation code is designated by two octal digits; for example, 01 refers to the operation A Right Shift.

The next three bits are the designator (b), which ordinarily refers to the index (B) register whose content is to be added to the base execution address, the remaining 15 bits of the instruction. In most cases the base execution address (m) is used in specifying the storage location of the operand used in execution of the instruction. The content of the B register denoted by b, that is, $B^b$, is added to m to give the actual address of the operand. When b = 0, no addition takes place and m itself specifies the operand address. When b = 7, indirect addressing is used (see chapter 2). The sum of $B^b$ and m is denoted by M.

A program step is a pair of 24-bit instructions, which together occupy one storage location as a 48-bit word. The higher-order 24 bits of such a word are called the upper instruction and the remaining 24 bits are called the lower instruction.

The P register, the program address counter, provides continuity by generating in sequence the storage addresses at which the individual steps of the program are contained. At the completion of each step the count in P is advanced by one to specify the address of the next step. Jump instructions enter a new address in P to begin a new series of program steps.

The program control register, $U^1$, holds a program step while the two instructions contained in it are executed. The 48-bit instruction word is taken from the storage location specified by P and entered into $U^1$. The upper instruction is always executed first. Execution of the lower instruction follows, except when the upper instruction is a jump or provides

for the conditional skipping of the lower one.

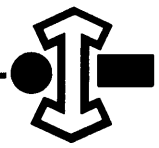
The auxiliary program control register, $U^2$, is an accumulator used primarily in the modification of the base execution address by the addition to it of $B^b$. $B^1$ through $B^6$, the six index registers, hold quantities used to modify the execution address of instructions.

The address buffer register, R, is used in transmissions to and from the B register. For instance, in the modification of the execution address, $B^b$ is transmitted to R; R is then added to $U^2$, which holds the base execution address. The R register is also a counter used during the execution of several instructions.

## EXECUTION OF TYPICAL PAIR OF INSTRUCTIONS

Address 00500 contains the following pair of instructions: upper instruction 14 Add and lower instruction 50 Enter Index; this pair is the next step to be performed in the program. The P register holds the address 00500. Because the lower bit is even, the address is in the even storage unit. The upper 14 bits of P are sent to $S^1$. The storage reference is initiated and the 48-bit word, the pair of instructions, is read from address 00500. The word is entered into $U^1$.

From this point on the operations of the computer are conditioned directly or indirectly by the 24-bit instruction in the upper half of $U^1$. This instruction, 14 Add, has as its purpose the addition of the quantity in the storage location specified by the execution address to the contents of A. The index code, b, and operation code, f, are now translated. Next, $B^b$ is transmitted to R. At the same time the 15-bit base execution address is transmitted from $U^1$ to $U^2$. The content of R is added to $U^2$ to yield the execution address, M, which specifies the location of the operand.



Depending on whether the lowest bit of $U^2$ is a "0" or a "1", the remaining 14 bits are transmitted to $S^1$ or $S^2$ in preparation for reading the operand which is to be added to A. The storage reference is initiated, and somewhat later the operand is entered into X.

The chief operation of the instruction adding X to A now occurs. All of the commands pertinent to the performance of this operation are conditioned by the operation code, f, which has the value 14. After the addition (the sum remains in A), the execution of the instruction is completed.

The computer is ready to execute the next instruction, which is presently contained in the lower 24 bits of $U^1$. The upper half of $U^1$ is cleared and the lower half transmitted to the upper half, so that $U^1$ upper holds instruction 50 Enter Index. The purpose of this instruction is to place the 15-bit base execution address in index register b. Following translation of f and b, the base execution address is transmitted from $U^1$ to $U^2$. Now $U^2$ is transmitted to R, and R in turn is transmitted to the B register specified by b. With this the execution of 50 is complete, and the next pair of instructions is read from storage in the manner described previously.

## NUMBER SYSTEMS

The basic number system used throughout the 1604 is binary. In this system the only digits (commonly called bits) are "0" and "1". All numbers, regardless of magnitude, are expressed by means of these digits. Information to be entered into the computer is in binary notation; similarly, data sent out from the computer are in binary. All operations within the computer use the principles of binary arithmetic.

## ONE'S COMPLEMENT NOTATION

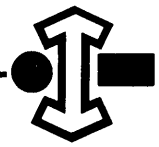
One's complement notation is used to express negative numbers. To generate the one's complement expression of negative five, for example, each bit of the binary expression of positive five, 0101, is complemented. The result is 1010. The fact that the leftmost digit is 1 indicates that the quantity is negative; if the leftmost digit is 0, the quantity is positive.

One's complement notation is used for expressing negative numbers because of the ease with which complementing can be accomplished. The use of this notation requires that the accumulators (registers with provisions for addition of a quantity to their content) be the closed type. A closed accumulator provides for an end-around carry in the addition of a quantity (or borrow, if the accumulator is subtractive). An end-around carry (or borrow) is one which is generated in the highest stage and sent to the lowest.

## TWO'S COMPLEMENT ARITHMETIC

The counters in the computer employ the principles of two's complement arithmetic. A counter is a register with provisions for increasing its content (if it is additive) or decreasing its content (if it is subtractive). A two's complement counter is open-ended; that is, there is no end-around carry or borrow. To illustrate the use of two's complement arithmetic in a counter, suppose that the content of a subtractive counter (such as R) is positive seven (0111) in binary and that this quantity is to be reduced by one. This is accomplished, in effect, by adding the two's complement expression of negative one, namely, 1111, to 0111 as shown below. The result is six.

```
 0111
 1111
------
 0110
```



OCTAL NUMBER SYSTEM

Operation of the computer is described by the octal number system because three binary digits may be expressed by one octal digit, thus reducing the total digits involved by a factor of 3. The octal system is selected as a basis for obtaining shortened expressions for numbers because of the ease with which numbers can be converted from binary to octal and from octal to binary notation. Such conversions can be performed much more easily than conversions between binary and decimal numbers.

The following example shows how these conversions between binary and octal numbers are performed. The binary expression for thirty-eight decimal is 100110. To convert this expression to octal, separate it into groups of three digits each as follows: 100 110. Each group of binary digits is converted individually to the corresponding octal digit. Performing this conversion, 46 is obtained as the octal expression.

The conversion of a number in octal notation to binary is simply the reverse. To convert the octal expression 513 to its binary equivalent, each octal digit is converted individually to its binary equivalent, as shown below:

| 5 | 1 | 3 |
|---|---|---|
| 101 | 001 | 011 |

The binary expression for an octal digit must be filled out with 0's to make it three digits in length.

Octal notation is used to refer to the content of a register, the value of a control designator, addresses in storage and to the codes for instructions. The octal number system is not actually used in the operation of the computer.

## ARITHMETIC PROPERTIES OF REGISTERS

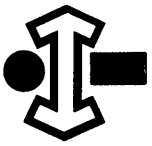
Arithmetic operations are performed in five registers of the computer. Their arithmetic properties are listed in table 1-1.

TABLE 1-1. ARITHMETIC PROPERTIES OF REGISTERS

| Register | No. of Stages | Modulus | Complement Notation | Sub. or Add | Closed or open ended | Result: Signed or Unsigned |
|---|---|---|---|---|---|---|
| A accumulator | 48 | $2^{48}-1$ | one's | subtractive | closed | signed* |
| Q | 48 | $2^{48}-1$ | one's | | | signed |
| $U^2$ accumulator | 15 | $2^{15}-1$ | one's | subtractive | closed | signed** |
| P | 15 | $2^{15}$ | two's | additive | open | unsigned |
| R | 15 | $2^{15}$ | two's | subtractive | open | unsigned |

* The result of an arithmetic operation in A satisfies $A \leq 2^{47}-1$ since A always is treated as a signed quantity. When the result in A is zero, it is always represented by 000...00 except when 111...11 is added to 111...11. In this case, the result is 111...11, which is commonly called "negative zero".

** When $U^2$ is used as an operand for an instruction, it is regarded as a signed quantity, and $U^2 \leq 2^{14}-1$. When it is used to specify the address of an operand, it is regarded as an unsigned quantity, and $U^2 \leq 2^{15}-1$.



## BUILDING BLOCK

The basic building block of the computer is a transistorized single inverter circuit. This circuit is used: (1) alone, as a single inverter; (2) in a pair to form a flip-flop; and (3) in configuration of three to form a control delay. The major portion of the computer is constructed by interconnecting these circuits, which are packaged on 2 1/2 inch by 2 1/8 inch printed circuit cards (figure 1-3). Each card is equipped with a 15-pin male connector for plugging into the major equipment chassis.

In the following paragraphs the single inverter circuit, the flip-flop (FF) and the control delay are described. After a treatment of logical equations and logic diagram symbols (the two methods showing the connections between building blocks), some typical uses of the block in registers and counters are discussed.

## ANALYSIS OF SINGLE INVERTER

Within the computer two signal levels are used, namely, -3.0 volts and -0.5 volts. The former represents the logical "1" and the latter represents the logical "0". The logical function of the single inverter is to invert one of these signal levels into the other. A -3.0 volt input to an inverter causes a -0.5 volt output, and vice versa. The inverter circuit permits the use of varying numbers of inputs and outputs.

The standard inverter circuit is shown in figure 1-4. Transistor Q01 is connected as an emitter follower; Q02, as an amplifier. The collector circuits of the transistors are provided with two feedback loops which prevent the transistors from being driven to cutoff or saturation. As a result, switching from one state of conduction to the other is accomplished in a minimum time of 50 millimicroseconds and a maximum time of 100 millimicroseconds.

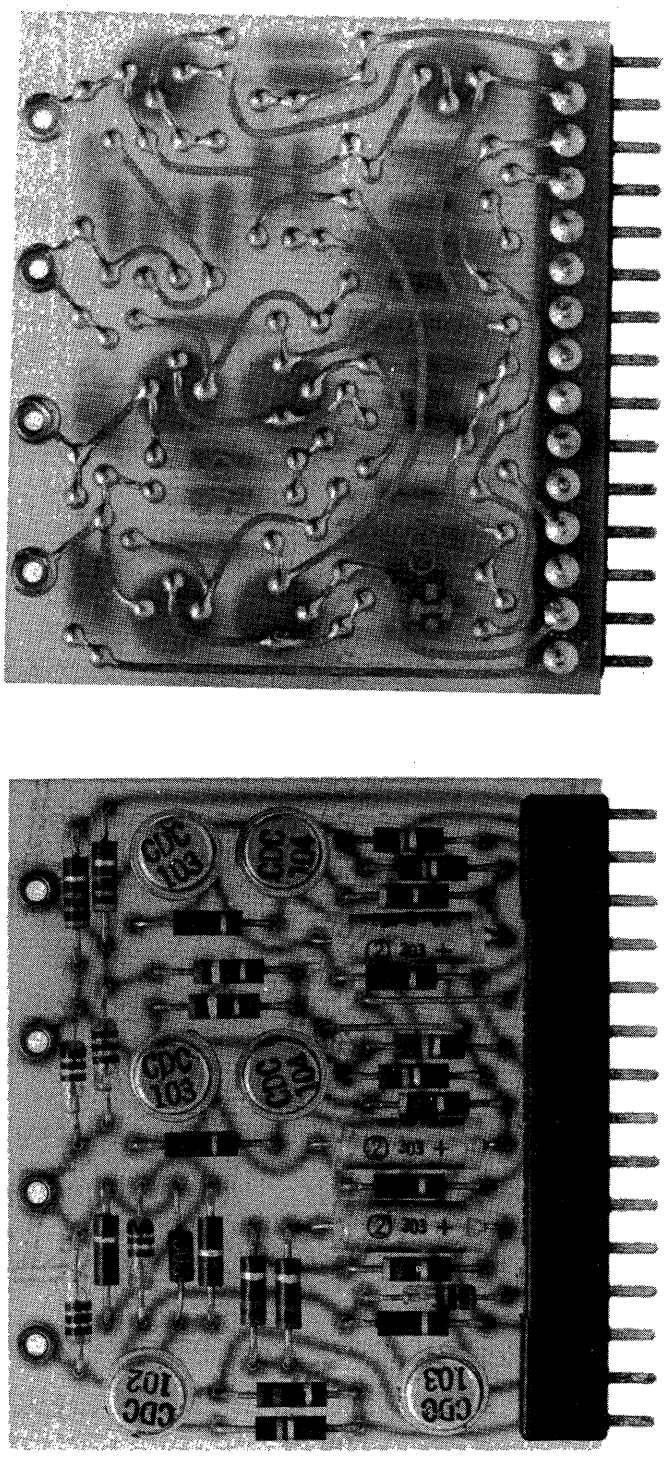



Figure 1-3. Typical Printed Circuit Card.

The input and output signal levels of the inverter circuit are -3.0 volts and -0.5 volts. An input signal is applied via isolation diodes CR01 or CR02 to a voltage divider network composed of resistors R07, R08, R09, R10 and R11. An input signal of -0.5 volts (point A) results in -1.5 volts at point B and 0.8 volts at the base of Q01 (point C). CR01 is thus biased 1 volt in the backward direction, which provides for noise suppression at the input of the inverter.

Capacitor C01, connected between CR01 and the base of Q01, provides rapid coupling of input signal changes to Q01, improving the switching time of the circuit.

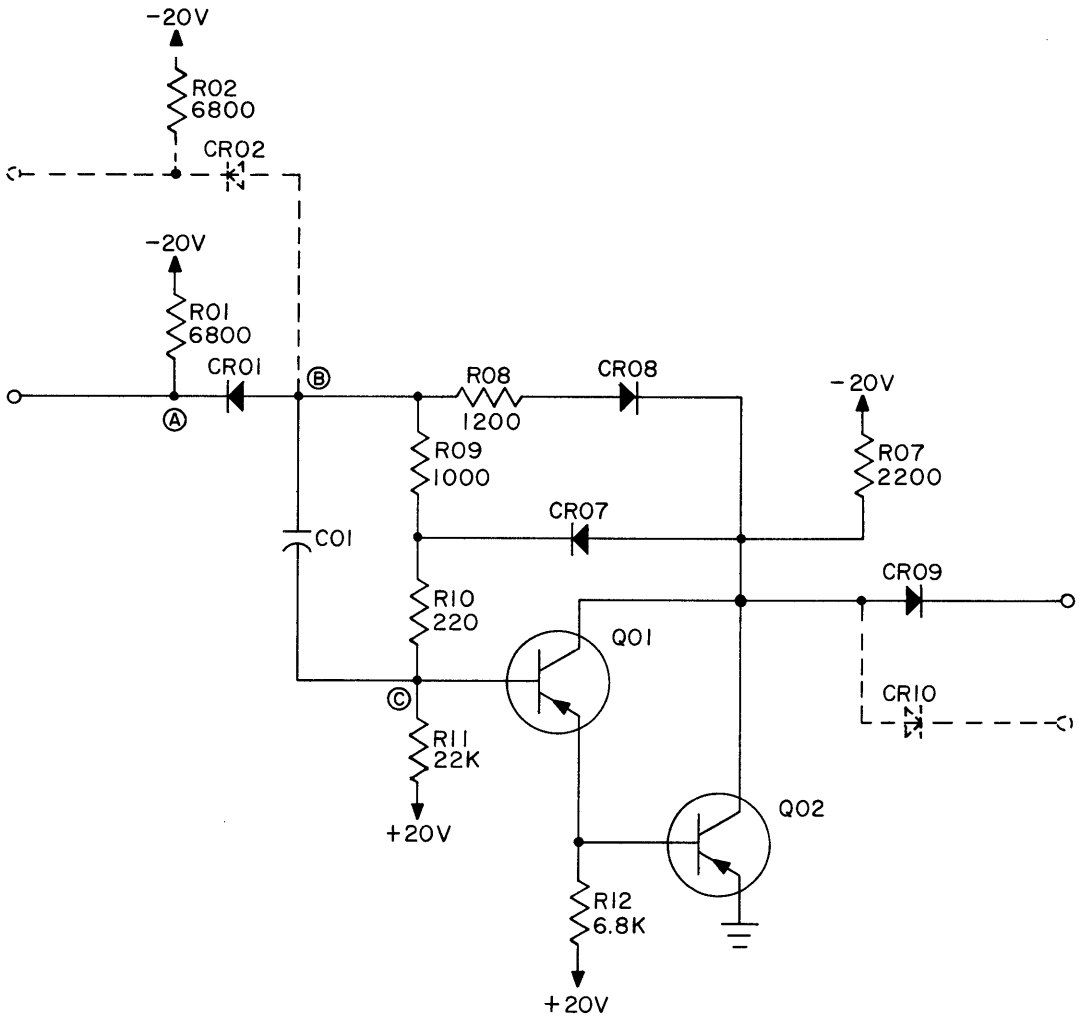



Figure 1-4. Schematic Diagram of Standard Inverter Circuit.

Transistors Q01 and Q02 each provide beta* current gains of approximately 100. Thus, loop gain of the two transistors is on the order of $10^4$. The collector current of $Q^{01}$ and $Q^{02}$ develops the output voltage across resistor R07. Output diode CR09 isolates the output line from the other output line connected to CR10.

The feedback loops which prevent transistors Q01 and Q02 from being driven to cutoff or saturation consist of diodes CR07 and CR08. The two feedback loops establish positive-going and negative-going limits for the base voltage swings of Q01. The positive-going limit allows a maximum transistor conduction that is less than saturation. Similarly, the negative-going limit fixes a minimum conduction for the transistors. When the transistors approach cutoff, their collectors approach -3 volts. The collector potential is coupled back to the base of Q01 through CR08, R08, R09 and R10. As a consequence the base of Q01 always is held at a sufficiently negative voltage to permit some minimum conduction of Q01 and thus Q02.

When the transistors approach saturation, the collectors approach 0 volts. The collector potential is coupled back to the base of Q01 through CR07 and R10. The base of Q01 is thus prevented from becoming so negative that saturation occurs.

FLIP-FLOP

All short-term storage of information in the computer is accomplished by flip-flops (FFs); thus, the various registers such as A, X, Q, etc., consist of flip-flops. A FF is composed of two single inverter circuits interconnected as shown in figure 1-5 (each rectangle represents a single inverter). One of the inverters constitutes the set side of the FF; the other, the clear side. The FF is placed in the "1" state, i.e., set, by a set input that is "1". Conversely, it is placed in the "0" state, i.e., cleared, by a clear input that is "1". (Set and clear inputs are never "1" at the same time).

* The beta current gain is the ratio of collector current to base current.

The storage capability of a FF means simply that it remains in a state that is indicative of the last "1" input received. Specifically, if a "1" pulse is present at the set input, then the output of inverter $A^{000}$ (figure 1-5) becomes "0". This output is applied as an input to $A^{001}$, whose output then becomes "1". The output of $A^{001}$ is fed back to $A^{000}$. Thus, when the set input returns to "0", the feedback connection between $A^{000}$ and $A^{001}$ permits the storage of the state to which the "1" pulse on the set input forced the FF. Should the clear input later receive a "1" pulse, the output of $A^{001}$ becomes "0", and hence the feedback input to $A^{000}$ is "0". Consequently, $A^{000}$ furnishes a "1" output which is returned to $A^{001}$ to replace the "1" pulse at the clear input.

When the FF is set, $A^{001}$ has a "1" output, and $A^{000}$ has a "0" output. Conversely, when the FF is cleared, $A^{001}$ has a "0" output, and $A^{000}$ has a "1" output.

The conventional square or box symbol for a FF is given in figure 1-5 to show the relationship between it and the inverter configuration which forms the FF. The square used on diagrams to represent the FF encompasses the crossover of outputs shown at the right of Figure 1-5.

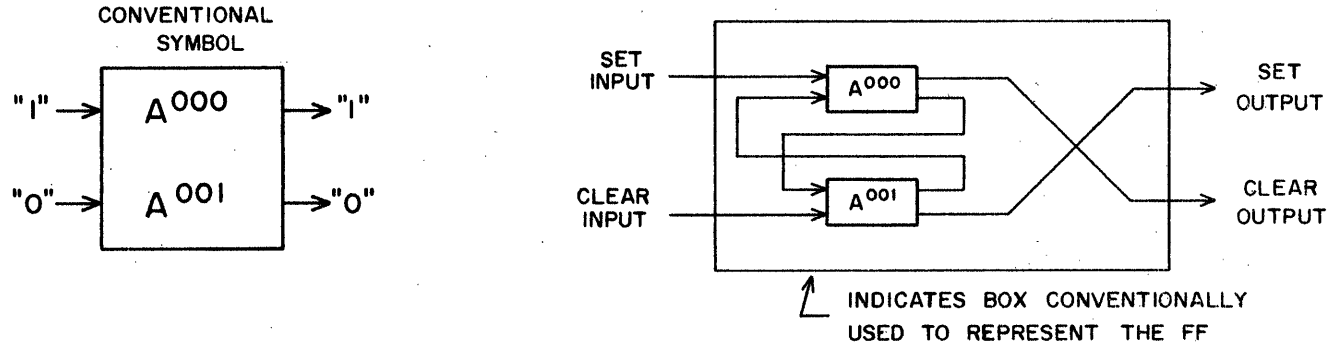


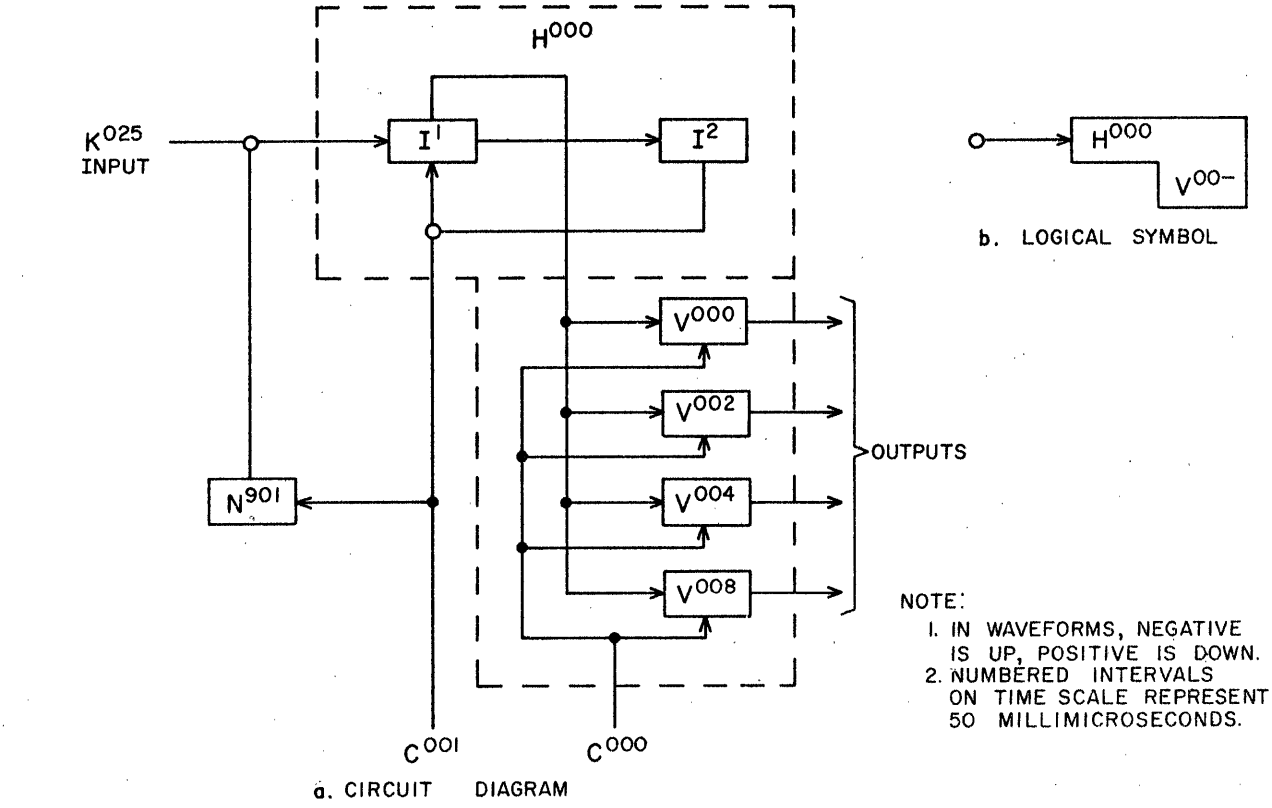

Figure 1-5. Interconnection of Inverters to Form a Flip-flop.

CONTROL DELAY

The single inverter and the FF described above are static, unclocked devices; that is, the output of the inverter is a steady-state inversion of its inputs. Likewise, once a FF is set, it provides a steady "1" from the set output and "0" from the clear output until it is cleared. However, an essential part of the internal operation of the computer is the occurrence of timed and properly shaped pulses. The function of control delays is to reshape signals and resynchronize them to furnish timed output signals. As its name indicates, the control delay inserts a controlled interval between the occurrence of its input and the occurrence of its output. This interval, or delay, is a single phase time of the master clock, namely, 0.2 microseconds. Furthermore, the minimum duration of the output pulse is 0.2 microseconds.

The two-phase master clock is essential in the operation of control delays. The outputs of the clock are two sine waves $180^{o}$ out of phase. However, for the analysis of the control delay, which is the major application of the master clock, it is sufficient to understand that the clock furnishes rectangular waves of two phases, odd and even, as shown in figure 1-6c, where they are labeled $C^{001}$ and $C^{000}$, respectively. The designations "odd" and "even" are given the clock phases because in the equation symbols (see page 23) for clocked circuits, the odd and even character of the third digit indicates the clock phase during which outputs from the circuit occur.

The control delay consists of a special FF (represented by an $H^{---}$ equation symbol) and one or more inverters (represented by $V^{---}$ or $N^{---}$ symbols) connected to the "0" output of the FF (figure 1-6a and b). The special flip-flop has set inputs only, namely, those going to $I^{1}$. The logic inputs are always signals formed from other building blocks, one of which must be clocked. Feedback from $I^{2}$ to $I^{1}$ is gated by one of the clock phases, which is opposite to that applied to the output inverters. Thus, in figure 1-6a the odd phase ($C^{001}$)



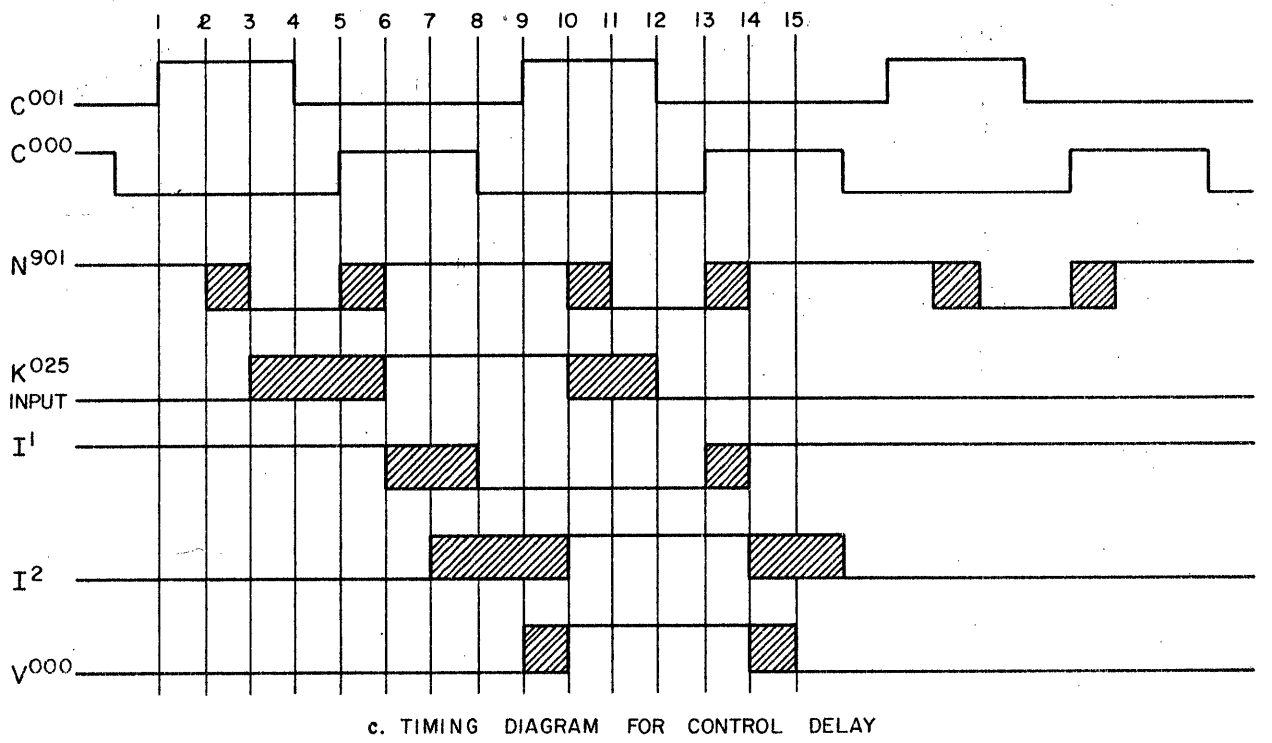




Figure 1-6. Control Delay

gates the feedback as well as clocking the input from $K^{025}$. The even phase ($C^{000}$) is fed to inverters $V^{00-}$.

During the odd clock phase ($C^{001}$) the input signal is allowed to set the special FF ($H^{000}$). The internal feedback also is gated during this clock phase so that the FF action extends or delays the original input signal. The even clock phase ($C^{000}$) acts as a gate for the FF output. The duration of the output from inverter $V^{000}$ is established by the even clock phase.

Waveforms for the various elements in the control delay are shown in figure 1-6c. For these waveforms it has been assumed that the internal switching time of each inverter is the minimum value of 50 millimicroseconds. Shaded areas indicate the variations in the occurrence of pulses due to external wiring delays. If, for example, these wiring delays were reduced to zero, then the output of $N^{901}$ would go to "0" at time 2 and remain so until time 5. On the other extreme, the delays could amount to a maximum of 50 millimicroseconds. In this case the output of $N^{901}$ would go to "0" at time 3 and remain so until time 6.

The time at which the output of $I^1$ may go to "0" varies over a 100 millimicrosecond period. Half of this period is due to the fact that delays introduced at $N^{901}$ are felt at $I^1$ also. Thus, if $N^{901}$ had the maximum delay but $I^1$ had zero delay, then the $I^1$ output goes to "0" at time 7 and remains there until time 13. If both $N^{901}$ and $I^1$ have the full delay, then the $I^1$ output is "0" from time 8 to time 14.

Assuming that capacitive wiring delays are zero, the leading edge of the output from $V^{000}$ occurs at time 9 because the clock input to $V^{000}$ (from $C^{000}$) does not go to "0" until time 8. The logic input signal goes to "0" at time 10; however, $C^{001}$ allows this signal to be replaced by gating the feedback from $I^2$ to $I^1$ until time 12. As a result, the original input signal is provided as an output from $I^1$ until at least time 13.

The output of $I^1$ encompasses the "0" portion of $C^{000}$ (figure 1-6c). Since the output of $V^{000}$ is actually the AND function of "not $C^{000}$" and "not $I^1$", it is a "1" only while both are "0". Therefore, the time of occurrence and duration of the $V^{000}$ output is determined by the period that $C^{000}$ is "0".

The 0.2 microsecond delay produced by the control delay is shown in figure 1-6c by the relationship of the $K^{025}$ and $V^{000}$ waveforms. The input from $K^{025}$ will go to "1" by time 6 regardless of the wiring delays. The result of this "1" input will be felt at the output of $V^{000}$ by time 10. Thus the signal has been delayed by 0.2 microseconds.

AND CIRCUIT

The AND circuit is shown in figure 1-7. The diodes of an AND circuit are the output diodes of inverters. As many as four diodes, each from different inverters, may be connected in an AND. The common cathode connection of the diodes is tied to the input of an inverter, which furnishes the remaining elements of the AND circuit. In order for the output of the AND to be a "1", that is, at -3 volts, inputs A, B and C must all three be at -3 volts. If any of the inputs are at -0.5 volts, a "0", then the cathodes of all three diodes are held at this potential, as is the output at D.

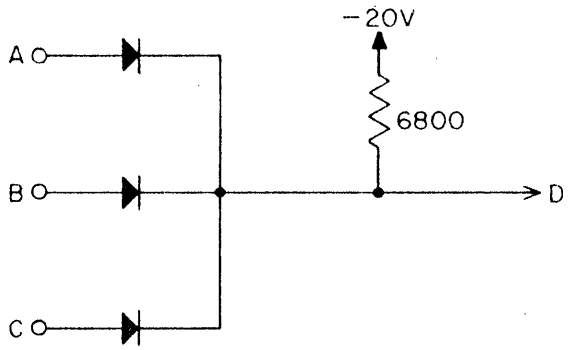



Figure 1-7. AND Circuit.

OR CIRCUIT

The OR circuit consists of components at the input of an inverter. The inverter shown in figure 1-4 has a two-input OR circuit, which involves R01, CR01, and R02 as well as voltage divider R09, R10 and R11 connected to -20 volts.

The potential at B, the common junction of the anodes of the OR diodes, is -1.5 (indicating a "0" in the circuit) only if both inputs at the cathodes of CR01 and CR02 are at -0.5 volts. If either OR input goes to -3.0 volts (a "1") then the potential at B is forced more negative than -1.5 volts. This more negative potential indicates a "1".

## LOGICAL EQUATIONS

The logical interconnections of virtually all circuits in the computer are described by means of logical equations found in the File of Equations. As a preliminary step to formulating such equations, every circuit is assigned a unique symbol consisting of a base letter and a 3-digit superscript (figure 1-8).

The base letter of the symbol associates the building block with one of 26 major logical areas, such as the A register, B register, etc. The superscript digits provide a unique identification of the block within the area. In addition, the odd or even character of the third digit may identify the output clock phase of a circuit or set and clear sides of a FF.

These symbols are used in writing the equation, which are actually specialized instances of Boolean algebraic equations. An equation represents a single inverter with the exception of two classes of equations dealing with circuits in the storage section.

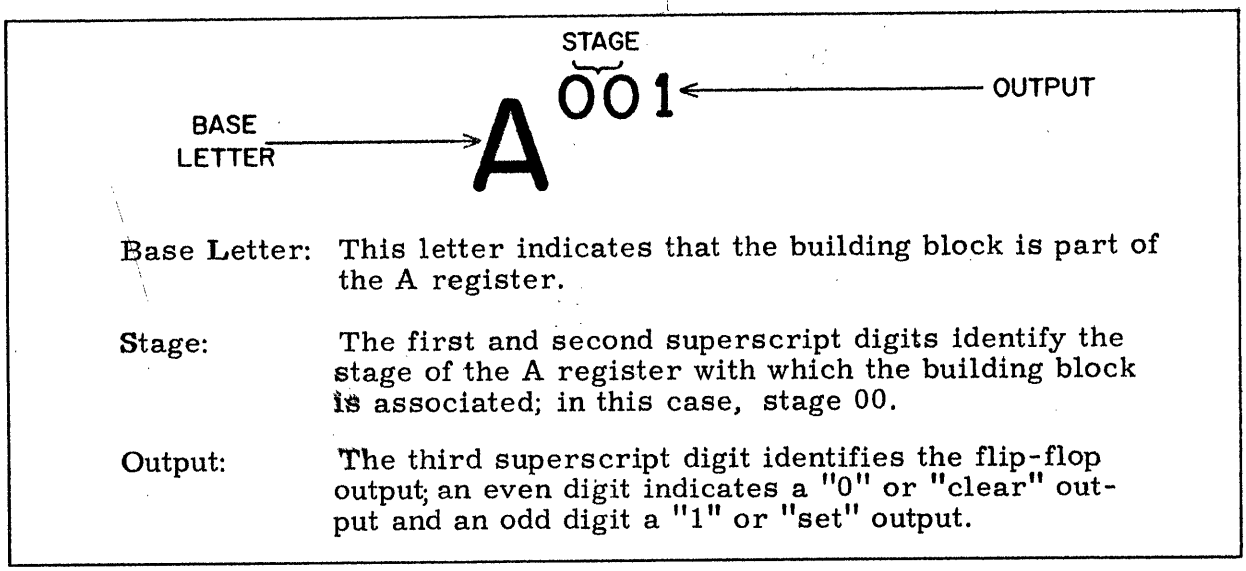



Figure 1-8. Typical Designation for a Building Block used in a Register.

From the logical viewpoint a single inverter is simply a circuit which provides as an output the inverted form of its input. Thus if any of the inputs to an inverter is a "1", its output is a "0"; conversely, its output is a "1" only if all of its inputs are "0". An equation is a logical representation of the inverter. For example:

$$K^{310} = K^{311} + V^{220}\ F^{585}\ K^{415} + V^{676}\ F^{940}\ J^{134}$$

The symbol on the left of the equal sign, called the subject term, denotes the inverter described by the equation. The expression on the right of the equal sign describes the logical configuration of the inputs.

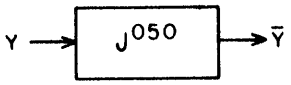
The + sign represents the OR function or logical sum, while the absence of a sign between symbols represents the AND function or logical product. In the context of equations, the word "term" is used to designate a single symbol or group of symbols that is a logical product. The equation given above for inverter $K^{310}$ has three terms, each representing an input to the inverter. Thus $K^{310}$ has a "0" output if: (1) $K^{311}$ is a "1"; (2) the AND function of $V^{220}$, $F^{585}$ and $K^{415}$ is satisfied, that is; if each of them is a "1"; or, (3) the AND function of $V^{676}$, $F^{940}$ and $J^{134}$ is satisfied, that is, each of them is a "1".

A two-phase master clock times computer operations. Circuits which receive timing signals from the clock are denoted by symbols with base letters H, V and N. The base letter of master clock symbols is C and of clock slaves, $N^{9--}$. In these symbols (those with the base letters H, V, N, C and $N^{9--}$) the even or odd character of the third superscript digit indicates timing relations. A $C^{---}$ or $N^{9--}$ symbol with an odd third superscript digit represents a circuit furnishing odd phase clock pulses; these symbols with even third superscript digits represent circuits furnishing even clock pulses. The $H^{---}$, $V^{---}$ and $N^{---}$ circuits with an odd third digit provide output during the odd clock phase and receive input during the even phase. The same circuits with even third digits provide output during the even clock phase and receive input during the odd phase. Certain circuits (those with symbols having base letters of D, G, L, M and Y) are not represented by complete equation entries. In these circuits only inputs or outputs (but not both) are represented by equation symbols. Thus either the inputs or the outputs are missing from the entry.

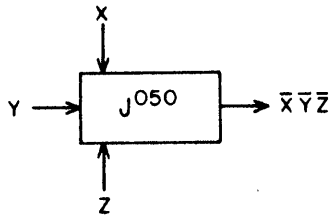
## LOGIC DIAGRAM SYMBOLS

The logic diagrams use five basic symbols for representing the logical properties of circuit configurations in the computer. The shape of each symbol designates a basic logical function. Equation symbols (such as $A^{001}$) enclosed by the diagram symbol provide a complete identification of the building blocks which are interconnected to perform the logical function. Inputs to the diagram symbol are identified by arrows; outputs, by the absence of arrows.

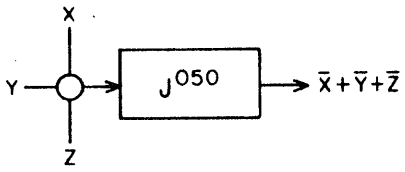
The symbols used in portraying the equations in diagrammatic form are shown and defined in figure 1-9. Since other logical elements such as the flip-flop and control delay are fundamentally configurations of two or more inverters (figure 1-9a), the diagrammatic symbols for these complex elements are formed from combinations of the inverter symbols. Each inverter rectangle contains the logical designation (such as $J^{050}$) of the inverter.



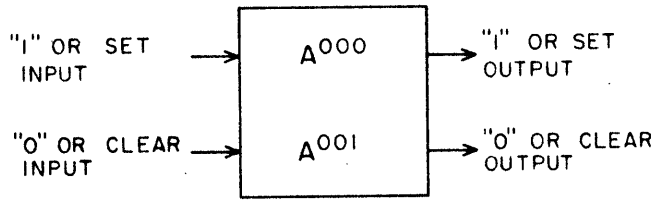

a. SINGLE INVERTER



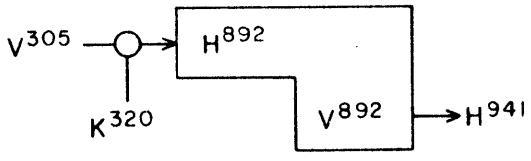

b. SINGLE INVERTER WITH THREE "OR" INPUTS




c. SINGLE INVERTER WITH "AND" INPUT




d. FLIP-FLOP




e. CONTROL DELAY

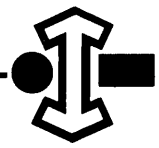
Figure 1-9. Logic Diagram Symbols.

The diagrammatic form of the OR function is the representation of the inputs (by arrows) to the inverter (figure 1-9b). The AND function is represented by a small circle. An input to the AND is represented by a line; the output from the AND (which is input to a logical element such as an inverter) is represented by an arrow.

The flip-flop (FF) is a storage device with two stable states, designated "1" (or set) and "0" (or clear), and is composed of two inverters. The logical symbol (Figure 1-9d) is a square that is formed from the rectangles which represent the two inverters. The logical designations of the two inverters appear within the square. In a logic diagram, the inverter which receives the set input is at the top and the inverter which receives the clear input is at the bottom. Set outputs are received from the top inverter and clear outputs from the bottom. This diagrammatic convention simplifies the actual interconnection of the inverters, which is shown in figure 1-5.

With the exception of the FFs which form the B registers, the logical designation of the set side of a FF has an even last digit and the clear side of the same FF is designated by the next odd digit; for example, $K^{942}$ (set side) and $K^{943}$ (clear side).

A control delay (figure 1-9e) consists of an $H^{---}$ part, which receives the input, and a $V^{---}$ or $N^{---}$ part, which provides the output. Control delays are clocked configurations which receive inputs during one clock phase and furnish a resultant output during the opposite clock phase.



TYPICAL USES OF BUILDING BLOCK

REGISTERS

A register is a device capable of storing a quantity or word. The register is made up of stages, each of which stores an individual bit of the word. A stage, therefore, may be considered as a bit register. There are two types of registers in the computer, single-rank and double-rank registers. The single-rank register consists of a single FF per stage and has storage properties only. The double-rank register consists of two FFs per stage and has either shifting, counting or complementing properties in addition to storage properties, depending upon the circuit connections.

A simple, three-stage, single-rank register is shown in figure 1-10. Each stage consists of a flip-flop and one or more input gates which allow the insertion of bits into the flip-flop. As shown in figure 1-10, the input gate of each stage of the Z register is enabled by the "1" output of the corresponding stage of the X register. The signal Clear Z prepares the Z register for receipt of a word by setting each of its stages to "0". The signal X → Z, when applied to the input gates, sets those stages of Z to "1" which receive "1" enables from X.

A simple three-stage double-rank register with shifting properties is shown in figure 1-11. The left-hand rank of this register is designated $Q^1$ and the right-hand $Q^2$. The input word, from the $A^2$ register, is initially entered into $Q^1$ by the signal $A^2 \rightarrow Q^1$. The signal Circular Left Shift $Q^1 \rightarrow Q^2$ next transfers the word to $Q^2$ and shifts it left one bit. Thus, the bits in stages "0" and "1" of $Q^1$ are transferred to stages "1" and "2", respectively, of $Q^2$, and the bit in stage "2" of $Q^1$ is transferred to stage "0" of $Q^2$. Following this, the signal $Q^2 \rightarrow Q^1$ transfers the word from $Q^2$ to $Q^1$ in preparation for the next shift. The word in $Q^2$ is available as an output to the $A^1$ register whenever the desired number of shifts have been performed.

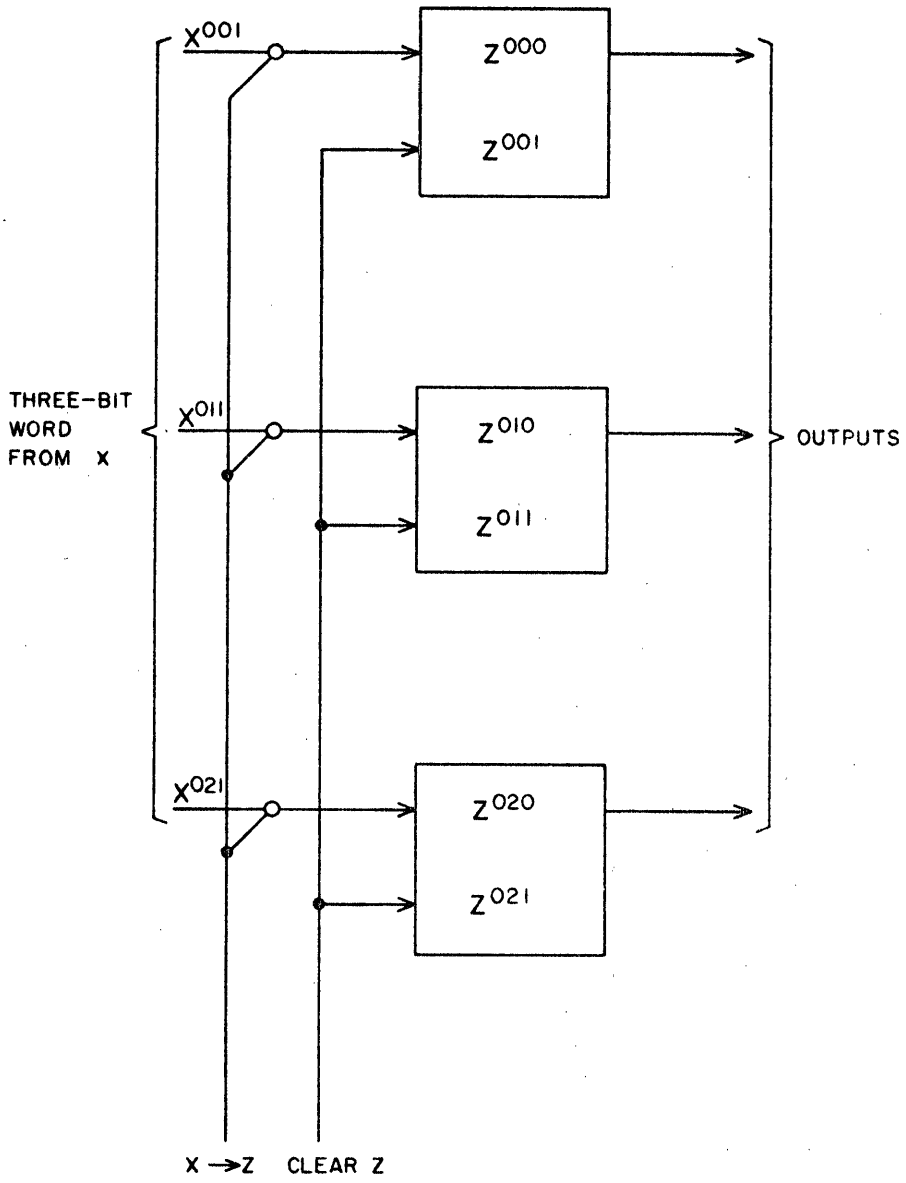



Figure 1-10. Three Stage Single-Rank Register.

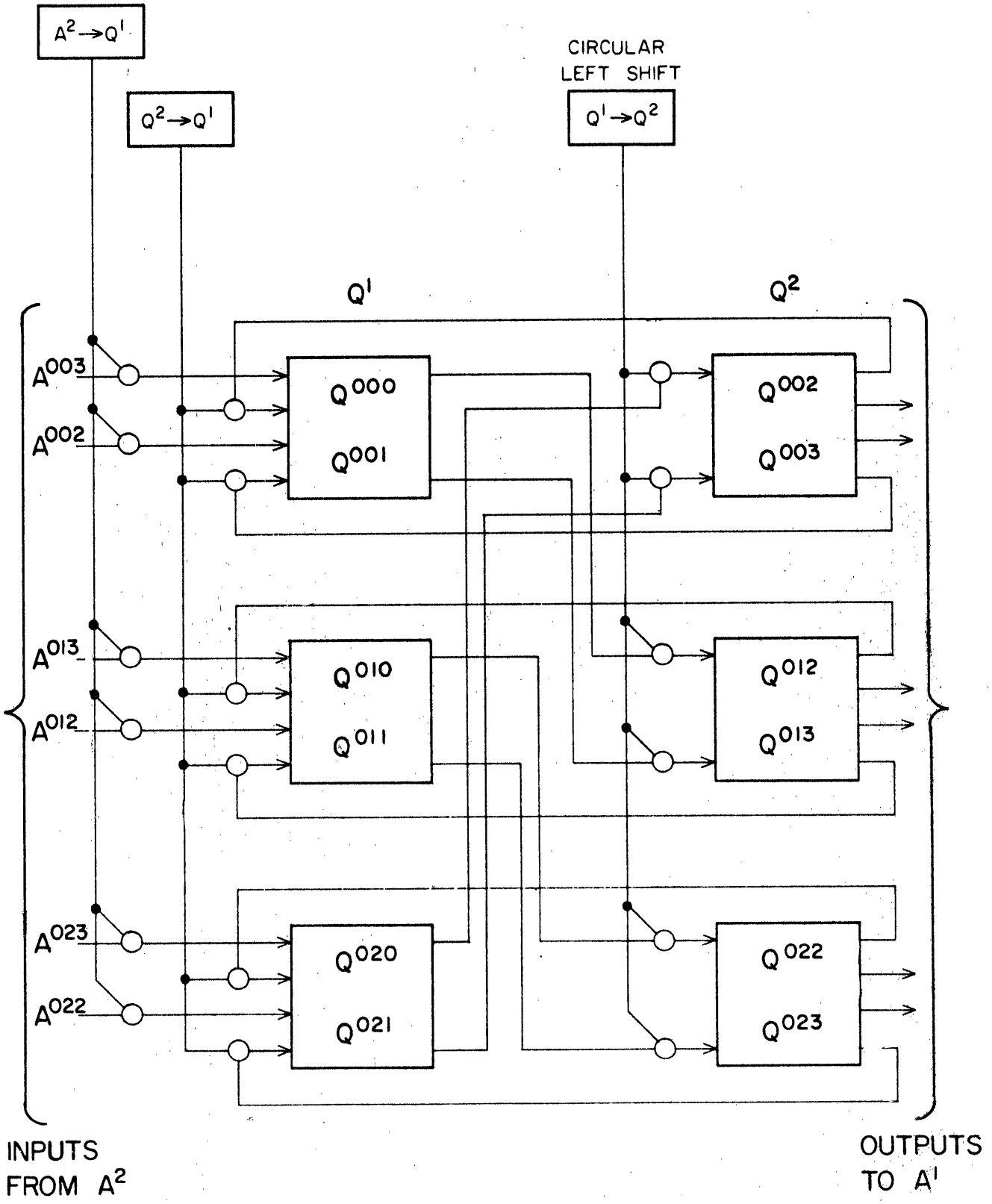



Figure 1-11. Three-Stage, Double-Rank Register with Shifting Properties.

COUNTERS

A counter is basically a double-rank register with circuitry which makes it possible to increase or decrease, by an increment, the quantity stored in the register.

Basic Three-Stage Counter

A three-stage counter circuit is shown in figure 1-12. This circuit is capable of additive counting; that is, from binary 000 through 111. Repetitive sequences consisting of the command Advance, followed by the command $P^1 \rightarrow P^2$, operate the counter.

The Advance command probes both the set ("1" side) and clear ("0" side) input gates of each FF of $P^1$. These gates provide the additive counting feature of the circuit. Each set input gate is enabled by the "0" output of the corresponding FF of $P^2$, and each clear input gate by the "1" output of the same FF. In addition, the pair of input gates to each FF is enabled by the "1" outputs of the lower-order FFs of $P^2$.

To analyze the operation of the counter, assume that both ranks of each stage initially contain a "0". The counting sequence is listed in table 1-2. The first Advance command finds the input gate to $P^{000}$ enabled and therefore enters the count 001 into $P^1$. The command $P^1 \rightarrow P^2$ transfers the count to $P^2$. The following Advance command finds the input gates to $P^{001}$ and $P^{010}$ enabled and therefore enters the count 010 into $P^1$. The operation continues in this manner until the count 111 is reached. This is the highest count which the three-stage counter is capable of reaching. This is followed by a command sequence which returns both ranks to the count 000, in preparation for the next counting cycle.

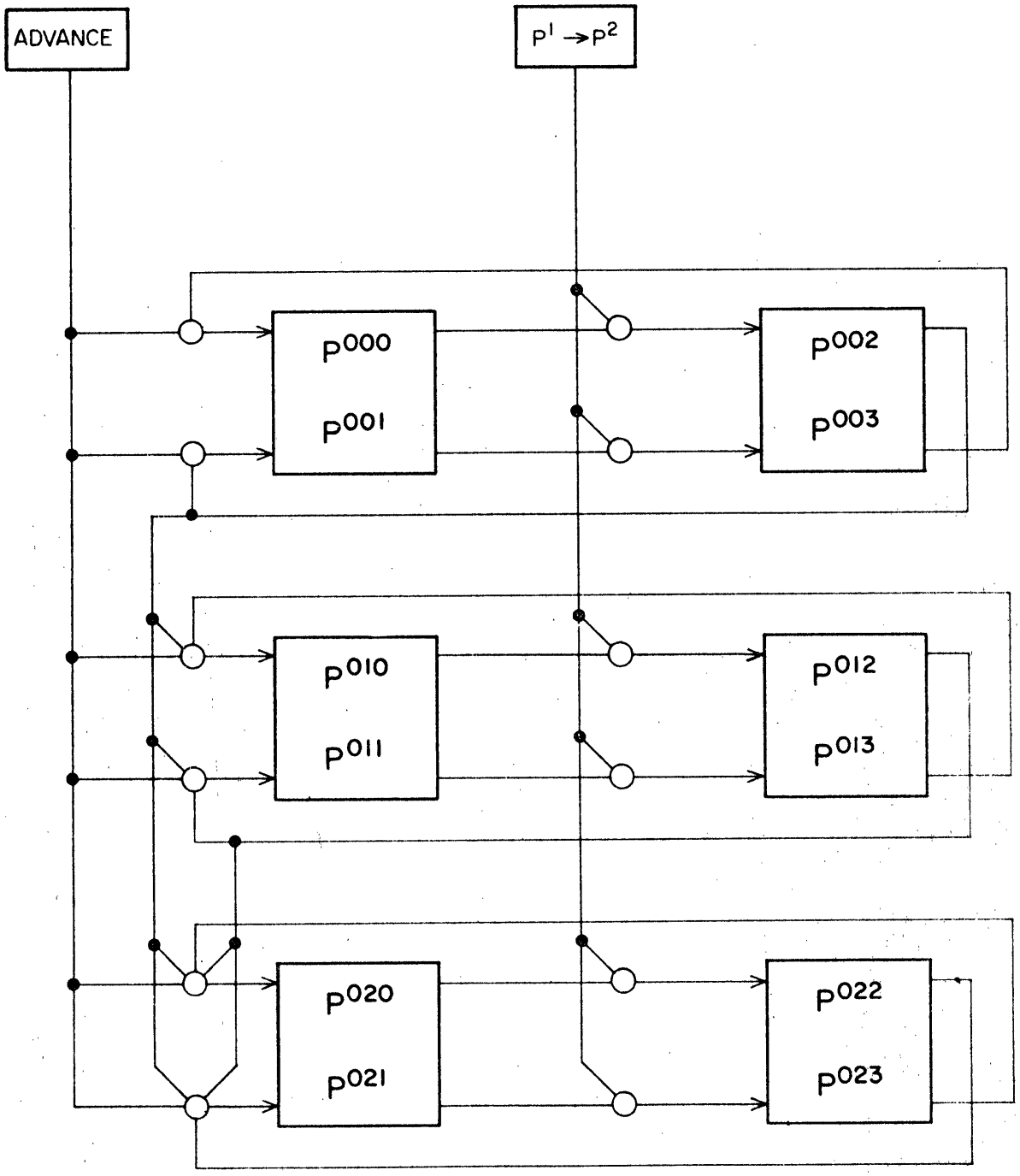
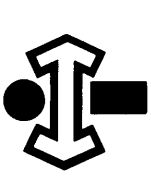



Figure 1-12. Three-Stage Counter.

TABLE 1-2. COUNTING SEQUENCE FOR THREE-STAGE COUNTER.

| Command | $P^1$ | | | $P^2$ | | |
|---|---|---|---|---|---|---|
| | $P^{02}$ | $P^{01}$ | $P^{00}$ | $P^{02}$ | $P^{01}$ | $P^{00}$ |
| Initially | 0 | 0 | 0 | 0 | 0 | 0 |
| Advance | 0 | 0 | 1 | 0 | 0 | 0 |
| $P^1 \rightarrow P^2$ | 0 | 0 | 1 | 0 | 0 | 1 |
| Advance | 0 | 1 | 0 | 0 | 0 | 1 |
| $P^1 \rightarrow P^2$ | 0 | 1 | 0 | 0 | 1 | 0 |
| Advance | 0 | 1 | 1 | 0 | 1 | 0 |
| $P^1 \rightarrow P^2$ | 0 | 1 | 1 | 0 | 1 | 1 |
| Advance | 1 | 0 | 0 | 0 | 1 | 1 |
| $P^1 \rightarrow P^2$ | 1 | 0 | 0 | 1 | 0 | 0 |
| Advance | 1 | 0 | 1 | 1 | 0 | 0 |
| $P^1 \rightarrow P^2$ | 1 | 0 | 1 | 1 | 0 | 1 |
| Advance | 1 | 1 | 0 | 1 | 0 | 1 |
| $P^1 \rightarrow P^2$ | 1 | 1 | 0 | 1 | 1 | 0 |
| Advance | 1 | 1 | 1 | 1 | 1 | 0 |
| $P^1 \rightarrow P^2$ | 1 | 1 | 1 | 1 | 1 | 1 |
| Advance | 0 | 0 | 0 | 1 | 1 | 1 |
| $P^1 \rightarrow P^2$ | 0 | 0 | 0 | 0 | 0 | 0 |



Multistage Counters

Multistage counters are formed by interconnecting several three-stage counter units, as shown in figure 1-13. A Carry Enable, indicating the count 111, is obtained from each unit. Such enables condition the advancing of the higher-order units. The Advance command is applied to a higher-order unit only if the appropriate enables are obtained from all the lower-order units. Thus the third unit is advanced by one count each time the two lower-order units change from "1's" to all "0's".

The counting sequences for multistage counters are identical to those listed in table 1-2, except for the larger numbers involved.

For the sake of convenience in drawing, the $H^{---}$ and $N^{---}$ parts of the control delay have been shown separately in figure 1-13. A brief analysis of the conditions required for advancing the third group will aid in understanding the operation of the counter. Advancing this group requires a "1" output from $N^{004}$; therefore all four inputs to $N^{004}$ must be "0". The occurrence of the ADVANCE command causes the output of $H^{970}$ to be "0". The output of $H^{002}$ is "0" when the three stages of the first group are each "1". Similarly, the output of $H^{004}$ is "0" when the three stages of the second group are each "1". (The clock input from $C^{000}$ is "0" during every odd clock phase.)

STANDARD CARD TYPES

The majority of printed circuit cards consist of one or two standard inverters on a single card. The cards differ in the number of inverters on the card, the number of input and output diodes associated with each inverter, and the electrical interconnections, if any. A maximum of six inputs may be applied to an inverter, and a maximum of eight outputs may be taken from it. Since an unused input terminal is sensed as a "1" input, no more than the exact number of input terminals required can be present. Inverter cards, therefore, are

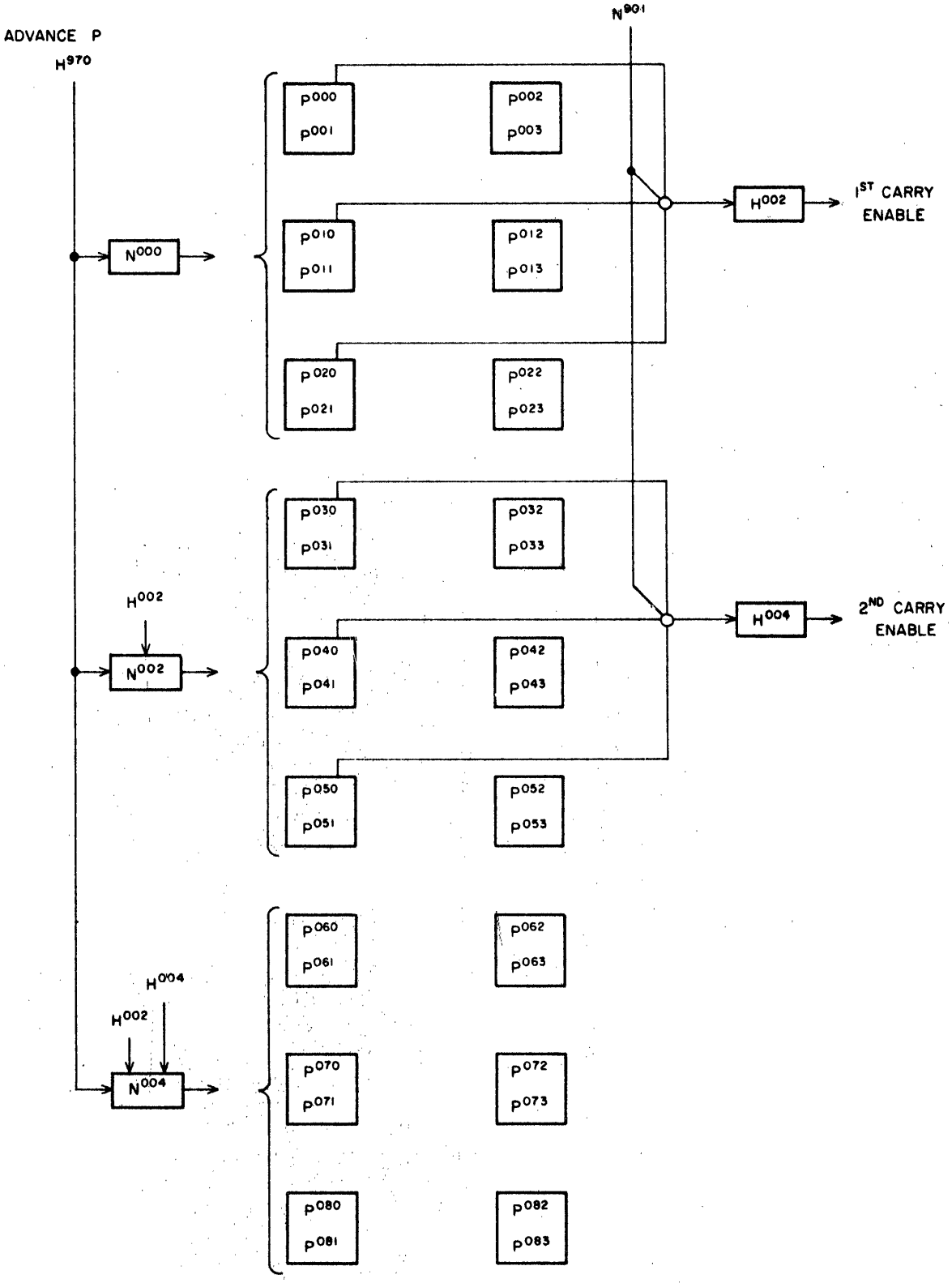
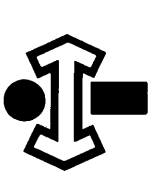



Figure 1-13. Interconnection of Three-Stage Counters to Form Nine-Stage Counters.



provided with varying numbers of input and output terminals to handle the various situations of logic.

The inverter cards are assigned two-digit numbers; the highest-order digit designates the type of card, and the lowest-order digit the number of inputs associated with each inverter on the card. (In the case of the Control Delay cards, only one inverter has external inputs.) The various types of inverter cards, and the pin assignments for each, are listed in table 1-3. The significance of letters is as follows:

I - input

O - output

A - as subscript one of the inverters on a card having two

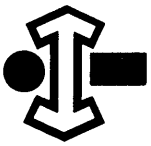
C - as subscripts other inverter

C - not as subscript, a clock pulse

## TABLE 1-3. DESCRIPTION OF STANDARD CARD TYPES

| Type Designation | No. of Inverters | No. of inputs (per inverter) | No. of outputs (per inverter) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 1 | 1 | 8 | I | | | | O | O | O | O | O | O | O | O | -20V | Gd | +20V |
| 12 | 1 | 2 | 8 | I | I | | | O | O | O | O | O | O | O | O | -20V | Gd | +20V |
| 13 | 1 | 3 | 8 | I | I | I | | O | O | O | O | O | O | O | O | -20V | Gd | +20V |
| 14 | 1 | 4 | 8 | I | I | I | I | O | O | O | O | O | O | O | O | -20V | Gd | +20V |
| 15 | 1 | 5 | 7 | I | I | I | I | I | O | O | O | O | O | O | O | -20V | Gd | +20V |
| 16 | 1 | 6 | 6 | I | I | I | I | I | I | O | O | O | O | O | O | -20V | Gd | +20V |
| 21 | 2 | 1 | 5 | $I_A$ | $O_A$ | $O_A$ | $O_A$ | $O_A$ | $O_A$ | $I_C$ | $O_C$ | $O_C$ | $O_C$ | $O_C$ | $O_C$ | -20V | Gd | +20V |
| 22 | 2 | 2 | 4 | $I_A$ | $I_A$ | $O_A$ | $O_A$ | $O_A$ | $O_A$ | $I_C$ | $I_C$ | $O_C$ | $O_C$ | $O_C$ | $O_C$ | -20V | Gd | +20V |
| 23 | 2 | 3 | 3 | $I_A$ | $I_A$ | $I_A$ | $O_A$ | $O_A$ | $O_A$ | $I_C$ | $I_C$ | $I_C$ | $O_C$ | $O_C$ | $O_C$ | -20V | Gd | +20V |
| 24 | 2 | 4 | 2 | $I_A$ | $I_A$ | $I_A$ | $I_A$ | $O_A$ | $O_A$ | $I_C$ | $I_C$ | $I_C$ | $I_C$ | $O_C$ | $O_C$ | -20V | Gd | +20V |
| 31* | 2 | 1 | 5 | $I_A$ | $O_A$ | $O_A$ | $O_A$ | $O_A$ | $O_A$ | $I_C$ | $O_C$ | $O_C$ | $O_C$ | $O_C$ | $O_C$ | -20V | Gd | +20V |
| 32* | 2 | 2 | 4 | $I_A$ | $I_A$ | $O_A$ | $O_A$ | $O_A$ | $O_A$ | $I_C$ | $I_C$ | $O_C$ | $O_C$ | $O_C$ | $O_C$ | -20V | Gd | +20V |
| 33* | 2 | 3 | 3 | $I_A$ | $I_A$ | $I_A$ | $O_A$ | $O_A$ | $O_A$ | $I_C$ | $I_C$ | $I_C$ | $O_C$ | $O_C$ | $O_C$ | -20V | Gd | +20V |
| 41** | 2 | 1 | 6 | I | | | | | C | O | O | O | O | O | O | -20V | Gd | +20V |
| 42** | 2 | 2 | 6 | I | I | | | | C | O | O | O | O | O | O | -20V | Gd | +20V |
| 43** | 2 | 3 | 6 | I | I | I | | | C | O | O | O | O | O | O | -20V | Gd | +20V |
| 44** | 2 | 4 | 6 | I | I | I | I | | C | O | O | O | O | O | O | -20V | Gd | +20V |
| 45** | 2 | 5 | 6 | I | I | I | I | I | C | O | O | O | O | O | O | -20V | Gd | +20V |

* The types 31, 32, and 33 are two-inverter units which have internal feedback or flip-flop connections.
** The types 41, 42, 43, 44, and 45 are two-inverter units used in Control Delay circuits; a clock pulse applied to pin 6 controls the internal feedback connection.



# BOOLEAN ALGEBRA

Boolean algebra is both similar to and different from ordinary algebra. Because of the differences, it is simpler at first to regard it as unrelated to ordinary algebra. The logical equations (page 22) which describe the connections of building blocks in the computer are a specialized form of Boolean algebra as applied to switching circuits. A brief discussion of this type of algebra will aid in use of the File of Equations.

In Boolean algebra, there are only two values or quantities to be considered, namely, "1" and "0". In a Boolean equation the variables, or literals, are restricted to these values, which can be considered as opposites; one is the negation of the other.

There are three operations used in such equations. The first is the logical product or the AND function of two terms, which is indicated by a dot between the terms, or by the absence of any symbol between them. This function is satisfied only when both terms are "1"; it is not satisfied for all other combinations of values of the terms.

The second operation is the logical sum or the OR function of two terms, which is indicated by a plus sign between the terms. An OR function of two terms is satisfied when one or the other, or both, of the terms are "1"; it is not satisfied only when both terms are "0". Thus, this is the inclusive OR rather than the exclusive.

The third operation is negation, or the NOT function of a term, and is indicated by a bar over the term. If a term is "1", then the negation of that term has the value "0", and vice versa.

The grouping of terms is indicated by parentheses. It is customary in Boolean equations for switching circuits to represent the circuit inputs by literals which appear on the right side of the equal sign. The literal on the left side of the equal sign represents the circuit output,

which is a function of the inputs on the right side of the equation. Table 1-4 is a glossary of the various symbols used in the algebra of switching circuits.

There are two types of Boolean equations: (1) identities; and (2) transfer formulas. An identity consists of two equivalent expressions separated by an equality sign. For example, the equation (A + B)C = AC + BC is an identity, which states that either A or B in combination with C, is equivalent to either A in combination with C, or B in combination with C; while on the other hand, the equation C = A + B is a transfer formula, which states that at some particular instant, a "1" is transferred to element C if a "1" is in either element A or element B.

A Boolean expression may be reduced to its simplest equivalent by applying the basic identities of table 1-5. This procedure not only provides a means of understanding the circuitry, but also of simplifying switching circuits, thus reducing the number of components necessary to perform the operations specified by a particular transfer equation. For example, the transfer equation D = AB + B + C can be reduced to its equivalent D = B + C by applying the theorems of table 1-5 to strike out the redundant expression AB. Because the right side of the reduced transfer equation is simpler than the right side of the unreduced equation, a circuit built by using the simplified equation uses fewer components and yet performs the same logical functions.

The theorems are also applied to convert the final simplified expressions into "standard" forms, that is, forms which are more readily adaptable to a given type of switching circuit. For example, the equation D = (A + C) ($\overline{B}$) can be changed to the formula D = A$\overline{B}$ + C$\overline{B}$, which is applicable to the building block used in the computer.

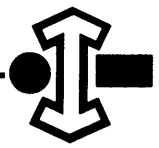


TABLE 1-4. GLOSSARY OF BOOLEAN SYMBOLS

| | |
|---|---|
| A<br>B<br>C<br>D<br>etc. | Letters used to denote switching elements. When a letter appears in a Boolean expression, it is a literal that denotes an active "1" input when the corresponding switching element is in the "1" state, or a passive "0" input when the corresponding switching element is in the "0" state. |
| $\bar{A}$<br>$\bar{B}$<br>$\bar{C}$<br>etc. | The bar notation denotes negation. When a letter with a bar appears in a Boolean expression, it denotes an input that is active when the corresponding switching element is in the "0" state, and passive when the element is in the "1" state. |
| 1 | Denotes an active ("1") input in a Boolean expression. |
| 0 | Denotes a passive ("0") input in a Boolean expression. |
| + | Denotes the logical sum, or the OR function of two or more literals. |
| ( ) ( )<br>etc. | Denotes the logical product, or the AND function of two or more literals. |
| = | Denotes equality. Used to separate equivalent Boolean expressions, or to separate the two halves of a transfer formula. |

TABLE 1-5. BOOLEAN IDENTITIES

A. COMMUTATIVE LAWS

1. $A + B = B + A$
2. $B + A = A + B$

B. ASSOCIATIVE LAWS

3. $(A + B) + C = A + (B + C)$
4. $(AB)\,C = A\,(BC)$

C. DISTRIBUTIVE LAWS

5. $AB + AC = A(B + C)$
6. $A + BC = (A + B)(A + C)$

D. FORMS INVOLVING 1 AND 0

7. $0 + 0 = 0$
8. $0 + 1 = 1$
9. $1 + 1 = 1$
10. $0 \cdot 0 = 0$
11. $0 \cdot 1 = 0$
12. $1 \cdot 1 = 1$
13. $A + 0 = A$
14. $A + 1 = 1$
15. $A \cdot 1 = A$
16. $A \cdot 0 = 0$

E. FORMS WITH REPEATED LITERALS

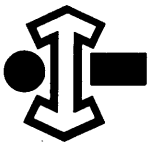
17. $A + A = A$
18. $A \cdot A = A$

F. FORMS WITH NEGATION

19. $A = A$
20. $A\,\overline{A} = 0$
21. $A + \overline{A} = 1$
22. $\overline{A \cdot B \cdot C} = \overline{A} + \overline{B} + \overline{C}$ De Morgan's Theorem
23. $\overline{A + B + C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$ De Morgan's Theorem

G. FORMS DERIVED FROM PREVIOUS IDENTITIES

24. $A + AB = A$
25. $A(A + B) = A$
26. $(A + \overline{B})\,B = AB$
27. $A\,\overline{B} + B = A + B$
28. $BC = ABC + \overline{A}BC$



The proofs of these identities are all of a similar nature. A simple proof of Identity 6 is presented below in table 1-6.

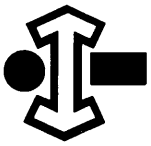
TABLE 1-6. PROOF OF IDENTITY 6

| Conditions A B C | Left-Hand Expression A + BC | = | Right-Hand Expression (A + B) ( A + C) |
|---|---|---|---|
| 0 0 0 | 0 | | 0 |
| 0 0 1 | 0 | | 0 |
| 0 1 0 | 0 | | 0 |
| 0 1 1 | 1 | | 1 |
| 1 0 0 | 1 | | 1 |
| 1 0 1 | 1 | | 1 |
| 1 1 0 | 1 | | 1 |
| 1 1 1 | 1 | | 1 |

Identity 6 contains three literals, each capable of being in either the "0" or the "1" state. The various combinations of "1's" and "0's" for these literals are listed in the Conditions column of table 1-6. In the Left-Hand Expression column, the "logical value" of the left-hand side of the expression for each ABC condition is listed. Similar values are also listed in the Right-Hand Expression column next to each ABC condition to represent values obtained from the right-hand expression of the identity. It is apparent that the left-hand and right-hand values are equivalent for each ABC condition; therefore, the two expressions are identical.

Identities 22 and 23 are worthy of special attention. By application of them, an equation using the AND function can be converted into one using the OR function, and vice versa. For example, the equation Z = ABC is to be converted into an equivalent one in which the OR

function is used in the right side. This is done by first negating both sides of $Z = ABC$ and thereby obtaining $\overline{Z} = \overline{ABC}$. Now by applying Identity 22 to the right side, the desired result $\overline{Z} = \overline{A} + \overline{B} + \overline{C}$ is obtained. This procedure, which is often called inverting the logic, has been employed at various times in the equations for the computer.



# CHAPTER 2

# CONTROL SECTION

The control section of the computer directs the operations required to execute instructions and to exchange data with external equipment. In addition, it establishes the timing relations required to perfom the operations in the proper sequence.

This section consists of main or overall control and the specialized control units: storage control, arithmetic control and input-output control. Main control, which performs many operations itself, also initiates action in these specialized units; these units produce the commands to carry out specific operations. This chapter deals primarily with main control and its relationship to these units.

The major elements of main control are the sequences, several networks of FFs and single inverters which sense and store static conditions, and several registers (U, P, R and $B^1$ through $B^6$).

The execution of instructions and the exchange of data with external equipment are accomplished by many simple unit operations called "commands". A command, which is issued by a sequence, causes one simple action to occur, such as the transmission of the content of a register to another or the setting of a control FF. In the execution of instructions the issuing of commands is controlled directly or indirectly by the instruction word in the program control register. Commands may also be conditioned by the presence or absence of specified conditions in some part of the computer.

The control section senses conditions, determines operations that are required, and issues commands in an order suitable for the performance of such operations.

An instruction is a 24-bit quantity consisting of three parts which are arranged as shown below in Figure 2-1:

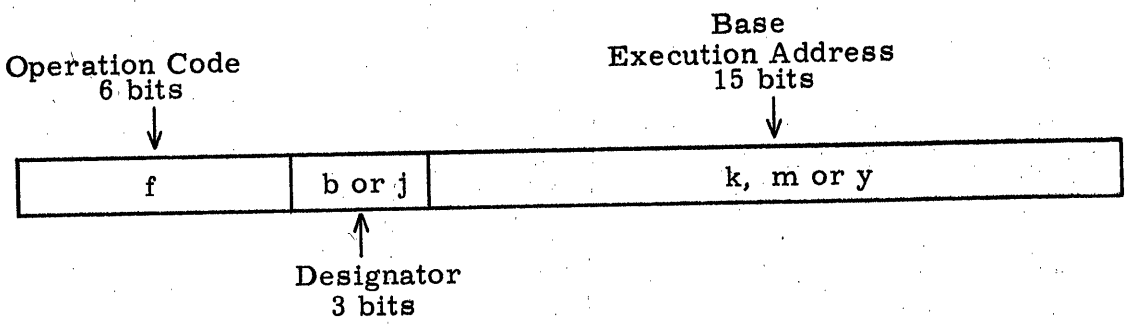



Figure 2-1. Instruction Format.

Each of the 62 instructions has a unique 6-bit operation code, f, which designates the instruction. The translation of f establishes the condition required within the control section for the execution of the instruction. The 3-bit designator usually specifies the index register, B, whose content is to be added to the base execution address; when so used the designator is denoted by the letter b. For some instructions it serves different purposes (described later). The execution address of an instruction is normally a base address quantity that is modified by the addition of $B^b$ to yield the actual address of the instruction operand. Some instructions use the execution address in a different manner.

A 48-bit instruction word is read from storage and entered in $U^1$. Execution of the upper instruction occurs first. Following this, the lower instruction is transmitted to the upper half of $U^1$ so that it can be executed. Thus the instruction currently being executed is always located in the upper half of $U^1$.

The primary function of $U^2$ is to modify the execution address, m, of the instruction in the upper half of $U^1$ by adding $B^b$ to it. In preparation for modification the designator of the upper instruction is translated. The translation specifies the B register the content of which is to be added in the modification.

## PROGRAM CONTROL REGISTER

The program control register holds the 48-bit instruction word during the execution of the two 24-bit instructions contained in the word. All operations that are necessary to successfully execute an instruction are governed by the content of this register. Because the letter U is used as the base letter in the logical symbols for the elements making up the program control register, it is often called the U register.

The U register consists of two ranks of FFs. Rank $U^1$, which is 48 bits in length, stores an instruction word during the execution of the two instructions contained in it. Rank $U^2$, 15 bits in length, has a borrow pyramid and therefore is a small subtractive accumulator. Transmission paths connect $U^2$ to the m portion of the upper half of $U^1$ (figure 2-2).

After transmitting m from $U^1$ to $U^2$ and adding $B^b$ to R, R is added to $U^2$. The modified execution address in $U^2$ for most instructions specifies the location of the operand in storage. In such cases $U^2$ is transmitted to the appropriate S register.

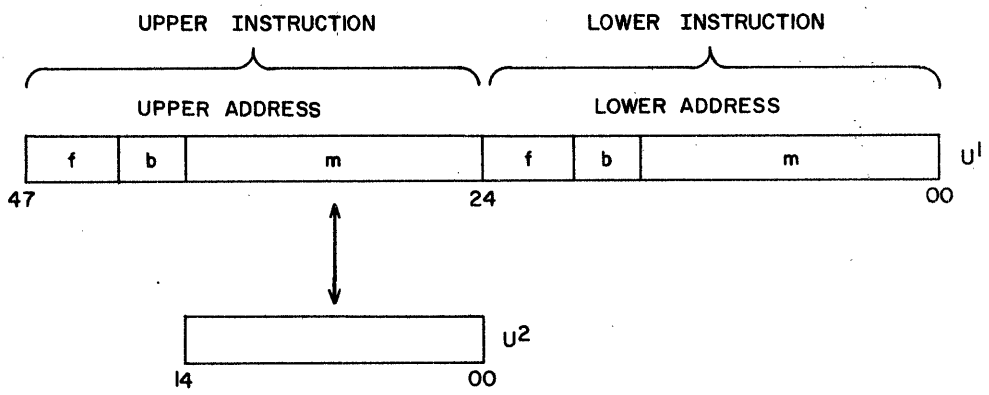



Figure 2-2. Relation of $U^2$ to $U^1$.

OPERATION CODE

This six-bit code specifies an instruction and controls the operation of the computer during the execution of the instruction. Of the 64 possible values of this code, 62 specify instructions; codes 00 and 77 (codes are expressed in octal) represent fault conditions that halt computation.

Prior to the actual execution of the instruction designated by the value of f, the operation code is translated by a network of single inverters which samples the upper six bits of $U^1$. The results of the translation go to the various sections of the machine to condition the occurrence of the commands which will actually carry out the required operations.

The translator uses several levels of logic in forming the outputs which actually gate commands. Figure 2-3 shows the fundamental structure of the translator. The large number of outputs from the upper six FFs of $U^1$ are obtained by single inverter slaves. The "1" side of each FF has a $U^{--4}$ and a $U^{--6}$ inverter slave; the "0" side of each FF has a $U^{--5}$ and a $U^{--7}$ inverter slave.

The first level translation is divided into two parts, one concerned with the lower octal digit of f (that is, $U_{42}$, $U_{43}$, and $U_{44}$) and the other with the upper octal digit ($U_{45}$, $U_{46}$, and $U_{47}$). All $F^{0--}$ and $F^{2--}$ inverters translate the lower octal digit. All $F^{1--}$ inverters translate the upper octal digit.

Unique (single-valued) translations of the lower octal digit are provided by $F^{000}$ through $F^{007}$. When the output of one of these inverters is "1", then the lower octal digit has the value given in the third superscript digit of the inverter designation; that is, a "1" from $F^{007}$ indicates that the lower digit is 7.

The $F^{2--}$ inverters provide partial, or incomplete, translations of the lower octal digit. These translations are duplicated by several slave inverters. The $F^{1--}$ inverters, which uniquely translate the upper octal digit, use the last digit of the inverter designation to indicate the value translated. Thus $F^{105}$ indicates that the upper octal digit is 5.

In the second level of the translator, the outputs of first level inverters that translate individually the upper and the lower octal digits are combined to specify either a unique value

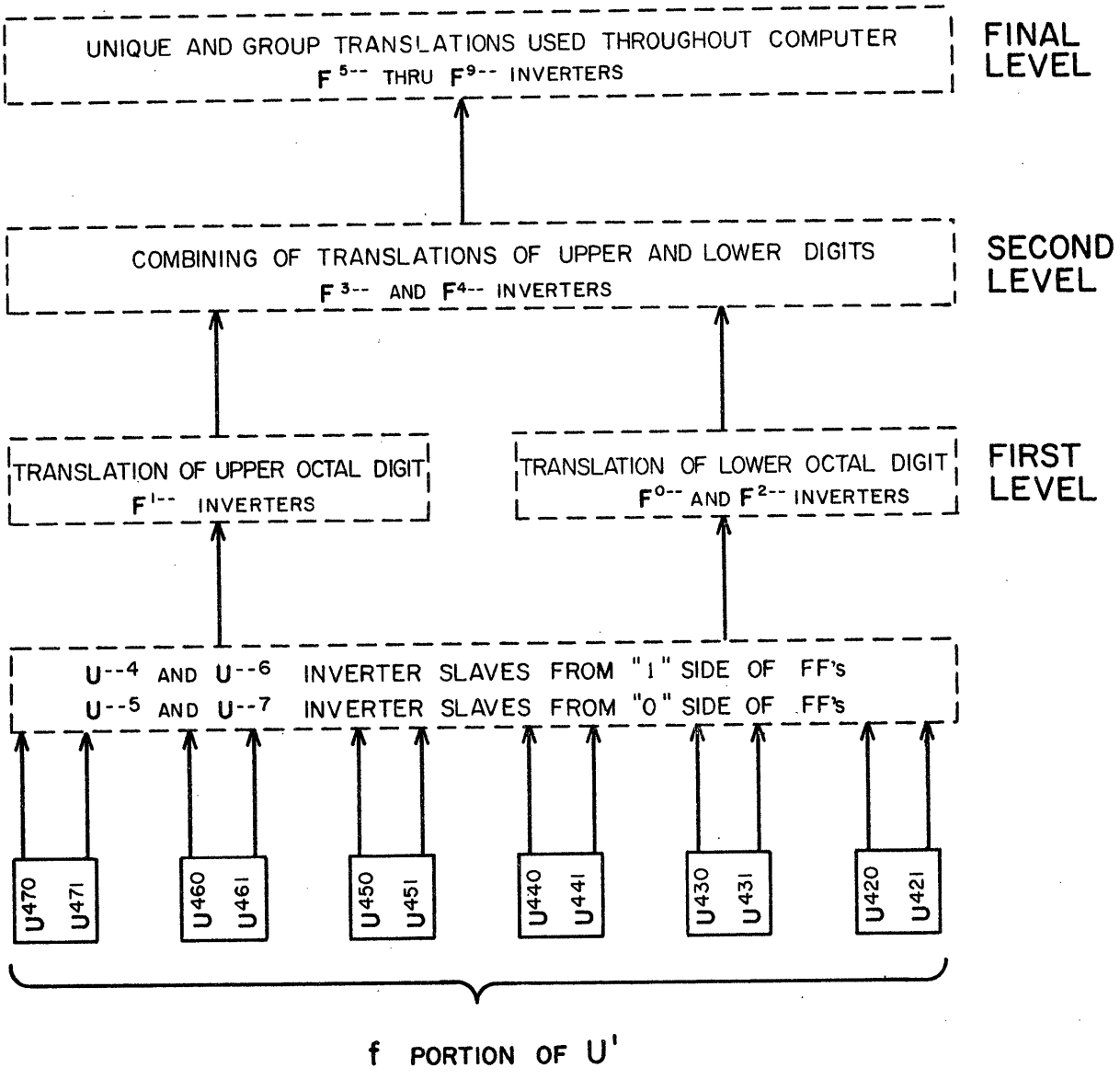



Figure 2-3. Structure of Operation Code Translator.

of the operation code or a group of values. The $F^{3--}$ and $F^{4--}$ inverters accomplish most of this combining of translations of the upper and lower digits. Outputs of the $F^{3--}$ and $F^{4--}$ inverters go to $F^{5--}$ or $F^{6--}$ inverters which actually supply the translations throughout the computer. A typical translation of f is shown in figure 2-4.

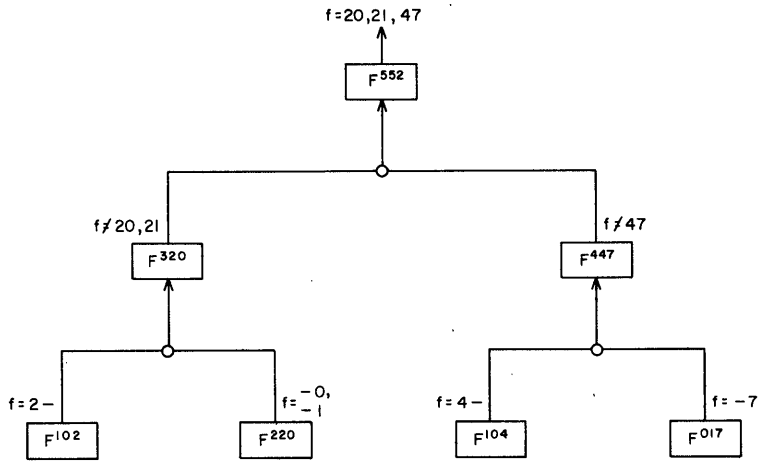
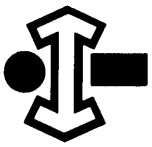



Figure 2-4. Typical Translation of Operation Code.

DESIGNATOR

The primary function of the 3-bit designator in an instruction is to specify the index, or B, register whose content is used in executing the instruction. When used as an index designator it is denoted by the letter b. For most of these cases namely, the 01-21, 24-33, 36, 37, 40-47, and 70-73 instructions, $B^b$ is added to the execution address. With b=0 no modification of the execution address occurs; with b=7 indirect addressing is used.

Instructions 34, 35, 50-57, 62-67 make use of $B^b$ in performing the basic operation of the instruction; no address modification occurs. The case of b=0 in instructions 34, 35 and 50-57 produces no significant result from their execution so, instruction 50 with b=0 is reserved as a "do-nothing" instruction to be used in filling out programs with an odd number of instructions. Instructions 62-67 interpret b=0 as specifying that exactly one word is to be transferred or searched. Indirect addressing is specified in instructions 34, 35, 50-57 and 62-67 when b=7.



Instructions 22, 23 and 74-76 use the designator to specify a condition for carrying out their execution. When used as a condition designator the 3-bit quantity is denoted by the letter j. The interpretation of j for instructions 22 A Jump and 23 Q Jump are as follows:

0 - Jump if register content is zero
1 - Jump if register content is not zero
2 - Jump if register content is positive
3 - Jump if register content is negative
4 - Return jump if register content is zero
5 - Return jump if register content is not zero
6 - Return jump if register content is positive
7 - Return jump if register content is negative

For instruction 75 Selective Jump, j is interpreted as follows:

0 - Jump unconditionally
1 - Jump if lever key one is set
2 - Jump if lever key two is set
3 - Jump if lever key three is set
4 - Return jump unconditionally
5 - Return jump if lever key one is set
6 - Return jump if lever key two is set
7 - Return jump if lever key three is set

For instruction 76 Selective Stop, only the stop may be conditioned; the jump occurs unconditionally. The interpretation is:

0 - Stop unconditionally (normal jump)
1 - Stop if lever key one is set (normal jump)
2 - Stop if lever key two is set (normal jump)
3 - Stop if lever key three is set (normal jump)
4 - Stop unconditionally (return jump)
5 - Stop if lever key one is set (return jump)
6 - Stop if lever key two is set (return jump)
7 - Stop if lever key three is set (return jump)

For instruction 74 External Function, j is interpreted as follows:

0 - Select external equipment
1 - Activate communication channel one
2 - Activate communication channel two
3 - Activate communication channel three
4 - Activate communication channel four
5 - Activate communication channel five
6 - Activate communication channel six
7 - Sense external condition

The various input-output operations conditioned by the value of j in this instruction are discussed in chapter 5.

Translation of the Designator

Single inverters $F^{700}$ through $F^{707}$ translate the eight possible values of the designator. A "0" output from one of these inverters indicates the value given by the third superscript digit in the symbol. Thus, a "0" output from $F^{705}$ indicates that the value is five. Since the outputs of these inverters are combined with that of inverters translating the operation code, the designator translation is in normal form, that is, a "1", when it is used for gating.

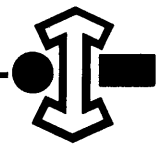
## BASE EXECUTION ADDRESS

The base execution address is the lower 15 bits of an instruction. It has three functions, depending upon the instruction used, each denoted by a different letter. (See table 2-1.)

TABLE 2-1. DESIGNATION OF BASE EXECUTION ADDRESS

| Instructions | Use | Denoted by | After modification by addition of $B^b$, denoted by |
|---|---|---|---|
| 12-33, 36-47, 52, 53, 55-73, 75, 76 | Specifies the storage location of operand | m | M |
| 14, 10, 11, 50 51, 54, 74 | As operand | y | Y |
| 01-03, 05-07, 34, 35 (shift instructions) | As shift count | k | K |

The transmission of the base execution address to $U^2$ from the upper instruction in $U^1$ occurs in the execution of all instructions. If it is to be modified, $B^b$ is transmitted to $R^1$ (the address buffer register) and then $R^1$ is added to $U^2$. When the operand is procured from or sent to storage $U^2$ is transmitted to $S^1$ or $S^2$. When $U^2$ itself is to be used as the operand it is transmitted to $R^1$ or $X^1$.



## THE $U^2$ ACCUMULATOR

In addition to the usual features of a FF register, $U^2$ is a 15-bit subtractive accumulator that provides for the addition of $R^1$ to its content. Because this accumulator is similar in structure to the 48-bit accumulator, the A register, it is not discussed in detail. (See appendix on A register.)

As in other registers in which arithmetic operations are performed, two ranks of FFs are necessary for the addition operation. Stages $U_{24}$ through $U_{38}$ of $U^1$ (that is, $U^1$ upper address, $U^1_{UA}$) form the rank that is sampled by the pyramid (see figure 2-5). The $U^2$ flip-flops constitute the other rank, which receives the sum.

Prior to the addition of $R^1$ to $U^2$, the appropriate $U^1_{UA} \rightarrow U^2$ or $U^2 \rightarrow U^1_{UA}$ command occurs to insure that both ranks hold the same quantity. Following this, the borrow pyramid samples $U^1_{UA}$ and $R^1$ to determine the stages of $U^2$ that must be toggled in order for $U^1_{UA} + R^1$ to be formed in $U^2$. Toggling the indicated stages forces the desired sum in $U^2$. This occurs when the command Add $R^1$ to $U^2$ is given.

The $U^2$ accumulator has provisions for disabling the borrow pyramid so that each bit of $U^2$ is toggled if the corresponding bit of $R^1$ is "1". The Partial Add in $U^2$ FF ($K^{530}$ $K^{531}$) disables the pyramid when it is set to "1". Thus, the Add $R^1$ to $U^2$ command accomplishes a selective toggling function (selection on the basis of bits of $R^1$ that are "1") when $K^{530}$ $K^{531}$ is set. However, when this FF is cleared, a full addition results from the Add $R^1$ to $U^2$ command. The partial addition of $R^1$ to $U^2$ is used often in transmitting B to some register such as X. Here, B goes to R and $R^1$ is partially added to $U^2$, with $U^2$ sent to X.

If either $B^b$ or $U^2$ has a "1" in the highest bit, then in their addition this number is treated as though it were a negative number expressed in one's complement form. Thus if the quantities below are added:

$m = 00005$

$B^b = 77776$

their sum is 00004. The same result is obtained when the values of $B^b$ and $U^2$ are interchanged.

In addition to the use of the $U^2$ accumulator in modifying the execution address (during the RNI sequence), it plays an important part in several other operations. Arithmetic operations on exponents in floating-point instructions are performed by the use of $U^2$. Buffer operations and transfer instructions employ $U^2$ in handling address words.

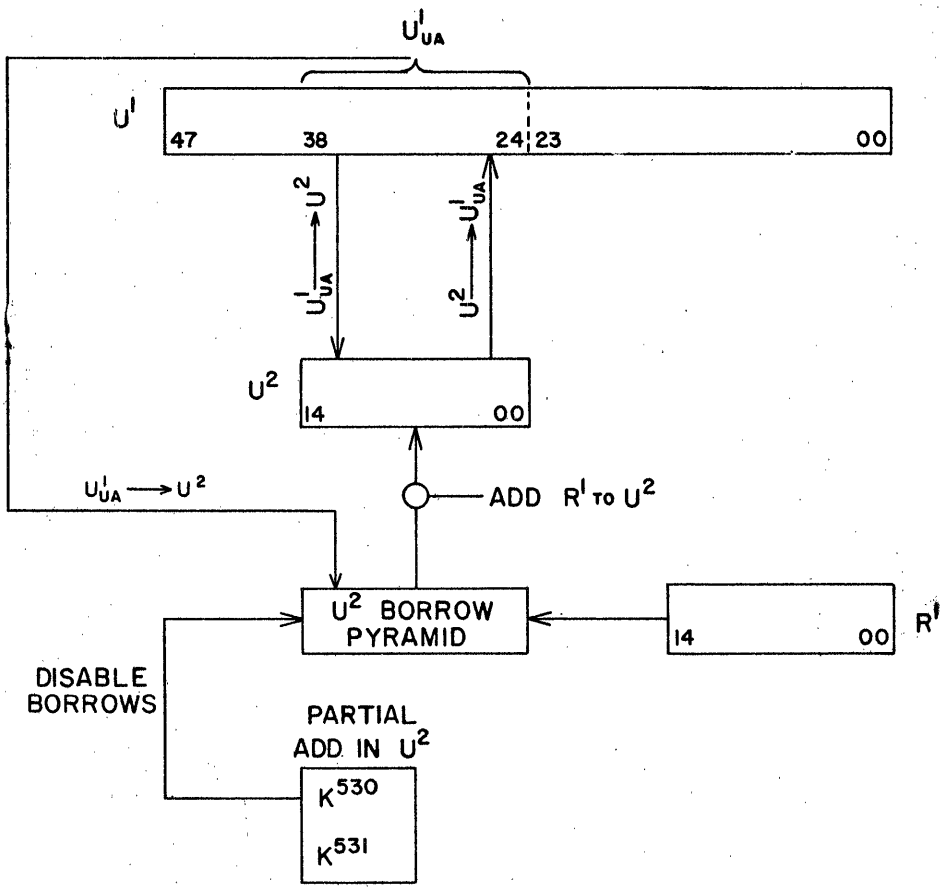
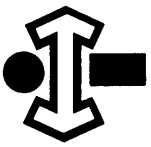



Figure 2-5. Adding in $U^2$ Accumulator.



## INDEX REGISTERS

Each of the six index registers, $B^1$ through $B^6$, provides storage for quantities which are used in a variety of ways, depending upon the instruction. In the majority of instructions the B registers hold quantities to be added to the base execution address. For search instructions (64-67), $B^b$ indicates the number of items to be searched. The B registers have no provision for arithmetic operations. When such an operation is required on an index quantity, $B^b$ is entered in R or $U^2$ and the operation performed there. Subsequently the result is returned to $B^b$.

Transmissions into a B register come from $R^2$ via the $I^4$ single-inverter rank (see figure 2-7). Each B register provides outputs to either $I^2$ or $I^3$, which are in turn gated to $R^1$. For this transmission the "0" output of the register FFs is used because of the single inversion of $I^2$ or $I^3$.

The assignment of equation symbols for the B registers follows a somewhat different pattern than that used in other registers. The first digit of the superscript identifies the particular register. The second and third digits indicate the stage. The "0" side is not indicated by the odd character of the third digit; instead, if the second and third digits are 50 or greater the symbol denotes the "0" side. Furthermore, the stage with which such a symbol is associated, for example, $B^{563}$, can be found by subtracting 50 from the last two digits. In the example, $B^{563}$ is the "0" side of stage 13.

## ADDRESS BUFFER REGISTER

The 15-bit address buffer register R has provisions for counting and complementing as well as storage. As a counter it operates subtractively. R is suitably connected to the $U^2$ accumulator for the addition of its content to $U^2$.

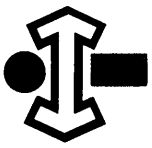
### FUNCTIONS OF THE R REGISTER

The R register is involved in the following functions:

1) All transmissions to and from the B registers are via R.
2) Modification of the base execution address of the current instruction is by addition of the quantity in R (obtained from $B^b$) to $U^2$.
3) In shift instructions, R acts as the shift counter.
4) In the integer and fractional multiply and divide instructions, R keeps a record of the number of partial multiplications and divisions which remain to be performed.
5) In floating-point instructions, R performs arithmetic operations on one of the two exponents.
6) During buffer operations, R increments the current buffer address and compares it with the terminal address.

### CONNECTIONS TO OTHER REGISTERS

The R register consists of two ranks of FFs, $R^1$ and $R^2$. The connection between R and other registers and inverter ranks ($I^2$, $I^3$, and $I^4$) are shown in figure 2-7. Rank $I^2$ receives inputs from control for setting R to various predetermined values. After receiving these control inputs, $I^2$ is transmitted to $R^1$.



COUNTING AND COMPLEMENTING IN R

The R register is a two's complement, open-ended subtractive counter with a modulus of $2^{15}$ (figure 2-6). Thus, in counting, the bits of R are toggled in such a manner as to form the quantity $(R^1)-1$ in $R^2$. This is accomplished by the command Reduce $R^1$ to $R^2$. The effect of this command is always to subtract "1" from the first stage, $R_{00}$. When $R_{00}$ is "0" at the time the command occurs, a borrow from $R_{01}$ is required. Similarly, a borrow is required from $R_{02}$ if both $R_{00}$ and $R_{01}$ are "0". Thus, in general, a borrow is required from stage n if all stages of lower order than n are "0". Borrows are accomplished by toggling a stage of $R^2$ with the corresponding stage of $R^1$ when the command occurs.

For the sensing of borrows, $R^1$ and $R^2$ are organized as five three-bit groups, each of which is itself a small counter. Just as "1" is borrowed from $R_{01}$ only if $R_{00}$ is "0", so "1" is borrowed from the second group only if the stages of the first group are each "0". Thus, in general, a borrow is made from a group only when all stages of lower groups are "0".

Each group has a $H^{87-}$ which senses when the three stages of the group are "0". The existence of such a condition in a group is indicated by a "0" output from the $H^{87-}$. Outputs of the $H^{87-}$ of lower-order groups are used by higher-order groups to determine whether a borrow is required from the latter. A borrow is required from the fifth group only when the outputs of $H^{870}$, $H^{872}$, $H^{874}$, and $H^{876}$ are "0". The Reduce $R^1$ to $R^2$ command is gated into the fifth group by the AND combination of outputs from these $H^{87-}$ elements.

Within a given group from which a borrow is required, the first stage is toggled by the occurrence of the Reduce $R^1$ to $R^2$ command. The second stage is toggled if the first stage holds a "0", that is, if a borrow is required from the second stage. The third stage is toggled only if both the first and second stages contain "0", since it is this condition that indicates that a borrow is required from the third stage.

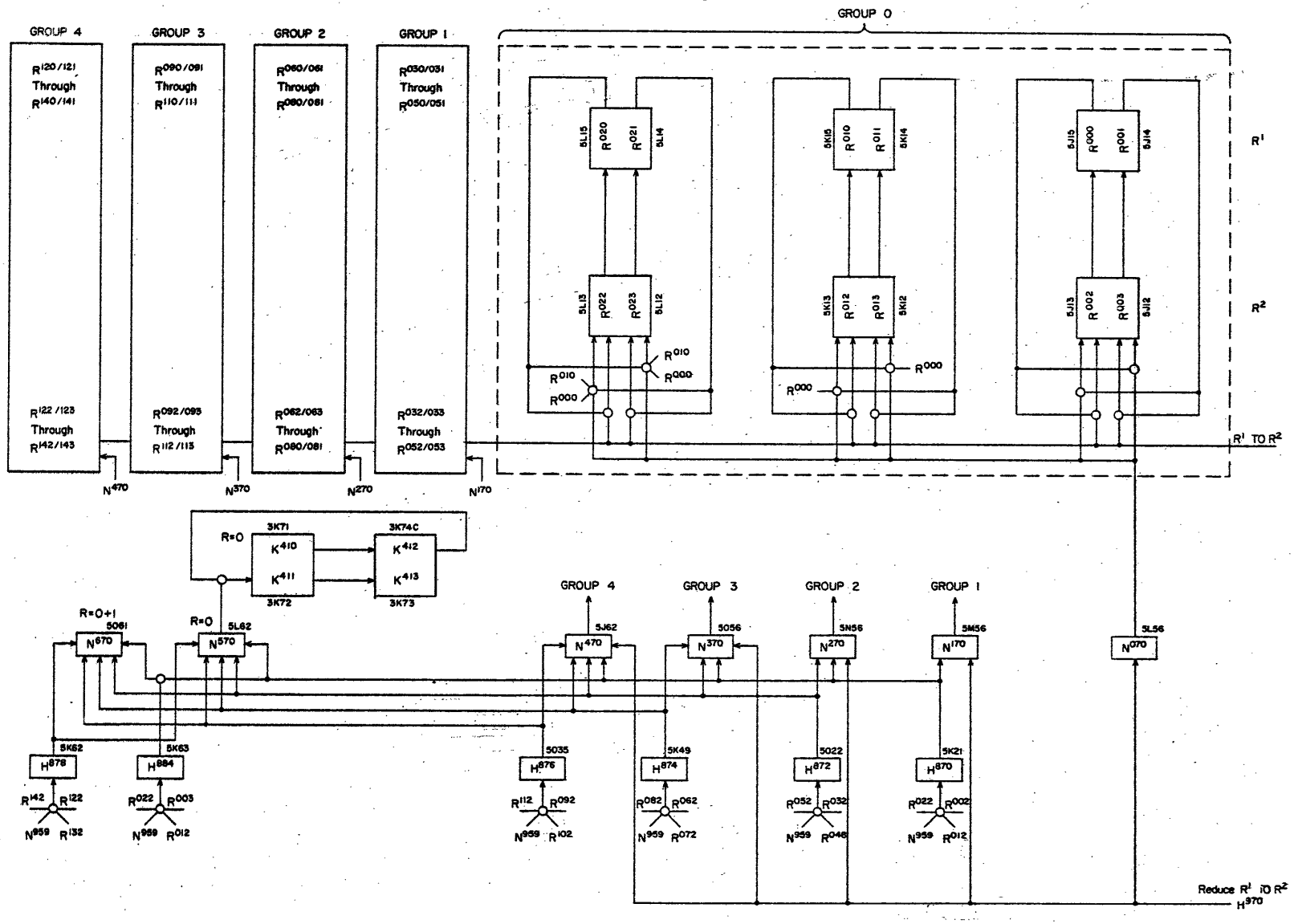
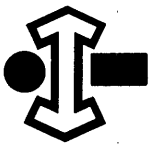



Figure 2-6. R Register Counting Structure.



Complementing is performed in R by transmitting the complement of the bits in $R^2$ (the "0" outputs of the FFs) to $R^1$. For many of the uses of R it is necessary to sense when the quantity in R is zero; the $H^{87-}$ terms are used in sensing this condition. The outputs of all five $H^{87-}$ terms are combined by both $N^{570}$ and $N^{670}$. When all five inputs to each of these are "0", all 15 stages of R contain "0". This condition is indicated by a "1" from $N^{570}$ or $N^{670}$. However, there is an important difference between the indications of the R = 0 condition given by $N^{570}$ and $N^{670}$.

To account for this difference it is necessary to consider the interval between the time when the Reduce $R^1$ to $R^2$ command is given and the time when the state of R resulting from this reduction is reflected at the output of $N^{570}$. The interval is two clock periods; one is required for the toggling of $R^2$ by the reduce command and the second is required for going through the R = 0 control delay. In certain cases (the shift instructions are an example) it is not permissible to have the sensing of the R = 0 condition lag the Reduce command by such an interval. Furthermore, in these cases it is known that R will be reduced to zero; since this is the case it is possible to anticipate the time when R will be reduced to zero. This is done by sensing when R = 1. The output of $N^{670}$ is a "1" not only when R = 0 but also when R = 1.

In addition to this aspect of the R = 0 indications there is in some cases a rather elaborate procedure for sampling the output of $N^{570}$. By the use of FFs $K^{410}$ $K^{411}$ and $K^{412}$ $K^{413}$ it is possible to obtain a pulse as soon as R is reduced to zero and, in addition, to obtain just one such pulse despite the fact that when R is reduced to zero the output of $N^{570}$ may be a "1" for some time. This is accomplished by setting $K^{410}$ $K^{411}$ to "1", which in turn sets $K^{412}$ $K^{413}$ to "1". As soon as the R = 0 condition exists the latter FF causes the former to be cleared. One clock time later this will, in turn, result in $K^{412}$ $K^{413}$ being cleared. In order to uniquely specify the first time the output of $N^{570}$ is "1" it is only necessary to combine a "1" output from $K^{412}$ $K^{413}$ in an AND with $N^{570}$.

In certain cases R is employed as an additive counter, despite the fact that it is actually subtractive. This is accomplished by the following sequence of commands:

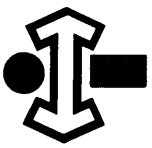
Complement $R^2$ to $R^1$

$R^1 \rightarrow R^2$

Reduce $R^1$ to $R^2$

Complement $R^1$ to $R^2$

Following this sequence of commands $R^2$ holds the initial quantity plus one.



## PROGRAM ADDRESS REGISTER

The program address register, or P register, holds the address from which each instruction word is obtained. After the execution of both the upper and lower instructions of this word (or when the decision is made to skip the lower instruction) the quantity in P is advanced by one to generate the address from which the next sequential instruction word is obtained. Thus the P register is a counter.

The initial address of a sequence of instruction words may be entered into P manually at the console, or it may be entered during the execution of a program by jump instructions. Such instructions always transmit a new program address quantity to P. If the jump instruction is a return jump, that is, when j = 4-7, then the previous content of P is stored, thereby permitting the return to the sequence of instructions from which the jump was made.

The P register consists of two ranks of FFs, $P^1$ and $P^2$. Rank $P^2$ follows unconditionally rank $P^1$; the signal transmitting $P^1$ to $P^2$ occurs every odd clock phase.

### COUNTING IN P

The P register is a two's complement additive counter with a modulus of $2^{15}$; thus, in counting, the bits of P are toggled in such a manner as to form the quantity $P^2 + 1$ in $P^1$. This is accomplished by the command Advance $P^2$ to $P^1$, which is given each time the RNI sequence is entered from a full exit. The effect of this command is always to add "1" to the first stage, $P_{00}$. When $P_{00}$ is "1" at the time the command occurs, a carry to $P_{01}$ is required. Similarly, a carry to $P_{02}$ is required if both $P_{00}$ and $P_{01}$ are "1". In general, a carry to any stage n is required if all stages of lower order than n are "1". Carries are accomplished by toggling a stage of $P^1$ with the corresponding stage of $P^2$ when the command occurs.

For the sensing of carries, $P^1$ and $P^2$ are organized as five three-bit groups, each of which is itself a small counter. Just as "1" is carried to $P_{01}$ only if $P_{00}$ is "1", so "1" is carried to the second group only if the stages of the first group are each "1". Thus, in general, a carry is sent to a group only when all stages of lower groups are "1". Except for the fact that P is an additive counter its counting structure is similar to that of the R register (figure 2-6).

The entry into the interrupt program and the exit from it to the main program are both accomplished by the two instructions held in address 00007. During entry into the interrupt program P is set to 00007. A P = 00007 translator is used in terminating the interrupt program.

The P register and the value of the breakpoint switch are compared and sampled during every execution of the RNI sequence. The pyramid for sensing the value of P is considered in conjunction with the breakpoint.

## PARALLEL TRANSMISSION INVERTER RANKS

Many of the parallel transmission paths between registers involve a rank of inverters. Figure 2-7 shows the major parallel transmission paths and the seven inverter ranks (called $I^0$ through $I^6$). In general, the purpose of inverter ranks is to augment the input or the output capacity of a register. For example, rank $I^1$ is a slave for $X^2$ and thereby increases the number of outputs that may be taken from rank $X^2$ of the X register. Each inverter of $I^1$ requires only one output from $X^2$; this inverter, in turn, provides five outputs to the output registers.

Rank $I^0$ serves a similar purpose for inputs to $X^1$. The one input to $X^1$ from $I^0$ handles the four inputs from data channels ($M^1$ through $M^4$). Ranks $I^2$, $I^3$, and $I^4$, which are

associated with the R register, are considered in the section treating this register.

Ranks $I^5$ and $I^6$ are in the transmission paths between the storage section and the rest of the computer. $I^5$ works in conjunction with even storage; $I^6$ works with odd storage. A word read from storage goes to $I^5$ or $I^6$ and thence to $U^1$ or $X^1$ depending on the purpose for which the storage reference was initiated. Information to be written in storage goes from $X^1$ to $I^5$ or $I^6$ and thence to $Z^1$ or $Z^2$ (which are sampled during the write part of the storage cycle).

The symbols assigned to represent the inverters of a rank use the letter I. The first digit of the superscript corresponds to that shown in Figure 2-7. For example, inverters of rank $I^5$ have symbols such as $I^{500}$, which denotes the lowest inverter of the rank.

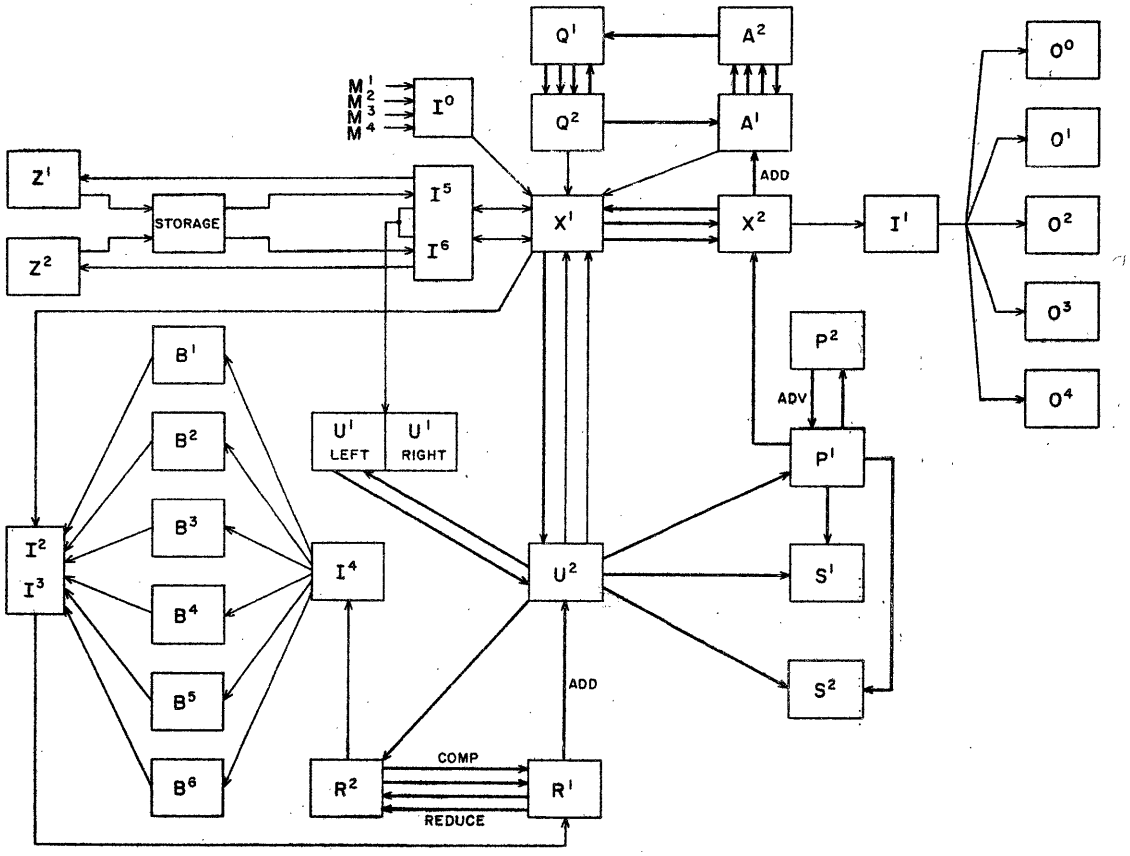



Figure 2-7. Parallel Transmission Inverter Ranks.

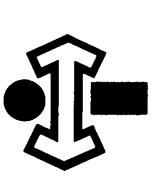
## CONTROL SEQUENCES

The execution of an instruction or the exchange of data with external equipment is divided into lesser operations called "sequences". The several sequences control the initiation and timing of commands which actually accomplish the execution of instructions, buffers, etc.

The nine sequences are listed in table 2-2 along with the associated $H^{---}$ equation symbol for the logical elements that constitute the sequence. This table also shows the instructions or operations for which the sequence is used. The group of sequences which includes all of them except Read Next Instruction and Auxiliary is referred to as the "instruction sequences".

Table 2-2. CONTROL SEQUENCES

| Sequence | Equation Symbol | Abbreviation | Instruction or Operation When Used |
|---|---|---|---|
| Read Next Instruction | $H^{09-}$ | RNI | All instructions |
| Normal Jump | $H^{1--}$ | NJ | (22, 23, 75, 76) (b=0-3) |
| Zero Address | $H^{2--}$ | ZA | 01-11, 34, 35, 50, 51, 54,55 |
| Read Operand | $H^{3--}$ | RO | 12-17, 36-46, 52, 53, 70-73 |
| Write Operand | $H^{4--}$ | WO | 20, 21, 47, 56-61<br>(22, 23, 75, 76) (b=4-7) |
| Search and Transfer | $H^{5--}$ | ST | 62-67 |
| Iterative | $H^{6--}$ | I | 24-33 |
| External Function | $H^{70-}$ | EF | 74 |
| Auxiliary | $H^{71-}$<br>$H^{76-}$ | AUX | Buffer, Advance Clock, and Interrupt |



Instructions are executed by the performance of the Read Next Instruction (RNI) sequence and one other. For example, instruction 14, ADD, is executed by the RNI sequence followed by the Read Operand (RO) sequence. RNI enters the 24-bit instruction word in the upper half of $U^1$. RO obtains the operand from storage and performs the actual addition.

The Auxiliary sequence is made up of three sub-sequences. One handles the execution of buffer operations, that is, the exchange of data with an external equipment. A second provides for the operation of advancing the real-time clock. The third handles the computer's recognition of an interrupt signal and initiates the routine that responds to the interrupt.

This section on control sequences begins with general, preliminary information concerning sequences. The generation of commands by sequences, initiation of sequences, and exiting from sequences are discussed; following this, the individual sequences are taken up. No attempt is made in the treatment of a given sequence to describe in detail all of its uses, although peculiarities and complicated aspects are taken up. A complete and detailed study of a sequence can be best made by assuming the sequence is to perform a given instruction or auxiliary operation. For each instruction and operation, Appendix presents a command timing chart, which shows the complete list of commands generated by the appropriate sequence to execute the instruction or operation. These charts should be studied in conjunction with individual sequences.

Each sequence consists of a series of control delays. Figure 2-8 shows a hypothetical example of such a series. Such important features of actual sequences as initiation, modes of generating commands, and exiting appear in this figure, and are discussed in the following paragraphs.

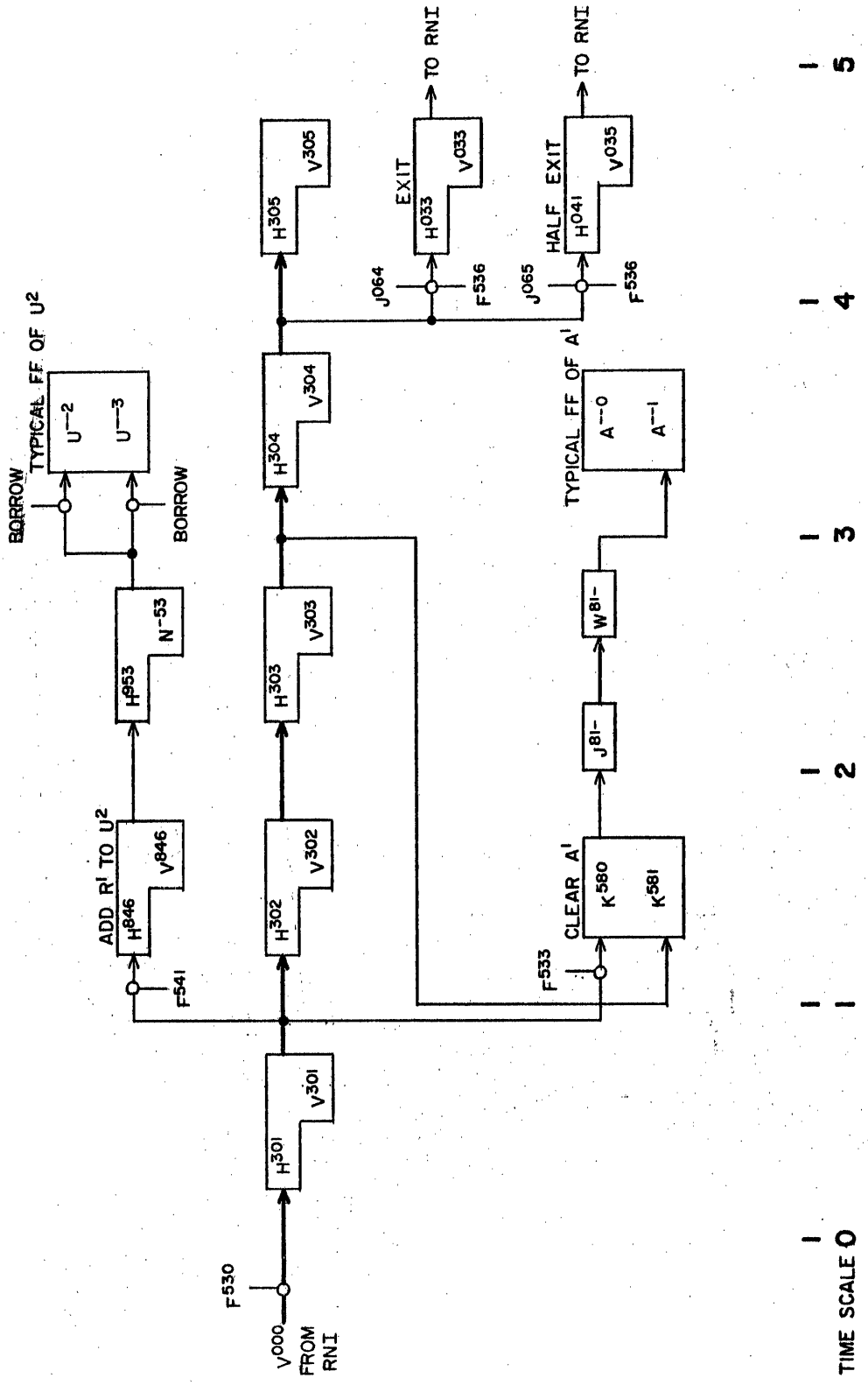
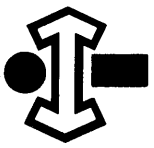



Figure 2-8. Example of a Sequence.



In the sequence of Figure 2-8 the basic series of control delays are those with $H^{3--}$ $-V^{3--}$ symbols. The sequence is initiated when a single pulse on the conditions represented by $F^{530}$, is sent from the RNI sequence ($V^{000}$) to $H^{301}$, the first control delay. This pulse moves down the chain of $H^{3--}$ $-V^{3--}$ control delays at a rate of one control delay per clock period (0.2 microseconds). The time at which $H^{301}$ receives this pulse is considered time 0. The scale at the bottom of the figure shows relative time positions for later control delays.

There are two methods for generating commands. The first is illustrated by the Add $R^1$ to $U^2$ command, for which pertinent logical elements are shown at the top of Figure 2-8. This means of generating commands is fully clocked, or timed, since in this case a control delay, whose output is clocked, applies the command signal to the FFs of the register being operated on at a definite time. After the sequence is initiated the $H^{846}$ $V^{846}$ control delay is set at time one if the condition given by $F^{541}$ is met. At time two, $H^{953}$ $N^{-53}$ is set by the output of $V^{846}$. At time three the command reaches the FFs of $U^2$. By time four the command is complete and thus the quantity is available at the outputs of the register FFs.

It is pertinent at this point to explain why the Add $R^1$ to $U^2$ command is, in this case, generated by the use of $H^{846}$ $V^{846}$. The command could be generated and occur at time three by providing an input to $H^{953}$ from $V^{302}$, instead of taking an output from $V^{301}$ and going through $H^{846}$-$V^{846}$ to $H^{953}$. In fact, some of the several instances for which the Add $R^1$ to $U^2$ command is generated do employ the former procedure. However, the total number of occasions for which this command is to be generated is greater than the maximum number of inputs possible for $H^{953}$. The occasions in excess of the input capacity of $H^{953}$ are provided for by $H^{846}$, which then requires only one input to $H^{953}$ to handle all of them. All control delays with $H^{8--}$ symbols perform this function of, in effect, increasing the input capacity of the $H^{9--}$ control delays which actually bring the command to the register FFs. The $H^{8--}$ control delays are called <u>initiates</u> -- a term intended to indicate that they do not actually supply the command to the register.

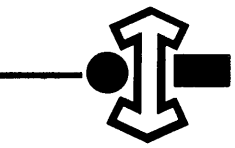
The second method of generating commands is illustrated by the Clear $A^1$ command shown at the bottom of Figure 2-8. This method involves setting a FF, Clear $A^1$. The "1" output of this FF is fed to single inverters $J^{81-}$. The output of $J^{81-}$ goes to $W^{81-}$ single inverters whose outputs feed the "0" side of the $A^1$ flip-flops. When FF $K^{580}$ $K^{581}$ is set to "1", $W^{81-}$ provides a "1" input to the "0" side of the $A^1$ FFs and thereby clears $A^1$. The Clear $A^1$ FF is set at time one and then cleared at time three. Since the transmissions from the output of the Clear $A^1$ FF to $J^{81-}$, then to $W^{81-}$, and finally to the $A^1$ FFs are not clocked, the time at which $A^1$ is cleared is not entirely determined. It occurs between time two (minimum) and time three (maximum).

At some point in the sequence, usually near the end a full exit or half exit is made to RNI. RNI is entered by full exit if the current instruction came from the lower part of the storage location. The half exit is used if the current instruction is in the upper position. The choice of full exit or half exit is determined on the basis of the Exit FF, which is sampled by single inverters $J^{064}$ and $J^{065}$.

Although the sequences employed in the execution of an instruction are performed successively, their periods often overlap somewhat. In other words, if the current instruction uses the RO sequence, then RO may exit to RNI before RO is completely executed.

## OVER-ALL RELATION OF SEQUENCES

When instructions are obtained from storage, two are read at one time and the upper instruction executed before the lower. The form of RNI which reads the instruction pair from storage is called the "full RNI", and precedes execution of the upper instruction. The form of RNI which precedes execution of the lower instruction is called the "half RNI". Therefore, exiting from an instruction sequence is conditioned by whether the next instruction is already in $U^1_L$ or must be obtained by a storage reference. The ordinary relation of instruction sequences to RNI is shown in figure 2-9.



An instruction is read by half or full RNI and executed by one of the instruction sequences. The sequence exits to RNI by means of one of the three exit control delays. The purpose of the Exit FF ($K^{060}$-$K^{061}$) is to aid in making the choice of an exit. During a full RNI it is set to indicate that the next instruction is already in $U_L^1$ and thus a half RNI is required. During a half RNI the Exit FF is cleared to indicate that the next instruction must be acquired by reading a pair from storage (through the use of full RNI).

Jump instructions, when the condition is satisfied, use the jump exit to initiate a full RNI. The jump exit is used rather than the full exit because the latter advances P and this is not required when the jump is satisfied. When the jump is not satisfied then either the half or full exit, depending on the state of the Exit FF, is used.

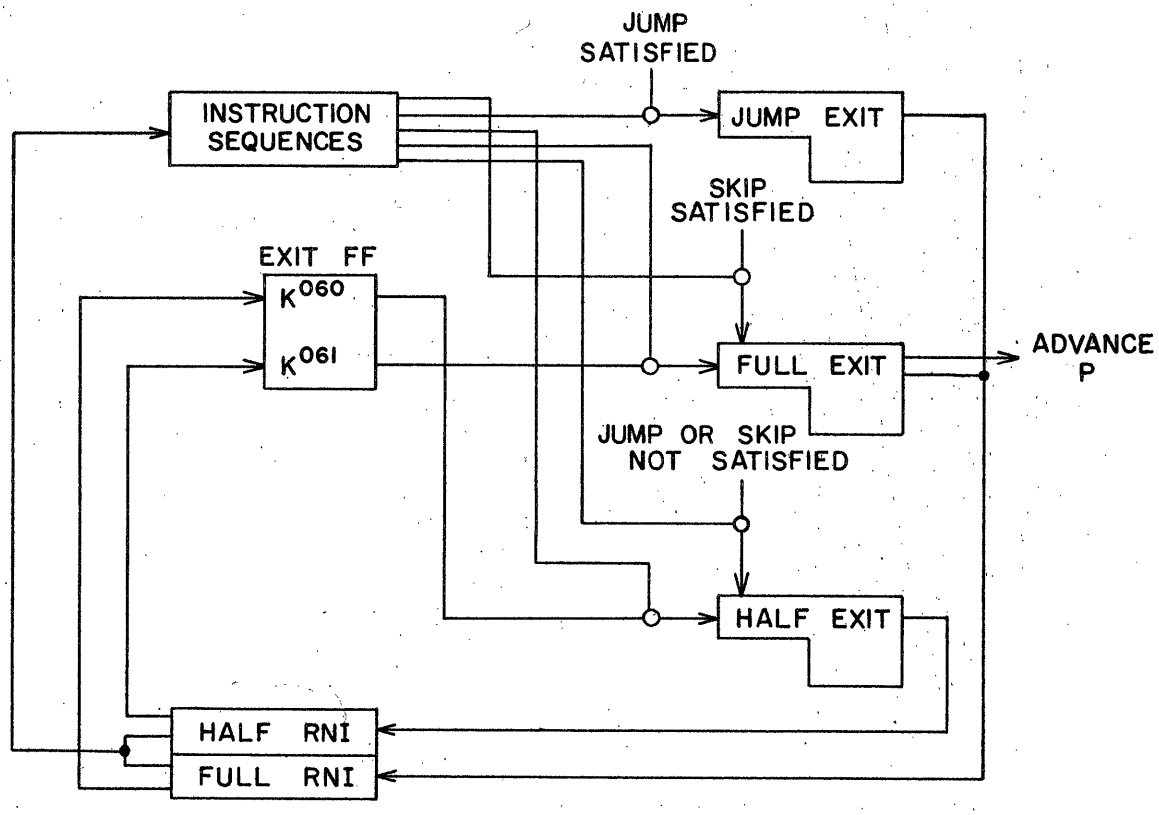



Figure 2-9. Overall Sequence Control.

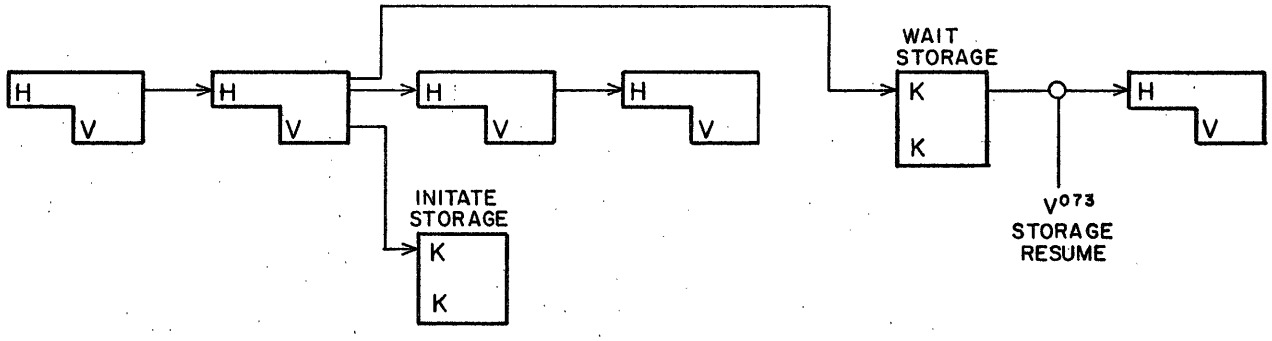
Certain instructions provide for a conditional skip of the next instruction. Such skip instructions are always located in the upper position of a pair. Thus skipping involves taking a full exit.

## RELATION OF CONTROL AND STORAGE SEQUENCES

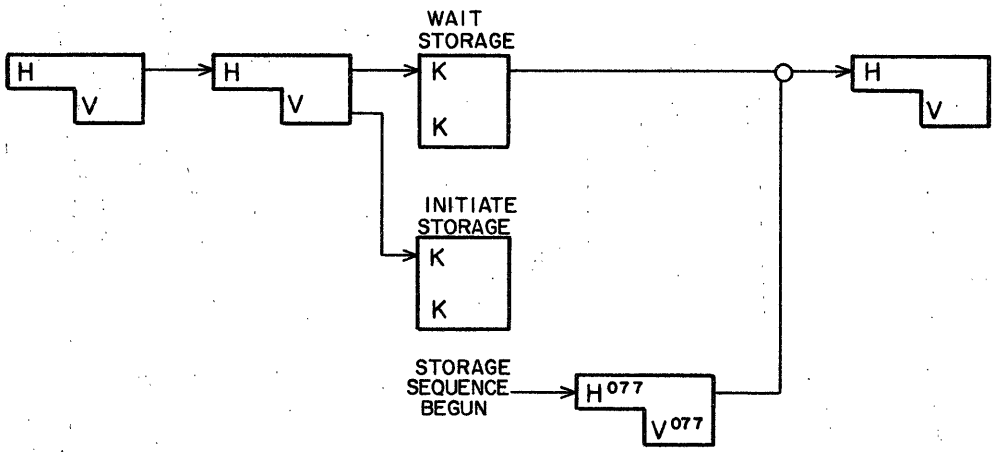
For those control sequences which initiate a storage reference it is necessary to maintain proper timing relations between the control sequence and the storage sequence. The maintenance of proper timing relations is made more complex because of the two storage units. Having two units introduces variability in the time when the selected storage unit is ready for use by the control sequence. If a control sequence requires a storage reference using the odd unit and the reference immediately preceding it used the even unit, then the reference to the odd unit can proceed before the reference to the even unit is completed. But when a reference to the odd unit is preceded by a reference to this unit, the second reference must wait until the first is complete. In short, successive storage references, each with a different unit, may overlap, whereas successive references to the same unit can not overlap.

The control sequences (1) allow for this variability in the time when a storage reference may begin, and (2) maintain synchronization with the storage sequence by means of the Wait Storage FFs. For each storage-reference of each control sequence there is a unique Wait Storage FF (see figure 2-10A). Ordinarily, this FF is set at the time the Initiate Storage FF is set. The control sequence continues with those commands that need not be timed with the storage sequence. Then the control sequence stops to await the occurrence of the Storage Resume. This signal indicates that the storage sequence has read the word from the specified address and that it can be sampled from $I^5$ or $I^6$. The Resume signal and the set output of the Wait Storage FF are combined in an AND which starts the control sequence again.

The preceding discussion considered the synchronizing of a control sequence with the latter part of the storage sequence. In some cases the control sequence must be synchronized with the first part of the storage sequence. (See figure 2-10B). Here, as in the case treated above, the Wait Storage FF is set at the time the Initiate Storage FF is set.




A. SYNCHRONIZING CONTROL SEQUENCE WITH READ TIME OF STORAGE SEQUENCE.




B. SYNCHRONIZING CONTROL SEQUENCE WITH BEGINNING OF STORAGE SEQUENCE.

Figure 2-10. Relation of Control and Storage Sequence.

The control sequence then stops. When the storage sequence actually begins, a special control delay, $H^{077}\ V^{077}$, receives a pulse. This is combined with the set output of the Wait Storage FF to begin the sequence.

READ NEXT INSTRUCTION SEQUENCE

The Read Next Instruction (RNI) sequence performs the following functions:

1) Acquistion of a new instruction word
2) Start and stop
3) Preliminary steps in address modification
4) Preliminary control and arithmetic steps
5) Interrupt termination
6) Indirect addressing

Acquisition Of New Instruction Word

Two 24-bit instruction words may be stored in a 48-bit storage location. Therefore, two alternatives are performed by the RNI sequence (figure 2-11). The first provides for transfer of the 48-bit quantity to the $U^1$ register making the 24-bit quantity in $U^1$ upper then available as the next instruction. The second alternative provides for transfer of the 24-bit quantity from $U^1$ lower to $U^1$ upper.

Figure 2-11 includes the sequence control delays and a number of other logical circuits pertinent to the instruction acquisition. The sequence is made up of two segments, the left represented by $H^{090}\ V^{090}$ and $K^{200}\ K^{201}$, and the right represented by $H^{094}\ V^{094}$ through $H^{099}\ V^{099}$. It can be seen that an entry to the left segment initiates storage and waits for the resume. Inputs to this segment come from the full exit, jump exit, and Initial Start FF. Inputs to the right segment come from the half exit and storage resume.

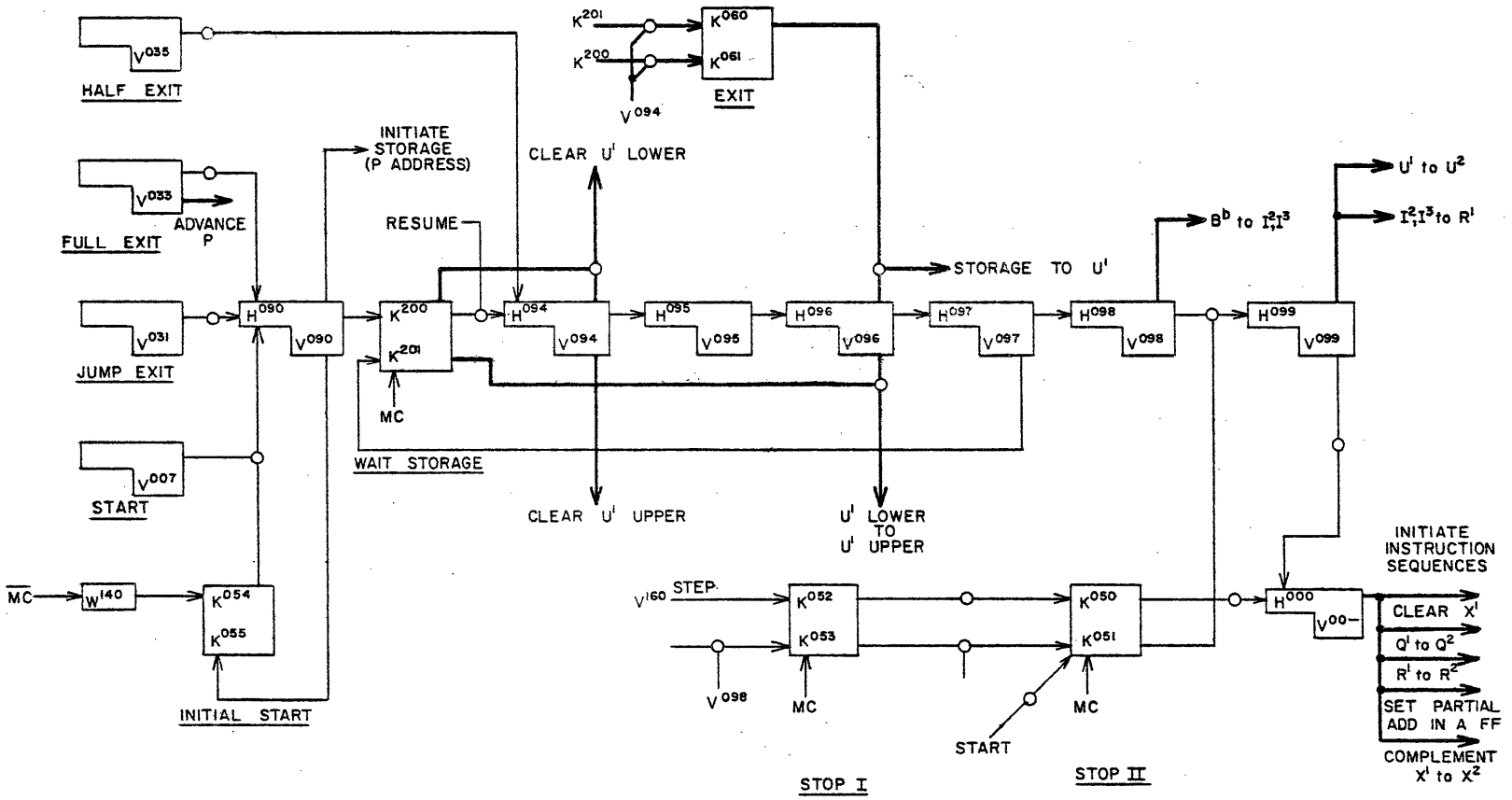



Figure 2-11. Form of RNI for Acquiring Instructions.

Examination of the sequence shows the following steps from initial start (also full exit or jump exit):

Initiate storage (P Address)

Wait storage

Storage resume

Set Exit FF

Clear $U^1$ upper and lower

Transmit storage to $U^1$

The corresponding steps performed from Half Exit are:

Clear Exit FF

Clear $U^1$ upper only

Transmit $U^1$ lower to $U^1$ upper

At the end of the above steps, a 24-bit instruction word is located in $U^1$ upper, ready to direct the execution.

Start and Stop

The RNI sequence provides for all manual and program stops and starts. Figure 2-12 shows the pertinent logical circuits.

Starts may be made from two conditions -- initial start and stop. Initial start, as shown in the figure, initiates the RNI sequence through $H^{090}$. This action occurs once only, on starts following master clear. Start from a stop condition begins the instruction execution through $H^{000}$.

A special case of instruction execution is the sweep mode selected by a key lever on the operator's console. This mode bypasses the instruction entirely and re-enters the RNI sequence through full exit or half exit. This case is also shown in figure 2-12. Sweep inputs to the $V^{00-}$ terms serve to disable them, and thereby disable any instruction initiates.

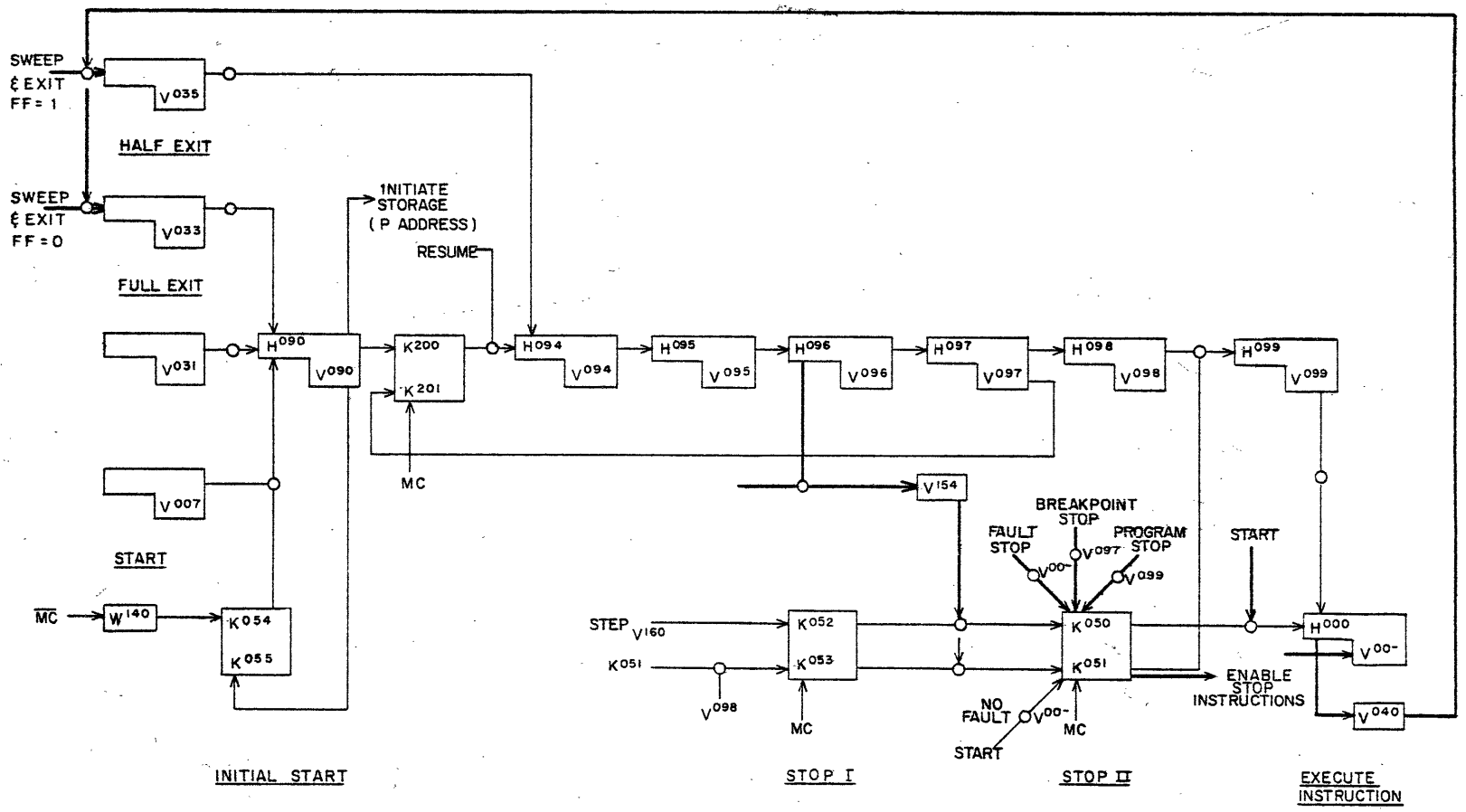



Figure 2-12. RNI for Start and Stop.

The $V^{040}$ term then selects the appropriate exit as determined by the Exit flip-flop.

A number of conditions may stop the computer operation. These are shown in figure 2-12 as inputs to Stop II, $K^{050}$ $K^{051}$. The first of these is an instruction stop produced by depressing the step key. This action sets Stop I, $K^{052}$ $K^{053}$, and waits for the RNI sequence (which may have been initiated from the step or from previous computer sequences). The RNI sequence transfers Stop I to Stop II in time to disable the last step of the sequence ($V^{098}$ to $H^{099}$).

A second stop condition is the breakpoint stop. The breakpoint condition is a comparison of the P register with the breakpoint switch. This condition is sampled in time to disable the last step of the RNI sequence.

Two more stop conditions, program stop and fault stop, are not sampled in time to disable the last step of the RNI sequence. Program stops are accomplished in the same manner, however, in the instruction sequence. The fault stops do not enter any sequence, and thereby stop by default. Re-starting from stop clears Stop II so that program stop sequences may proceed.

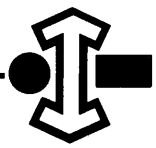
Preliminary Steps in Address Modification

When the b designator has the value 1-6 it may be necessary to modify the base execution address by the addition of $B^b$. While the actual addition is initiated by instruction sequences the preparation for the addition is handled by RNI. Figure 2-11 shows the following steps which are pertinent to address modification:

$B^b$ to $I^2$ or $I^3$

$I^2$ or $I^3$ to $R^1$

$U^1$ to $U^2$



Preliminary Control and Arithmetic Steps

Some control and arithmetic commands need to be performed early and for many instructions. They are produced at the end of RNI, as shown in figure 2-11. These commands are:

Clear $X^1$

Set Partial Add in A FF

$Q^1$ to $Q^2$

Complement $X^1$ to $X^2$

$R^1$ to $R^2$

Interrupt Termination

An interrupt signal is a request for action which may be originated by an external equipment or within the control section of the computer. In order for the computer to respond, the main program is suspended temporarily while a special routine of instructions performs the response.

The computer's acknowledgement of an interrupt request and the initiation of the interrupt routine are handled by the AUX sequence. The return to the next instruction of the main program after termination of the interrupt routine is handled, for the most part, by RNI. Figure 2-13 shows the pertinent logical circuits.

During the initiation of the interrupt routine by the AUX sequence, three steps are taken to prepare for terminating the routine and returning to the main program. They are:

Store address of next instruction of main program in upper address position of address 00007.

Store Exit FF in the Interrupt Exit FF.

Set Interrupt Lockout FF.

The first keeps a record of the address which contains the next instruction of the interrupted program. For that address the second keeps a record of the position (upper or lower) in

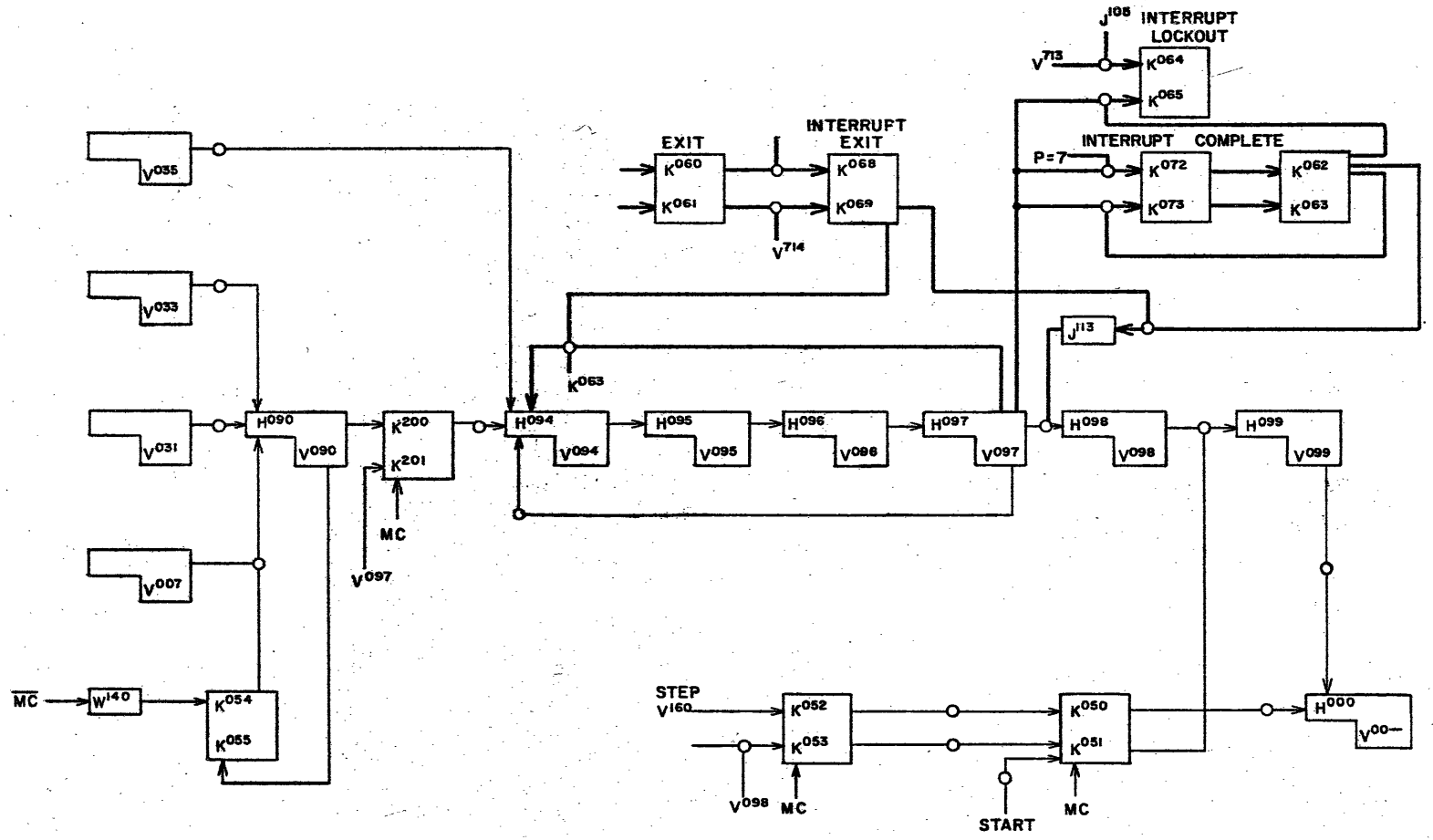
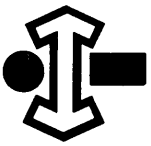



Figure 2-13. RNI for Interrupt Termination.



which holds the next instruction. The third prevents acknowledgement of a second request while the first request is being responded to.

In the termination of the interrupt routine a jump is made to address 00007. The steps below then bring about the return to the main program.

1) Jump to address 00007:
   a) sets P to 00007
   b) initiates full RNI from jump exit
2) Full RNI from step 1b:
   a) reads content 00007 into $U^{1}$
   b) sets Interrupt Complete FF ($K^{072}$ $K^{073}$)
3) Execution of upper instruction at 00007:
   a) jumps to next instruction of main program
   b) initiates full RNI from jump exit
4) Full RNI from step 3b:
   a) clears Interrupt Lockout FF
   b) clears $K^{072}$ $K^{073}$
   c) continues from $V^{097}$ to $H^{098}$ if Interrupt Exit FF=1
   d) goes to $H^{094}$ from $V^{097}$ if Interrupt Exit FF=0
      (this is a half RNI)
   e) after step 4d, $K^{062}$ $K^{063}$ is cleared to enable AND from $V^{097}$ to $H^{098}$

Indirect Addressing

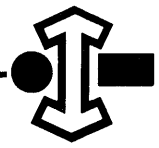
The preceding discussion pertained to RNI as used in direct addressing. Indirect addressing, also accomplished by RNI, is often chosen for programs involving a great deal of address modification because it simplifies programming and reduces the time for running the program.

With indirect addressing the execution address part of the instruction is used as the address of a storage location that holds the operand address, whereas in normal or direct addressing the operand address is obtained immediately by the execution address (modified by the contents of an index register when desired.) Thus in direct addressing the execution address indicates the location of the operand; in indirect addressing the execution address indicates the location of the operand address. An additional memory reference is required in the latter case to obtain the operand address.

All instructions except 22, 23, 74, 75 and 76 may be used with either direct or indirect addressing. Indirect addressing occurs when b=7, otherwise direct addressing is used. Note that the above statements apply to instructions which use the execution address as an operand, such as the shift instructions.

As examples of direct and indirect addressing, suppose that the two instructions in address 05012 are to be executed next (see table 2-3). Because b is 3 in the upper instruction (left) direct addressing is used in its execution. Thus $B^3$ is added to 71331 to produce the address of the operand. In the lower instruction (right) because b is 7 indirect addressing is used; therefore m is used as an address for obtaining a new operand address. Now the lower 18 bits are read out of address 00367 (see table 2-3B), while the remaining upper bits are ignored. These 18 bits are substituted in the program control register for the original 18-bit quantity made up of b and m. As a consequence the current instruction has been altered so that it is 14 2 11135.

The designator is examined again, and since it is not 7, the address of the operand is $1135 + B^2$. But, if the new value of b had been 7, a second indirect addressing operation would have resulted. This situation is illustrated by the upper instruction at address 05013. Since b is 7 in this instruction, the lower 18 bits at address 04006 are substituted in the program control register, which then holds 01 7 11466. Since b is 7 again, the lower 18 bits



in address 11466 are entered in the program control register. Because b is zero, 00012 is used as the execution address.

TABLE 2-3. EXAMPLES OF INDIRECT ADDRESSING

A. PROGRAM

| Address | Upper Instruction | Lower Instruction |
|---|---|---|
| | f b m | f b m |
| 05012 | 36 3 71331 | 14 7 00367 |
| 05013 | 01 7 04006 | 12 6 71331 |

B. OTHER MEMORY LOCATIONS

| Address | | Content |
|---|---|---|
| 00367 | 01436675 | 42 2 11135 |
| 04006 | | 7 11466 |
| 11466 | | 0 00012 |

With this background information, the actual accomplishment of indirect addressing by RNI may be examined. Figure 2-14 shows the relevant part of RNI. The choice between direct and indirect addressing is made at the output of $V^{099}$. By the time RNI has progressed to $V^{099}$ the f and b parts of the instruction just placed in $U^{1}_{u}$ have been translated. Conditions for the choice are:

Indirect Addressing - f ≠ 22, 23, 74-76 and b=7

Direct Addressing - f = 22, 23, 74-76 or b ≠ 7

Figure 2-14 indicates the following sequence of steps for indirect addressing:

Initiate storage (on address m)

Clear b and m parts of $U^{1}_{u}$

Wait storage

Transmit storage to b and m part of $U^{1}_{u}$

At the end of this sequence a new 18-bit quantity occupies the b and m portions of $U^1_u$. This new value of b specifies whether the new value of m is a direct or indirect address. If the former is specified, the execution of the instruction proceeds from $H^{000}$. If indirect addressing is specified (the new value of b is 7), then the steps listed above are performed again.

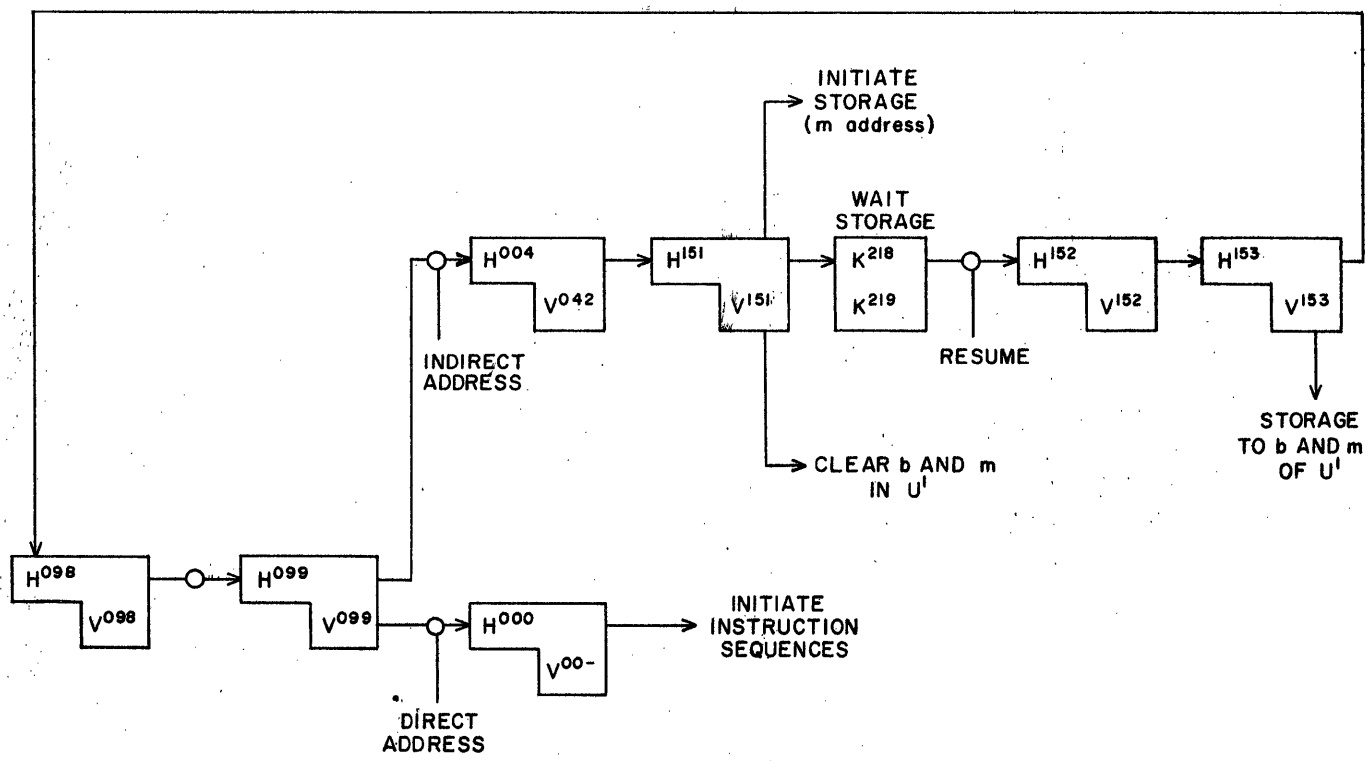



Figure 2-14. Indirect Addressing Part of RNI.

NORMAL JUMP SEQUENCE

The Normal Jump (NJ) sequence is a very short one used by instructions 22, 23, 75 and 76, when j = 0-3, that is, when these instructions perform normal jumps rather than return jumps.

Figure 2-15 shows that when the jump condition is met, this sequence transmits $U^2$ to $P^1$, thereby entering the address of the first instruction of a new program in P. The jump exit is then taken to initiate the full RNI and read the pair of instructions at address P. If the jump condition is not met, the full exit or half exit (depending on the state of the Exit FF) is taken to initiate the full or half RNI respectively.

Instruction 76, Selective Stop, provides for a jump regardless of whether the conditions for stopping are met. When the stop condition is met the AND to $H^{102}$ is disabled. This stops operation. When the operator resumes operation by either start or step, the NJ sequence is initiated and, since the AND to $H^{102}$ is enabled now due to removal of the stop condition, the jump is completed.

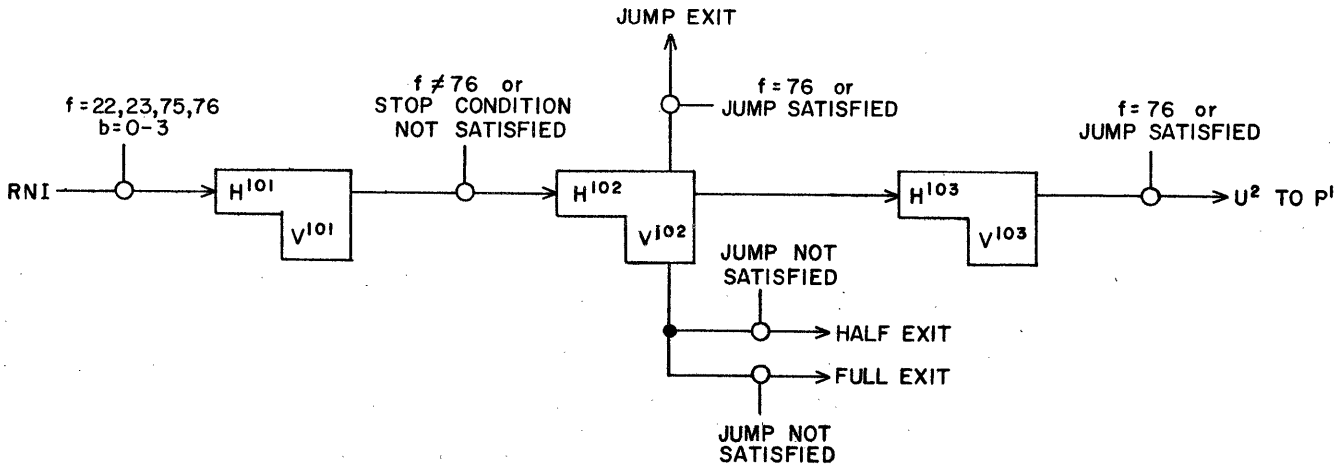



Figure 2-15. Normal Jump Sequence.

ZERO ADDRESS SEQUENCE

The Zero Address (ZA) sequence performs the basic operation for instructions 01-11, 34, 35, 50, 51, 54, and 55. As the name indicates, this sequence makes no storage reference. The execution address is employed as the operand by these instructions.

As shown in figure 2-16, the chain of control delays forming the basis of the ZA sequence consists of two parts. All ZA instructions use the first part while only instructions 04, 10, 11, and 54 (with R ≠ 0) use the last part. Most of the commands for executing ZA instructions are generated from this chain of control delays. However, the commands for shifting and reducing R in instructions 01-03, 05-07, 34, and 35 are generated independently of the main chain by the shift control. The chain merely establishes the enabling conditions which permit these commands to occur until the terminating conditions are reached. Table 2-4 lists the commands produced by this sequence.

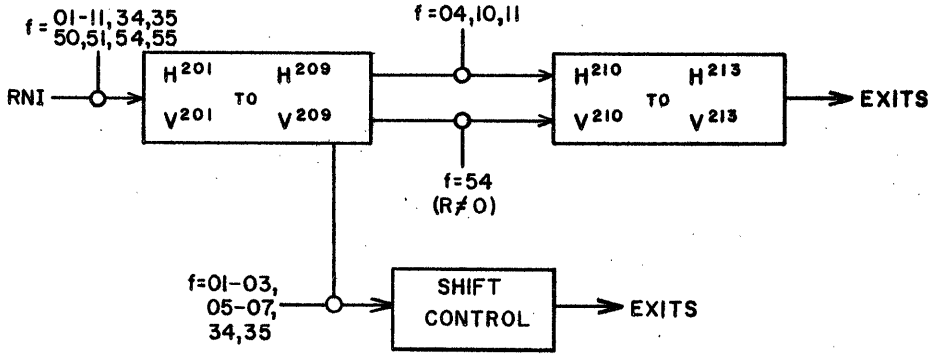



Figure 2-16. Basic Chain of Control Delays in Zero Address Sequence.

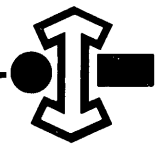
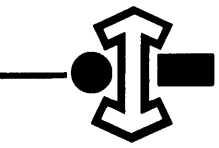


TABLE 2-4. COMMANDS FOR ZERO ADDRESS SEQUENCE
instructions 01-07, 10, 11, 34, 35, 50, 54, 55

| | | |
|---|---|---|
| 00 | $U^1 \rightarrow U^2$ | |
| 01 | Clear $X^1$ | |
| 02 | $R^1 \rightarrow R^2$ | 54+55 |
| 03 | $A^2 \rightarrow Q^1$ | 04 |
| 03 | Clear $A^1$ | 04+10 |
| 04 | Part. Add $R^1$ to $U^2$ | 04 |
| 04 | Add $R^1$ to $U^2$ | (b≠0) (34+35+50+55) |
| 05 | Comp $R^2$ to $R^1$ | 54 |
| 05 | $U^2 \rightarrow P^1$ | 55(R≠0 at time 03) |
| 05 | Clear B | 34+35+50+51+54+55 |
| 06 | $U^2 \rightarrow R^2$ | 01-03+05-07+34+35+50+51+54 |
| 06 | $U^2 \rightarrow X^1$ | 04+10+11 |
| 07 | Full Exit | |
| 07 | Half Exit | |
| 07 | Jump Exit | 55(R≠0 at time 03) |
| 07 | $R^2 \rightarrow R^1$ | 01-07+34+35 |
| 08 | $X^1 \rightarrow X^2$ | |
| 08 | Red. $R^1$ to $R^2$ | 55 |
| 08 | $R^2 \rightarrow B^b$ | 50+51+55 |
| 09 | Full Exit | 10+11 |
| 09 | Half Exit | 10+11 |
| 09 | Full Exit | 54(R=0 at time 08) |
| 09 | Full Exit | 50+51 |
| 09 | Half Exit | 50+51 |
| 10 | $R^1 \rightarrow R^2$ | (04+10+11) (R≠0 at time 08) |
| 10 | $R^2 \rightarrow B^b$ | 34 $A_{47} \neq A_{46}$ A=0+35$A_{47} \neq A_{46}$ AQ≠0 |

TABLE 2-4. (Continued)

| | | |
|---|---|---|
| 10 | Initiate Shift | (01-03+05-07)+(34$A_{47}$=$A_{46}$A≠0)+(35$A_{47}$=$A_{46}$AQ≠0) |
| 10 | A Right | (01+03)(R≠0-8) |
| 10 | Q Right | (02+03)(R≠0-8) |
| 10 | A Left | (05+07+34+35)(R≠0 at time 08) |
| 10 | Q Left | (06+07+34+35)(R≠0 at time 08) |
| 11 | Full Exit | |
| 11 | Half Exit | |
| 11 | Full Exit | 04 |
| 11 | Half Exit | |
| 11 | Shift One | (01-03+05-07)+(34$A_{47}$=$A_{46}$A≠0)+(35$A_{47}$≠$A_{46}$AQ≠0) |
| 12 | Reduce $R^1$ to $R^2$ | (01-03, 05-07, +34+35) (R≠0 at time 08) |
| 12 | Reduce $R^1$ to $R^2$ | 54 (R≠0 at time 09) |
| 13 | Full Exit | (01-03+05-07+34+35) (R=0 at time 12) |
| 13 | Half Exit | (01-03+05-07+34+35) (R=0 at time 12) |
| 13 | Full Exit | 34$A_{47}$≠$A_{46}$A≠0 |
| 13 | Half Exit | +35$A_{47}$≠$A_{46}$AQ≠0 |
| 13 | Comp $R^2$ to $R^1$ | |
| 13 | Part. Add $X^2$ to $A^1$ | 04+10 |
| 13 | Add $X^2$ to $A^1$ | 11 |
| 14 | $Q^1 \rightarrow Q^2$ | 04 |
| 14 | $R^1 \rightarrow R^2$ | 54(R≠0 at time 09) |
| 15 | $A^2 \rightarrow Q^1$ | 04 |
| 15 | $Q^2 \rightarrow A^1$ | 04 |
| 15 | $R^2 \rightarrow B^b$ | 54(R≠0 at time 09) |
| 16 | Half Exit | 54(R≠0 at time 09) |



READ OPERAND SEQUENCE

The Read Operand (RO) sequence is used for executing instructions 12-17, 36-46, 52, 53, and 70-73. To aid in following the operation of the sequence, figure 2-17 provides a simplified form of the main chain.

All the instructions executed by this sequence read an operand from storage. Instructions 37 and 70-73 also store an operand at the same address at the conclusion of their execution. Thus for all instructions using this sequence at least one storage reference is initiated, and for the instructions mentioned above two references are made.

The main chain of control delays have $H^{3--}$ $V^{3--}$ symbols. The first part of the chain, $H^{301}$ $V^{301}$ to $H^{312}$ $V^{312}$, performs those commands that must precede the storage reference. The period of waiting for storage occurs after $V^{312}$; the Storage Resume allows the remaining portion of the sequence to continue. The second storage reference in RO is performed for instructions 37, 70-73 in order to write the operand, after alteration, into the location from which it was read initially by the first storage reference. Table 2-5 lists the commands produced by this sequence.

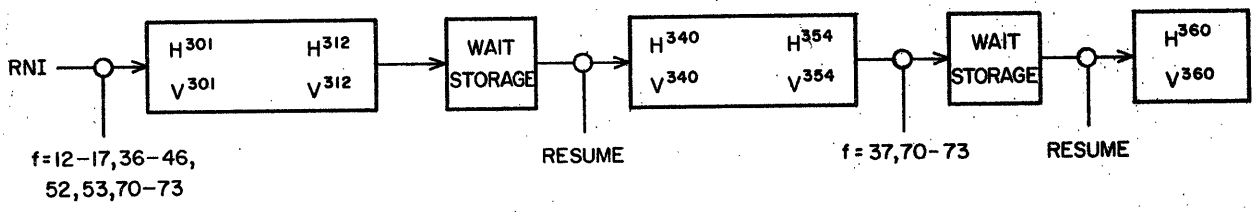



Figure 2-17. Basic Chain of Control Delays in RO Sequence.

TABLE 2-5. COMMANDS FOR READ OPERAND SEQUENCE
instructions: 12-17+36+37+40-46+52+53+70-73

| | | |
|---|---|---|
| 00 | $U^1 \rightarrow U^2$ | |
| 01 | Clear $X^1$ | |
| 03 | Set $X^2$ to 1 | 72+73 |
| 04 | Add $R^1$ to $U^2$ | 52+53 (b≠0) |
| 04 | Init. Storage | |
| 05 | Clear $R^1$ | |
| 06 | $Q^1 \rightarrow Q^2$ | 37+43+44-46 |
| 06 | $X^1 \rightarrow X^2$ | 72 |
| 06 | Comp $X^1$ to $X^2$ | 41+43+71+73 |
| 06 | Clear $A^1$ | 12+13+16+17+37+44+72+73 |
| 07 | $X^2 \rightarrow X^1$ | 43 |
| 07 | $A^2 \rightarrow Q^1$ | 16+17+37 |
| 07 | Part. Add $X^2$ to $A^1$ | 41+43+71 |
| 07 | Log. $Q^2$ to $X^1$ | 43 |
| 11 | $A^1 \rightarrow X^1$ | 43 |
| 11 | Part. Add $X^2$ to $A^1$ | 72+73 |
| 11 | Clear $A^1$ | 43 |
| 12 | Comp $X^1$ to $X^2$ | 43 |
| 13 | Clear $X^1$ | 43+72+73 Wait Storage |
| 14 | Clear B | 52+53 |
| 15 | $I^5I^6 \rightarrow X^1$ | |
| 15 | $A^1 \rightarrow X^1$ | 40+41 |
| 15 | Clear $A^1$ | 40+41 |
| 15 | Log. $Q^2$ to $X^1$ | 43-46 |
| 16 | $X^1 \rightarrow X^2$ | 12+14+16+37+40+42+70-73 |
| 16 | Comp $X^1 \rightarrow X^2$ | 13+15+17+41 |
| 16 | $X^1_L \rightarrow$ to $X^2_U$ | 53 |

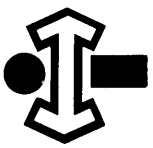
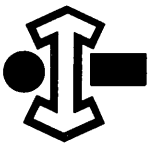


TABLE 2-5. (Continued)

| | | |
|---|---|---|
| 17 | $X^2_U$ to $X^1_U$ | |
| 17 | Part. Add $X^2$ to $A^1$ | 43 |
| 17 | Full Exit | 12-17+40+41+42 |
| 17 | Half Exit | |
| 18 | $X^1_U \rightarrow I^2$ | |
| 19 | Half Exit | 36 X pos |
| 19 | Full Exit | 36 (Exit =1 + X Neg) |
| 20 | Comp $X^1 \rightarrow X^2$ | 46 |
| 20 | $X^1 \rightarrow X^2$ | 43-45 |
| 20 | $I^2$ $I^3$ to $R^1$ | 52+53 |
| 21 | Part. Add $X^2$ to $A^1$ | 12+13+16+17+37+40-42 |
| 20 | Inhibit $A^1 \rightarrow A^2$ | 37 |
| 21 | Add $X^2$ to $A^1$ | 43-46+52+53+36 |
| 21 | Clear $X^1$ | 37+70-73 |
| 21 | Shift One | 37 |
| 21 | Full Exit | 43-46 |
| 21 | Half Exit | |
| 22 | $R^1 \rightarrow R^2$ | 52+53 |
| 22 | $Q^1 \rightarrow Q^2$ | 16+17 |
| 22 | $R^2$ to $B_b$ | |
| 23 | Full Exit | 53 |
| 23 | Half Exit | 53 |
| 23 | $A^2 \rightarrow Q^1$ | 16+17 |
| 23 | $Q^2 \rightarrow A^1$ | 16+17 |
| 23 | Half Exit | 52+53 |
| 23 | Full Exit | |
| 23 | $A^2 \rightarrow A^1$ | 37 |
| 25 | $A^1 \rightarrow X^1$ | 37+70-73 |

TABLE 2-5. (Continued)

| | | | |
|---|---|---|---|
| 25 | Part. Add $X^2$ to $A^1$ | 43+44 | |
| 25 | Add $X^2$ to $A^1$ | 45+46 | |
| 27 | $Q^2 \rightarrow A^1$ | 37 | |
| 28 | $Q^1 \rightarrow Q^2$ | 37 | |
| 28 | Init. Storage | 37+70-73 | Wait Storage |
| 29 | $A^2 \rightarrow Q^1$ | 37 | |
| 29 | $Q^2 \rightarrow A^1$ | 37 | |
| 37 | $X^1 \rightarrow Z^1 Z^2$ | 37+70-73 | |
| 39 | Full Exit | 37 (X Neg) +70-73 | |
| 39 | Half Exit | 37 (X pos) +70-73 | |



WRITE OPERAND SEQUENCE

The write operand (WO) sequence is used for executing instruction 20, 21, 47, 56-61, for all values of the designator and for instructions 22, 23, 75, and 76 with designator values 4-7. The latter group of instructions are return jumps. The instructions executed by this sequence are characterized by the fact that they all require entering an operand in storage.

The main chain of control delays ($H^{4--}$ $V^{4--}$), which produces the commands is shown in figure 2-18. For all WO instructions except 56 and 57 the sequence is initiated at $H^{401}$. The latter two instructions initiate the sequence at $H^{406}$ only after the storage sequence has actually begun. The WO sequence is thereby synchronized with the beginning of the storage sequence.

The conditional stop for instruction 76 is accomplished at the input to $H^{402}$. The AND is disabled when the stop condition is met. After stopping, the execution of the instruction is completed by re-initiating the sequence when the operator selects step or start. The sequence then goes to completion. Table 2-5 lists the commands produced by this sequence.

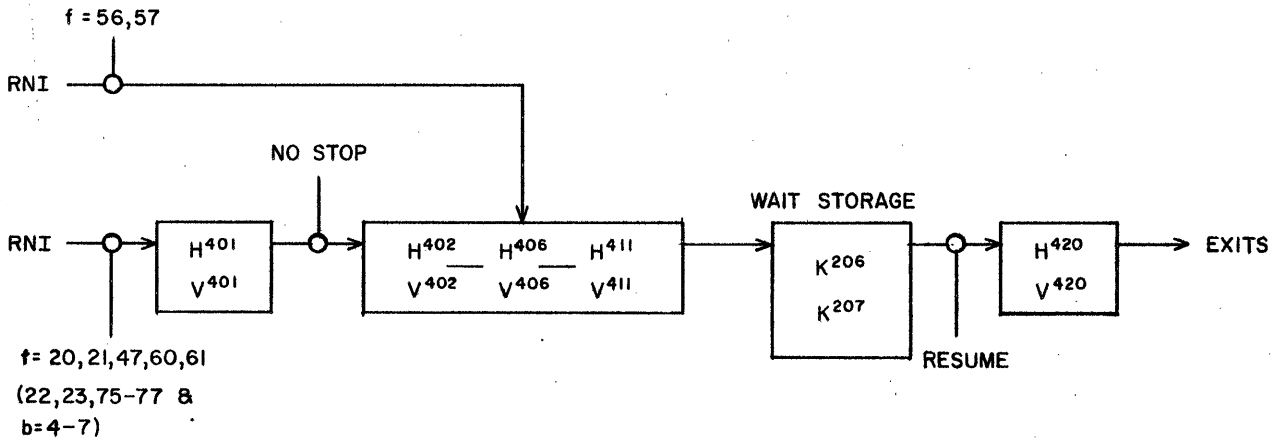



Figure 2-18. Write Operand Sequence.

TABLE 2-6. COMMANDS FOR WRITE OPERAND SEQUENCE
instructions; 20+21+47+56+57+60+61+(b=4-7)(22+23+75-77

| | | |
|---|---|---|
| 00 | $U^1 \rightarrow U^2$ | |
| 00 | Init. Storage | 56+57 Wait Storage |
| 01 | Set Stop | 76 and Key |
| 01 | Clear $X^1$ | |
| 03 | Clear $U^1_U$ | 56+57 |
| 03 | $A^2 \rightarrow Q^1$ | 21 |
| 03 | $Q^2 \rightarrow A^1$ | 21 |
| 04 | Add $R^1$ to $U^2$ | 20+21+47+60+61 |
| 06 | Init. Storage | (jump) 20+21+47+60+61 |
| 06 | Adv. $Q^1$ to $P^2$ | (Jump)(mode Switch Not up) |
| 06 | $Q^1 \rightarrow Q^2$ | 21+47 |
| 06 | $U^1 \rightarrow U^2$ | 56+57 |
| 07 | Full Exit | (22+23+75-77)(b=4-2) |
| 07 | Half Exit | (No Jump) |
| 07 | $A^1 \rightarrow X^1$ | 20+21+47+60+61 |
| 07 | Log $Q^2$ to $X^1$ | 47 |
| 08 | Add $R^1$ to $U^2$ | 56+57 |
| 08 | Part. Add R to $U^2$ | 56+57 |
| 08 | $P^1 \rightarrow X^2_L$ | (22+23+75+76)(Jump) |
| 08 | $X^1 \rightarrow Z^1 Z^2$ | 47+20+21 |
| 08 | Partial Write Lower | 57+61 |
| 08 | Partial Write Upper | 57+61 |
| 08 | $U^2$ to $P^1$ | 22+23+75+76 (Jump) |
| 09 | $U^2$ to $X^1_{LA}$ | 56+57 |
| 09 | $X^2 \rightarrow X^1$ | (22+23+75+76) (Jump)<br>Wait Storage |

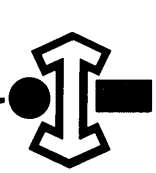


TABLE 2-6. (Continued)

| | | |
|---|---|---|
| 11 | Clear $U^1$ | 22+23+75+76 (Jump) |
| 11 | $A^2 \rightarrow Q^1$ | 21 |
| 11 | $Q^2 \rightarrow A^1$ | 21 |
| 12 | $X^1_L \rightarrow X^2_U$ | (22+23+75+76) (Jump)+56+60 |
| 13 | $X^2_U \rightarrow X^1_U$ | |
| 15 | Half Exit | (22+23+75+76) (Jump)+20+21+47+56+57+60+61 |
| 15 | Full Exit | 20+21+47+56+57+60+61 |
| 15 | $I^5I^6 \rightarrow U^1$ | (22+23+75+76) (Jump) |

SEARCH AND TRANSFER SEQUENCE

The Search and Transfer sequence (ST) performs the search instructions (64-67) and the two transfer instructions (62 and 63). The purpose of the search instructions is to rapidly inspect a specified list of operands for one which meets a certain condition. Essentially, the search conditions are two kinds: operand equal to A and operand greater than A. The purpose of the transfer instructions is to rapidly exchange a block of data with another equipment via the transfer channel.

For both types of instruction the latter part of the sequence is performed repeatedly, once for each word to be examined in the search or once for each word to be transferred. The content of $B^b$ indicates the number of repetitions required; this quantity is reduced by one during each repetition. The block of storage locations involved in search and transfer instructions is specified by $B^b$ and m, the execution address. The first word to be searched or transferred is at the address specified by $B^b + m - 1$. The last word is at address m. When b=0, a search or transfer instruction operates on just one word which is at the address given by m.

Figure 2-19 shows the main part of the ST sequence, which consists of control delays with $H^{5--}$ $V^{5--}$ symbols. The ST sequence consists of three segments or parts, preparation, loop, and termination. The first sets up the block of addresses to be used as well as other initial conditions. The loop produces the series of commands to search or transfer each word, depending on the instruction, and is repeated for each word. The termination part of the sequence takes care of housekeeping duties required when ST is temporarily terminated to allow an AUX sequence operation to be performed.

A search instruction provides for a conditional skip and is thus used in the upper position of a program step. Ordinarily its execution involves repeating the loop for each word until:

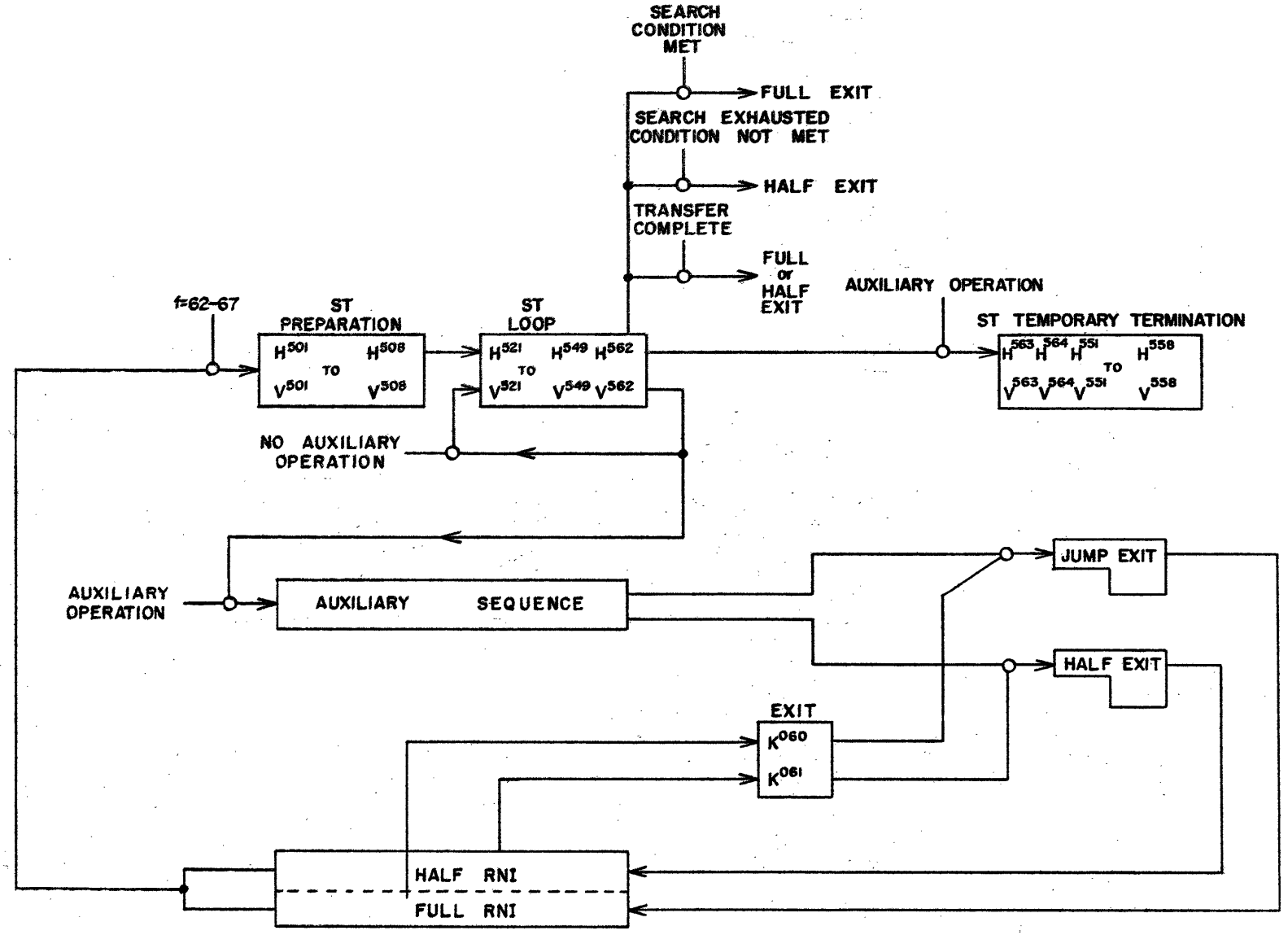



Figure 2-19. Search and Transfer Sequence.

1) A word is found that meets the search condition, at which point the search terminates and the full exit is taken to the next instruction step; or

2) The entire list of words has been searched without finding one that meets the search condition, in which case the half exit to the lower instruction of the current step is taken.

Likewise, the ordinary case of a transfer instruction consists of one repetition of the loop for each word. When all have been transferred, a full or half exit is taken, depending upon whether the instruction is in the lower or upper position.

Since the number of repetitions of the ST loop can be large, provision is made for temporarily suspending execution in order to handle an auxiliary request, that is, a request for one of the operations performed by the Auxiliary sequence. These operations (buffers, advancing real-time clock and interrupt) occur asynchronously to the main program. When the request has been taken care of by performing one of these operations, the execution of the ST instruction is begun again.

The relationship between the ST, AUX and RNI sequences is also shown in Figure 2-19. Note that the need for performing an auxiliary operation causes the termination part of the ST sequence to be executed. Upon completing the operation, AUX takes the jump or half exit to RNI. The condition for choice of exit are such that the form of RNI that is initiated by these exits is the same one that originally acquired the ST instruction. Thus when a ST instruction is suspended to perform an auxiliary operation, its execution is resumed by rereading it. This allows for the case where the suspension is due to an interrupt, which requires that a routine of instructions be executed before returning to the ST instruction.

A more detailed account of the ST sequence appears in chapter 5. The commands produced by the sequence are shown in table 2-7.

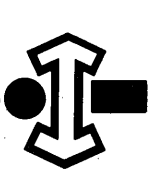
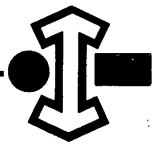


TABLE 2-7. COMMANDS FOR SEARCH AND TRANSFER SEQUENCES
Instructions; 62-67

| | | | |
|---|---|---|---|
| 00 | $U^1 \rightarrow U^2$ | | |
| 01 | Clear $X^1$ | | |
| 02 | Set R ≠ 0 FF | | |
| 02 | $R^1 \rightarrow R^2$ | | |
| 04 | Reduce $R^1$ to $R^2$ | | |
| 05 | $R^2 \rightarrow R^1$ | | |
| 05 | Set ST Not Complete | (R ≠ 0)+(b = 0) | |
| 08 | $Q^1 \rightarrow Q^2$ | | |
| 08 | Set Input Trans. Act. | (R ≠ 0) (b ≠ 0) (St Not Complete) | |
| 09 | Half Exit | (St Complete) (b = 0) | |
| 09 | Full Exit | | |
| 09 | Set Wait Storage | (Input) | |
| 09 | Clear $B^b$ | (Transfer Ready) | |
| 09 | Clear Input Transfer Act. | | 62 |
| 09 | Input Resume | | 62 |
| 10 | Add $R^1$ to $U^2$ | | |
| 10 | Initiate Storage | (Input Trans Ready) | |
| 10 | X = A | | 64 + 66 |
| 11 | X = A | | 65 + 67 |
| 11 | Clear $X^1$ | | |
| 14 | $R^2 \rightarrow B^b$ | | |
| 14 | Set R ≠ 0 FF | | |
| 16 | $U^1 \rightarrow U^2$ | | |
| 16 | Comp $X^1 \rightarrow X^2$ | | 62 |

TABLE 2-7. (Continued)

| | | | |
|---|---|---|---|
| 17 | $X^2 \rightarrow X^1$ | | 62 |
| 17 | $I^0 \rightarrow X^1$ | | 62 |
| 17 | Set ST Not Complete | (R ≠ 0) | |
| 18 | Reduce $R^1 \rightarrow R^2$ | | |
| 19 | $R^2 \rightarrow R^1$ | | |
| 20 | Set Input Trans. Act. | (ST Not Complete) | |
| 21 | $I^5\ I^6 \rightarrow X^1$ | | 63-67 |
| 21 | $I^5\ I^6 \rightarrow Z^1\ Z^2$ | | 62 |
| 21 | Set Output Trans. Act | | 63 |
| 21 | L QX | | 66+67 |
| 22 | Comp $X^1\ X^2$ | | 64+65 |
| 22 | $X^1 \rightarrow X^2$ | | 63 |
| 22 | Clear Wait Storage | | 62 |
| 22 | Output Resume | | 63 |
| 23 | $X^2 \rightarrow 0^4$ | | 63 |
| 23 | Clear Output Trans. Act. | (Buffer Request) | 63 |
| 24 | Comp $X^1 \rightarrow X^2$ | | 66+67 |
| 24 | Add $R^1 \rightarrow U^2$ | | 62+64+67 |
| 24 | Return To Time 08 | (R ≠ 0) (b ≠ 0) | 62+63 |
| 24 | Initiate Aux Seq. | | 62 |
| 25 | Comp $R^1 \rightarrow R^2$ | | 63 |
| 26 | $R^1 \rightarrow R^2$ | | 63 |
| 27 | Initiate Aux Seq. | (Buffer Request) | 63 |
| 28 | Reduce $R^1 \rightarrow R^2$ | | 63 |
| 29 | $R^2 \rightarrow R^1$ | | 63 |
| 29 | Half $_{\text{Exit}}$ | (b ≠ 0)(R = 0)+(b = 0) | 62+63 |
| 29 | Full | | |



| | | | |
|---|---|---|---|
| 29 | Half $_{\text{Exit}}$ | (b ≠ 0) (R = 0)+(b = 0) (X = A) | 64+66 |
| 29 | Full | | |
| 29 | Half $_{\text{Exit}}$ | (b ≠ 0)(R = 0)+(b = 0) (X A) | 65+67 |
| 29 | Full | | |
| 30 | Return to Time 08 | (R ≠ 0) (b = 0) | 64-67 |
| 30 | Reduct $R^1$ to $R^2$ | | 63 |
| 31 | Comp $R^2$ to $R^1$ | | 63 |
| 32 | $R^1 \rightarrow R^2$ | | 63 |
| 33 | $R^2 \rightarrow B^2$ | | 63 |
| 33 | Initiate Aux Sequence (Aux Request) | | 64-67 |

ITERATIVE SEQUENCE

The commands which cause the computer to execute the multiply, divide, and floating-point instruction (24-27, 30-33) rise from the Iterative sequence (I). This sequence is so called because it executes these instructions by iterative or repetitive action; for example, in multiplication the product is formed by the repetitive additions of the multiplicand. The Iterative sequence is composed of a chain of control delays labelled $H^{6--}$ (figure 2-20).

A detailed discussion of the Iterative sequence appears in chapter 3 where the complex arithmetic operations using this sequence are described; table 2-8 lists the Iterative sequence commands.

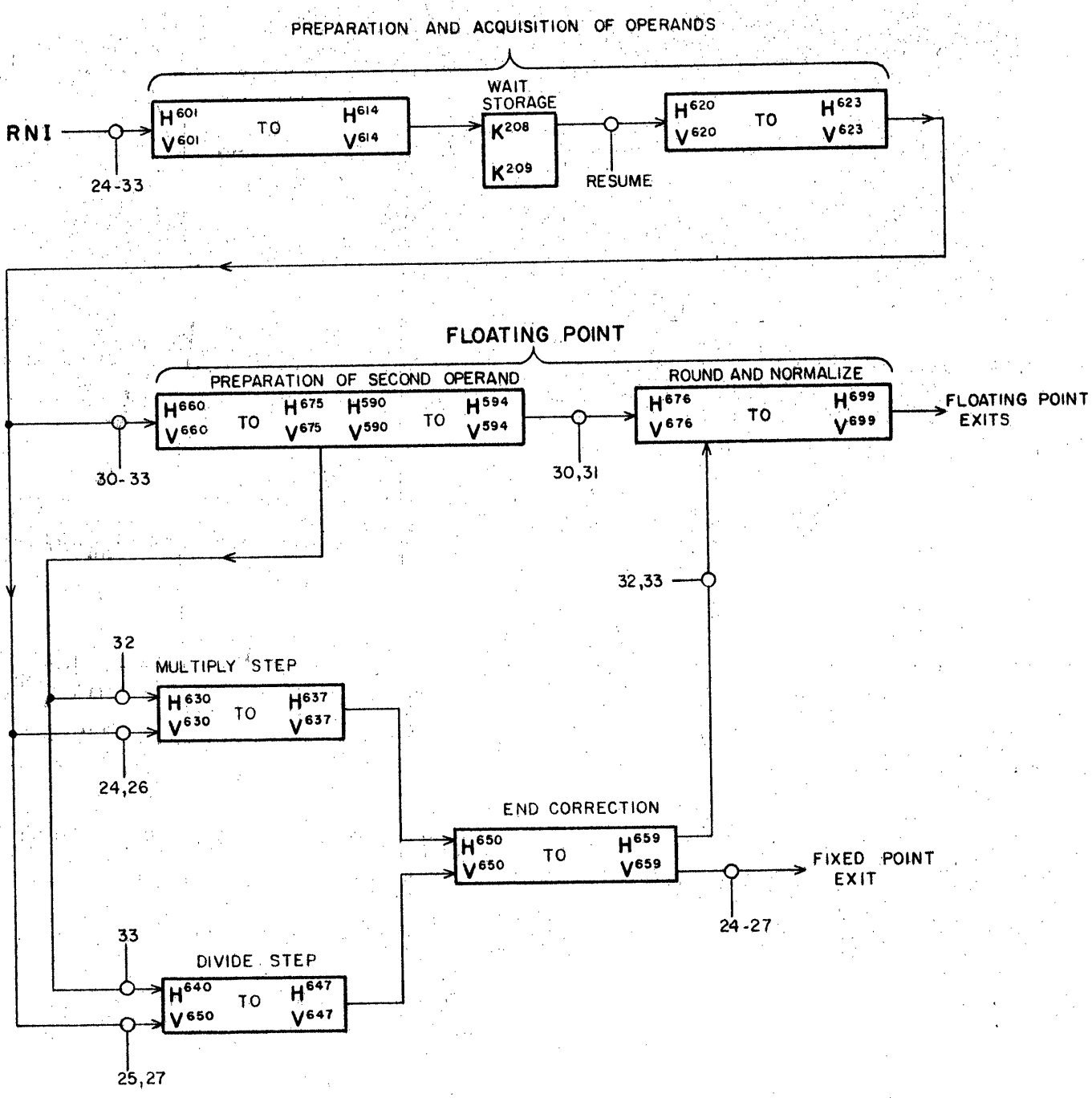



Figure 2-20. Iterative Sequence.

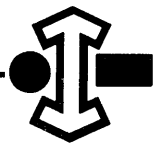


TABLE 2-8a. ITERATIVE SEQUENCE
Main Sequence; Instructions 24-27+30-33

| | | |
|---|---|---|
| 04 | Add $R^1$ to $U^2$ | 24-27+30-33 (b ≠ 0) |
| 04 | Comp $X^1 \rightarrow X^2$ | 24-27+30-33 |
| 05 | Part. Add $X^2 \rightarrow A^1$ | 25 (Q Neg) |
| 05 | Part Add $X^2 \rightarrow A^1$ | 30-33 (A Neg) |
| 04 | Initiate Storage | (32+33) (A = 0) |
| 05 | Set Dividend Sign FF | (25 Q Neg) + (27 A Neg) |
| 07 | Full Exit | (32+33) (A = 0) |
| 07 | Half Exit | |
| 07 | Part Add $X^2 \rightarrow A^1$ | (26+27) (A Neg) |
| 07 | $A^1 \rightarrow X^1$ | 30-33 |
| 05 | Set Sign Record | (24+26+27+30-33) (A Neg) + 25 (Q Neg) |
| 08 | Set $I^2$ | |
| 09 | Clear A | 30-31 |
| 10 | $Q^1 \rightarrow Q^2$ | 25+26+27 |
| 10 | $X^1$ Exp $\rightarrow U^2$ with sign ext. | 30-33 |
| 11 | $U^2 \rightarrow U^1$ | |
| 11 | Clear X Exp | 25+26+27 |
| 11 | $A^2 \rightarrow Q^1$ | 25+26+27 |
| 11 | $Q^2 \rightarrow A^1$ | 25+26+27 |
| 12 | Comp $X^1 \rightarrow X^2$ | (30+31) (SR = 1) |
| 12 | $X^1 \rightarrow X^2$ | |
| 12 | $I^2 I^3 \rightarrow R^1$ | 24-27 |
| 13 | Clear $X^1$ | |
| 13 | Part Add $X^2$ $A^1$ | 30-33 |
| 13 | Part Add $X^2 \rightarrow A^1$ | (24+25+27) (SR = 1) |
| 13 | $A^2 \rightarrow Q^1$ | 30-33 |
| 13 | $U^2 \rightarrow R^2$ | 30-33 |

TABLE 2-8a. (Continued)

| | | |
|---|---|---|
| 12 | Clear SR FF | 30-31 |
| 14 | $Q^1 \rightarrow Q^2$ | |
| 14 | Clear $A^1$ | 24+26+32 |
| 15 | $Q^2 \rightarrow A^1$ | 24+27 |
| 15 | $A^2 \rightarrow Q^1$ | 24+27+32 |
| 15 | Comp $R^2 \rightarrow R^1$ | 30+31+33 |
| 15 | $R^2 \rightarrow R^1$ | 32 |
| 15 | $I^5I^6 \rightarrow X^1$ | $F^{670}$ |
| 16 | $X^1 \rightarrow X^2$ | |
| 16 | $R^1 \rightarrow R^2$ | 24-27 |
| 15 | Exit to Mult. Step | 24+26 |
| 16 | INHIBIT $A^1 \rightarrow A^2$ | 27-33 |
| 18 | Comp $X^1 \rightarrow X^2$ | (25+27) (X Pos) |
| 17 | Exit to Divide Step | 25+27 |
| 17 | Exit to Floating Point | 30-33 |
| 18 | Comp $X^1 \rightarrow X^2$ | (24+26+30+31) (X Neg) |
| 17 | Comp Sign Record | X Neg |

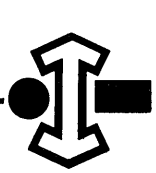


TABLE 2-8b. ITERATIVE SEQUENCE
Execute Multiply Step; Instructions 24+26+32

| | | | | |
|---|---|---|---|---|
| 00 | Reduce $R^1 \rightarrow R^2$ | | Short Loop | Long Loop |
| 00 | Shift AQ Right | | | |
| 01 | $R^2 \rightarrow R^1$ | | | |
| 01 | $A^2 \rightarrow A^1$ | | | |
| 01 | $Q^2 \rightarrow A^1$ | | | |
| 01 | Exit to 00 | R=0, $Q_{00}$=0 | | |
| 01 | Exit to End Correction | R=0 | | |
| 05 | Add $X^2$ to $A^1$ | $Q_{00}$=1 | | |
| 05 | Exit to 0 | | | |
| 05 | Exit to End Correction | R=0 | | |

TABLE 2-8c. ITERATIVE SEQUENCE
Execute Divide Step; Instructions 25+27+33

| | | |
|---|---|---|
| 00 | Red. $R^1$ to $R^2$ | |
| 00 | Shift AQ Left | |
| 01 | $R^2 \rightarrow R^1$ | |
| 01 | $A^2 \rightarrow A^1$ | |
| 01 | $Q^2 \rightarrow Q^1$ | |
| 01 | Exit to 00 | $R \neq 0$, $A < X$ |
| 01 | Exit to End Correction | R=0 |
| 05 | Add $X^2$ to $A^1$ | $A \geq X$ |
| 05 | Set $Q_{00}$ to 1 | $A \geq X$ |
| 05 | Exit to 00 | $R \neq 0$ |
| 05 | Exit to End Correction | R=0 |

TABLE 2-8d. ITERATIVE SEQUENCE
Execute End Correction

| | | |
|---|---|---|
| -02 | Set Divde Fault | (25+27) (Q Neg.) |
| 00 | $Q^1 \rightarrow Q^2$ | 26+32 |
| 01 | Clear $X^1$ | |
| 01 | $Q^2 \rightarrow A^1$ | 26+32 |
| 01 | $A^2 \rightarrow Q^1$ | 26+32 |
| 02 | Comp $X^1$ to $X^2$ | |
| 03 | Part Add $X^2$ to $A^1$ | (24+26+32) (SR = 1) |
| 03 | Full Exit | 24-27 |
| 03 | Half Exit | 24-27 |
| 03 | Part. Add $X^2$ to $A^1$ | (25+27) (Dividend Sign = 1) |
| 04 | $Q^1 \rightarrow Q^2$ | |
| 05 | $Q^2 \rightarrow A^1$ | |
| 05 | $A^2 \rightarrow Q^1$ | |
| 07 | Part. Add $X^2$ to $A^1$ | SR = 1 |
| 07 | Exit to Round, Normalize, Final Ass'y. | 32+33 |

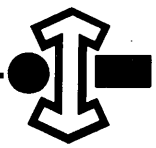


TABLE 2-8e. ITERATIVE SEQUENCE
Execute Floating Point; Instructions: 30-33

| | | |
|---|---|---|
| 00 | Inhibit $A^1 \rightarrow A^2$ | 30-33 |
| 01 | $X^2 \rightarrow X^1$ | 30+31 |
| 01 | Shift Right | +33 |
| 02 | $X^1 \rightarrow U^2$ | 30+31 |
| 03 | $U^2 \rightarrow U^1$ | 30+31 |
| 04 | Comp $X^1$ to $X^2$ | (32+33) (X Neg.) |
| 05 | $X^2 \rightarrow X^1$ | 32+33 |
| 05 | Clear X Exp. | 30+31 |
| 06 | $X^1 \rightarrow X^2$ | (30+31) (SR=0) |
| 06 | Comp $X^1$ to $X^2$ | (30+31) (SR=1) |
| 06 | $X^1 \rightarrow U^2$ Exit | 30+33 |
| 07 | $U^2 \rightarrow U^1$ | 32+33 |
| 07 | $X^2 \rightarrow X^1$ | 30+31 |
| 07 | Clear X Exp. | 32+33 |
| 07 | Clear S. R. FF | 30+31 |
| 08 | Add $R^1$ to $U^2$ | 30+31 |
| 08 | Comp $X^1$ to $X^2$ | 31+33 |
| 08 | $X^1 \rightarrow X^2$ | 32 |
| 09 | Clear X | 30+31 |
| 09 | Set $U^2$ S. R. | $U^2$ Neg. |
| 10 | $U^2 \rightarrow R^2$ | 30+31 |
| 10 | Clear A | (30+31) (SR=1) |
| 11 | $A^1 \rightarrow X^1$ | 30+31 |
| 11 | Clear $U^1$ | 30+31 ($U^2$ S. R. = 1) |
| 12 | Add $R^1$ to $U^2$ | 32+33 |
| 12 | $U^1 \rightarrow U^2$ | 30+31 |

TABLE 2-8e. (Continued)

| | | |
|---|---|---|
| 13 | $X^2 \rightarrow X^1$ | (30+31) ($U^2$ S. R. =0) |
| 13 | Clear $R^1$ | 32+33 |
| 14 | Part. Add $R^1$ to $U^2$ | 30+31 ($U^2$ S. R. =1) |
| 14 | $U^2 \rightarrow R^2$ | 33 |
| 14 | $I^2$ $I^3 \rightarrow R^1$ | 32+33 |
| 15 | Comp $R^2 \rightarrow R^1$ | (30+31)($U^2$ S. R. =1) |
| 15 | $R^2 \rightarrow R^1$ | (30+31)($U^2$ S. R. =0) |
| 15 | $X^2 \rightarrow A^1$ | (30+31)($U^2$ S. R. =1) |
| 15 | Mult. Step | 32 |
| 15 | Divide Step | 33 |
| 16 | $R^1 \rightarrow R^2$ | |
| 16 | $X^1 \rightarrow X^2$ | 30+31 |
| 20 | Initiate Shift | |
| | $R^2 \rightarrow R^1$ | 30+31 |
| | Reduce $R^1$ to $R^2$ | |
| | Shift One R ≠ 0 | |
| | Exit to 21 R = 0 | |
| | (Round, Normalize, Final Ass'y) | |

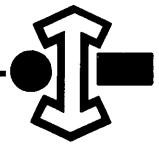


TABLE 2-8f. ITERATIVE SEQUENCE
Execute Round, Normalize, Final Assembly; Instructions (30+31)(R≠0)+32+33

| | | |
|---|---|---|
| 22 | Clear $R^1$ | |
| 23 | $U^2 \rightarrow U^1$ | |
| 23 | Set Execute Round FF | $(A^{47} \neq Q^{47})$ |
| 24 | $R^1 \rightarrow R^2$ | |
| 25 | Add $X^2$ to $A^1$ | 30+31 |
| 25 | Clear $X^1$ | |
| 25 | Comp. $R^2 \rightarrow R^1$ | |
| 25 | Set $X^2$ to 1 | A Neg |
| 26 | Comp. $X^1 \rightarrow X^2$ | A Pos |
| 26 | $X^1 \rightarrow X^2$ | |
| 26 | Part Add $R^1 \rightarrow U^2$ | (SR=1) (30+31+33) |
| 27 | $U^2 \rightarrow U^1$ | |
| 28 | $R^1 \rightarrow R^2$ | |
| 31 | Add $X^2$ to $A^1$ | $A^{47} \neq Q^{47}$ |
| 30 | Exit to Time 38 | A=0 |
| 33 | Left Shift | $A^{35}=A^{36}=A^{37}$ |
| 34 | Inhibit $A^1 \rightarrow A^2$ | $A^{37} \neq A^{36}$ |
| 35 | Right Shift | $A^{37} \neq A^{36}$ |
| 35 | Exit to Time 38 | $A^{37}=A^{36}$ |
| 37 | Clear $X^1$ | $A^{37} \neq A^{36}$ |
| 37 | Comp $R^2 \rightarrow R^1$ | $A^{37} \neq A^{36}$ |
| 41 | $A^1 \rightarrow X^1$ | |
| 41 | Set $X^1$ SR FF | A Neg |
| 41 | Clear $U^2$ SR FF | |
| 42 | Comp $X^1 \rightarrow X^2$ | A Neg |
| 42 | $X^1 \rightarrow X^2$ | A Pos |
| 43 | $X^2 \rightarrow X^1$ | |

TABLE 2-8f. (Continued)

| | | |
|---|---|---|
| 43 | Full Exit | |
| 43 | Half Exit | |
| 43 | Clear A | (30-33) (AQ≠0) |
| 44 | Add $R^1 \rightarrow U^2$ | AQ≠0 |
| 45 | $U^2$ Exp $\rightarrow X^1$ | AQ≠0 |
| 45 | Clear $X^1$ SR FF | AQ≠0 |
| 46 | Comp $X^1 \rightarrow X^2$ | ($X^1$ SR=0) (AQ≠0) |
| 46 | $X^1 \rightarrow X^2$ | ($X^1$ SR-1) (AQ≠0) |
| 47 | Part. Add $X^2 \rightarrow A^1$ | AQ≠0 |
| 47 | Set Exp. Fault FF | (AQ≠0) ($U^{707}$) |

## EXTERNAL FUNCTION SEQUENCE

The External Function (EF) sequence performs instruction 74, External Function. The value of the designator determines what type of operation is to be performed. When the designator is 0 (74.0), the instruction sends the external function select code, which is contained in the base execution address, to a specified external equipment or part of computer control. This code initiates a mode of operation in the equipment or part of control. Instructions 74.1 - 74.6 activate the buffer channel specified by the designator. This involves a storage reference on address 0000j to write the base execution address (the initial address of the buffer) in the upper address portion of the storage location.

Finally, a 74.7 instruction sends an external sense code, which is contained in the base execution address, to the specified equipment or part of computer control. The code designates a condition; to indicate the presence or absence of the condition a sense return signal is sent to the computer. The 74.7 instruction is also used to sense internal conditions of the computer such as faults.

The chain of control delays which make up the EF sequence is presented in simplified form by figure 2-21. Commands generated by this sequence appear in table 2-9. A complete examination of EF occurs in chapter 5.

TABLE 2-9. EXTERNAL FUNCTION SEQUENCE
Instructions; 74.0 - 74.7

| | | |
|---|---|---|
| 00 | $U^1 \rightarrow U^2$ | |
| 01 | Clear $X^1$ | |
| 02 | b to Aux Ref. Desig. | |
| 02 | $P^2 \rightarrow P^1$ | 74.1 - 74.6 |
| 02 | Initiate Storage | 74.1 - 74.6 |
| 03 | $U^2 \rightarrow X^1$ with ext. | |
| 04 | $X^1 \rightarrow X^2$ | 74.0 + 74.7 |
| 05 | Set Select FF | 74.0 |
| 05 | Set Sense FF | 74.7 |
| 05 | Set Wait Storage FF | 74.1 - 74.6 |
| 05 | Exit to Aux Seq. | 74.1 - 74.6 |
| 06 | Complement Exit FF | 74.0 + 74.7 |
| 12 | Full Exit | 74.0 + 74.7 |
| 12 | Half Exit | 74.0 + 74.7 |

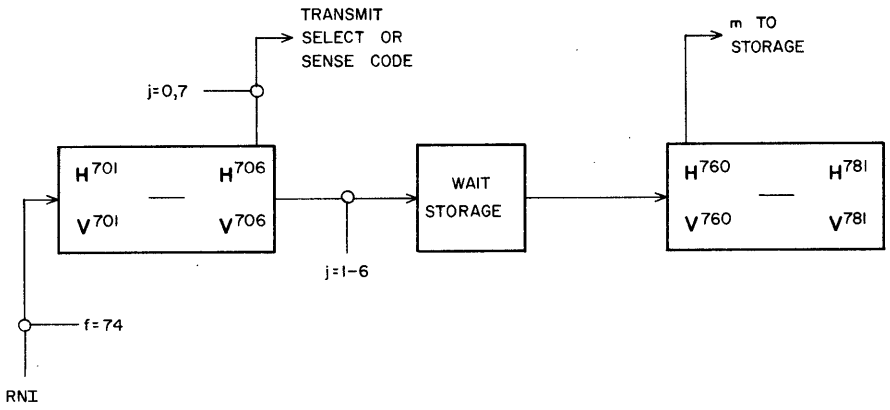



Figure 2-21. Basic Chain of Control Delays in EF Sequence.

AUXILIARY SEQUENCE

The Auxiliary sequence (AUX) is a relatively long one that performs three different operations: (1) buffering, which is the computer's exchange of a word with an external equipment via a buffer channel; (2) advancing the real time clock; and (3) initiating the interrupt routine or program upon receipt of an interrupt signal from an external equipment or part of computer control.

All three of these operations are characterized by the fact that the necessity for performing them does not arise (directly) from the execution of a computer program. These operations are not a direct part of the execution of any computer instruction. Thus the occurrence of one of these operations is relatively independent of the program of instructions being executed.

The basic chain of control delays for the AUX sequence is shown in figure 2-22. Buffer operations use all four segments; the advance-clock operation uses the first three segments; the interrupt operation uses only segments 1 and 4.

There are six buffer communication channels, which are entirely independent of one another and thus may be used simultaneously. The occurrence of a ready signal on an input buffer channel or a resume signal on an output channel constitutes a request for a buffer operation. Every 16 milliseconds a request signal is produced to indicate the need for advancing the real-time clock. An interrupt signal is sent by an external equipment or part of the computer control to indicate a somewhat unexpected need or demand for action by the computer. The occurrence of any of these three requests for operations performed by the Auxiliary sequence is denoted by the expression AUX request. There are thus eight possible sources of AUX requests: six for buffer and one each for interrupt and advancing the clock. (There are actually several interrupt request lines, but since they are combined in an OR function it is possible to consider them as one with respect to the use of the Auxiliary sequence.)

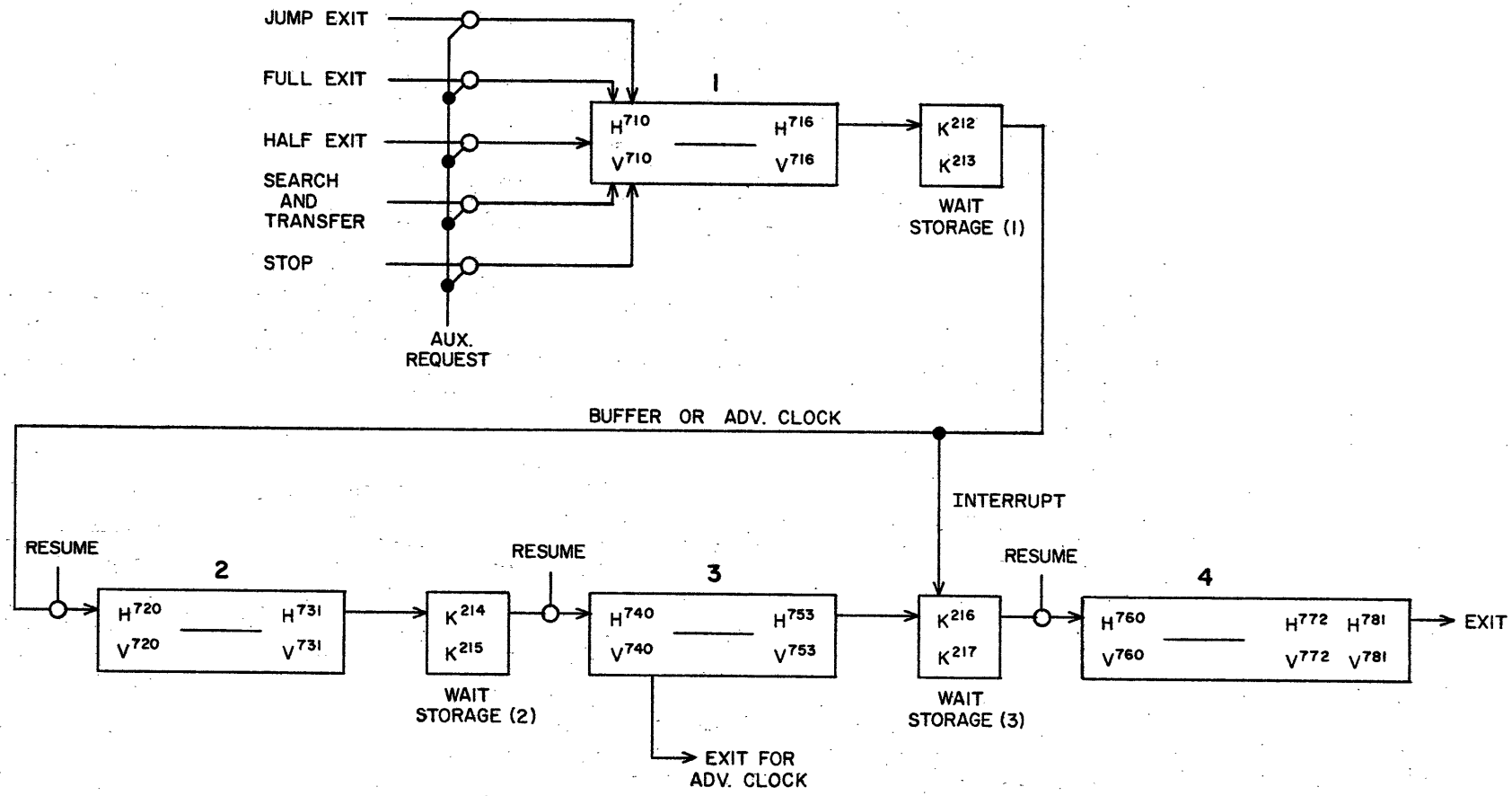



Figure 2-22. Auxiliary Sequence.

Since sources of AUX requests are independent and operate asynchronously with respect to one another, it is possible for more than one AUX request to occur at a time. Therefore, the order for acknowledging requests is established by a scanner that sequentially examines each of the sources of AUX requests, and assures that each source has equal priority. After examining one source, the remaining seven are examined before the first is re-examined. Each time that a request is present the scanner stops, and the operation appropriate to that AUX request is performed by the AUX sequence.

As shown in figure 2-23, the Auxiliary sequence is executed in response to an AUX request only at the completion of one of the instruction sequences and before the upcoming performance of RNI. Thus the performance of the AUX sequence occurs between an instruction sequence and the RNI sequence. If two or more AUX requests are present and they come from sources examined successively by the scanner, then the Auxiliary sequence is executed once for each such request before RNI occurs. From the foregoing it is seen that the execution of the Auxiliary sequence causes a brief suspension of the execution of the main program by the instruction and RNI sequences.

The Auxiliary sequence, the scanner and associated logical circuitry are taken up in Chapter 5. Table 2-10 lists the commands generated by this sequence.

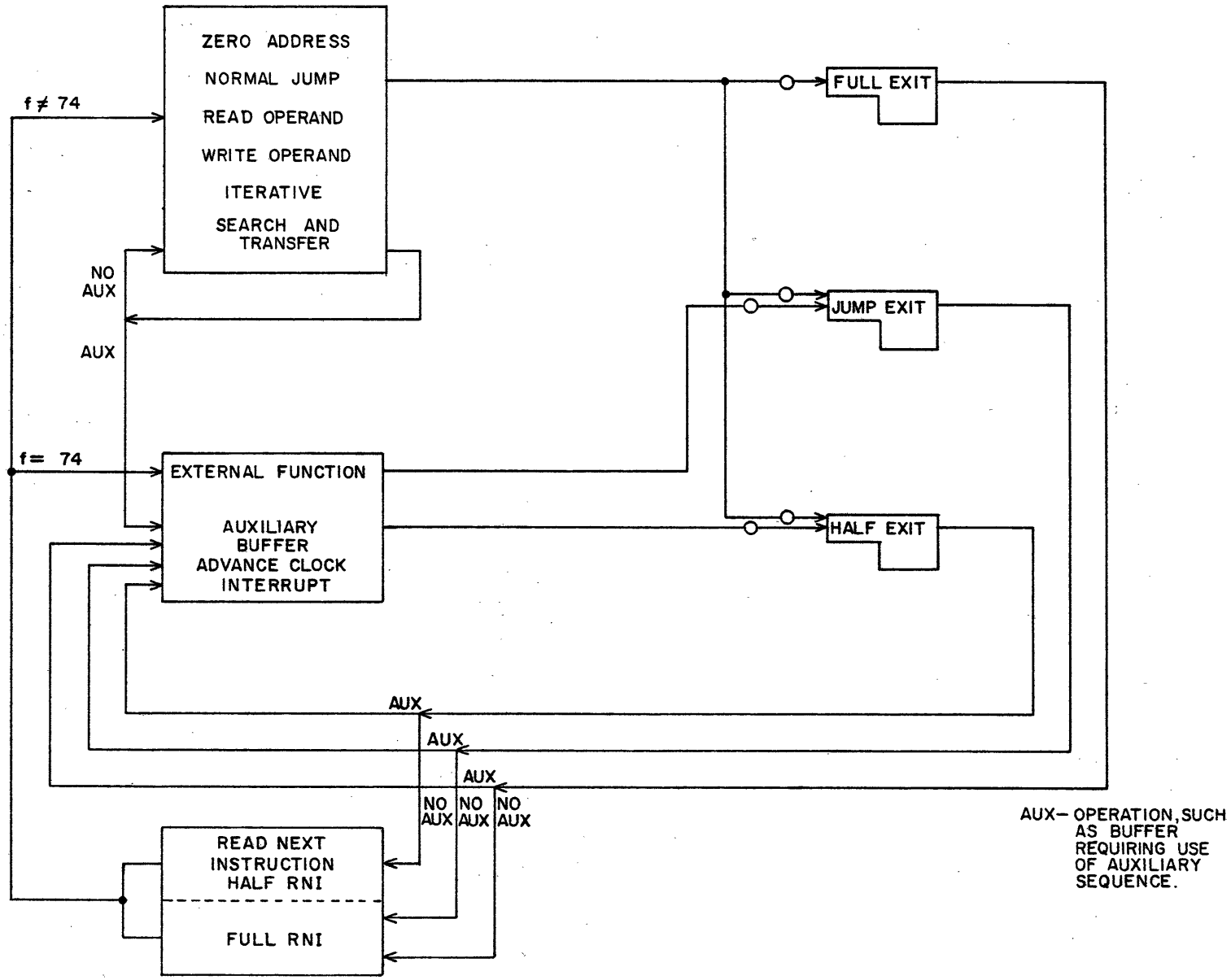



Figure 2-23. Relation of Auxiliary to Other Sequences.

TABLE 2-10a. AUXILIARY SEQUENCE
Buffer

| | | | | | |
|---|---|---|---|---|---|
| 00 | Initiate Storage | | 34 | Part. Add. $R^1$ to $U^2$ | |
| 01 | Set Aux. Ref. Desig. | | 35 | $U^2 \rightarrow X^1$ (with extension) | |
| 03 | Set Wait Storage I FF | | 36 | $X^1{}_L \rightarrow X^2{}_U$ | |
| 05 | Clear $R^1$ | | 37 | $X^2 \rightarrow X^1$ | |
| 07 | Clear $U^1{}_U$ | | 39 | Set Ready/Resume FF | $f \neq 74$ |
| 11 | $I^5\ I^6 \rightarrow X^1$ | | 39 | $U^2 \rightarrow U^1$ | |
| 11 | $I^5\ I^6 \rightarrow U^1{}_U$ | | 39 | $X^1 \rightarrow I^2$ | |
| 12 | Initiate Storage | | 40 | $X^1{}_L \rightarrow X^2{}_U$ | |
| 12 | $U^1 \rightarrow U^2$ | | 40 | Clear Buffer Request | |
| 17 | Clear $X^1$ | | 41 | $X^2 \rightarrow X^1$ | |
| 18 | Comp. $X^1$ to $X^2$ | In. Buf. | 43 | Clear $R^1$ | |
| 19 | $X^2 \rightarrow X^1$ | In. Buf. | 44 | $I^2\ I^3 \rightarrow R^1$ | |
| 20 | $U^2 \rightarrow R^2$ | | 46 | Part. Add. $R^1$ to $U^2$ | |
| 20 | $I^0 \rightarrow X^1$ | In. Buf. | 48 | $U^2 \rightarrow R^2$ | |
| 20 | Set Wait Storage II FF | | 48 | Set $R \neq 0$ FF | |
| 23 | $I^5\ I^6 \rightarrow X^1$ | Out. Buf. | 51 | Half Exit | |
| 24 | $X^1 \rightarrow X^2$ | Out. Buf. | 51 | Jump Exit | |
| 24 | Set Wait Storage III FF | | | | |
| 25 | Comp. $R^2 \rightarrow R^1$ | | | | |
| 25 | Clear $U^1{}_U$ | | | | |
| 26 | $R^1 \rightarrow R^2$ | | | | |
| 28 | Reduce $R^1$ to $R^2$ | | | | |
| 28 | Initiate Storage | | | | |
| 29 | Comp. $R^2$ to $R^1$ | | | | |
| 32 | $U^1 \rightarrow U^2$ | | | | |
| 33 | Clear $X^1$ | | | | |

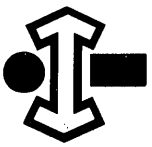


TABLE 2-10b. AUXILIARY SEQUENCE
Interrupt

| | | | |
|---|---|---|---|
| 00 | Init. Storage | 09 | Enable Partial Write Upper |
| 03 | Set Wait Storage III FF | 10 | $X^1_L \rightarrow X^2_U$ |
| 04 | $P^1$ to $X^2_L$ | 11 | $X^2 \rightarrow X^1$ |
| 05 | Set Interrupt Lockout FF | 11 | $I^5\ I^6 \rightarrow X^1$ |
| 05 | Clear $U^1_L$ | 15 | Half Exit |
| 06 | Set $P^1$ to 00007 | | |
| 07 | $X^2$ to $X^1$ | | |

TABLE 2-10c. AUXILIARY SEQUENCE
Advance Clock

| | | | |
|---|---|---|---|
| 00 | Initiate Storage | 17 | Add $X^2$ to $A^1$ |
| 01 | Set $S^1$ to 0 | 19 | $A^1 \rightarrow X^1$ |
| 03 | Clear $A^1$ | 20 | Initiate Storage |
| 03 | Set Wait Storage I FF | 20 | Set Wait Storage II FF |
| 05 | Clear $X^1$ | 21 | $Q^2 \rightarrow A^1$ |
| 05 | Set $X^2$ to 1 | 22 | $Q^1 \rightarrow Q^2$ |
| 06 | $Q^1 \rightarrow Q^2$ | 23 | $A^2 \rightarrow Q^1$ |
| 06 | $X^1 \rightarrow X^2$ | 23 | $Q^2 \rightarrow A^1$ |
| 07 | $A^2$ to $Q^1$ | 31 | Half Exit |
| 09 | Part Add. $X^2$ to $A^1$ | 31 | Full Exit |
| 11 | $I^5\ I^6 \rightarrow X^1$ | | |
| 12 | $X^1 \rightarrow X^2$ | | |

STATIC CONTROL

## BREAKPOINT ADDRESS

A digit switch assembly at the console provides for setting an address at which the program is to stop; this address is called the breakpoint. The breakpoint address is continually compared with **P**, the program address. When the two quantities are equal, the computer stops just before performing the upper instruction located at the breakpoint address (see figure 2-12); in other words, stopping occurs only during the full RNI. The breakpoint feature permits the program to be executed at high speed to a preset address of interest.

There are five digit switches in the switch assembly; each has eight positions numbered 0 through 7. Thus every possible combination of positions of the five wheels represents one of the 32,768 (decimal) storage addresses. The breakpoint is disabled by setting it to an address not used by the program. Addresses 00000 through 00006 are allocated for special uses which preclude their use as storage for instructions. Thus a convenient method of disabling the breakpoint is by setting it to one of these seven values.

## CONSOLE DISPLAY

The contents of operational registers are displayed in octal form at the console (figure 2-24) only when the computer is stopped. A display of the register contents during operation would be useless because of the rapidity with which such contents change.

Figure 2-25 shows the circuits involved in the display of the lowest octal digit (lowest three binary digits) of the A register. The digit display gate is turned on when the computer is stopped. Output amplifiers ($L^{500}$ thru $L^{502}$) sample the state of the A register. Each $L^{5--}$ energizes a relay if the associated bit is a "1". The three relays are interconnected

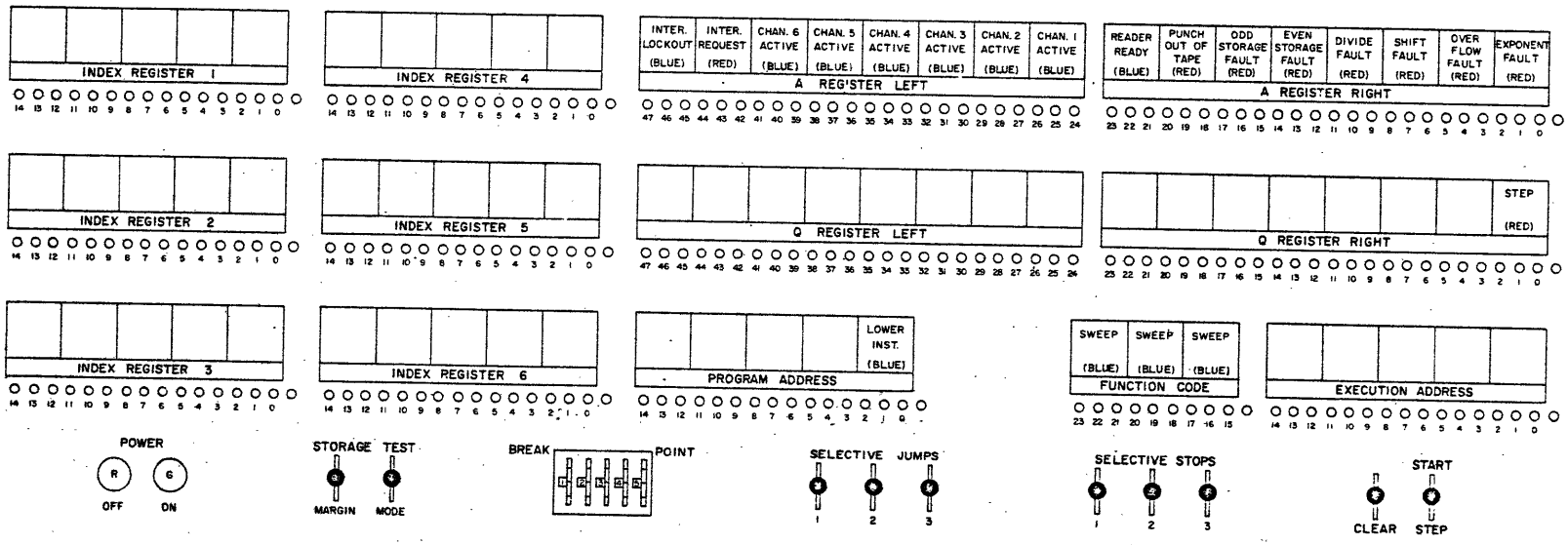



Figure 2-24. Console Display.

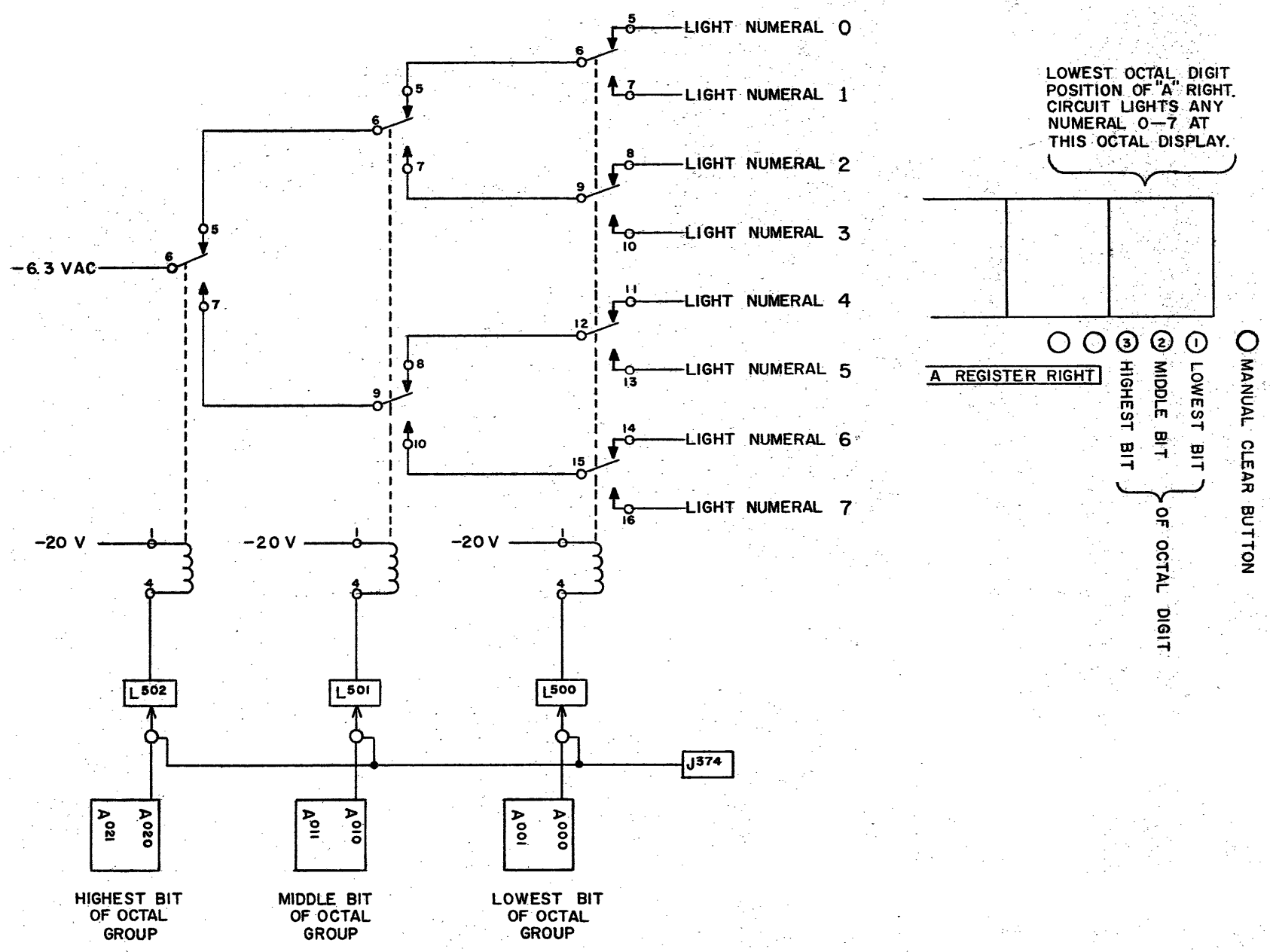
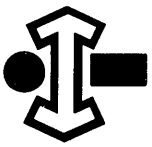



Figure 2-25. Typical Digit Display, Lowest Octal Digit of A.



to form a binary to octal translation; the output of the translator illuminates the appropriate octal digit at the associated position.

In addition to the digit display, the console provides visual indication of several other conditions in the computer and console input-output equipment. This form of indication is accomplished by means of the colored background lights at some of the lamp modules. In contrast with the digit display, the background lights can be illuminated at any time, including times when the computer is operating. Figure 2-24 shows the various background lights, the color used and the significance of each. With only a few exceptions the description of the significance of the light involves the name of the FF which must be set in order for the light to be illuminated.

The first of the exceptions is the blue Reader Ready digit, which is turned on when the Reader End-of-Tape FF is cleared. The red Punch-Out-of-Tape light is turned on by a switch at the punch; this switch is closed when the tape supply reel is nearly empty. The blue lower Instruction light is illuminated when the Exit FF is cleared, which is the case during the execution of the lower instruction.

RESYNCHRONIZING

Signals from external equipment and switches enter the computer via cables. Such signals must undergo special treatment in order to make them useable within the computer. Such special treatment is called Resynchronizing. One purpose of resynchronizing is to convert a signal which is asynchronous with respect to the timing of the computer to one that is timed by the computer clock. This involves insuring that only one synchronized pulse results from an asynchronous signal, regardless of the duration of such a signal. Another purpose of resynchronizing is to resolve runt pulses.

Resynchronizing Circuit

To accomplish the resynchronization of a signal, a special logical configuration, or resynchronizing circuit (figure 2-26), is used. Two kinds of resynchronizing pulses (referred to as sync pulses) are crucial to the operation of this circuit. Every 1.6 microseconds an odd sync pulse comes from $V^{021}$; likewise, every 1.6 microseconds an even sync pulse comes from $V^{020}$. The timing relation of the odd and even sync pulses is given in Figure 2-26. Note that: (1) the even sync pulse occurs first; and (2) the interval between an odd sync pulse and the succeeding even pulse is 1.4 microseconds.

To analyze the operation of the resynchronizing circuit, its state must be considered when no input signal is present; that is, the input to $M^{164}$ is -20 volts which at its output represents a "0". In this circumstance $K^{994}$ $K^{995}$ is cleared and $K^{996}$ $K^{997}$ is set.

Now the next odd sync pulse after $M^{164}$ receives a signal, $K^{994}$ $K^{995}$, is set. During the following even sync pulse, $H^{295}$ receives a pulse. The output of $V^{295}$ clears $K^{996}$ $K^{997}$ to prevent another pulse on the second even sync pulse. The input to $M^{164}$ must change to bring the circuit to its no input state. This prepares the circuit to act on the next signal from $M^{164}$.

The resync circuit thus performs the first objective of resynchronizing, namely, converting one occurrence of an asynchronous signal to one pulse timed by the computer.

Runt Pulses

A runt pulse is, specifically, an input to $M^{164}$ that at approximately 0.3 microseconds before an odd sync pulse has a value between -20 volts and 0 volts. In such a case the output of $M^{164}$ does not definitely indicate either a "1" or "0", but rather the voltage level of the output is somewhere between -3.0 volts and -0.5 volts respectively.

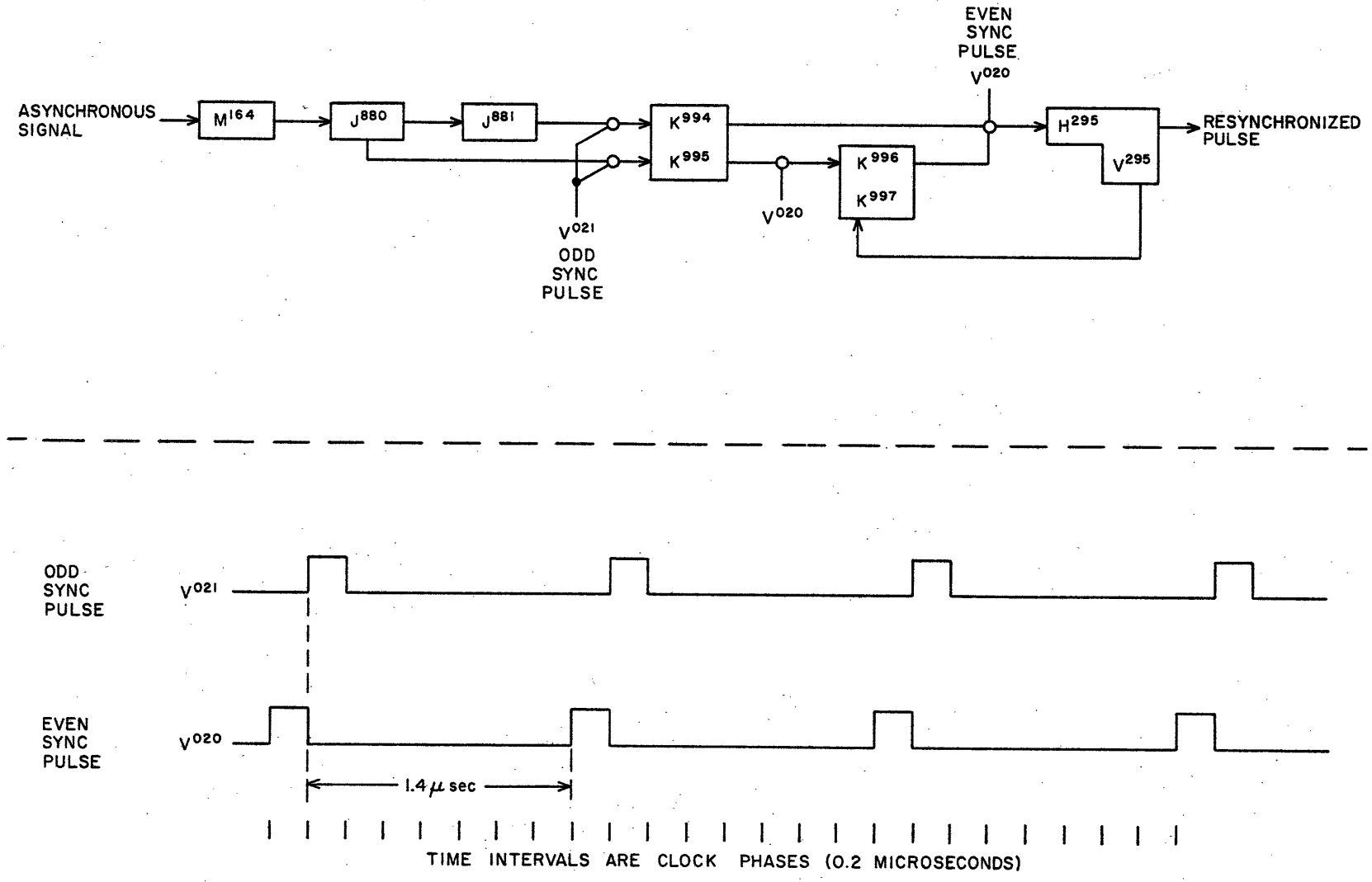



Figure 2-26. Typical Resynchronizing Circuit.

As an example, suppose that the output of $M^{164}$ is -1.5 volts. As a consequence, the level of the output from both $J^{880}$ and $J^{881}$ is also -1.5 volts approximately. If this is the case when the odd sync pulse from $V^{021}$ occurs, both the set and clear side inputs to $K^{994}$ $K^{995}$ receive half-sized inputs. In short, there is an attempt to both set and clear this FF with half-sized signals.

The behavior of a FF becomes indeterminate in this circumstance; it may finally be in the full "1" state or the full "0" state. But it will not be fully set or cleared in 0.2 microseconds, the normal switching time of a FF when a full 3.0 volts pulse is applied to one side or the other. Instead, when such runt pulses are applied to a FF, a much long period is required for the FF to settle into either the full "1" state or the full "0" state. The maximum period required for such settling down is 1.4 microseconds, which is the interval between the odd sync pulse (when runt pulses may be applied to $K^{994}$ $K^{995}$) and the even sync pulse (when the state of this FF is sampled at the AND input to $H^{295}$). This interval is the resolution time of a FF. Only when the runt input is resolved into a full "1", does $H^{295}$ receive an input pulse.

It now becomes apparent that the purpose of the odd and even sync pulses is to provide for the resolution of runt pulses in the first FF of the resynchronizing circuit. The odd sync pulse sets the time for sampling the asynchronous signal, and the even sync pulse sets the time for sampling the first FF of the circuit.

Resynchronizing Counter And Pulses

The sync pulses used in resynchronizing circuits throughout the computer are produced by a two-stage counter (figure 2-27). This counter is advanced every even clock phase, that is, every 0.4 microseconds. Each time the counter reaches three (11 in binary), $H^{020}$ receives a pulse. This pulse produces even sync pulses which are distributed throughout

the computer by $V^{020}$, $V^{022}$, $V^{024}$, and $V^{026}$. In addition, $V^{020}$ sends a pulse to $H^{021}$. This causes the odd sync pulses that are distributed by $V^{021}$, $V^{023}$, $V^{025}$, and $V^{027}$.

The counter returns to 0 (00 in binary) on the next even clock phase after reaching the count of three. A period of 1.6 microseconds elapses between the times when the counter holds the count of three. Thus the rate of the sync pulses is one each 1.6 microseconds; that is, for example, even sync pulses occur every 1.6 microseconds. As the timing chart on figure 2-26 shows, the interval between an odd sync pulse and the succeeding even sync pulse is 1.4 microseconds.

Even and odd sync pulses used in control logic of the console input-output equipment are obtained from control delays involving $H^{150}$ and $H^{151}$.

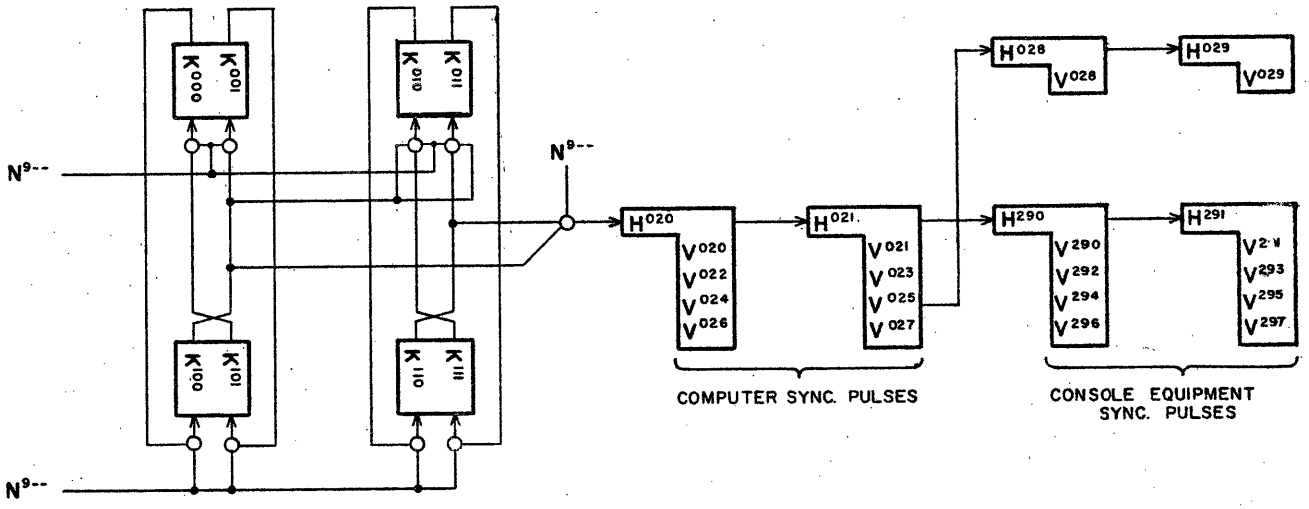



Figure 2-27. Resync Counter and Pulse Distribution.

## JUMP INSTRUCTIONS

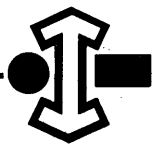
The instructions which accomplish program jumps or stop program execution are 22, 23, 55, 75, and 76. All of these involve jumping, and 76 also provides for stopping. Instructions 22, 23, 75, and 76 with j = 0-3 and instruction 55 with all values of j cause a normal jump. Instructions 22, 23, 75, and 76 with j = 4-7 cause a return jump.

When the jump condition is satisfied, the execution of a jump instruction causes the termination of the current program of sequential instructions and initiates a new sequence of instructions at the storage location given by the execution address of the jump instruction. Whereas a normal jump instruction accomplishes only this, a return jump instruction in addition prepares for the return to the original instruction sequence upon completion of the new sequence. For both normal and return jump instructions, when the jump is conditional and the condition is not satisfied, the instruction immediately after the jump in the original sequence is executed in a normal manner.

A jump instruction may appear in either the upper or the lower position of an instruction word. When it appears as the upper instruction and the jump is taken, the lower instruction is never executed. This is true, also, for return jumps.

### NORMAL JUMP

An instruction involving a nomal jump is executed by the Normal Jump sequence, with the exception of instruction 55, which is executed by the Zero Address sequence. For conditional jumps an examination is first made to determine if the condition is satisfied. If the jump is to be taken then the unmodified m portion of the jump instruction is transmitted from $U^2$ to $P^1$. The jump exit is then taken to the RNI sequence. Entering RNI by the jump exit differs from entering by the full exit only in that P is not advanced before the $P^1$ to $S^1$ or $S^2$ transmission. Thus the execution of RNI following a jump that is executed reads



the instruction from the storage location given by m of the jump instruction.

If the jump condition is not met, then a half or full exit, depending upon whether the jump instruction occupied the upper or lower positions, respectively, is taken to RNI.

RETURN JUMP

A return jump instruction is always executed by the Write Operand sequence. Table 2-11 shows a typical program situation in which such an instruction is used. Pertinent, relative storage addresses and the instructions contained in them are given for both the main program and the subroutine to which the return jump leads.

Suppose that the upper instruction at relative address c has been executed and, furthermore, that the instruction word in c + 1 has been read. The left instruction in c + 1 is a return jump which is to be performed if A $\neq$ 0. Suppose that this condition exists. The resultant execution of the return jump involves four basic steps:

Step 1. Advance P to c + 2.

Step 2. Initiate at address d a storage reference which reads the contents of d into $U^1$ and writes the contents of P, that is, c + 2, in the m portion of the upper instruction at d.

Step 3. Transmit d from $U^2$ to P.

Step 4. Half exit.

The first step yields the address of the next instruction of the main program. It is the return to this address that the instruction prepares for. Step 2 accomplishes two things: 1) it reads the pair of instructions at address d, of which the lower is the first to be executed in the subroutine; 2) it stores the quantity c + 2 as an address in the upper instruction at location d. Step 3, by entering the quantity d in P, prepares for the sequential execution of the instructions in the subroutine. Finally, the half exit to RNI in step 4 results in the

TABLE 2-11. TYPICAL USE OF RETURN JUMP INSTRUCTION

A. Main Program

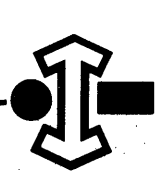
| Relative Address | Instruction | |
|---|---|---|
| c | Upper | Lower |
| c + 1 | f b m | f b m |
| c + 2 | 22 5 d | f b m |
| | f b m | f b m |

B. Subroutine

| | | |
|---|---|---|
| d | 75 0 c + 2 | f b m |
| - | | |
| - | | |
| - | | |
| d + n | f b m | 75 0 d |

subsequent execution of the lower instruction in address d. The upper instruction at d will be executed in the actual return to the main program after completion of the subroutine.

After step 4 the return jump instruction is completed and the subroutine commences with the lower instruction at d. When the upper instruction at d + n has been executed the basic function of the subroutine is accomplished. It then remains to re-enter the main program. The re-entrance begins with the normal jump in the lower part of d + n; this instruction enters the quantity d in the P register and initiates RNI through the jump exit. Consequently, both instructions in address d are entered in $U^1$ and the upper one is now executed. This instruction is the one which received as its execution address the address c + 2. Since this instruction is another unconditional normal jump, the quantity c + 2 is entered in P and RNI is initiated through the jump exit. With the execution of the upper instruction in c + 2, the main program is resumed.



## SKIP INSTRUCTIONS

There are several instructions, namely, 36, 37, 54, 64-67, and 74.7, which provide for skipping the next instruction of the program when a certain condition is met. These instructions are limited to use in the upper position of an instruction word, since they provide for skipping the lower instruction. Skipping is accomplished by using the full exit when the condition is met. If this condition is not met a half exit is used to return to RNI.

The 74.7 instruction (External Function Sense) may also be employed as a lower instruction However, in this case it is no longer a skip instruction; rather it is used to cause an indefinite period of waiting until the specified condition is met. The half exit is used and thus the 74.7 is simply repeated until the condition is met; then a full exit is taken. Further details on 74.7 are found in chapter 5.

## COMPUTER OPERATING CONTROLS

The logical networks associated with the switch on the lower center panel of the console (figure 2-24) are discussed in the following paragraphs. The switches are:

Start - Step

Clear

Selective Jump

Selective Stop

Storage Test

### START-STEP SWITCH

The START-STEP switch selects the mode of operation of the computer. Both the up and down positions are momentary. The up or START position selects the high-speed mode in

which a program of instructions as well as auxiliary operations proceeds until completed, or until program stop occurs. The down or STEP position selects a mode in which a single instruction is executed; operation then stops to await further manual selection. Thus one instruction is executed per depression of the switch to STEP position. Operation of the START-STEP switch is ineffective so long as any buffer channel is active.

The STEP position has priority over the START position. This means that if the computer is operating high-speed as a result of previously selecting START, then depressing the switch to STEP causes operation to stop. Selecting STEP removes the previous effect of switch selection of a mode made by the START position. All the effects of selecting either the START or STEP postion of the switch are felt at the RNI sequence (figure 2-28). The Start, Step, and Neutral contacts of the switch are connected to a resynchronizing circuit which converts the d-c levels produced by closing these contacts to single pulses.

Either a Start or Step pulse causes the digit display to be turned off. The pulse is then delayed 16.6 milliseconds before being applied to RNI. This delay allows transients, which result from turning off the lights, to settle before computer operation begins. In analyzing the effect of a Start or Step pulse, it is necessary to distinguish between a pulse following immediately after master clear and one that does not follow master clear. A master clear, in addition to clearing registers and control FFs, sets the Initial Start FF. Figure 2-28 shows that a start pulse following master clear:

Initiates a full RNI (at $H^{090}$)

Clears Stop II

Begins execution of program at high speed

A step pulse following a master clear:

Initiates a full RNI (at $H^{090}$)

Sets Stop I

Sets Stop II from Stop I (by $V^{096}$)

Halts RNI after $V^{098}$

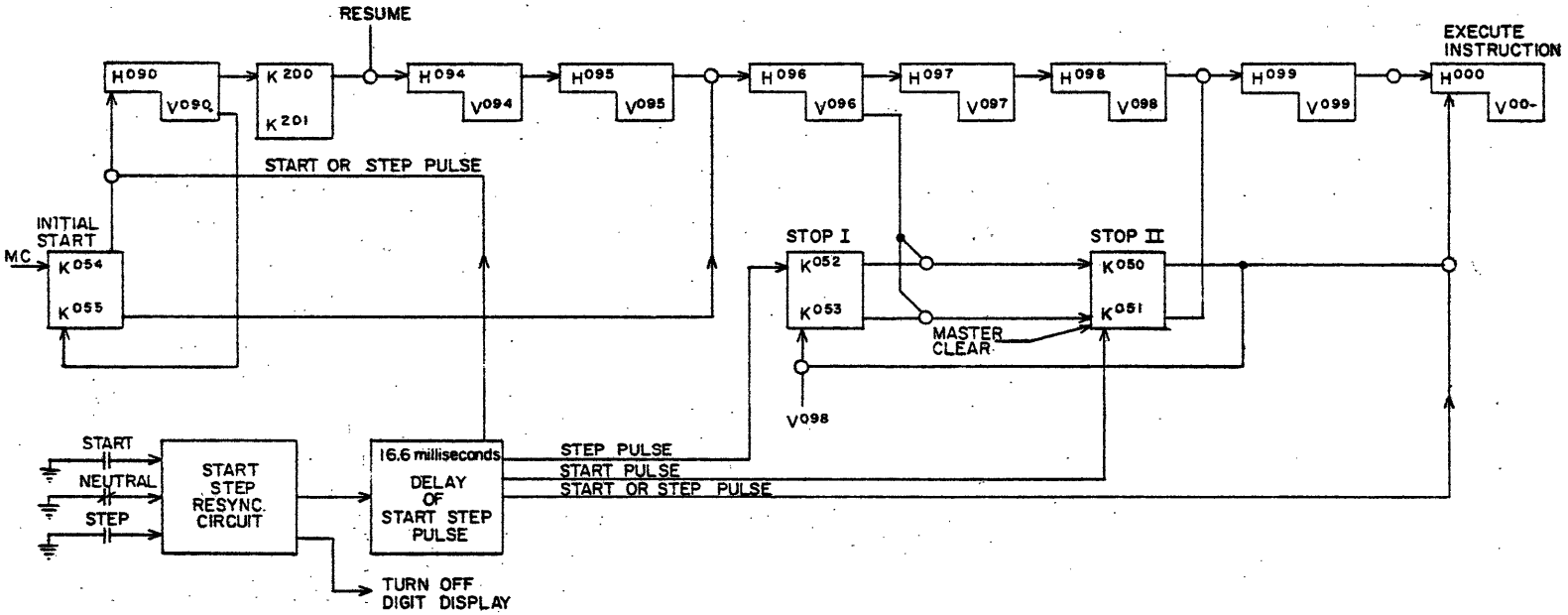



Figure 2-28. Connection of Start-Step Switch to RNI.

This pulse reads a pair of instructions. Another step pulse is necessary to execute the first instruction.

A step pulse not preceded by a master clear:

Sets Stop I

Initiates execution of instruction (at $H^{000}$)

Sets Stop II from Stop I during following RNI

Halts following RNI at $V^{098}$ because of Stop II

A start pulse not preceded by a master clear:

Clears Stop II

Initiates execution of instruction (at $H^{000}$)

The program execution continues since Stop II is cleared and remains cleared.

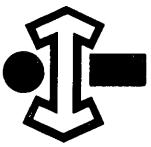
CLEAR SWITCH

The CLEAR switch at the console provides for master clearing the computer (down position) and the external equipment connected to the computer (up position). The purpose of either a computer master clear or an external master clear is to completely erase all traces of the previous mode of operation and thereby to prepare for an entirely new mode of operation. Either type of master clear may occur at any time the operator wishes; regardless of whether the equipment is operating or stopped, or of the mode of operation, the master clear takes effect immediately.

COMPUTER MASTER CLEAR

Placing the CLEAR switch down effects a computer master clear by:

1) forcing all stages of the registers displayed at the console indicator panel to the "0" state



2) forcing a great number of control FFs throughout the computer to the "0" state
3) setting the Initial Start FF ($K^{054}$ $K^{055}$) to "1"

As a consequence operation stops, if the computer was running. Regardless of whether or not the computer was running, the master clear prepares for the resumption of operation at the operator's discretion. The operator must select the mode of operation by manually entering quantities in the registers provided with set buttons at the indicator panel. A computer master clear does not alter any quantity placed in core storage before the clear.

It must be emphasized that it is not the case that every FF in the computer is cleared by a master clear; it does not clear the secondary registers and many control FFs.

The master clear is static. Thus, so long as the CLEAR switch is held down the FFs affected by master clear are forced to the "0" state.

EXTERNAL MASTER CLEAR

When the CLEAR switch is placed in the up position, a master clear signal is sent out to all the equipments connected to the computer. An external master clear is not selective; it affects every equipment. Within each equipment the master clear signal forces the critical registers and control FFs to the "0" state. (A few exceptional FFs may be set to "1" by the master clear.) All operation stops in the equipment upon receipt of the clear signal. Following the master clear, each equipment is ready for a mode of operation entirely unrelated to its previous mode. However, an external master clear does not in any way alter the information recorded on the storage medium of the external equipment.

SELECTIVE JUMP SWITCHES

The three SELECTIVE JUMP switches provide the manual conditions for performance of normal jumps for instruction 75 with j = 1-3 and return jumps for instruction 75 with j = 5-7. The connection of these switches to the logical circuitry of the computer is shown in figure 2-29. Each switch has a resynchronizing circuit. The outputs of all three circuits are sampled at the inputs to $F^{910}$. The switches lock in up position and are momentary in down position. While either position meets the manual condition for jumping, only the up (locking) position is used.

SELECTIVE STOP SWITCHES

The three SELECTIVE STOP switches function for instruction 76 in a manner similar to the Selective Jump switches for instruction 75. The stopping of operation by execution of 75 and b = 1-3, 5-6, is conditioned by the switch positions. Three resynchronizing circuits are associated with the switches, and the circuit outputs are combined with the translations of j associated with the switch. The switches lock in up position and are momentary in

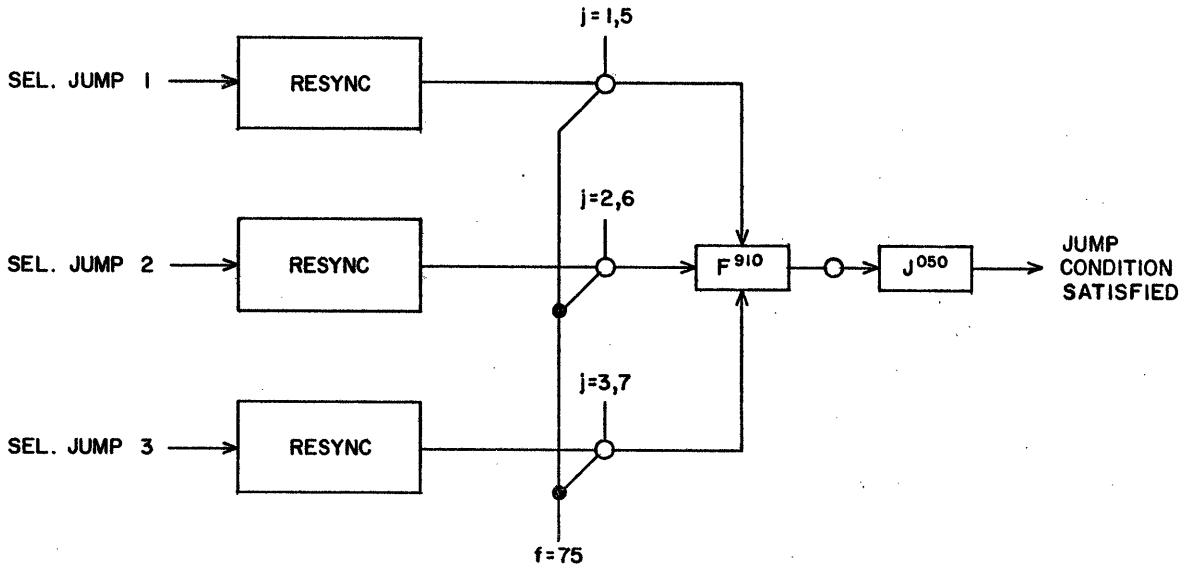
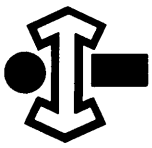



Figure 2-29. Sampling Selective Jump Conditions.



down position. Either position meets the manual condition for jumping, but ordinarily only the up position is used.

STORAGE TEST SWITCHES

There are two STORAGE TEST switches, MODE and MARGIN. The switches lock in both up and down positions. Normal operation occurs when the switches are in the neutral position.

The MODE switch in up position provides for repeatedly reading and executing the same pair of instructions. This is accomplished by disabling the advance P command which is normally initiated by the performance of a full RNI.

In down position the MODE switch provides for sweeping through (successively) all the addresses in storage. Figure 2-12 shows that in this type of operation instructions are not executed; rather, only the RNI sequence is performed. A full RNI, which initiates a storage reference, is performed and followed immediately by a half RNI, which does not initiate a storage reference. Another full RNI follows immediately upon the half RNI. Since this next full RNI advances P, the storage address referred to is one greater than the address referred to in the preceding full RNI.

Note that the rate at which storage references are initiated in the sweep mode is fixed in the case of high speed operation. Sweeping through storage may also be accomplished by stepping. In this case the operator can view the content of each address as it is read. The upper 24 bits of the word at the address appear visually in the program control register at the console when the full RNI is performed. The succeeding half RNI transfers the lower 24 bits into the program control register for visual inspection.

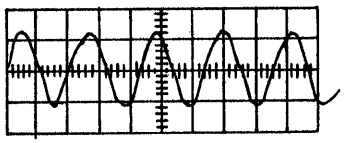
# MASTER CLOCK

The master clock provides the timing pulses used throughout the computer. Directly or indirectly the timing of all signals is determined by this clock. The master clock consists of a system of over 100 interconnected oscillators (each contained on a type 01 card). In addition the system sometimes employs single inverters as slaves to provide additional outputs from the oscillators.

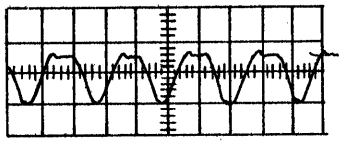
Each oscillator operates at 2.5 megacycles. There are ten sine wave outputs provided by an oscillator. Five of the outputs are 180 degrees out of phase with the remaining five. One set of outputs is designated as "even" and the other set as "odd" (the reason for this designation is given later). The circuits which receive the sine wave outputs of the oscillators convert them into rectangular waves. The even and odd outputs along with the rectangular waves produced from them are shown in Figure 2-30. The asymetrical form of the rectangular wave is due to the bias used in clipping the sine wave peaks. Only the negative or smaller portion of the rectangular wave is effective in gating. Thus the clock pulses used in the computer are 0.2 microseconds in duration.

The elements or circuits of the master clock appear in the File of Equations. Oscillator circuits are denoted by symbols of the form $C^{---}$. Each oscillator has two such symbols associated with it, for example, $C^{220}$ and $C^{221}$. The two symbols always have consecutive superscripts. The smaller superscript has an even digit in the third position. The even symbol, $C^{220}$ in this case, denotes the part of the circuit concerned with clock pulses of one phase, which is called "even". The odd symbol, such as $C^{221}$, denotes the other part of the circuit providing clock pulses of the opposite phase, which is called "odd".

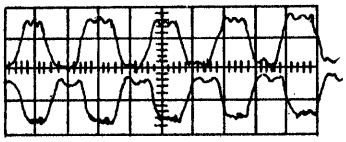
The first digit of a $C^{---}$ symbol indicates the chassis on which the oscillator is located. Thus, $C^{220}$ appears on chassis 02. The even part of each oscillator in the computer is



A. OSCILLATOR TEST POINT
(VERTICAL: 5v/cm;
SWEEP: 0.2 μsec/cm)



B. OSCILLATOR OUTPUT PIN
(VERTICAL: 5v/cm.;
SWEEP: 0.2 μsec/cm)



C. COMPARISON OF PHASES AT THE
OUTPUT OF SLAVE INVERTER
(VERTICAL: 2v/cm.;
SWEEP: 0.2 μsec/cm)

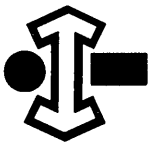
Figure 2-30. Master Clock Oscillation Waveforms.

connected to the even part of every other oscillator. Similar connections exist between the odd parts of the oscillators. These interconnections of the oscillators insure synchronization of all of them.

Each chassis has one card type 00 on it. This card connects the system of oscillator cards on that chassis to the oscillators on the remaining seven chassis. The 00 card thus functions as a switch. When it is inserted in its jack it closes the connection of the oscillators on its chassis and those on the other chassis. When the 00 card is removed from its jack it opens this connection. This facilitates maintenance of the master clock.

Since the outputs of the oscillator draw some current, it is desirable to load the oscillator as symmetrically as possible. Thus an attempt is made to use nearly as many even outputs as odd outputs from an oscillator. In cases where a great many outputs of one phase are required, symmetrical loading of the oscillator is maintained by the use of single inverter slaves. The slaves in effect increase the available outputs. Such single inverters are denoted by $N^{9--}$ symbols. Suppose, for example, that additional requirements for odd clock pulses have arisen. They can be supplied by a single inverter, for example, $N^{953}$, which receives an input from an odd part of an oscillator, say $C^{221}$. The odd output of $C^{221}$ is inverted by $N^{953}$ which then provides up to six odd outputs.

The master clock begins operation as soon as power is applied to the central computer. Operation of the clock continues so long as power is applied, regardless of whether computation is going on.



# CHAPTER 3

# ARITHMETIC SECTION

The arithmetic section of the 1604 central computer is composed of the three registers A, Q and X. Associated logic circuitry provides for the entry of the registers into the operation of the computer or tests the condition of the registers' contents. The arithmetic section (figure 3-1) also borrows circuits of the control section to aid in the performance of some arithmetic operations.

The procedures followed in the arithmetic section are derived from the fundamental logic processes, and the arithmetic is performed according to the rules of one's complement binary arithmetic. Although this section of the computer is directly concerned with all arithmetic, elements of it are also used in a number of operations not directly associated with arithmetic.

This chapter describes registers A, Q and X and the operations in which they are involved. In some instances entire instructions are analyzed to demonstrate how various parts of the arithmetic section are used.

## ARITHMETIC REGISTERS

### ACCUMULATOR

The principal arithmetic register is the Accumulator (A register), so called because one of its contributions to computer operation is to form the results of arithmetic operations by the process of accumulation. Some of the more important functions of A are:

1) Arithmetic operation - A initially holds one of the operands in addition, subtraction, multiplication and division. The result is usually held in A.

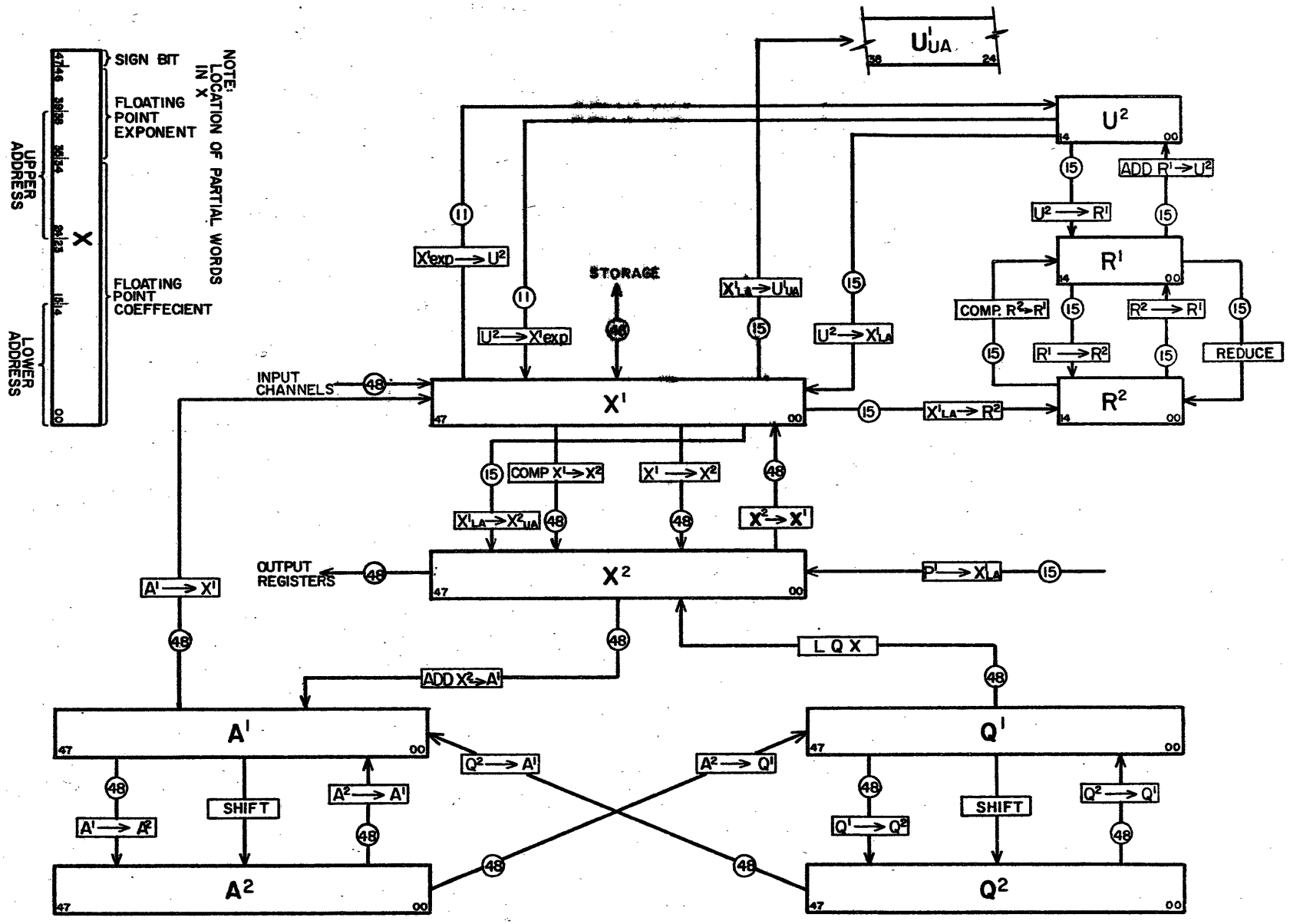
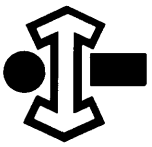



Figure 3-1. Over-all Block Diagram of Arithmetic Section



2) Shifting - A may be shifted separately or in conjunction with Q to the right or left. Right shifting is open-ended with lowest bits discarded and sign extended.
3) Control for conditional instructions - A holds the word which conditions jumps and search instructions.

The A register is composed of two major functional parts: 1) two ranks ($A^1$ and $A^2$) of 48 flip-flops each, which provide for storage of words; and 2) the borrow pyramid, a network for sensing and generating the borrows required in subtracting the complement of X, that is, $X^1$, from A. Figure 3-2 shows a typical stage of A. The upper FF is in rank $A^1$ and the lower in rank $A^2$.

Normally, rank $A^1$ is permitted to follow rank $A^2$ unconditionally. The signal $A^1 \rightarrow A^2$ is produced each phase time as long as the separate storage facilities of $A^2$ are not required. Thus, for example, the commands Add $X^2$ to $A^1$ and $Q^2 \rightarrow A^1$ reference $A^2$ as well, duplicating in $A^2$ the data transferred to $A^1$. Similarly, the command Clear A is applied only to $A^1$, but its effect is felt by $A^2$. $A^2$ receives the next higher or lower bits in the right or left shift operations and supplies the quantity for the transfer $A \rightarrow Q$.

Only the Q and X registers communicate with A (figure 3-1). The transmission from Q to A is accomplished in the ordinary manner. Transmission of $X^2$ to $A^1$ makes use of the add path; $A^1$ is first cleared, then $X^2$ is added to $A^1$.

Q REGISTER

The Q register is the auxiliary arithmetic register. It is composed of 48 stages of double-rank FF storage, and associated circuitry. The two ranks operate independently of each other. The signals causing transmission between them are generated conditionally. The

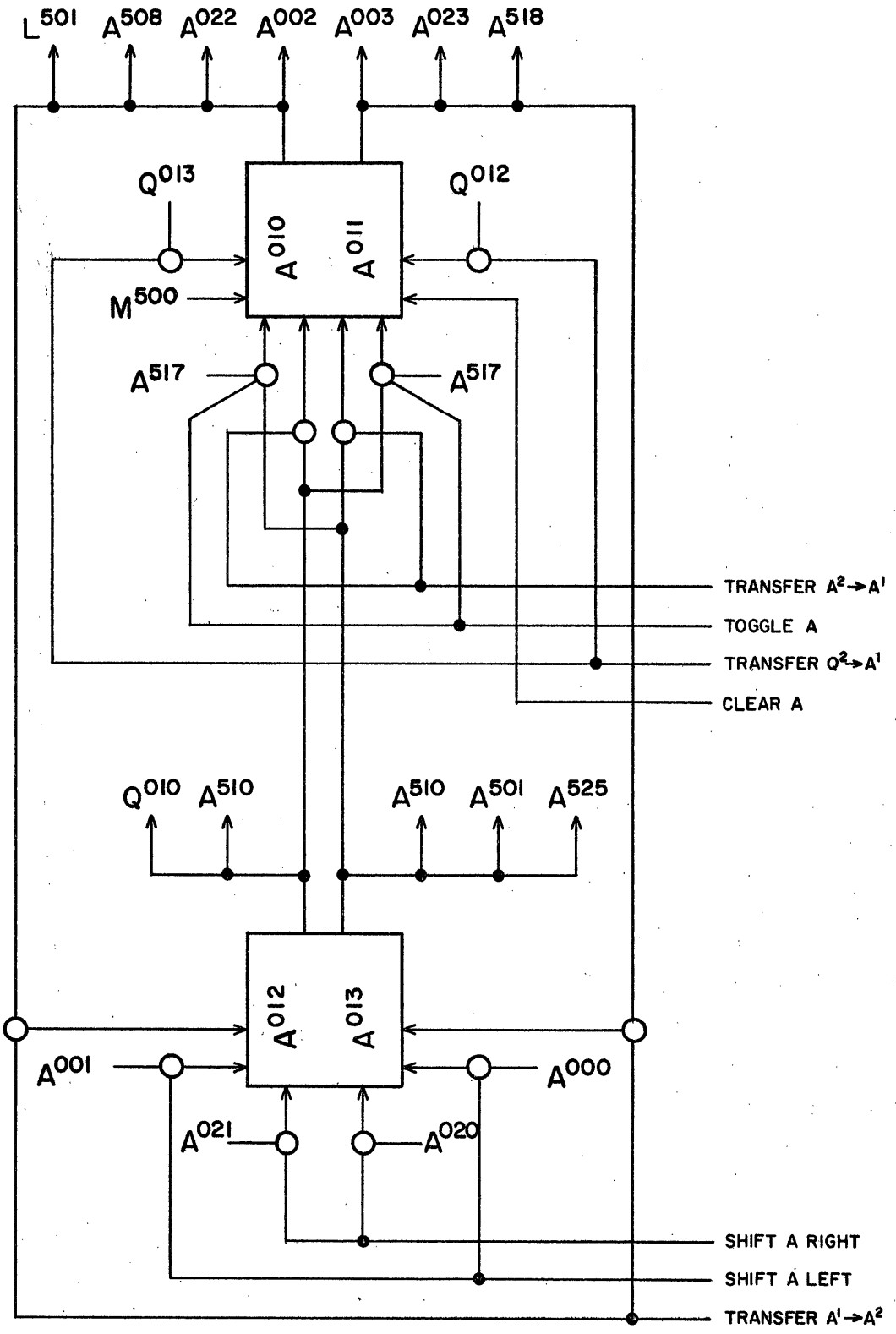
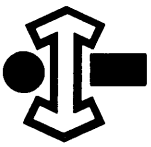



Figure 3-2. Typical Stage of A Register.



principal functions of Q are:

1) providing temporary storage of contents of A
2) forming a double-length register, AQ or QA
3) shifting to the right or left, separately or in conjunction with A
4) participating in multiplication, division and logical product operations (masking).

As shown in figure 3-1, Q communicates directly with the A register only. In the formation of logical products the clear outputs of $Q^1$ are transmitted to clear inputs of $X^1$.

X REGISTER

The Exchange, or X, register is the communication center of the computer. All internal transmissions between the arithmetic section or the input-output section and the rest of the computer are made through X.

The X Register is composed of 48 stages of double-rank FF storage and associated circuitry. Communication between the ranks is dependent upon the conditional generation of the transfer $X^1 \rightarrow X^2$ or $X^2 \rightarrow X^1$ commands. The principal uses of and operations involving the X register are:

1) participating in arithmetic operations
2) complementing
3) formation of logical products
4) assembly and disassembly of floating point words
5) communication between various sections of the computer (see figure 3-1)

## CONTROL REGISTERS USED IN ARITHMETIC OPERATIONS

Certain arithmetic operations borrow elements of the control section to effect their completion. In the case of the multiply, divide and shift instructions a control quantity is placed in R to govern the operation; and in the case of the floating-point instruction use is made of the arithmetic qualities of the U and R registers.

Multiplication and division operations proceed as repetitions of a two-part step, which consists of: (1) an addition (for multiplication) or a subtraction (for division); and (2) shifting. The number of repetitions of the step is controlled by the R register, and in preparation for the operation, R is preset to indicate the number of repetitions. After each step is performed R is reduced by one; when R=0, the operation concludes.

# BASIC OPERATIONS

## BINARY ARITHMETIC

The binary number system is the basis for the representation and manipulation of all information within the computer. Only two digits, "0" and "1", are used in this system. This characteristic is fundamental to computer operation since it permits the assignment of a pair of conditions, in this case voltage levels, to be sufficient to encode data presented in binary form. A voltage of -3.0v represents a "1"; a voltage of -0.5v represents a "0".

A binary number uses the digit 2 as its basis of notation (radix) in the same manner that a decimal number uses 10. To illustrate, the decimal number 653 breaks down as follows:

$$6 \times 10^2 + 5 \times 10^1 + 3 \times 10^0 = 600 + 50 + 3$$

Similarly, a binary number such as 1011 can be analyzed:

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1$$

which is equivalent to decimal 11.

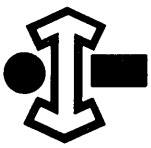


Table 3-1 lists the decimal numbers 0 through 16 and their binary equivalents.

TABLE 3-1. DECIMAL AND BINARY EQUIVALENTS

| DECIMAL | BINARY | DECIMAL | BINARY |
|---|---|---|---|
| 0 | 00000 | 9 | 01001 |
| 1 | 00001 | 10 | 01010 |
| 2 | 00010 | 11 | 01011 |
| 3 | 00011 | 12 | 01100 |
| 4 | 00100 | 13 | 01101 |
| 5 | 00101 | 14 | 01110 |
| 6 | 00110 | 15 | 01111 |
| 7 | 00111 | 16 | 10000 |
| 8 | 01000 | | |

Binary numbers are added together according to the following rules:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 with a carry of 1

The addition of two binary numbers proceeds as follows (the decimal equivalents verify the result):

| | | |
|---|---|---|
| Augend | 0111 | (7) |
| Addend | +0100 | +(4) |
| Partial Sum | 0011 | |
| Carry | 1 | |
| Sum | 1011 | (11) |

Subtraction may be performed as an addition. These decimal examples illustrate this method:

| | | | |
|---|---|---|---|
| 8 | (minuend) | or | 8 (minuend) |
| -6 | (subtrahend) | | +4 (10's complement of subtrahend) |
| 2 | (difference) | | 2 (difference - omit carry) |

The second method shows subtraction performed by the "adding the complement" method. This process is proved in the following identities:

$$8 - 6 =$$

$$8 + (10 - 6) - 10 =$$

$$8 + 4 - 10$$

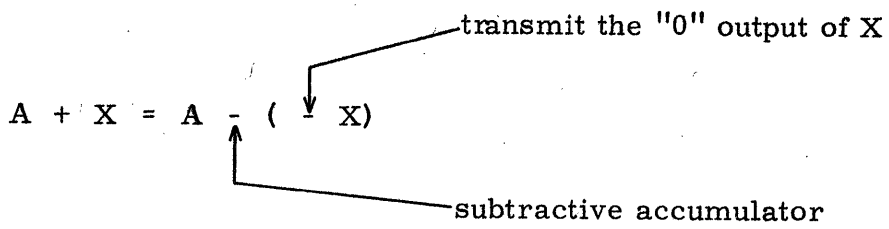
The omission of the carry in the illustration has the effect of reducing the result by 10.

A method of complementing a binary number, known as the one's complement, is formed by subtracting each bit of the number from 1. For example:

| | |
|---|---|
| 1111 | |
| -1001 | (9) |
| 0110 | (one's complement of 9) |

The one's complement of a binary number may also be formed by substituting "1's" for "0's" and "0's" for "1's" in the number.

Performing subtraction by the complement method means that the computer need perform only one basic arithmetic operation, namely, addition. The accumulator circuits that perform the arithmetic were chosen to be subtractive; thus addition as well as subtraction is accomplished subtractively. The following equation shows that it is possible to achieve the result of addition by a subtractive process:




The equation states that the quantity in X is added to the quantity in A by subtracting the complement of X, usually written X', from A. The equation indicates that even though A is subtractive, the net result is additive. The notes above and below the equation indicate the manner in which the computer accomplishes addition.

Fundamental to the performance of subtraction is the generation of borrows which completes the operation. A "borrow" in a subtractive type accumulator compares to a "carry" in an additive accumulator, the exception being that a borrow "backs" a stage one count, whereas a carry "advances" a stage one count. It is this borrow feature which makes A subtractive rather than additive.

ADDITION

Perhaps the most important function of the arithmetic section is the basic operation of addition because it is involved in most other operations. Both the accumulator and the X register are used to accomplish addition. In the addition of X to A the result is governed by two conditions: (1) what must be done to $A_n$ to produce the difference without borrows of $A_n$ and $X'_n$; and (2) whether the arithmetic difference of $A_{n-1}$ and $X'_{n-1}$ requires that a borrow be made from $A_n$.

Sensing for First Condition

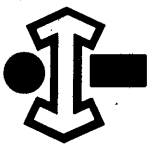
Table 3-2 shows that "0" subtracted from a minuend of either "0" or "1" results in a difference which has the same value as the minuend; conversely, a "1" subtracted from a minuend of either "0" or "1" results in a difference which is just the reverse of the minuend in each case. Where the minuend is "0" ("1" is subtracted from "0") a borrow from the next higher-order bit is required. The following rules summarize these observations on binary subtraction: (1) whenever the subtrahend is "0", the difference is the same as the minuend regardless of whether it is "0" or "1"; and (2) whenever the subtrahend is "1", the difference is the reverse of the minuend, with a borrow required from the next higher-order stage when the minuend is "0".

If the borrow generated in the last case of table 3-2 is disregarded, the difference column is the logical or bit-by-bit difference resulting from the associated minuend and subtrahend.

In the addition of X to A, the minuend is converted into the logical difference of the minuend and the subtrahend, by complementing the bits of the minuend for which the corresponding bits of the subtrahend are "1" (table 3-2). The minuend is in A and the subtrahend is the complement of the quantity in X. The logical difference of the minuend A and the subtrahend X is formed by toggling (individually complementing) each bit of A for which the corresponding bit of X is a "0".

TABLE 3-2. BINARY SUBTRACTION

| Minuend | | Subtrahend | | Difference |
|---|---|---|---|---|
| 1 | - | 0 | = | 1 |
| 1 | - | 1 | = | 0 |
| 0 | - | 0 | = | 0 |
| 0 | - | 1 | = | 1 with a borrow required from the next higher-order bit. |



Sensing for Second Condition

The second condition in the addition of X to A indicates what borrows must be made to form not merely the logical difference but the arithmetic difference of A and X. It is necessary to borrow from stage $A_n$ when both $A_{n-1}$ and $X_{n-1}$ were initially "0"; in other words, a borrow from $A_n$ means that the logical difference of $A_{n-1}$ and $X_{n-1}$ is "0". This follows, since the complement of X is subtracted from A. When $X_{n-1}$ is a "0", its complement is "1", and it is the complement of $X_{n-1}$ which is subtracted from $A_{n-1}$.

Combining the Two Conditions

The addition of X to A is accomplished by toggling A as determined by the two conditions described above. For each stage of A the two conditions for toggling are combined to gate the command Add X to A. If this AND is satisfied for $A_n$, then $A_n$ is toggled. Following the occurrence of the toggle command, A holds the sum of A and X.

Since the two conditions are independent, it is possible for both of them to be satisfied at the same time for a given stage. If both require $A_n$ to be toggled, then $A_n$ should not be toggled at all since toggling a stage twice restores it to the initial state. Also, if neither is satisfied, $A_n$ is not to be toggled. Only if one or the other condition (but not both) exists should $A_n$ be toggled. The exclusive OR combination of the two conditions is required for gating the Add command. Thus, $A_n$ is to be toggled for either of these cases: (1) $X_n$ is "0" and no borrow is required from $A_n$; and (2) $X_n$ is "1" and a borrow is required from $A_n$.

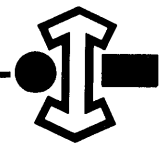
If a borrow is required from $A_n$ and $A_n$ is "0", then a borrow will be required from $A_{n+1}$. If $A_{n+1}$ is a "0", then a borrow will be required from $A_{n+2}$, and so on. The propagation of a borrow continues in this manner until a stage is reached which is "1". If a borrow is generated in $A_{47}$, the stage of highest order, it is made from $A_{00}$. This is the end-around borrow which makes the accumulator a one's complement arithmetic device. Note especially

that a borrow may be required from $A_n$ and yet it will not be toggled by the Add command. This situation exists whenever $X_n$ is "0".

Addition Example

The example given below illustrates the addition procedure. A 4-bit system is used for convenience; however, the 48-bit system of the accumulator is exactly the same. The logical difference of A initial ($A_i$) and $X_i$ is formed by complementing each bit of $A_i$ where the corresponding bit of $X_i$ is "0"; the other bits of the logical difference are the same as those of $A_i$. The determination of borrows can be made with the aid of this logical difference. A borrow is generated in any stage $A_n$ where $A_n$ and $X_n$ are both "0". The borrow is represented by an arrow which points to stage $A_{n+1}$, from which the borrow is made. In the case where the borrow points to a stage of the logical difference of A and $X^1$ that is "0" another borrow is required. To determine what stages of A must be toggled by the command Add X and A to produce the sum, a T is placed in that stage position when $X_n$ is "0" and no arrow points to it, or X is "1" and an arrow points to it.

| | | |
|---|---|---|
| Augend, in A initially | 0 1 0 1 | + 5 |
| Addend, in X initially | 0 0 1 1 | + 3 |
| Logical Difference of A and $X^1$ | ⌐1 0 0 1←⌐ | |
| Stages to be Toggled | T T T | |
| Sum of A and X (produced by toggling $A_i$ in each bit with a T) | 1 0 0 0 | + 8 |



Borrow Pyramid

The borrow pyramid senses the stages of A from which borrows are required and sends signals to them. These signals accomplish borrows by toggling the necessary stages. The pyramid is a network of inverters, part of which senses borrows (the $A^{--0}$, $A^{--1}$, $A^{--2}$, $A^{--3}$, $A^{--4}$, $A^{--5}$ and $A^{--6}$ inverters) and part of which senses where A must be toggled (the $A^{--7}$ inverters). The borrow sensing is accomplished by sampling $A^2$ and $X^2$. The result of borrow sensing is combined with $X^2$ as the input to the toggle control. The latter then determines what stages of $A^1$ are to be toggled by the Add $X^2$ to $A^1$ command. Appendix A at the back of this volume contains a detailed examination of the borrow pyramid.

Figure 3-3 shows the relation of the pyramid to the A and X registers. The purpose of this portion of the accumulator is to sense for each stage of A whether one or the other of two conditions exists which requires that $A_n$ be toggled in the addition of X to A. These conditions for toggling are, once again: $X_n$ is "0" and no borrow is required from $A_n$; and $X_n$ is "1" and a borrow is required from $A_n$.

For the full additions, the Partial Add in A FF is cleared. By setting it, the borrow sensing is disabled. This permits the Add command to simply toggle $X^2$ into $A^1$. Since this amounts to an Add without carries it is called a Partial Add.

## SUBTRACTION

The basic operation of subtraction is accomplished in a manner similar to addition. The minuend is located in A; the subtrahend is in X. Subtraction is prepared for by first complementing $X^1$ with the command Complement $X^1 \rightarrow X^2$. It is X' which is subtracted from A.

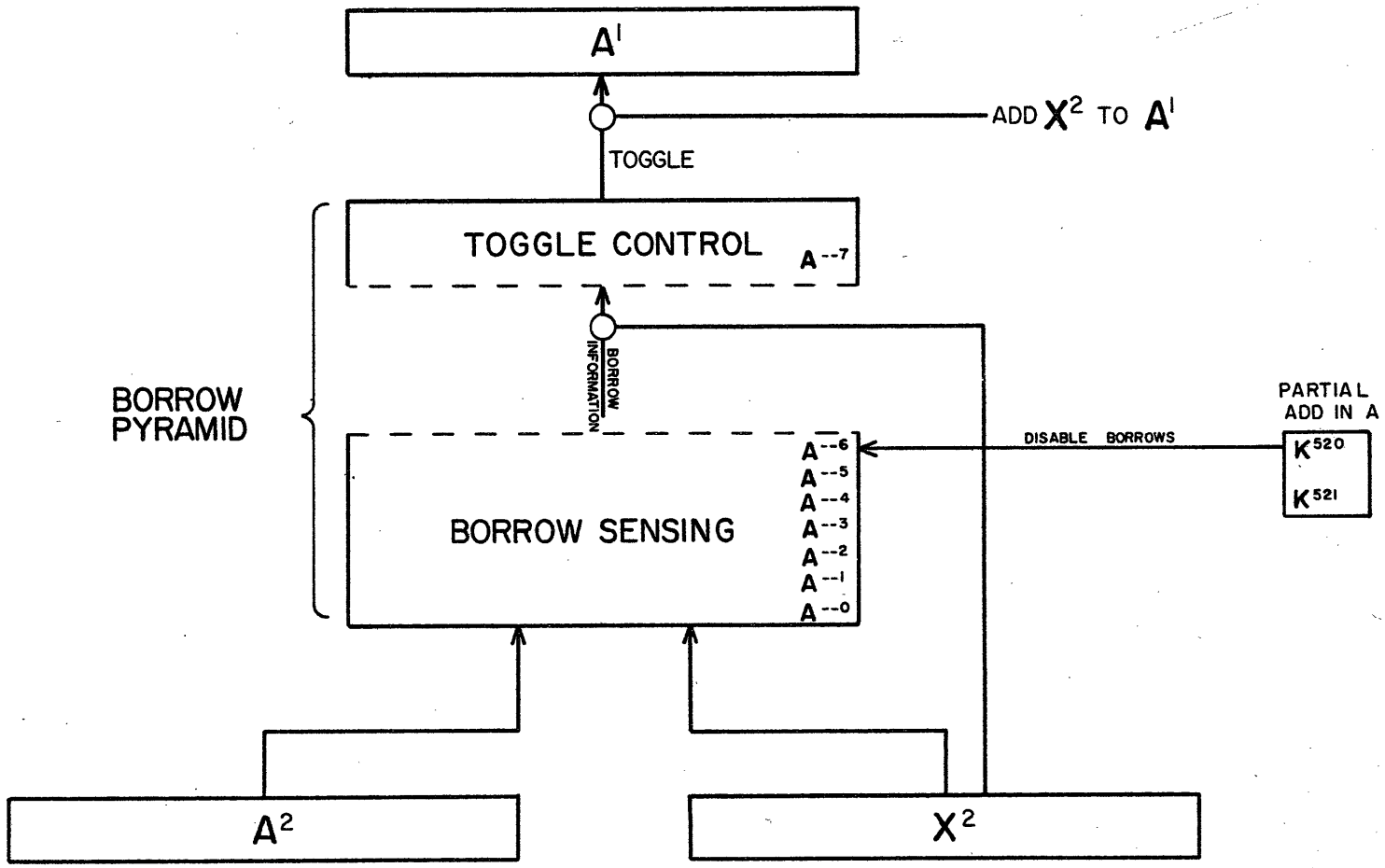
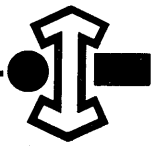
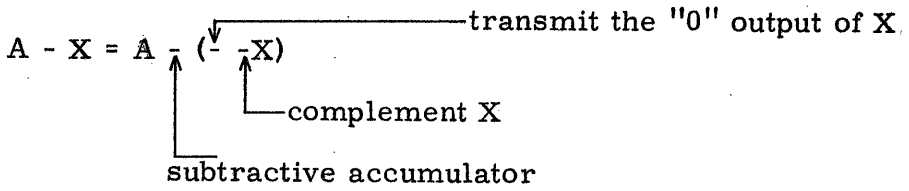



Figure 3-3. Relation of Borrow Pyramid to A and X registers.



Since this means X is in effect complemented twice during the operation, it follows that the procedure reduces to A - X, as is shown in the following equation:




Subtraction is accomplished in three steps: (1) complement X; (2) sense borrows by means of the logical difference of A and X; and (3) form the arithmetic difference by toggling as indicated by the borrow situation. Except for the first step the procedure is the same as that used in addition.

SHIFTING

Fundamental to the arithmetic operations are the shifting properties of A and Q. These registers can be shifted either to the right or to the left separately or as a combined 96-bit unit. Shifts are accomplished in increments of one stage at a time, up to the register width.

The transmission paths for shifting are always from the upper rank of the register to the next stage in the lower rank, the direction depending upon the type of shift, right or left, being performed. A subsequent command places the shifted word into the upper rank. When A and Q shift as a unit, $A_{00}$ is connected directly to $Q_{47}$.

The right shifts are all open-ended; as a word is shifted to the right the least-significant bit is lost after each stage shift. The sign bit is maintained in the most-significant stage and is extended to the right for as many stages as the word is shifted.

All left shifts are circular in fashion; the left shift sequence connects the highest stage directly to the lowest stage. Thus the content of the sign bit stage appears in the least-significant bit stage after each stage shift, and all other positions move one place to the left.

The shift operation can be called for directly by instruction and occurs as subsequences in the multiply, divide, floating-point and scale instructions.

Shift Instructions

The following instructions are used in programs to directly specify a shift operation:

| Instruction | Function |
|---|---|
| 01 | Shift A to the right |
| 02 | Shift Q to the right |
| 03 | Shift AQ to the right |
| 05 | Shift A to the left |
| 06 | Shift Q to the left |
| 07 | Shift AQ to the left |

The number of places to be shifted is designated by the operand address portion of the instruction. The shift count can have any value up through 127; however, 96, the number of stages in the double-length register AQ, should be the largest significant count. (The lowest eight bit positions of the execution address are required to encode 96, since the capacity of these eight bit positions is 127. Thus numbers in the range 97-127 are acceptable counts.) Any "1's" in bit positions greater than $2^7$ cause a fault indication which may be sensed by the external function instruction.

The zero address sequence initiates shifting by setting one or more of the four FFs ($K^{310}$ - $K^{317}$) in the shift control circuits (figure 3-4). Once set, these FFs enable the shift and Reduce R commands to occur every even phase. During the odd phase the $R^2$ to $R^1$ and appropriate $A^2$ to $A^1$ or $Q^2$ to $Q^1$ commands occur to prepare for the upcoming shift and reduce commands.

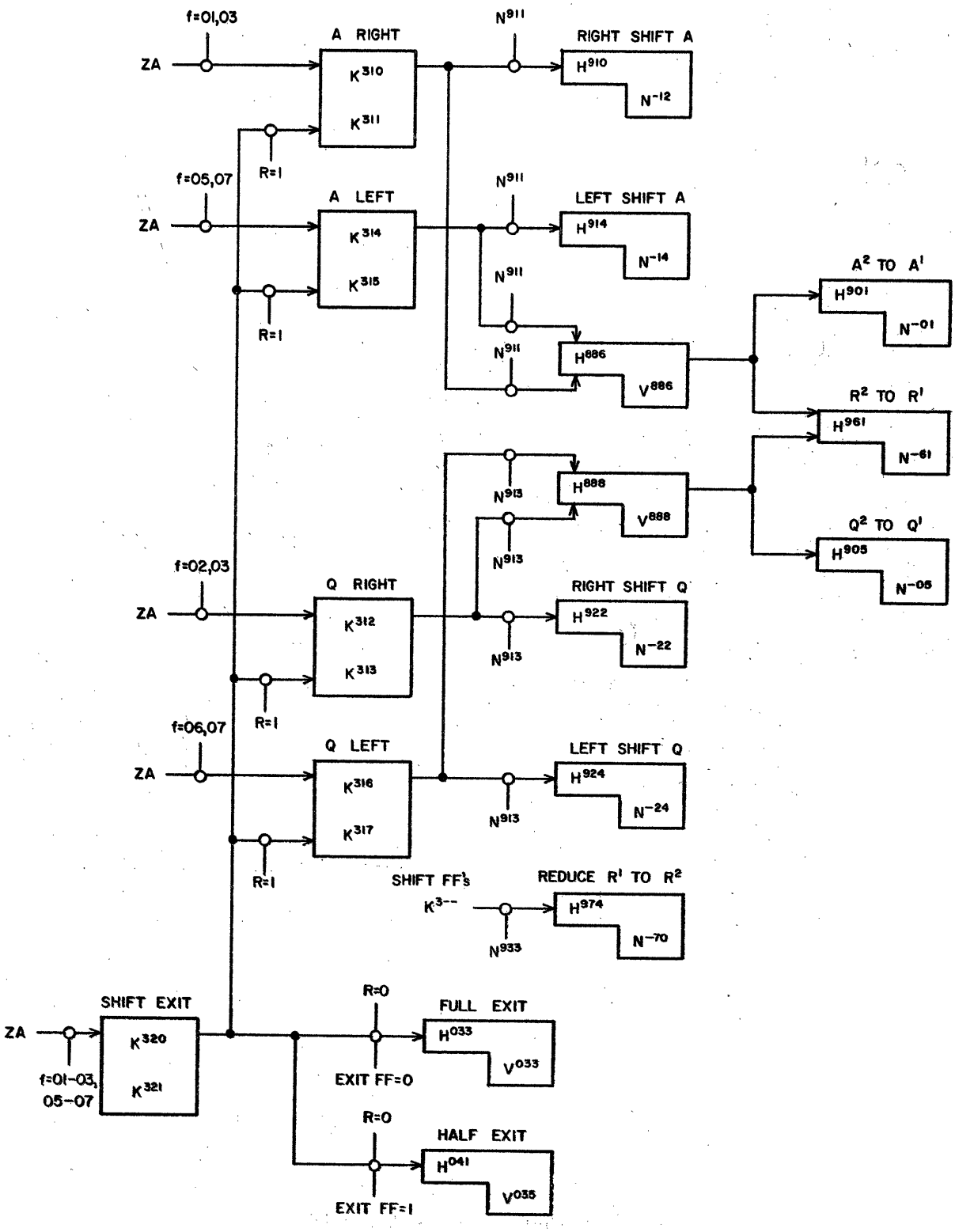



Figure 3-4. Shift Control.

The bottom of figure 3-4 shows the circuits concerned with shift termination. Shift Exit FF ($K^{320/321}$) is set to maintain a record of the fact that one of the shift instructions is being executed. When R is reduced to R=1 the shift FFs ($K^{310}$ - $K^{317}$) are cleared (on the even phase). This terminates shifting at the correct point, since a shift command is being executed at the same time.

The Reduce R command accompanying this last shift command reduces R from one to zero. The signal indicating R=0 (even phase) allows the appropriate exit to be taken.

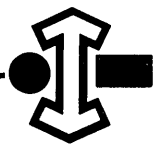
Shifting in Multiplication and Division

Shifting is fundamental to the arithmetic operations of multiplication and division. In multiplication, the AQ Right Shift is used to position the current sum of partial products in AQ for the formation of the next partial product. The shift also positions the next multiplier bit in $Q_{00}$ to determine the action required for generating the next partial product. The number of shifts required to produce a product depends on the type of multiplication (integer, fractional or floating-point) being performed. The control for the number of shifts is preset in R by the instruction.

In division, left shifts are employed to properly position the partial dividend as a minuend for the subtraction of the divisor to determine the individual bits of the quotient. The shift also causes the quotient bits to be assembled in proper order. The type of division being performed determines the number of shifts required to complete the quotient.

Shifting in Floating-Point Instructions

Shifting is employed as described above for determining the product or quotient of two floating-point operands. In addition, floating-point operations may require additional shifting in order that the final result may be expressed in proper fashion.



Shift in Scale Instructions

The scale instructions 34 and 35 are essentially left shift instructions, the main difference being the control that causes the operation to conclude. The content of A or AQ is shifted to the left until R = 0, as is the case for the normal shift instructions; or the shift concludes when a "1" appears in the stage immediately to the right of the sign bit.

The means of accomplishing shifting for the scale instructions is similar to that for the shift instructions (figure 3-4). The A Left FF or both the A Left and Q Left FFs are set to begin shifting. Shifting terminates by clearing these FFs when R = 0 or when scaling is achieved.

## ITERATIVE SEQUENCE

The longer arithmetic operation of multiplication and division and all floating-point operations are performed by the Iterative sequence. This sequence consists of control delays with $H^{6--}$ and $V^{6--}$ symbols (figure 3-5).

For any of the instruction (24-33) using the Iterative sequence one of the operands is initially in A as the result of a previous instruction. The sequence begins at $H^{601}$. The first part of the chain acquires the second operand from storage and begins the preparation of the operands. Various portions of the chain are then used as shown in figure 3-4 depending upon the instruction.

## MULTIPLICATION

The multiplication instructions, (24) Multiply Integer, (26) Multiply Fractional and (32) Floating Multiply, cause the computer to form the product of two operands, one contained in A and the other in the address specified by the instruction. The word in A is the multi-

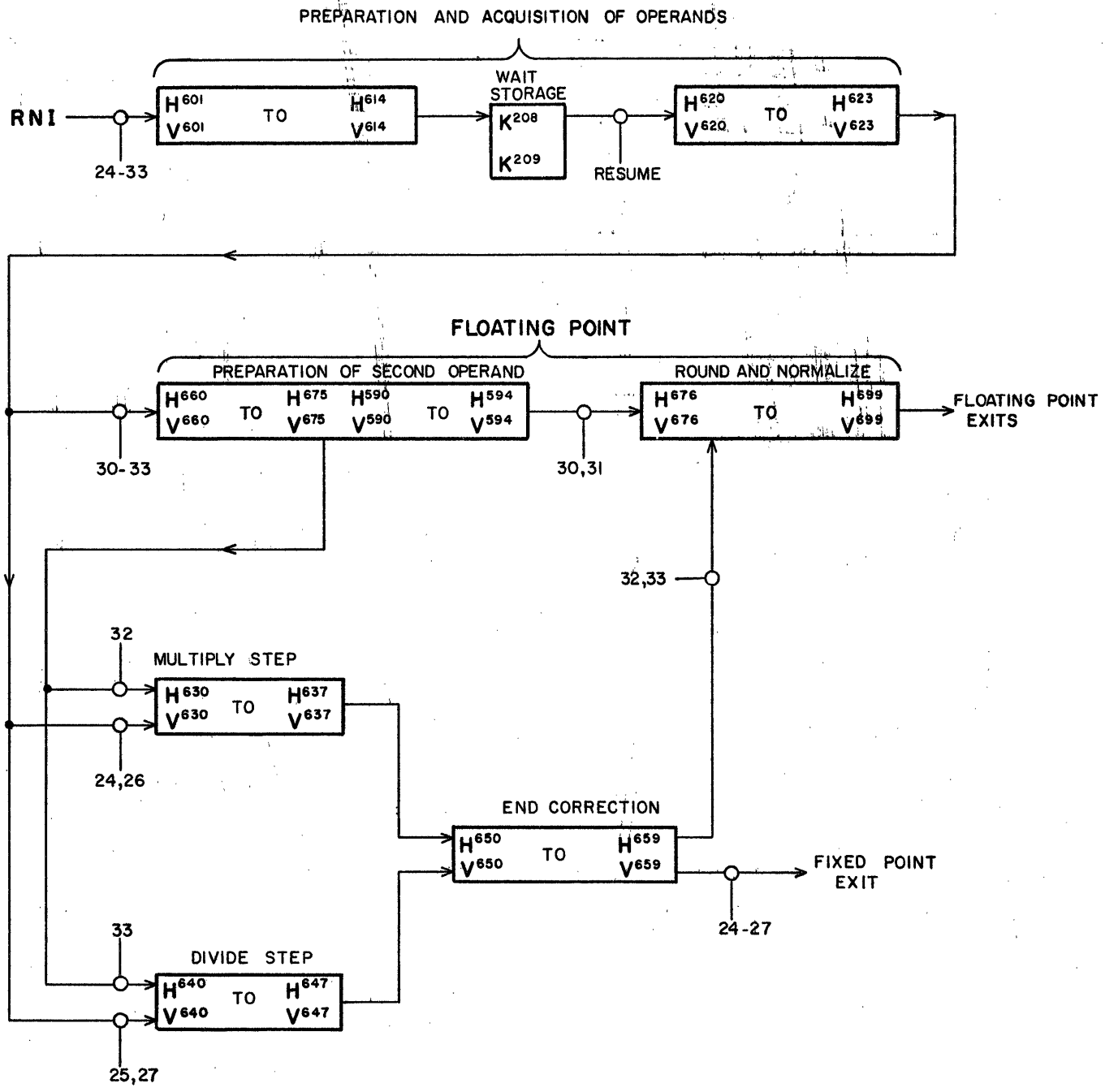
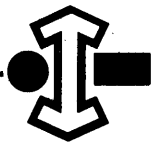



Figure 3-5. Iterative Sequence.



plier; the word in storage is the multiplicand. The result of the 24 and 26 instructions is a 96-bit quantity, held in QA for 24 and in AQ for 26.

The computer performs a multiplication by repeated additions and shifts. The multiplicand which is in storage is transferred to X from which it can be added to the partial product being developed in AQ. The multiplier digits, initially stored in A, are transferred to Q and then sequentially inspected beginning with the least-significant bit. The magnitude of each multiplier bit determines whether or not the multiplicand is to be added to the current sum of partial products during the particular step for which it is the control.

Binary multiplication proceeds according to the following rules:

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

Multiplication is always performed on a bit-by-bit basis as carries do not result from multiplication, since the product of any two bits is always a single bit.

For a more detailed explanation of binary multiplication, consider first the following decimal example:

| | | |
|---|---|---|
| multiplicand | 14 | |
| multiplier | 12 | |
| partial products | 28 | |
| | 14 | shift one place left |
| product | 168 | |

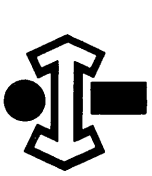
Note the apparent shift in the expression of the second partial product; however, this is a shorthand method for writing the true value 140.

Re-expressing this example as the product of the two binary numbers, the problem under consideration becomes:

| | | |
|---|---|---|
| multiplicand (14) | 1110 | |
| multiplier (12) | 1100 | |
| | 0000 | |
| | 0000 | shifts to place digits in proper columns |
| partial products | 1110 | |
| | 1110 | |
| product (168) | 10101000 | |

To avoid the difficulties that arise in the summation of more than two numbers, the computer determines the running subtotal of the partial products. Rather than shifting the partial product to the left to position it correctly as is done in paper and pencil multiplication, the computer accomplishes the same function by right shifting the summation of the partial products one place before the next addition is made. When the multiplier bit is "1", the multiplicand is added to the running total and the results are shifted to the right one place. When the multiplier bit is a "0", the partial product subtotal is shifted to the right (in effect, the quantity has been multiplied by $10_2$).

After each multiplier bit in Q is considered it is discarded and Q is shifted right one place. This always positions the current multiplier bit in $Q_{00}$. Because of this right shift the upper stages of Q become available to receive the lower-order digits of the product as it develops and is right shifted in A. Visualizing the example as the process evolves, the contents of the X, Q and A registers would appear as follows (for brevity, the registers are considered as containing four bits and sign):



X

0 1 1 1 0

| | A | Q | |
|---|---|---|---|
| Initial | 0 1 1 0 0 | 0 0 0 0 0 | |
| Step 1 | 0 0 0 0 0 | 0 1 1 0 0 | Interchange A and Q |
| Step 2 | 0 0 0 0 0 | 0 0 1 1 0 | Add $Q_{00}$ times X to A Shift right one |
| Step 3 | 0 0 0 0 0 | 0 0 0 1 1 | Add $Q_{00}$ times X to A Shift right one |
| Step 4 | 0 0 1 1 1 | 0 0 0 0 1 | Add $Q_{00}$ times X to A Shift right one |
| Step 5 | 0 1 0 1 0 | 1 0 0 0 0 | Add $Q_{00}$ times X to A Shift right one |
| Step 6 | 1 0 0 0 0 | 0 1 0 1 0 | Express the lower bits of the product in A, the upper bits in Q. |

The actual multiplication procedure which is carried out by the Iterative Sequence may be divided into three parts:

1) initial sign correction
2) multiplication phase
3) final sign correction

During initial sign correction both the multiplier and multiplicand are made positive, if they were not so originally. In order to given the final product the proper sign, the Sign Record FF is set if the signs of the two were unlike (product is to be negative). If the signs are alike the FF is cleared. Later during final sign correction the Sign Record FF indicates whether the product obtained from the multiplication phase must be negative and hence complemented.

Following the initial sign correction, the sequence generates a control quantity which determines the number of steps necessary to conclude the multiplication. This number, which varies with the type of multiply being performed, is set in $I^2$ and transmitted to R. As each step is completed R is reduced by one. When R = 0, conditions are produced which cause the sequence to perform its concluding operation.

The step control quantities set in $I^2$ for the different multiplication instructions are as follows:

48 for instruction 24, Multiply Integer

47 for instruction 26, Multiply Fractional

36 for instruction 32, Floating Multiply

The difference in the control quantities provides for the proper positioning of the product within Q and A (the A register in the case of Floating Multiply). Integer products are positioned with regard to the binary point being at the right-hand side of A; the fractional products are positioned with regard to the binary point which is located immediately to the right of the sign bit, $A_{47}$. For floating-point products the point is just left of stage $A_{35}$.

The determining of a product with operands packed in floating-point format proceeds in a manner similar to that described here. However, floating-point multiplicands and multipliers contain 36 bits; therefore, multiplication is accomplished in fewer steps. The product is then rounded to 36 bits.

After the preparatory measures described above have been accomplished, the multiplication phase begins. This part of the procedure is accomplished by repeating the multiply step (figure 3-6) the number of times specified by the control quantity. Each repetition forms the partial product of a bit of the multiplier and the multiplicand and then adds this to the running sum of partial products.

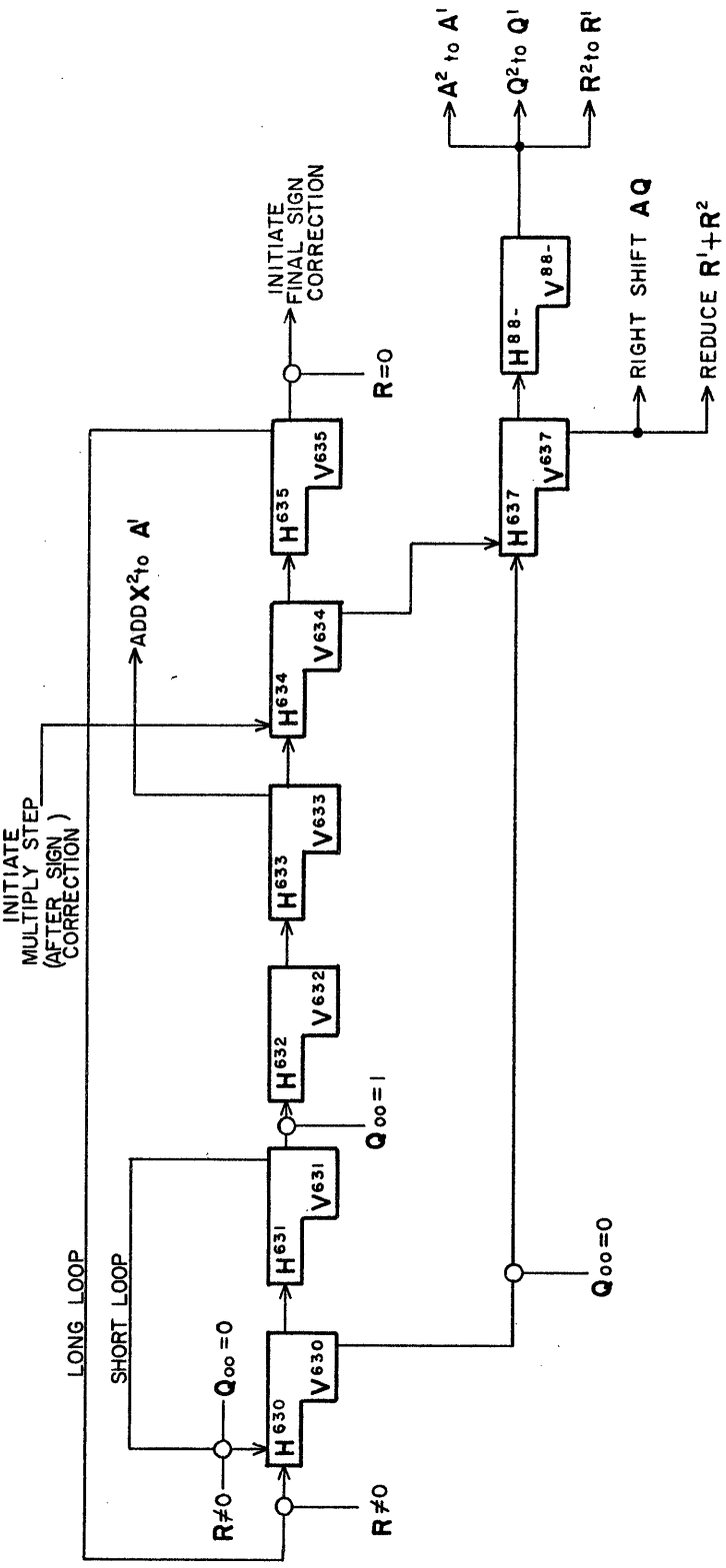


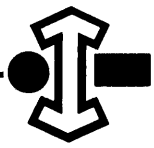

Figure 3-6. Multiply Step.

The first multiply step begins by shifting AQ right. This disposes of the multiplier bit in Q; however, the effect of $Q_{00}$ is maintained in slaves long enough to determine the path (the short loop or long loop) for the first trip around. Following the shift, the sequence reaches a decision point: if $Q_{00}$ was a "0", the short loop is selected; if $Q_{00}$ was a "1", the long loop is selected. The short loop merely right shifts the partial product in AQ; this, as was explained previously, has the effect of a multiplication by a zero bit. The long loop first adds the multiplicand X to the sum of partial products in A and then initiates a shift AQ right, performing a multiplication of one bit. As each shift, on either the short or long loop, is performed, R is reduced by one. When R = 0 the product accumulation is terminated.

The product as it has been assembled is a positive number, it has a "0" sign, and its bits are presented in non-complement fashion. If the product is to be expressed as a negative number, that is, if the operands were initially of opposite signs, AQ is complemented. In the case of 24, Integer Multiply, A and Q are interchanged to place the least-significant part of the product in A.

DIVISION

The divide instructions call for the division of a word already in the A and Q registers by a divisor which is in storage at an address specified by the instruction. There are three division instructions: (25) Divide Integer; (27) Divide Fractional; and (33) Floating Divide. The execution of a division is performed in three phases: initial sign correction, division phase, and final sign correction.



The following example shows the familiar method of decimal division:

```
                    14          Quotient
Divisor      13 ) 185           Dividend
                  13
                  ---
                  55            Partial Dividend
                  52
                  ---
                   3            Remainder
```

The computer performs division in a similar manner (using binary equivalents):

```
                   1110         Quotient
Divisor     1101 ) 10111001     Dividend
                   1101
                   ----
                  10100
                   1101
                   ----
                   1110         Partial Dividends
                   1101
                   ----
                     11         Remainder
```

However, instead of shifting the divisor right to position it for subtraction from the partial dividend (shown above), the computer shifts the partial dividend left, accomplishing the same purpose and permitting the arithmetic to be performed in the A register. The computer counts the number of shifts, which is the number of quotient digits to be obtained, and after the appropriate number of counts, the length of the quotient register, the routine is terminated.

In programming a division the relationship of the size of the dividend and the divisor must be considered in order to express the quotient within the capacity of the quotient register.

Initial Sign Correction

Since the division phase requires a positive divisor and dividend, the first phase, initial sign correction, complements either or both of these operands if they are negative. At the same time the sign of the quotient is determined by the normal algebraic rules: the quotient is positive if the signs of the operand are alike, negative if the signs are unlike. An indication of the sign of the quotient is stored in the Sign Record FF. Later, during the final sign correction, this FF directs whether to complement the positive quotient that always results from the division phase. In a similar manner the sign of the dividend is recorded so that the remainder may be given the same sign.

For the division phase the dividend must be positioned in AQ, i. e., with higher-order bits in A. Therefore, during initial sign correction for 25 Divide Integer, A and Q are interchanged (initially the dividend is in QA for 25).

Division Phase

The divide step which performs the actual division (figure 3-7) involves:

1) left shifting AQ one place
2) subtracting X (divisor) from A (partial dividend) if $A \geq X$
3) setting $Q_{00}$ to "1" if subtraction was made; otherwise entering "0" in $Q_{00}$.

The divide instructions require two different control quantities to govern the number of repetitions of the divide step, which performs each partial division and generates the individual quotient bits. The quantity is 48 for both the integer and fractional divide instructions (25 and 27); for the floating-point divide (33) it is 36. $I^2$ is set to the value of the control quantity and is then transmitted to the R register. Each repetition of the divide step reduces R by one count. After the step has been repeated until R = 0, the full quotient is in

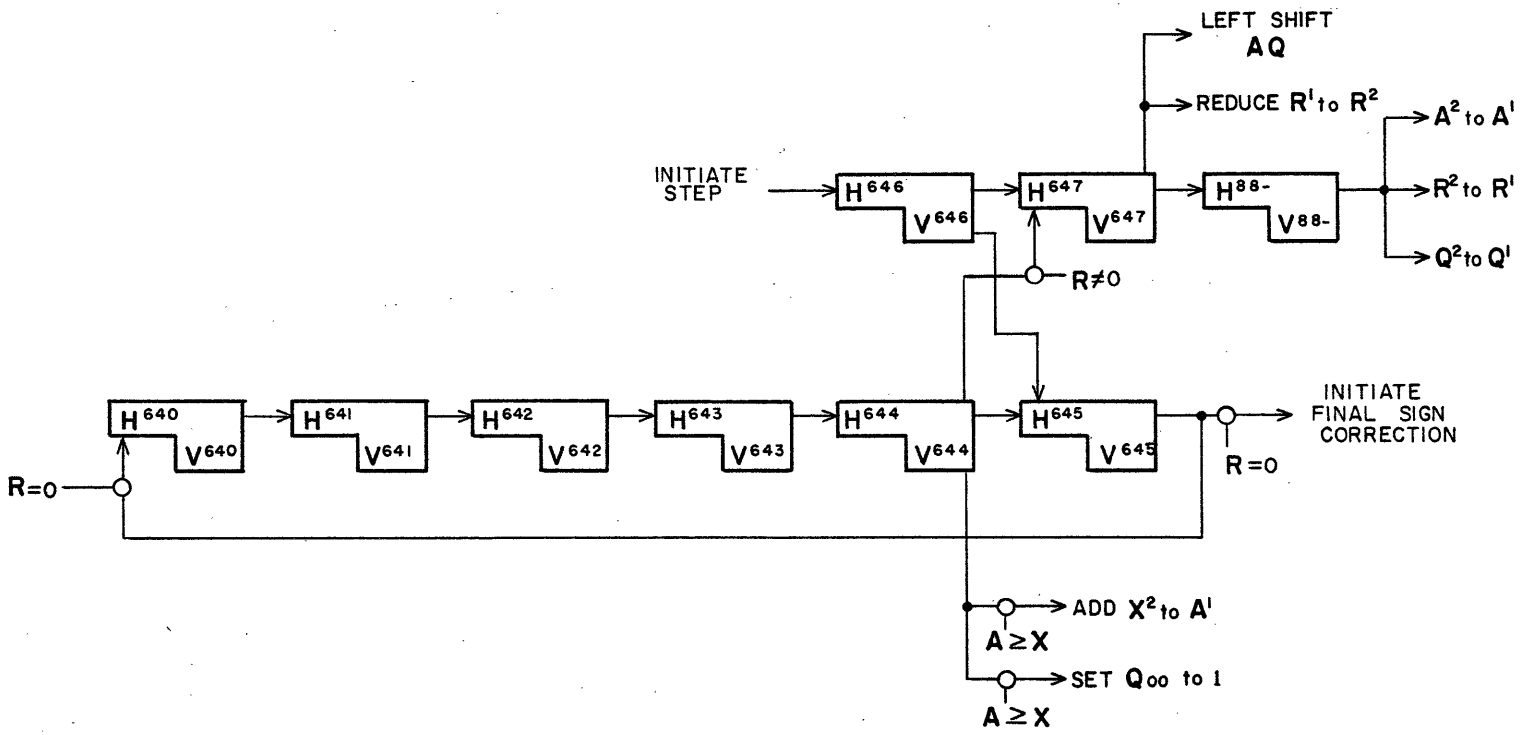



Figure 3-7. Divide Step.

Q and the remainder is in A.

Subsequent commands position the quotient in A and the remainder in Q. The following example, using 5-bit registers for brevity, indicates how division proceeds in the computer. Remarks in the right-hand column clarify the accomplishment of each step.

| | | | |
|---|---|---|---|
| Divisor | 0 1 1 0 1 | | |
| | | R = 5 | |
| Dividend | 0 0 1 0 1 \| 1 1 0 0 1 | | |
| | 0 1 0 1 1 \| 1 0 0 1 0 | | Shift AQ left |
| | | R = 4 | |
| | 1 0 1 1 1 \| 0 0 1 0 0 | A < X | Shift AQ left |
| | | R = 3 | |
| | 0 1 0 1 0 \| 0 0 1 0 1 | A ≥ X | Subtract X from A<br>Set $Q_{00}$ to 1 |
| | 1 0 1 0 0 \| 0 1 0 1 0 | | Shift AQ left |
| | | R = 2 | |
| | 0 0 1 1 1 \| 0 1 0 1 1 | A ≥ X | Subtract X from A<br>Set $Q_{00}$ to 1 |
| | 0 1 1 1 0 \| 1 0 1 1 0 | | Shift AQ left |
| | | R = 1 | |
| | 0 0 0 0 1 \| 1 0 1 1 1 | A ≥ X | Subtract X from A<br>Set $Q_{00}$ to 1 |
| | 0 0 0 1 1 \| 0 1 1 1 0 | | Shift AQ left |
| | Remainder Quotient | R = 0 | Initiate Final Sign Correction |

Final Sign Correction

The Final sign correction phase:

1) senses for a divide fault

2) complements Q (the quotient) if the Sign Record FF is set (dividend and divisor initially have unlike signs)

3) complements A (the remainder) if Dividend Sign FF is set (dividend negative initially)

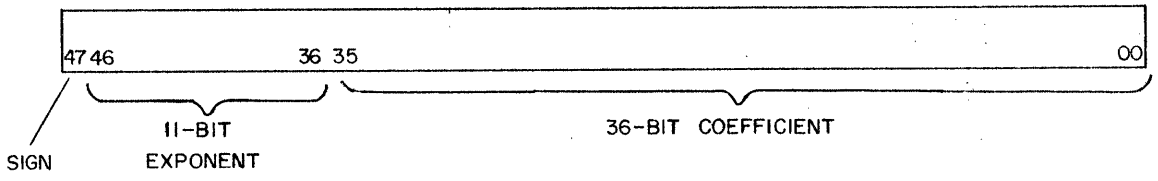
4) places quotient in A and remainder in Q

A divide fault (quotient register overflow) results when the correct quotient requires more than 47 bits to express its magnitude. If a division involves a fault, then after the first shift of the division phase A >X and a "1" is entered in $Q_{00}$. But if a divide fault is not involved, then after the first shift A < X and a "0" is entered in $Q_{00}$. In either case this bit appears in $Q_{47}$ after the 47 remaining shifts of the division phase. Thus, at this point if Q is negative there is a fault; if Q is positive there is not a fault. Sensing a divide fault is thus sensing the sign of Q. The Divide Fault FF is set when Q is negative. The fault condition can be sensed later by the 74 External Function instruction.

After the remaining three functions of the final sign correction phase, instructions 25 and 27 take the appropriate exit. Instruction 33, Floating Divide initiates the round and normalize part of the Iterative sequence.

## FLOATING-POINT

Any number can be described by the general expression $kB^n$, where k is a coefficient, B a base number, and the exponent n the power to which the base number is raised. The floating-point mode of operation takes advantage of this means of expression by including in its operand format the coefficient, the sign of the number and the exponent. The make-up of a floating-point word is shown below:




### Coefficient

The coefficient is made up of a 36-bit fraction located in the lower-order positions of the floating-point word and a single bit located in the conventional highest-order bit

position. The fraction has a value ranging from one-half to one (not including one). Negative numbers are represented in one's complement notation.

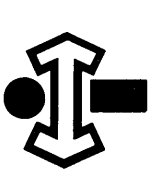
Exponent

The exponent is an 11-bit number with a value ranging from 0 to $2^{11}$ -1 (2047 in decimal, 3777 in octal). A bias of $2^{10}$ (1024 in decimal, 2000 in octal) is added to the true exponent when the floating-point number is formed. Thus a number with true exponent equal to zero would appear as 2000 (octal). A number with exponent equal to 264 would appear as 2264; a number with exponent equal to minus 137 would appear as 1641. As in algebra, positive exponents are used for integral numbers and negative for fractional numbers.

When a number is negative, the exponent is included in the one's complement representation. Thus if the above examples of exponents were for negative numbers, they would appear in the floating-point word as the bit-by-bit complement.

The bias used with the exponent makes floating-point operation more versatile since now floating-point operands can be compared with each other in the normal fixed-point mode. The transition from fixed-point to floating-point format is given below for five numbers to illustrate the encoding of numbers by fixed-point methods and the way this format lends itself to comparisons. For simplicity the fraction is limited to three bits and the exponent to four bits including the sign. The exponent is in parentheses.

| Fixed-Point | | | Floating-Point |
|---|---|---|---|
| +4.0 | = $0100 \times 2^{0}$ | = | $0.100 \times 2^{3}$ = 0(1011) .100 |
| +0.04 | = $0.000100 \times 2^{0}$ | = | $0.100 \times 2^{-3}$ = 0(0100) .100 |
| -4.0 | = $1011 \times 2^{0}$ | = | $1.011 \times 2^{3}$ = 1(0100) .011 |
| -0.04 | = $1.111011 \times 2^{0}$ | = | $1.011 \times 2^{-3}$ = 1(1011) .011 |
| +0.4 | = $0.100 \times 2^{0}$ | = | $0.100 \times 2^{0}$ = 0(1000) .100 |
| | | | or 0(0111) .100 |



In a bit-by-bit comparison of two numbers the larger or more positive has a "1" in the first higher position for which the bits of the two numbers are unlike. The signs (first bit) are compared independently; a positive (0) quantity always is indicated as larger than a negative (1) quantity. Now, since 4.0>0.04, the floating-point form of 4.0 should and does have a "1" in a more significant position than does 0.04. Thus fixed-point comparison of the two floating-point numbers yields the desired result. Similarly, -0.04>-4.0, since the first is less negative. Thus the floating-point form of -0.04 contains a "1" in a higher position than the highest "1" of -4.0.

The two forms possible for +0.4 present a special case. The first is considered standard: that is, if all floating-point operands that fall in the range which has an exponent of magnitude zero are encoded in the positive form, then results of floating-point operations which fall in this range are also represented by exponents of positive form. In short, unless the original operands have the negative representation of an exponent of magnitude zero, this form of exponent is not generated by the computer. These comments apply to the state of the 11-bit exponent before any complementing is performed to care for the fact that the entire number may be negative.

Floating-Point Operation

Any of the four floating-point instructions are performed by the following sequence of steps:

Unpack
Execute coefficient and exponent arithmetic
Round
Normalize
Pack

All operands should first be in floating-point format.

The unpack step involves both operands: one in the accumulator and one from storage.

Each operand is separated into its exponent and coefficient. The coefficients are sent to the arithmetic registers and the exponents to the address modification registers, $U^2$ and R. In unpacking, the sign of the number is examined and if it is negative both the exponent and coefficient are complemented.

To illustrate actual floating-point format several sample quantities are shown below, encoded in binary. The floating-point word for the decimal quantity + 1604. 0 (3104 in octal) is:

| Exponent | Coefficient |
|---|---|
| 010 000 001 011 | 110 010 001 000 000 000 000 000 000 000 000 000 |

Sign

The floating-point word for the decimal quantity - 1604. 0 is:

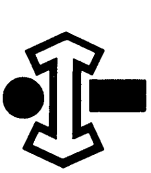
| Exponent | Coefficient |
|---|---|
| 101 111 110 100 | 001 101 110 111 111 111 111 111 111 111 111 111 |

In this case the negative character is indicated by the "1" in the sign position of the fraction. The fraction is expressed in one's complement form.

The floating-point word for the decimal quantity - .1604 is:

| Exponent | Coefficient |
|---|---|
| 110 000 000 010 | 010 110 111 100 000 000 011 010 000 000 111 010 |

In the arithmetic step the exponent operations are handled in the $U^2$ and R registers. Coefficient arithmetic is performed as a fractional fixed-point operation. In addition and subtraction the coefficients are aligned by shifting to make the exponents alike.



The round step modifies the coefficient answer by adding one to A for positive answers or by subtracting one for negative answers. Rounding is necessary since the result of the operation on coefficients may yield a quantity containing more than 36 bits. The condition for rounding is inequality of the sign bits of A and Q. This means, in effect, that the next lower significant bit to the right of the number in A is equal to or greater than one-half.

Coefficient arithmetic may yield rounded answers in the range from zero to $2^{37}$. The normalize step brings this answer back to a fraction ranging from one-half to one with the binary point to the left of the 36th bit. In other words, the final normalized number in A will range from $2^{36}$ to $2^{37}-1$. Normalizing is performed by either a right shift or the required number of left shifts. A correction is made to the exponent for every shift. The residue in Q is not shifted.

The pack step positions the final exponent and coefficient in A. If the sign of the answer if negative both the coefficient and exponent are complemented. The exponent range is tested at this time to determine overflow of the exponent and set the fault indicator.

Addition (Table 3-3)

The Floating Add instruction, 30, produces the sum of the floating-point operands in A (augend) and address **M** (addend). In accordance with the floating-point format, the coefficients and exponents of the operands are separated. Operations are performed on the coefficients in the A and X register while the exponents are operated on in the R and $U^2$ registers. At the conclusion of the separate operations the resultant exponent and coefficient are assembled into proper format in A.

TABLE 3-3. BASIC STEPS IN FLOATING ADD

1) Enter augend in A by previous instruction; complement if negative.
2) Acquire addend from address **M** and place in X; complement if negative.
3) Compare exponents; save larger in $U^2$; difference in R.
4) Position coefficients so that one with smaller exponent is in A.
5) Shift AQ right by number of places indicated by content of R.
6) Add coefficients (add X to A).
7) Round off portion of coefficient in Q, if $A_{47} \neq Q_{47}$.
8) Normalize A so that $A_{35} \neq A_{36}$. If normalizing requires left shifting reduce R by one for each shift. If $A_{36} \neq A_{37}$ right shift one place and increase R by one.
9) Test for exponent fault (exponent greater than $2^{10} - 1$; Set FF in this case).
10) Assemble floating point sum in X: exponent comes from $U^2$ and coefficient from A.
11) Transmit X to A.

Steps 3, 4 and 5 (in table 3-3) align the two exponents, that is, make them equal by right shifting of the coefficient with the smaller exponent. Notice that step 5 shifts both A and Q to the right in aligning the exponents. Thus bits of the augend coefficient are moved into Q. As shown in figure 3-8a, the addition of X to A in step 7 may produce a sum of more than 36 bits. Step 8, therefore, rounds off the excess portion in Q by adding one to A if $Q_{47} = 1$.

Normalizing (step 8) provides for expressing the sum in proper floating-point format. If the sum of the coefficients has more than 36 bits, it must be shifted right and the exponent increased. (At most the sum in A can have 37 bits, and thus only one right shift will ever be required). If the sum in A has less than 36 bits (its most significant bit is not in $A_{35}$) then it must be shifted left and the exponent increased by one for each place shifted.

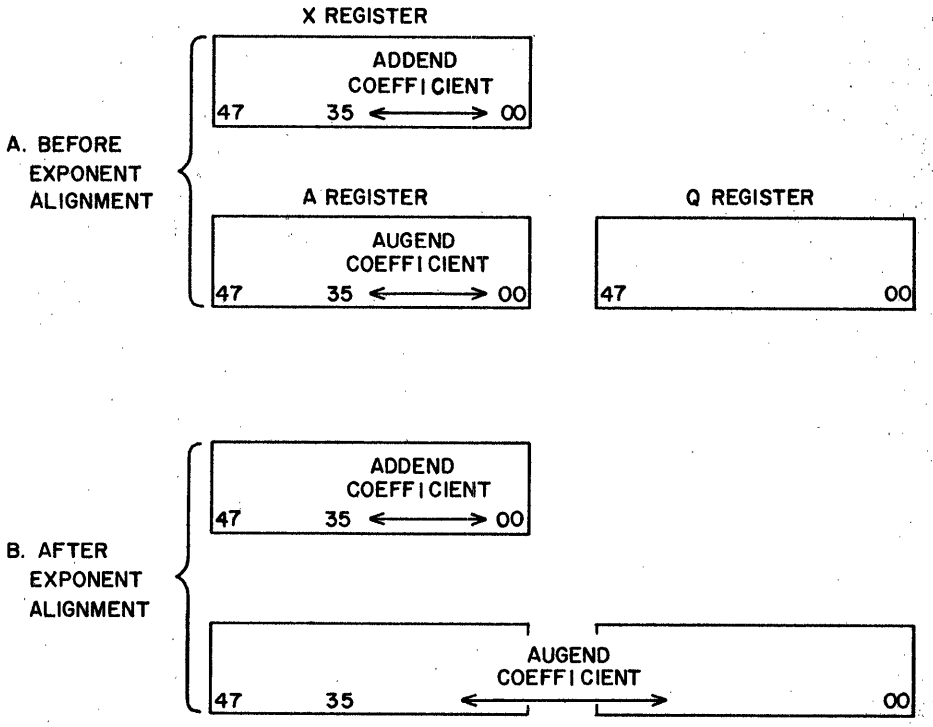



Figure 3-8. Typical Relation of Coefficients in Addition.

Subtraction (Table 3-4)

The Floating Subtract instruction, 31, produces the difference of two floating-point operands in A (minuend) and address M (subtrahend). As in addition, the exponents and coefficients are separated and the operation performed on them separately.

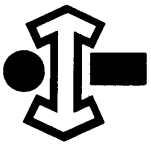
TABLE 3-4. BASIC STEPS IN FLOATING SUBTRACT

1) Enter minuend in A by previous instruction; complement if negative.
2) Acquire subtrahend from address M and place in X; complement if negative.
3) Separate and compare exponents; save larger in $U^2$; difference in R.
4) Position coefficients so that one with smaller exponents is in A.
5) Shift AQ right by number of places indicated by content of R.
6) Subtract coefficients (subtract X from A).
7) Round off portion of coefficient in Q, if $A_{47} \neq Q_{47}$.
8) Normalize A so that $A_{35} \neq A_{36}$. If normalizing requires left shifting reduce R by one for each shift. If $A_{37} \neq A_{36}$ right shift one place and increase by one.
9) Test for exponent fault (exponent greater than $2^{10}$ -1; Set FF in this case).
10) Assemble floating point difference in X; exponent comes from $U^2$ and coefficient from A.
11) Transmit X to A.

Multiplication (Table 3-5)

The Floating Multiply instruction, 32, forms the floating-point product of two operands in A (multiplier) and address M (multiplicand). Exponents and coefficients are separated and operated on independently.

The multiplication of the two coefficients is performed in a manner identical to that used for fixed point. However, the multiply step is repeated only 36 times instead of 48. Figure 3-9 shows the location of the double-length product in AQ. In order to express the



product in floating-point format the least-significant 36 bits of the product (located in Q) are rounded off by adding one to A if $Q_{47} \neq A_{47}$.

TABLE 3-5. BASIC STEPS IN FLOATING MULTIPLY

1) Enter multiplier in A by previous instruction; complement if negative.
2) Acquire multiplicand from storage address M and enter in X; complement if negative.
3) Set Sign Record FF if product is to be negative.
4) Extract exponents from A and X; send to $U^2$ and R.
5) Add R to $U^2$; leave sum in $U^2$.
6) Initiate multiply step to form product of coefficients; repeat step 36 times; final product in AQ.
7) Complement if Sign Record = 1.
8) Round product to 36 bits by adding one to A if $Q_{47} \neq A_{47}$.
9) Transmit sum of exponents from $U^2$ to R.
10) Normalize product so that $A_{35} \neq A_{36}$; shift AQ left and increase R by one per shift, or right shift AQ once and reduce R by one.
11) Test for exponent fault (exponent greater than $2^{10} - 1$) and set Exponent Fault FF.
12) Assemble floating point product in X: exponent from R and coefficient from A.
13) Transmit X to A.

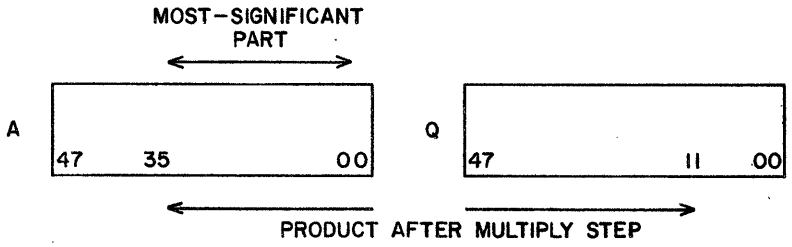



Figure 3-9. Location of Double-length Product for Floating Multiply.

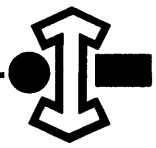
Division (Table 3-6)

The Floating Divide instruction, 33, produces the quotient of the two operands in A (dividend) and address M (divisor). At the outset the dividend is examined and if found to be zero, the instruction terminates (with a dividend equal to zero the quotient would be zero).

The quotient of the two coefficients is formed in a manner identical to that used for fixed-point division. However, the divide step that performs each partial division is repeated only 36 times.

TABLE 3-6. BASIC STEPS IN FLOATING DIVIDE

1) Enter dividend in A by previous instruction; complement if negative.
2) Acquire divisor from storage address M and enter in X; complement if negative.
3) Set Sign Record FF if quotient is to be negative.
4) Extract exponents from A and X and send to $U^2$ and R.
5) Subtract R from $U^2$; leave difference in $U^2$.
6) Initiate divide step to form quotient of coefficients (repeat step 36 times); quotient in Q at conclusion.
7) Complement if Sign Record FF = 1.
8) Transmit quotient from Q to A.
9) Transmit exponent from $U^2$ to R.
10) Normalize quotient if $A_{35} = A_{36}$ (shift A left and reduce R by one per shift until $A_{35} \neq A_{36}$ or shift right one place and increase R.)
11) Test for exponent fault (exponent greater than $2^{10} - 1$; set FF in this case).
12) Assemble floating point quotient in X: exponent from R and coefficient from A.
13) Transmit X to A.



## LOGICAL PRODUCT

A logical product is the result of the bit-by-bit multiplication of two numbers. A bit in the logical product is "1" only when both bits used in its determination are "1"; if either or both bits are "0" the bit of the logical product is "0". Thus:

| | |
|---|---|
| 1 0 1 1 | multiplicand |
| 1 1 0 0 | multiplier |
| 1 0 0 0 | logical product |

The logical product of two quantities is formed by the computer in X as the result of interaction between X and Q. Figure 3-10 shows the circuit which forms the logical multiplication of one stage of Q and X. Setting the LQX FF clears every stage of X for which the associated stage of Q holds "0". But for stages of Q holding a "1", the associated stages of X remain undisturbed. Thus the interaction between Q and X as governed by $K^{540/541}$ obeys the laws of binary multiplication, namely:

| $Q_i$ | | $X_i$ | | $X_f$ |
|---|---|---|---|---|
| 0 | x | 0 | = | 0 |
| 0 | x | 1 | = | 0 |
| 1 | x | 0 | = | 0 |
| 1 | x | 1 | = | 1 |

Forming the logical product of Q and X is a discrete logical function of the arithmetic section of the computer. However, this function is used to accomplish more complex operations; most important is the ability to select specific portions of an operand for entry into another operation. As it passes through X the operand is subject to a mask which has been loaded in Q. The mask, which is comprised of a pattern of "0's" and "1's", causes X to retain its original content only in those stages which have corresponding "1's" in Q. Once the selected bits are all that remain in X the instruction proceeds to its conclusion.

In some instances it is more feasible to have the mask in storage and the operand to be masked in Q. The result is the same and in either case will appear in X. The computer instructions in which the formation of a logical product is involved areas follows: (43) Selective Substitute, (44) Load Logical, (45) Add Logical, (46) Subtract Logical, (47) Store Logical, (66) Masked Equality and (67) Masked Threshold.

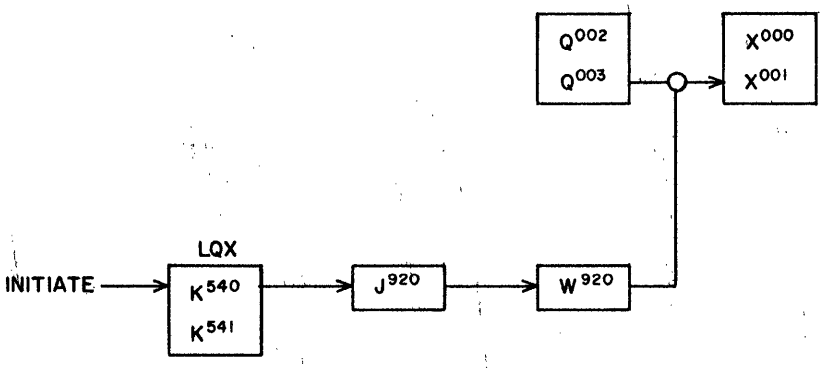
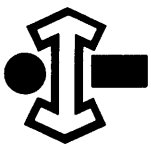



Figure 3-10. Formation of the Logical Product.

## REGISTER SENSING CIRCUITS

For several computer operations or commands the choice of whether or not to perform them is conditioned on the state of arithmetic registers. Some examples are:

1) In the divide step the condition A ≥ X conditions the subtraction of X from A and the entering of a "1" as a partial quotient digit.
2) In the Equality Search instruction (64), which searches a list of operands for one equal to A, the exit from the search is conditioned by X = A.
3) In the A Jump instruction with j = 0, the choice of jumping is conditioned by A = 0.



For this case, a similar whiffletree samples the "1" side of each stage in A, the output of which is supplied in OR fashion with the A = +0 signal to $Q^{986}$.

A pyramid similar to that shown for A tests for Q = 0. The logical connections and symbols are similar. The A = 0 and Q = 0 signals are brought together in AND fashion to $J^{126}$ which provides the indication that the double-length register AQ = 0.

Slave inverters connected to the outputs of the sign bit stage of the A, Q and X registers provide indication of the sign of the content of these registers.

In some operations (test for divide fault, scale, and floating point) the results of the comparison of two bits determines the course of action the computer is to take. (1) An output from $J^{110}$, indicating inequality between $A_{47}$ and $Q_{47}$, is used to test for a proper division. (2) A "0" output indicates overflow during the formation of the quotient. (3) $J^{124}$ is used in the scale instruction to indicate whether scaling has been achieved, that is, whether a "1" has been shifted into $A_{46}$. (4) $J^{131}$ and $J^{123}$ aid in properly positioning the coefficient during the normalizing process of the floating-point instruction. The logic of these comparisons is shown by the inverters in the bottom row in figure 3-11.

Figure 3-11 shows the pyramids which provide specific indication of the register's condition. The upper pyramid tests for equality between A and X. The first level inverters of the accumulator borrow pyramid are used in determining this condition. Since the inputs of this level require similarity between A and X to produce a "0" output, any "1's" emanating from this level of inverters indicates disimilarity and this will produce a "0" output from $A^{994}$.

The middle pyramid illustrates how the test is made for A=0. The "0" sides of the A register FFs are brought together in "whiffletree" fashion. Any "1's" suppress the generation

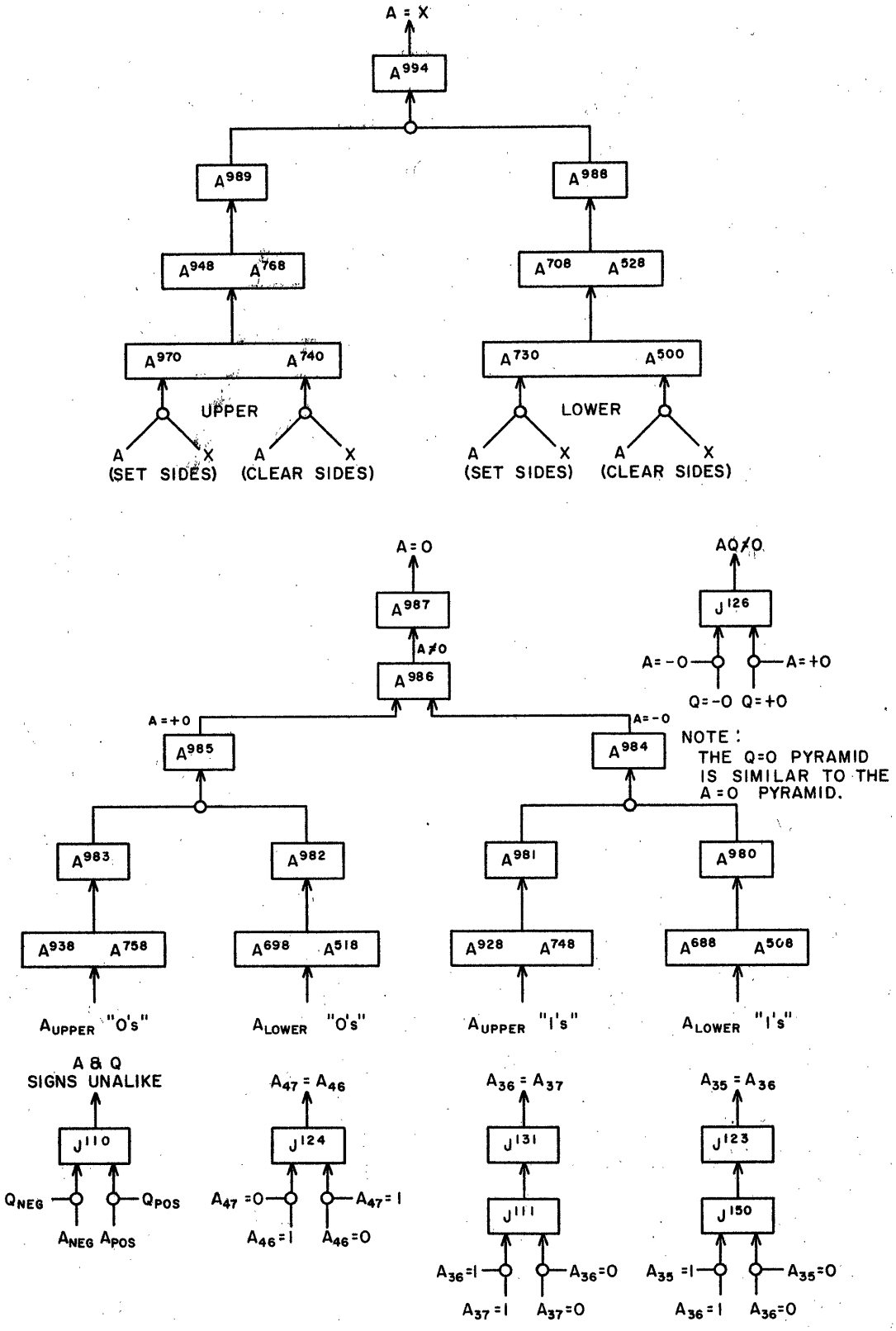
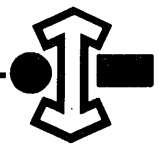



Figure 3-11. Register Sensing Networks.



of the A = 0 signal. The test for A = 0 is complicated by the instance when A = -0, or "all ones".

## ARITHMETIC FAULTS

The registers used in the arithmetic processes are monitored for fault conditions by four FFs; the error-checking circuits are shown in figure 3-12. The situations to which each FF responds are: (1) divide faults; (2) shift faults; (3) overflow faults; and (4) exponent faults. The FF recognizes the error condition immediately and lights an indicator on the console. An interrupt can be produced to notify the program of the occurrence of the fault if the program has previously selected interrupt on arithmetic fault by means of the 74.0 instruction. The interrupt program, which is initiated by the interrupt signal, examines each of the four arithmetic fault FFs to determine which type of fault has occurred.

Faults may also be detected by following an instruction which might produce one, such as a divide, with a 74.7 instruction having the code for sensing the Divide Fault FF.

### DIVIDE FAULTS

The divide fault occurs in the execution of the fixed-point divide instructions (25 and 27) if the quotient exceeds the capacity of the quotient register.

In division the operands are first set as positive quantities. (The sign of the quotient is established by the initial sign condition of the operands and stored by the Sign Record FF). Thus the quotient is initially positive and will be corrected later if it is to be expressed negatively. In testing for a divide fault the sign of the quotient is inspected before the final correction. At this time the sign bit is "1" only if overflow has occurred. Overflow will occur if the divisor is zero, or if an improper relationship exists between the dividend and divisor.

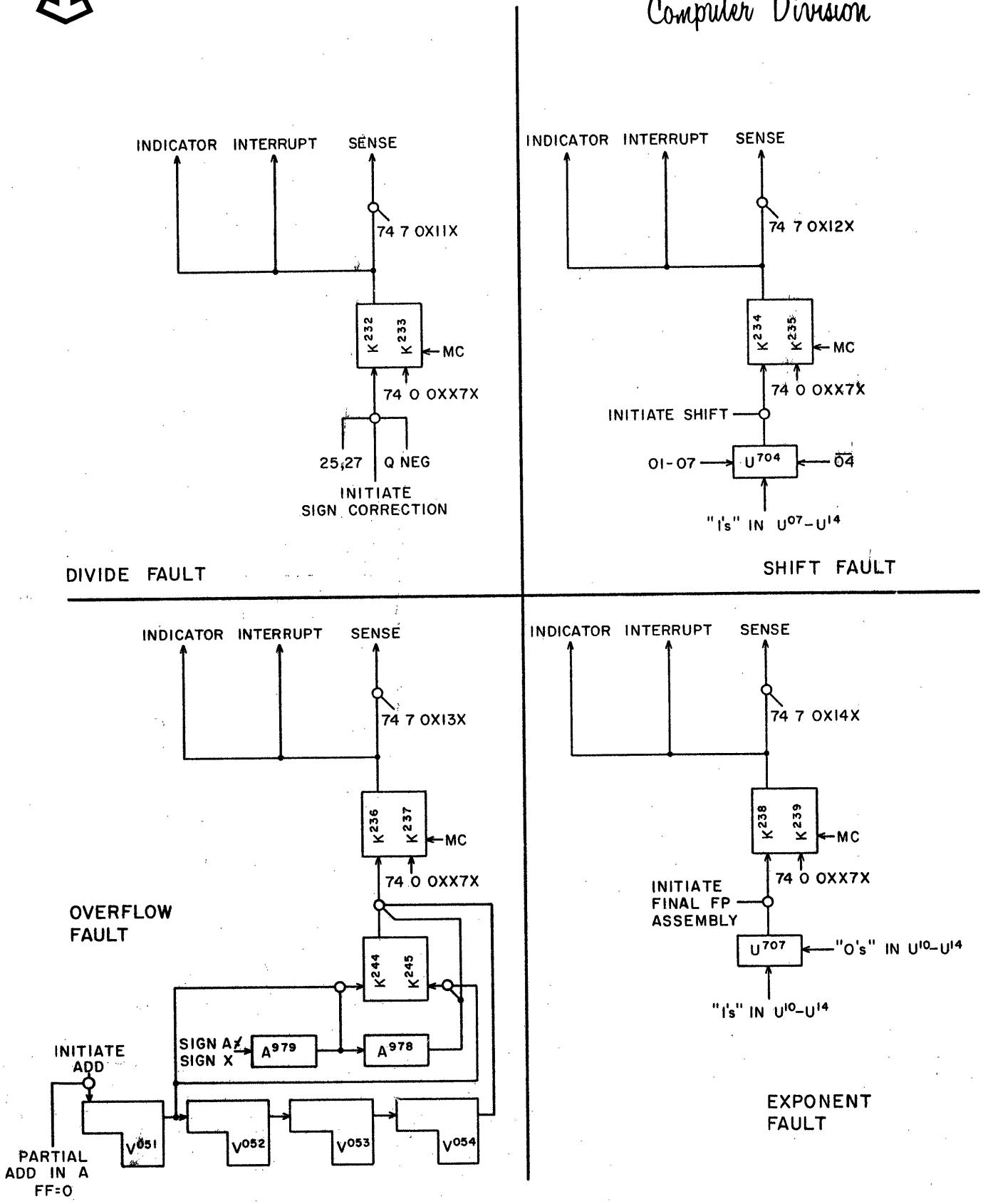
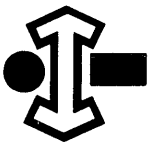



Figure 3-12. Arithmetic Faults.



Indication of a Divide Fault is provided by FF $K^{232/233}$, which is set by the combination of the following signals: $W^{773}$, which indicates that Q is negative; $F^{325}$, which is selected by instructions 25 and 27; and $V^{650}$, which indicates the quotient has been determined and the sign corrections have not been made.

SHIFT FAULTS

The shift instructions 01, 02, 03, 05, 06 and 07, provide for shifting the contents of the A, Q, or AQ registers to the right or left up to 127 (decimal) places. Any attempt to shift a register more than 127 places results in a shift fault.

The shift count is specified by the base execution address. Since 127 is the greatest number that can be coded in the last 7 binary positions, "1's" in positions greater than the $2^6$ position in $U^2$ indicate a fault. A "1" in any of the monitored stages attempts to set the Shift Fault FF $K^{234/235}$.

OVERFLOW FAULTS

Overflow faults occur when the sum or difference of two quantities exceeds the capacity of A. The possibility of overflow exists only after the addition or subtraction of two similarly-signed quantities. When two quantities of the same sign are to be added or subtracted a signal from $A^{979}$ sets FF $K^{244/245}$ at the initiation of the arithmetic. Since the sum or difference of two like-signed quantities retains the sign of its operands, any change in the sign of A indicates overflow.

Three phase times after the initial inspection of the signs of A and X a test is made for any difference in signs, if they were originally alike. This test is made at the AND on the set input to $K^{236/237}$. $K^{245}$ is present if the signs of A and X were originally alike; $V^{054}$ says

that the arithmetic has been done; $A^{978}$ indicates that the signs of A and X are unlike. Setting $K^{236/237}$ provides for the record of an overflow fault until such time as it is called for by the operator.

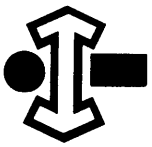
## EXPONENT FAULTS

The exponent fault occurs during the floating-point instructions when the exponent of the result, after rounding and normalizing, cannot be expressed as a quantity less than $2^{10}$. The 11-bit signed exponent for a floating-point operation is manipulated in the 15-bit registers R and U. The presence of any "1" bits in the upper stages of $U^2$ after the final determination indicates an improper exponent.

The exponent fault test is made by comparing the sign bit of the exponent, $U_{10}$, with the conditions of $U_{11}$ - $U_{14}$. Unless a fault has occurred, they must be the same.

## FAULT CONTROL

The Arithmetic Fault FFs are cleared by the master clear from the console or by the clear arithmetic faults code as ordered by the 74 0 00070 instruction. The cleared state represents the "no fault" condition. The presence of a fault condition can be sensed by the program, using the 74.7 instruction with the codes indicated in figure 3-12. Furthermore, the program can select to be notified by an interrupt each time a fault occurs. A 74.0 instruction with the code 00100 enables interrupts to be produced upon occurrence of any of the four faults.



# CHAPTER 4

# STORAGE SECTION

The storage section of the 1604 Computer is a large-capacity magnetic core system. It provides high-speed, non-volatile, random-access storage for 32, 768 48-bit words, and consists of two independent magnetic core storage units each with a capacity of 16, 384 48-bit words. These units operate together during the execution of a stored program and thus may be logically considered as a 32, 768 word storage system. All odd storage addresses reference one storage unit; all even addresses, the other.

A word is transferred to or from a storage location by a single instruction. The operation code of the instruction specifies the type of reference (read or write) and the register which serves as the source or destination. The execution address of the instruction identifies the storage location involved. A read reference is performed by transferring the word at a selected storage address to a specified destination via the X register, and restoring the word at the address. A write reference is performed by clearing the selected storage address, then transferring the word at a specified source to the address via the X register.

The cycle time, or time for a complete storage reference, is 6. 4 microseconds. The access time, or time from request to delivery of data from storage, is 2. 2 microseconds. The storage cycles of the two sections overlap one another to a considerable extent in the execution of a program and result in an effective cycle time considerably less than 6. 4 microseconds.

The basic logical divisions of the storage section are shown in block representation in figure 4-1. The odd and even storage units are identical, each consisting of four principal parts:

1) Memory plane assembly, containing the magnetic core storage elements of the system

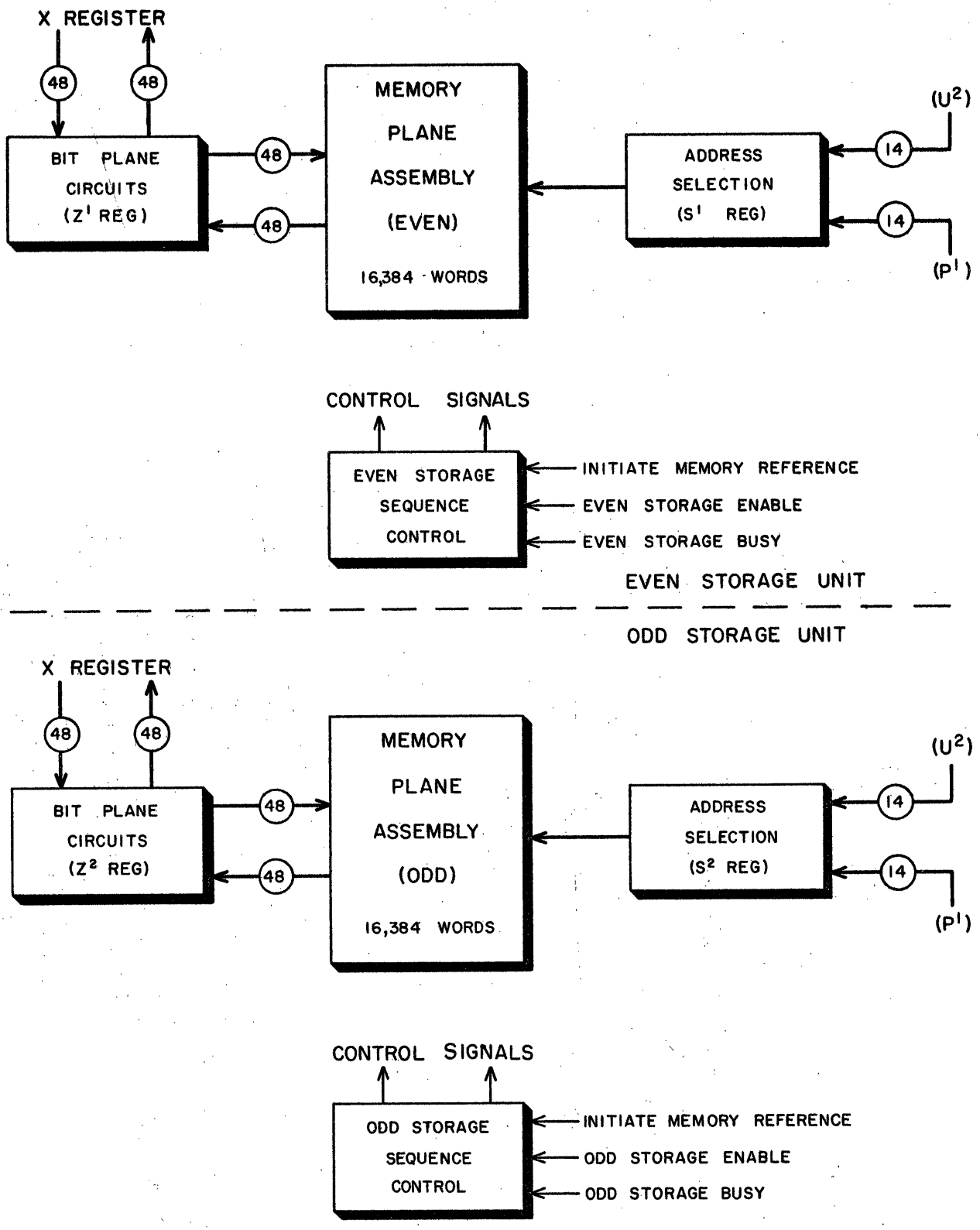
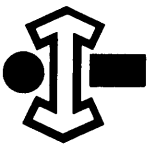



Figure 4-1. Logical Divisions of the Storage Section.



2) Address selection, which interprets the address from the control section and selects the specified storage location
3) Bit plane circuits, which handle the transfer of information between the memory plane assembly and the X register
4) Storage sequence control, which generates the signals that control the storage references

The magnetic core assemblies of the storage section, along with their associated addressing, driving and control circuitry, are evenly distributed among the eight chassis of the central computer, occupying approximately one-half of each chassis. Data transmissions into storage are channeled through $Z^1$ and $Z^2$, the storage restoration registers. The storage address registers, $S^1$ and $S^2$, hold the address of the storage location involved in a given cycle of operation. The input-output, arithmetic and control sections of the computer have independent access to the storage registers by utilizing the appropriate Z and S registers.

## PRINCIPLES OF MAGNETIC CORE STORAGE

The storage section uses the permanent magnetic properties of ferrite cores to store the bits of computer words. A magnetic core is a bistable device capable of storing a "1" or a "0", depending upon its state of remanent magnetization. The cores are General Ceramics type MC-113, size F-394, and are composed of S-4 ferrite material.

These cores are magnetized in one direction or the other by current-carrying wires which pass through them. The direction of magnetization is determined by the direction of the current flow. The characteristics of the cores are such that approximately 800 ma of current in one turn for a period of one microsecond is required to switch them.

The cores are assembled in square matrices, as shown in figure 4-2. Five wires pass through each core: a horizontal H wire, a vertical V wire, a horizontal I wire, a vertical I wire and a diagonal S wire. The coincident-current switching technique is employed. A core is addressed by simultaneously passing half-amplitude current pulses through a selected V wire and a selected H wire.

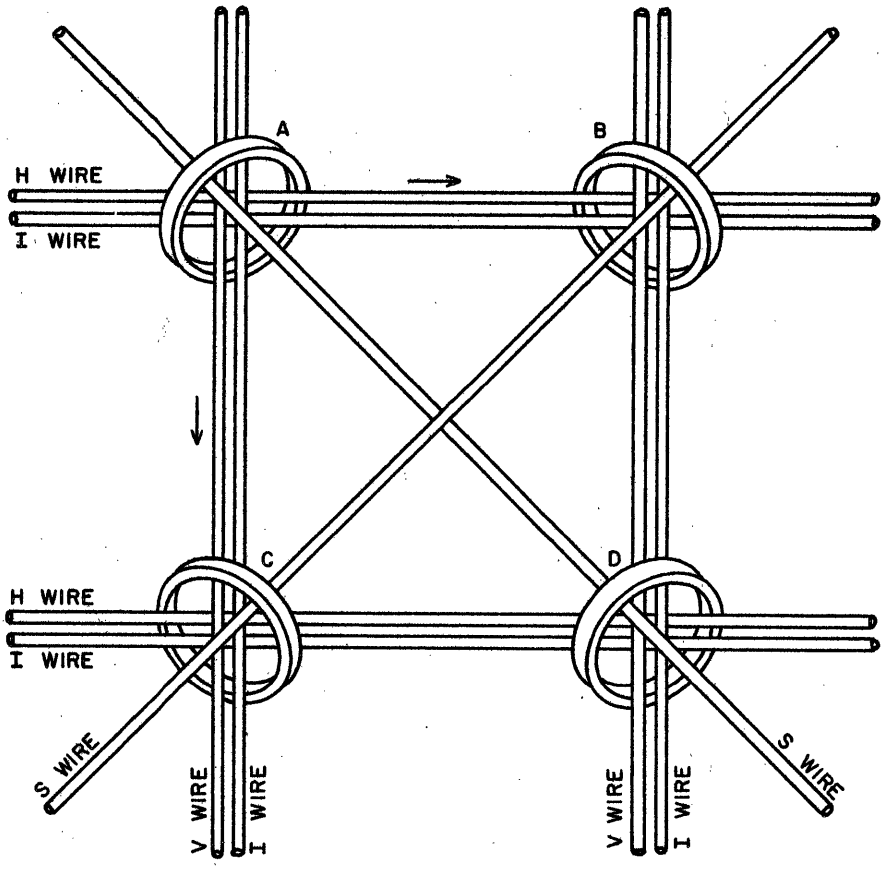
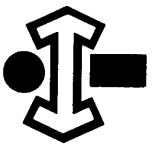



Figure 4-2. Magnetic Core Matrix

Only one core among all those in the memory plane will be subjected to a magnetizing force sufficiently large to switch its magnetic state. This core is at the intersection of the selected H and V wires. All other cores in the same row or column as the selected core receive half-amplitude current pulses and are said to be half-selected. In figure 4-2, where the left-most V wire and the upper H wire carry current pulses, core A is



selected, cores B and C are half-selected, and core D is unselected.

Binary information stored in a core is determined by the polarity of its residual magnetization. A "O" is stored by the magnetizing force of read current pulses on the selected H and V wires and remains stored when current in the I wires inhibits (cancels) the effect of write current pulses in the H and V wires. The absence of current in the I wires permits a "1" to be stored by the write pulse.

The information is extracted (read) from the core by applying read current pulses to the selected H and V wires. If the core stored a "1", the pulses drive it to "O" and a pulse is induced in the S wire. This voltage is interpreted as a "1" bit from the core. If the core stored a "O", it is unaffected by the pulses and no pulse is induced in the S wire. The absence of a voltage is interpreted as a "O" bit from the core.

The matrix arrangement of magnetic cores which is the basic storage unit of the storage system is called a memory plane. This consists of 16, 384 cores in a 128 x 128 array. In order that the coincident-current storage technique may be employed, each bit of a word must be stored on a separate memory plane. Thus, the 48-bit words used in the computer are stored by 48 memory planes. This array is called a memory plane assembly.

There are two memory plane assemblies in the storage section, one associated with the odd storage unit and the other with the even storage unit. Both assemblies are evenly divided among the chassis of the computer. Thus, a stack of six "even" and six "odd" memory planes is mounted on each chassis. A photograph of a memory plane stack is shown in figure 4-3.

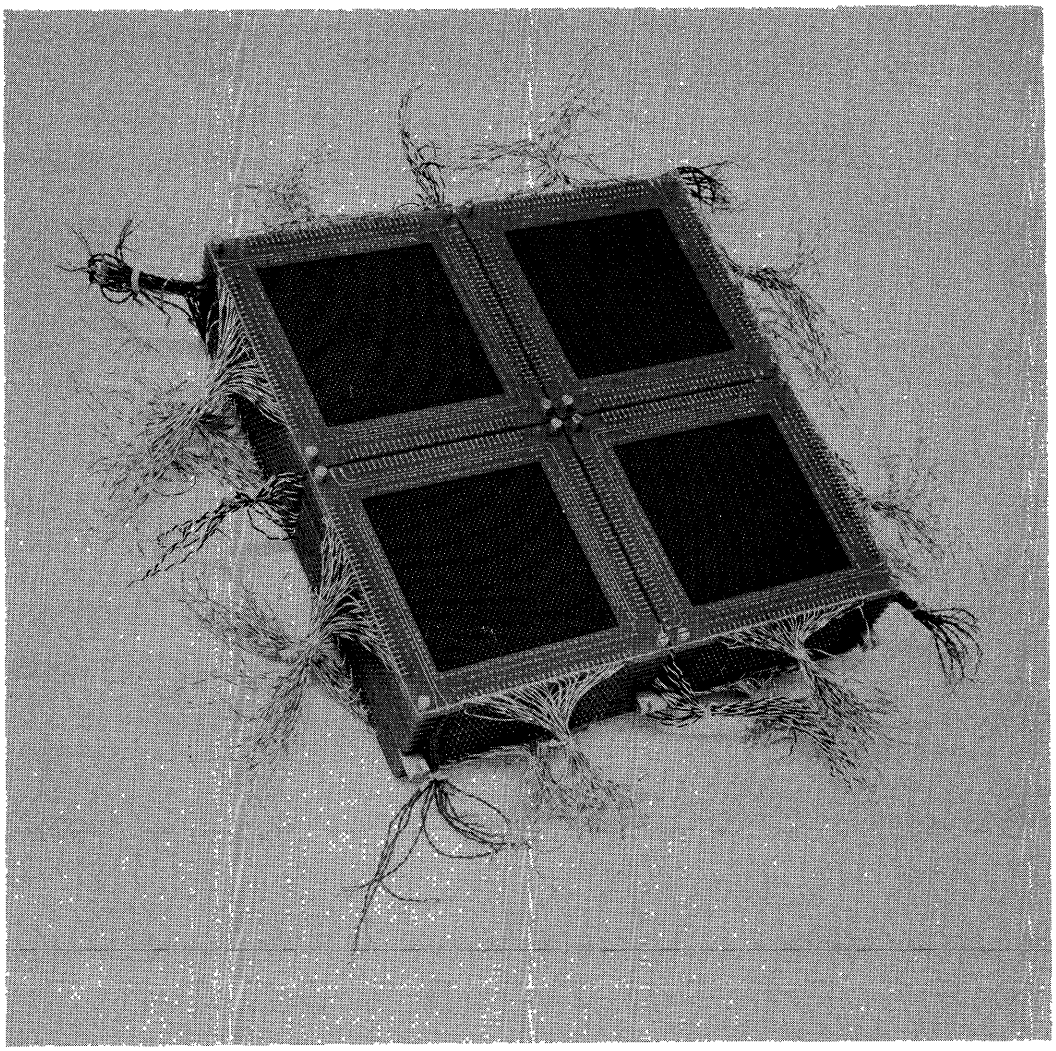


Figure 4-3. Memory Plane Stack

## MAGNETIC CORES

The magnetic properties of a core are represented by its hysteresis diagram, which plots magnetic flux density (B) as a function of the field intensity (H). A typical hysteresis diagram is shown in figure 4-4. If current flow sufficient to cause a field intensity of $+H_m$ is applied to the drive lines, the flux density increases to saturation ($+B_s$). When the current is removed, the flux density drops to the residual positive value ($+B_r$), which has been designated the "O" state, and remains there. Another pulse of $+H_m$ would merely shift the core to $+B_s$ again and after the pulse is removed, it would drop back to $+B_r$.

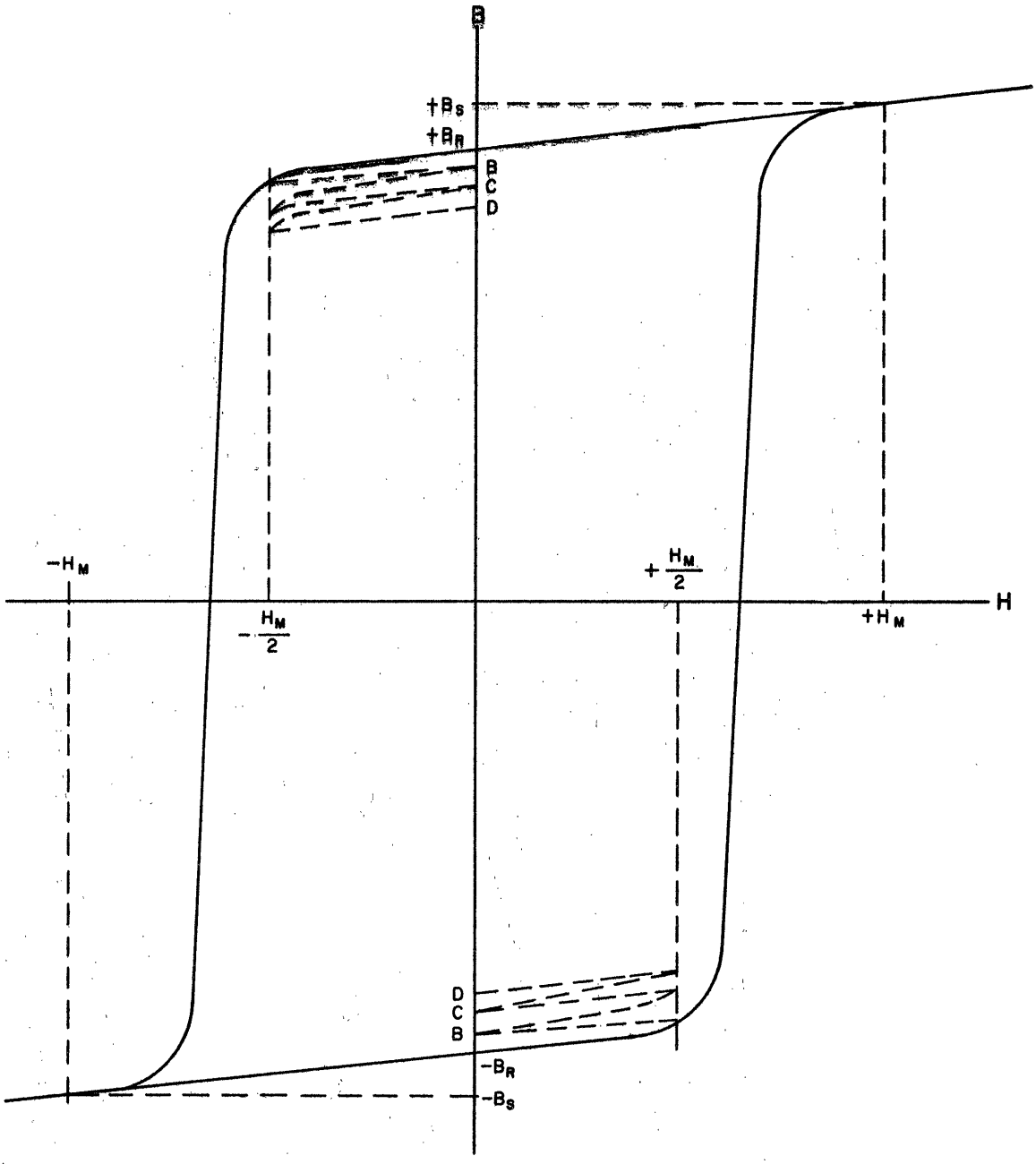



Figure 4-4. Typical Hysteresis Diagram.

Application of current flow sufficient to cause a field intensity of $-H_m$ reverses the flux density to $-B_s$ and, when the current is removed, the flux density drops to the residual negative value ($-B_r$), the "1" state.

The basic memory cycle is composed of half-amplitude pulses, each capable of producing a field intensity of $H_m/2$. A half-amplitude pulse is insufficient to switch the core; instead the flux density returns to the residual value, or a slightly lower value, after the pulse is removed. The coincidence of two half-amplitude pulses, one on the H drive line and the other on the V drive line of a core, produces a net field of $H_m$ which is sufficient to switch the core. When a half-amplitude pulse drives the flux density toward the knee of the hysteresis loop, the flux travels up (or down) the knee somewhat and then returns to a slightly lower residual value, such as B. Since the core is now operating on a smaller loop, further half-pulses reduce this remanent flux again, but this effect soon reaches a limit, as at point D.

Any change in the magnetic state of a core causes a change in the total flux linking the core and any winding passing through it. Such a change produces a voltage output on the sense winding (figure 4-5). During the period that H is applied, the voltage is sampled to see if the core switches. If a large voltage is sensed, the core was in the "1" state and has switched. If only a small voltage is sensed, the core was in the "0" state and has merely shifted from $+B_r$ to $+B_s$ and back again.

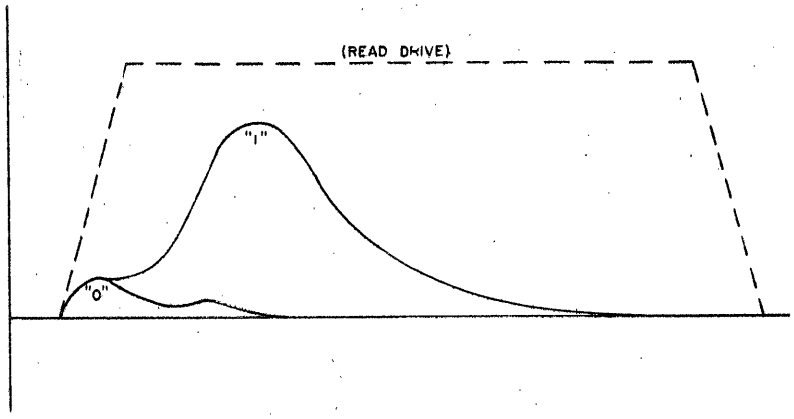
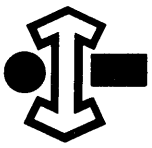



Figure 4-5. Voltage on Sense Winding as a Result of Read Drive



## MEMORY PLANES

A memory plane consists of four quadrants or boards, numbered 0, 1, 2 and 3, as indicated. Each board consists of a phenolic frame with printed circuit wiring on both sides. The board holds 4096 cores, held in a 64 x 64 array by 64 horizontal H wires and 64 vertical V wires. One I wire threads all the cores both horizontally and vertically and one S wire threads all the cores diagonally.

A memory board is shown in figure 4-6. The H and V wires thread across the board vertically and horizontally and terminate at tabs on either side of the edge of the board. Examination of figure 4-6 shows that a wire connected to a front tab on one edge of the board terminates on a rear tab on the opposite edge. This feature allows close spacing of the wires. The ends of each I wire terminate at two tabs in a corner of the board, from which connections are made to the inhibit drivers.

The ends of each S wire terminate at a pair of tabs in a second corner of the board, from where connections are made to the sense amplifiers. The four boards of a memory plane are so oriented that the sense wire tabs are at the corners of the memory plane. Corresponding H wires and corresponding V wires of adjoining boards are connected together by short lengths of wire.

## MEMORY PLANE ASSEMBLY

Twelve memory planes, stacked one behind another, make up a stack. (See figure 4-3.) The memory planes are held together by bolts passing through the four corners of each and are separated by aluminum spacers. An aluminum plate at the back and a plexiglass plate at the front shield the stack. A memory plane assembly is made up of six memory planes from each of eight such stacks, or a total of 48 planes. Since each plane stores one bit of a word, and there are 16, 384 cores on a plane, the memory plane assembly provides storage for 16, 384 48-bit words.

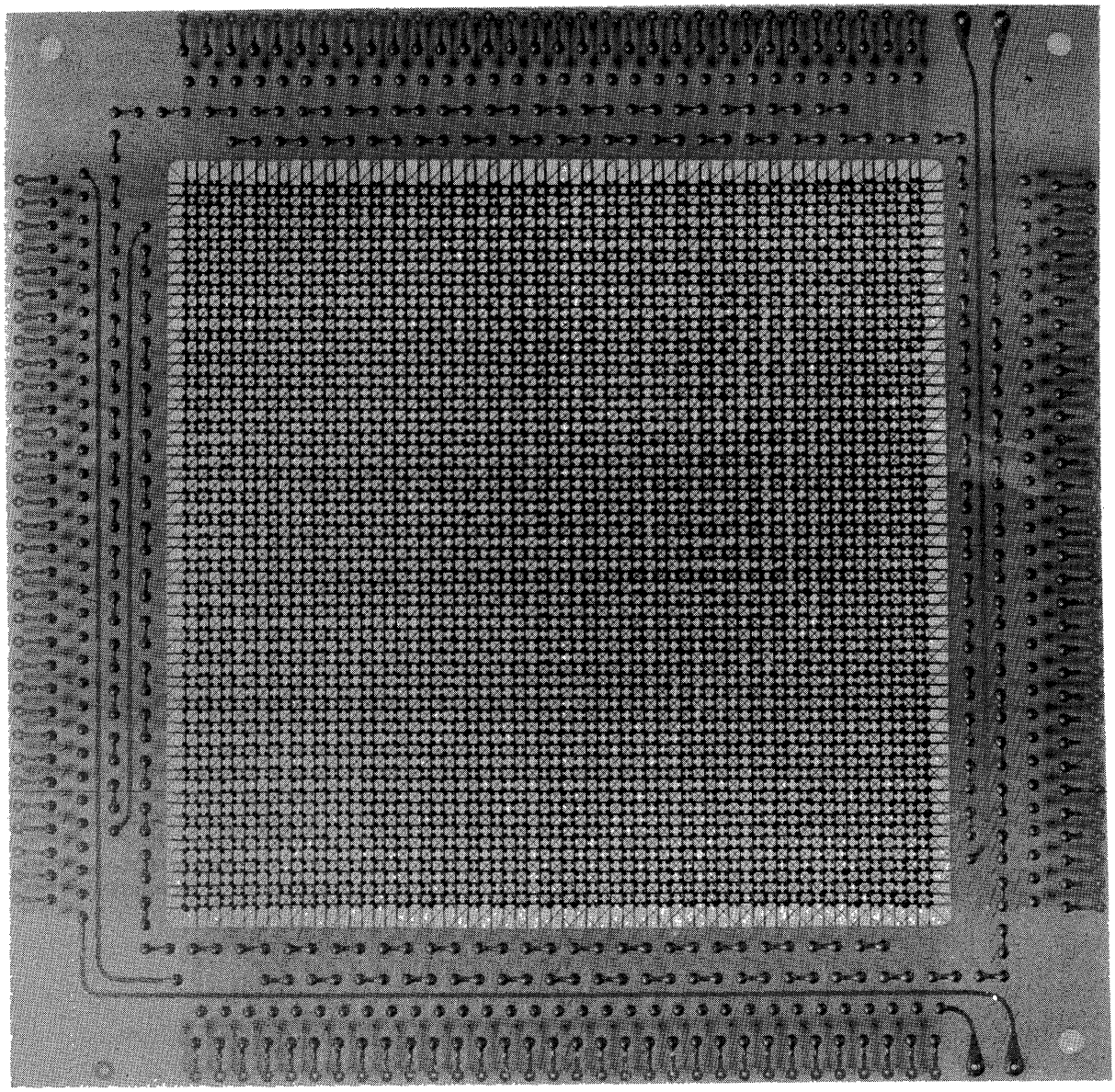


Figure 4-6. Memory Board

The storage section contains two memory plane assemblies, one for the odd storage unit and one for the even one. (Thus, the total storage capacity is 32,768 words.) The memory planes of each are evenly distributed among the eight computer chassis, as shown in figure 4-7. The stack of chassis 1 stores bits 0 through 5 of all the words; the stack of chassis 2 stores bits 6 through 11, etc. In each case, the memory planes associated with the even memory unit are on the card side of the chassis and the memory planes associted with the odd memory unit are on the wiring side.

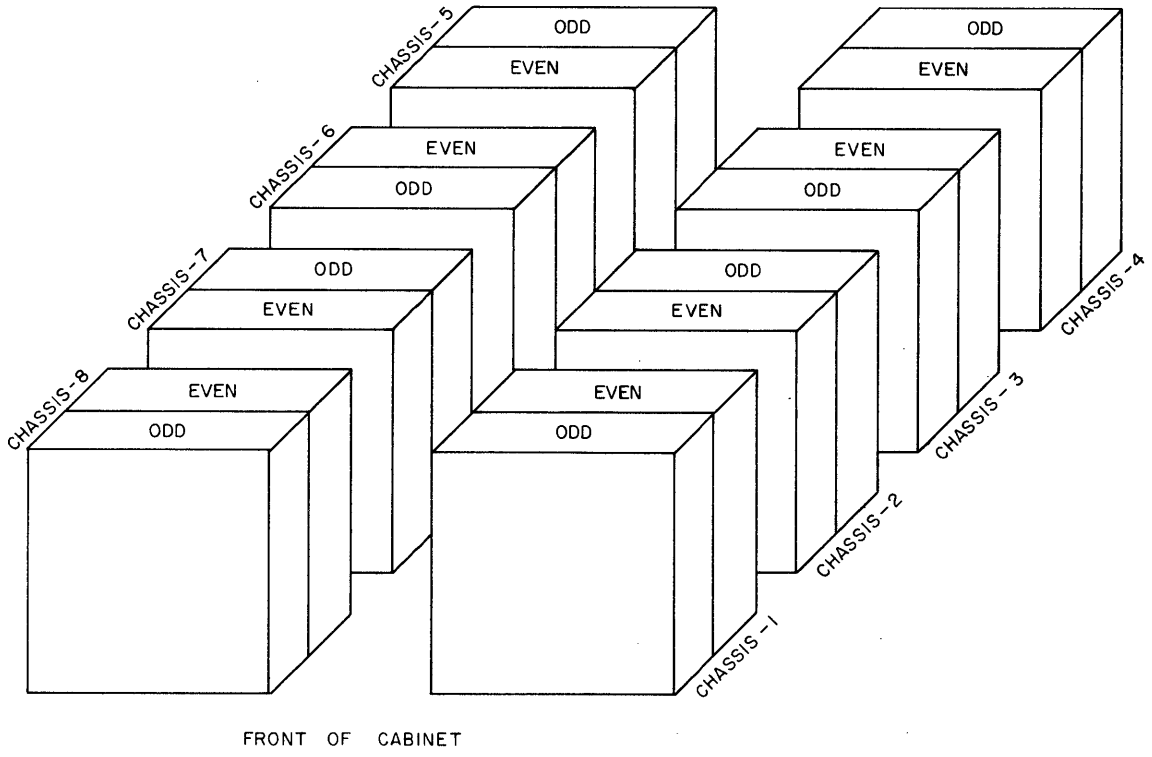



Figure 4-7. Distribution of Odd and Even Memory Plane Assemblies Among Chassis of Computer

Corresponding H wires and corresponding V wires of each group of six memory planes of a stack are connected in series by short lengths of wires soldered to the tabs at the edges of the boards. The H and V wires of each of the eight such groups of memory planes which form a memory plane assembly are connected to separate driver and diversion circuits. In each case, these circuits are located on the same chassis as the memory planes

with which they are associated. The corresponding drivers and diverters of the even memory are connected in parallel to the $S^1$ register, located on Chassis 3; those of the odd memory unit are connected in parallel to the $S^2$ register, located on Chassis 4. Thus, a complete memory plane assembly is the logical equivalent of 48 memory planes, stacked one behind another.

As shown in figure 4-8, which is a simplified representation of a memory plane assembly, the cores in each horizontal plane are effectively connected in series by an H wire. Similarly, the cores in each vertical plane are effectively connected in series by a V wire. Coincident currents on a selected pair of H and V wires affect only those cores at the intersection of the horizontal and vertical planes formed by those wires.

Separate connections are made to the sense and inhibit lines of each quadrant of a memory plane. Each sense line is brought to a pair of tabs at a corner of the memory plane. These tabs are connected to the sense amplifier circuits by twisted pairs. Each inhibit line is brought to a pair of tabs which is located approximately half way between two of the corners of the memory plane. The tabs are connected to the inhibit circuits by twisted pairs.

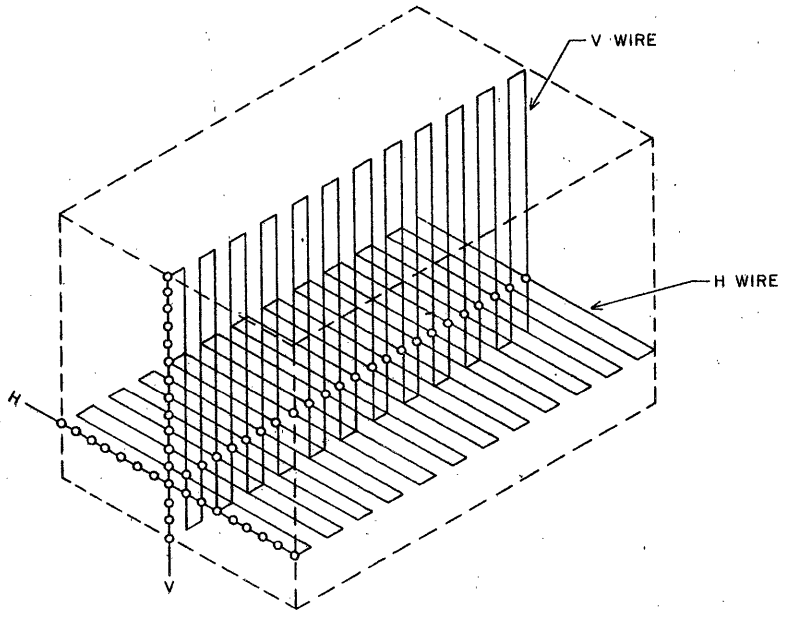
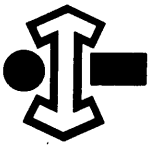



Figure 4-8. Intersection of H and V Wires in a Simple Memory Plane Assembly



## ADDRESS SELECTION

A storage location has been defined as the row of cores at the intersection of a selected pair of H and V wires of a memory plane assembly. Each memory plane assembly thus has 16, 384 storage locations. It is the function of the address selection circuit to select a discrete storage location for each different address in $S^1$ or $S^2$.

Exactly 15 bits are required to separately identify each of the 32, 768 storage locations. The identifications, called addresses, range from 00000 to 77777 inclusive, expressed in octal notation. Two address selection systems, one associated with the odd storage unit and the other with the even storage unit, are provided to interpret the address involved in a storage reference and select the appropriate "odd" or "even" storage location. The lowest-order bit of the 15-bit address selects either the odd or even address selection system. The remaining 14 bits are placed in the S register of the selected system and translated to select a single H wire and a single V wire of each matrix plane of the memory plane assembly.

The address selection circuits for the odd and even storage units are identical. Each is composed of four logical systems:

1) The vertical drive system, consisting of a translator and eight drivers for each of the storage units on a chassis, selects a group of 16 V wires, among which is the desired V wire, and supplies the necessary drive current for this wire.
2) The horizontal drive system, consisting of a translator and eight drivers for each of the storage units on a chassis, selects a group of 16 H wires, among which is the desired H wire, and supplies the necessary drive current for this wire.
3) The vertical diversion system, consisting of a translator and 16 diverters for each of the storage units on a chassis, selects the desired V wire from the group of V wires selected by the vertical drive system.

4) The horizontal diversion system, consisting of a translator and 16 diverters for each of the storage units on a chassis, selects the desired H wire from the group of H wires selected by the horizontal drive system.

The address selection system logically divides each stack into eight vertical and eight horizontal sections, as illustrated in figure 4-9. Each horizontal section consists of 16 H wires; each vertical section consists of 16 V wires.

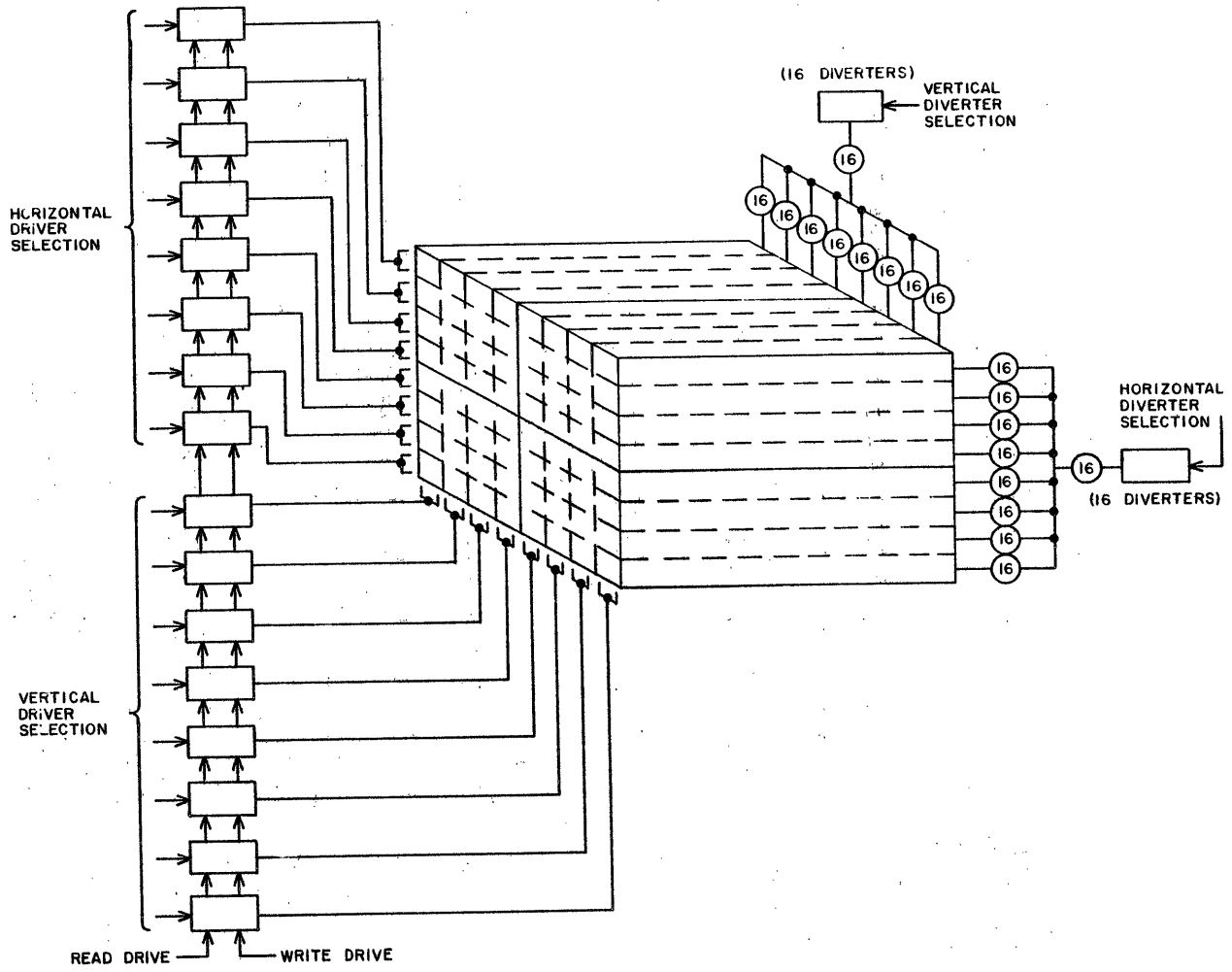
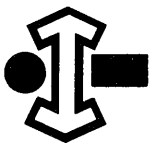



Figure 4-9. Connection of Drive Lines and Diversion Lines to the Odd or Even Portion of the Stacks

The region generated by the intersection of an H section and a V section and common to both is called a quarter section. One end of each of the wires in a horizontal section is connected to a common horizontal driver; similarly, one end of each of the wires in a



vertical section is connected to a common vertical driver. Thus, the selection of a horizontal and vertical driver by the respective translators selects a region of 256 storage locations, one of which is the desired one. The other end of the H wires are connected to the diverters; corresponding H wires of each horizontal section are connected to a single diverter. The selection of a horizontal and vertical diverter completes the electrical circuits to the selected horizontal and vertical drivers, allowing current to flow from the drivers through the respective H and V wires, through the diverters, and back to the current source.

As an aid to maintenance, the selection of drivers and diverters by consecutive storage addresses follows an orderly pattern. The bits of a storage address are combined, for selection purposes, in the pattern shown at the top of figure 4-10. The lowest-order bit specifies the odd or even memory unit; the next two higher-order bits specify the quadrant within that unit. Thus, the three lowest-order bits divide each stack into eight portions, and are termed the "octant selection bits". The next six higher-order bits of the address select the vertical drivers and diverters in the octant while the six highest-order bits select the horizontal drivers and diverters in the octant.

The selection of a vertical driver is given by bits 1, 8, 7 where bit 1 is treated as the highest-order one. Similarly the selection of a horizontal driver is given by bits 2, 14, 13 where bit 2 is treated as the highest-order one.

The ordered selection of drivers and diverters within an octant is shown in the lower part of figure 4-10. The selection of vertical diverters proceeds from left to right, in repetitive cycles, at 1/8 the frequency of address change. The selection of vertical drivers proceeds from left to right, in repetitive cycles, at 1/128 frequency of address change. Horizontal diverters and drivers are selected in the same manner but at 1/64 the rate of the vertical.

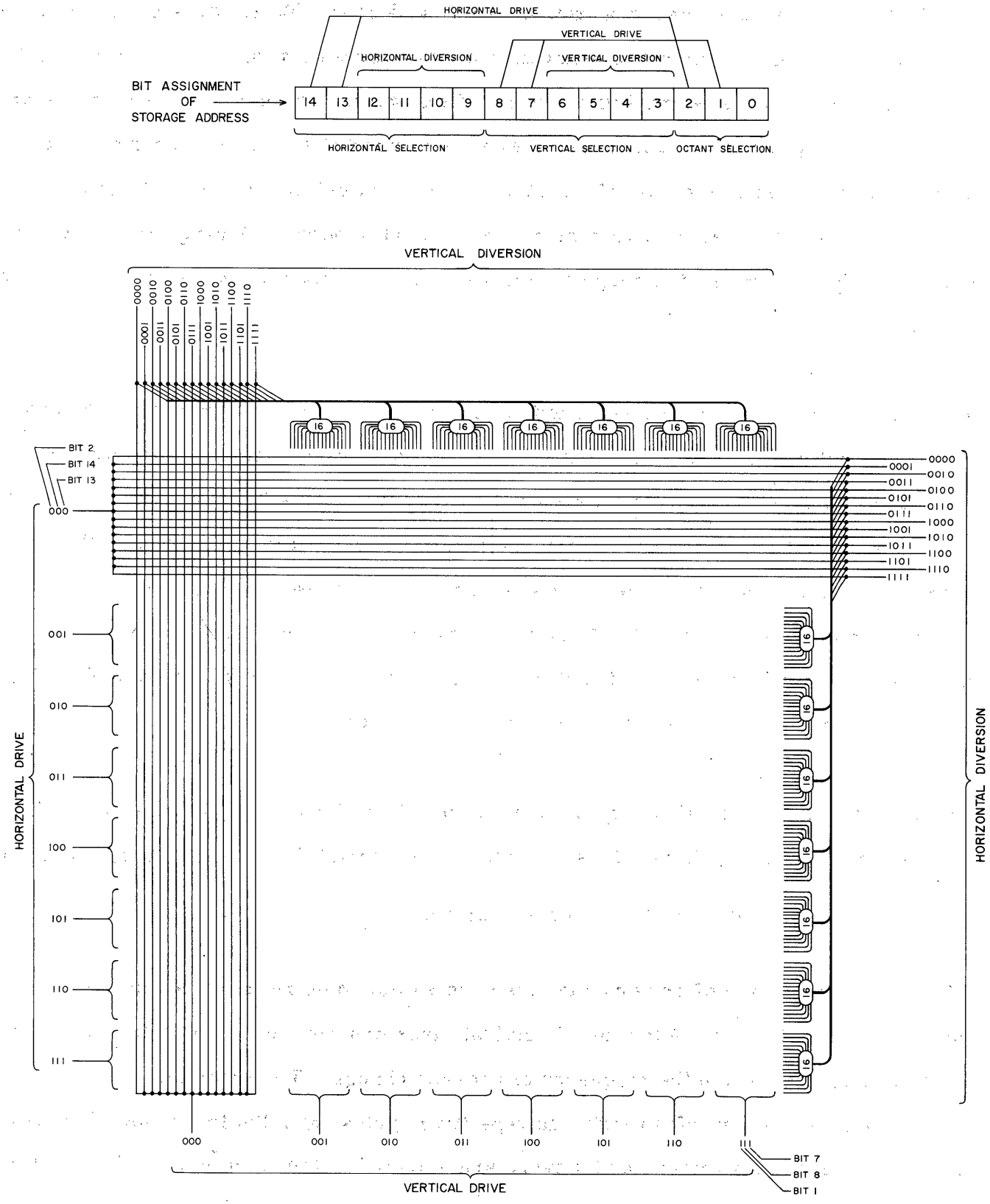
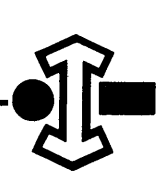
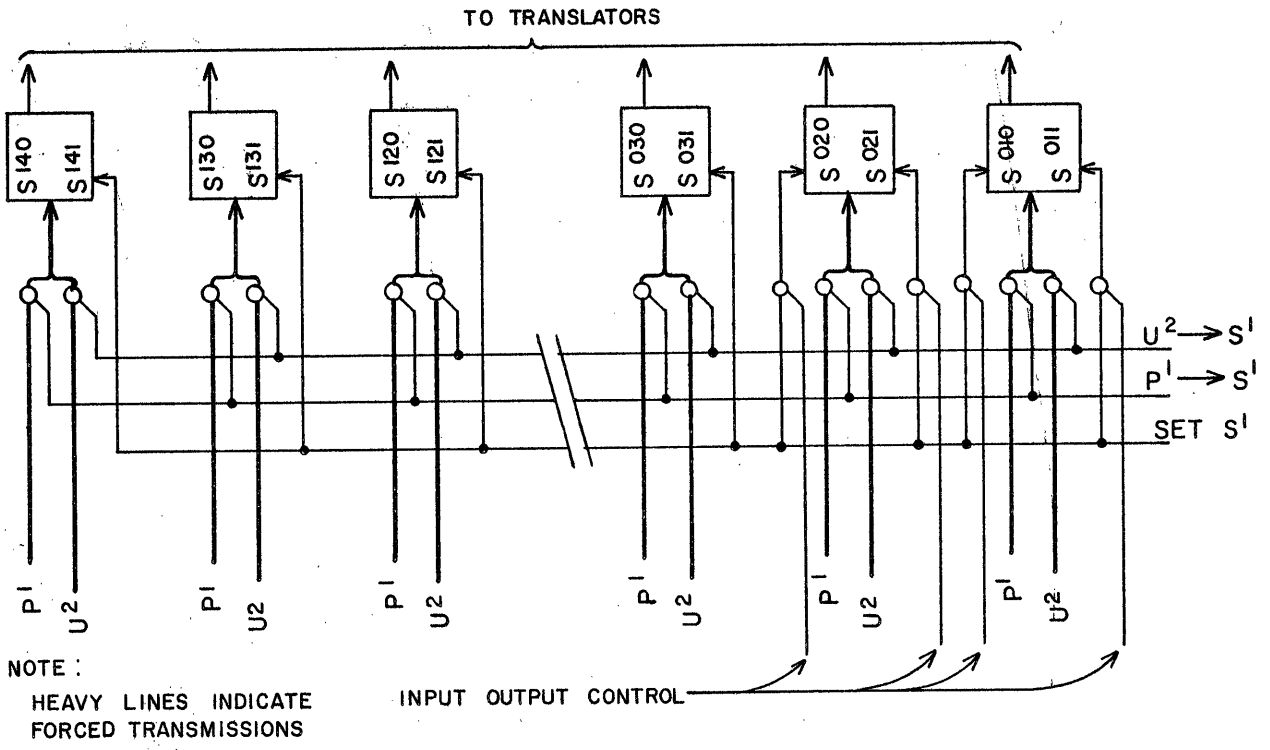



Figure 4-10. Selection of Drivers and Diverters by $S^1$ or $S^2$ Registers.

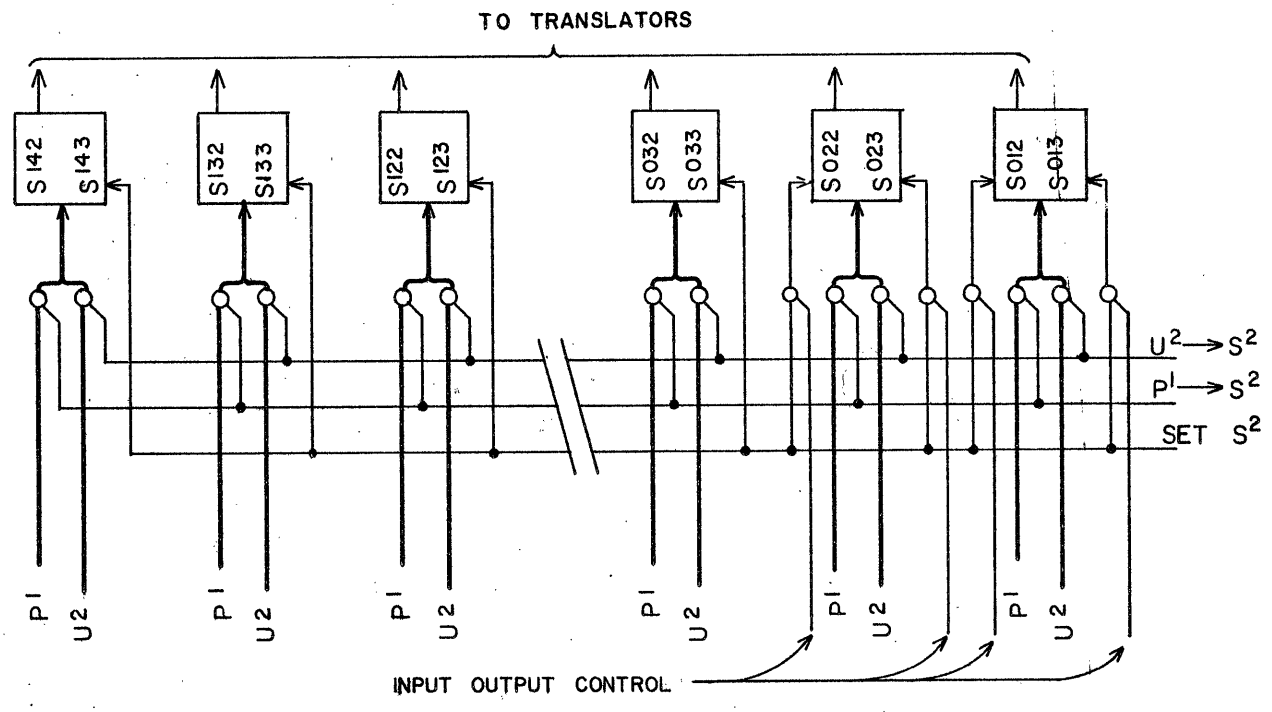


S REGISTER

The $S^1$ and $S^2$ registers, shown in figure 4-11 hold the storage address during a storage reference. Each register consists of 14 single-rank FF stages and has no properties other than storage. An address is entered into one of the registers by one of three commands. The command $U^2 \rightarrow S^1$ (or $S^2$) enters the quantity stored in $U^2$ into S; the command $P^1 \rightarrow S^1$ (or $S^2$) enters the quantity in $P^1$ into S. The command Set $S^1$ (or $S^2$) inserts one of eight addresses (00000 through 00007) into the appropriate S register during buffer operations. The $U^2 \rightarrow S$ and $P^1 \rightarrow S$ transfers are forced transmissions; i. e., they transfer both outputs of a stage of $U^2$ or $P^1$ to the corresponding stage of S.




S-1 Register (Even Memory)




S-2 Register (Odd Memory)

Figure 4-11. S-1 and S-2 Registers.

## HORIZONTAL AND VERTICAL DRIVERS

There are eight vertical and eight horizontal driver circuits for the odd memory plane assembly and the same number for the even on each chassis. The driver circuits are identical; a typical circuit is shown in figure 4-12. The vertical driver circuits are connected in parallel to stages 01, 07 and 08 of the appropriate **S** register; the horizontal driver circuits are connected in parallel to stages 02, 13 and 14 of the appropriate **S** register.

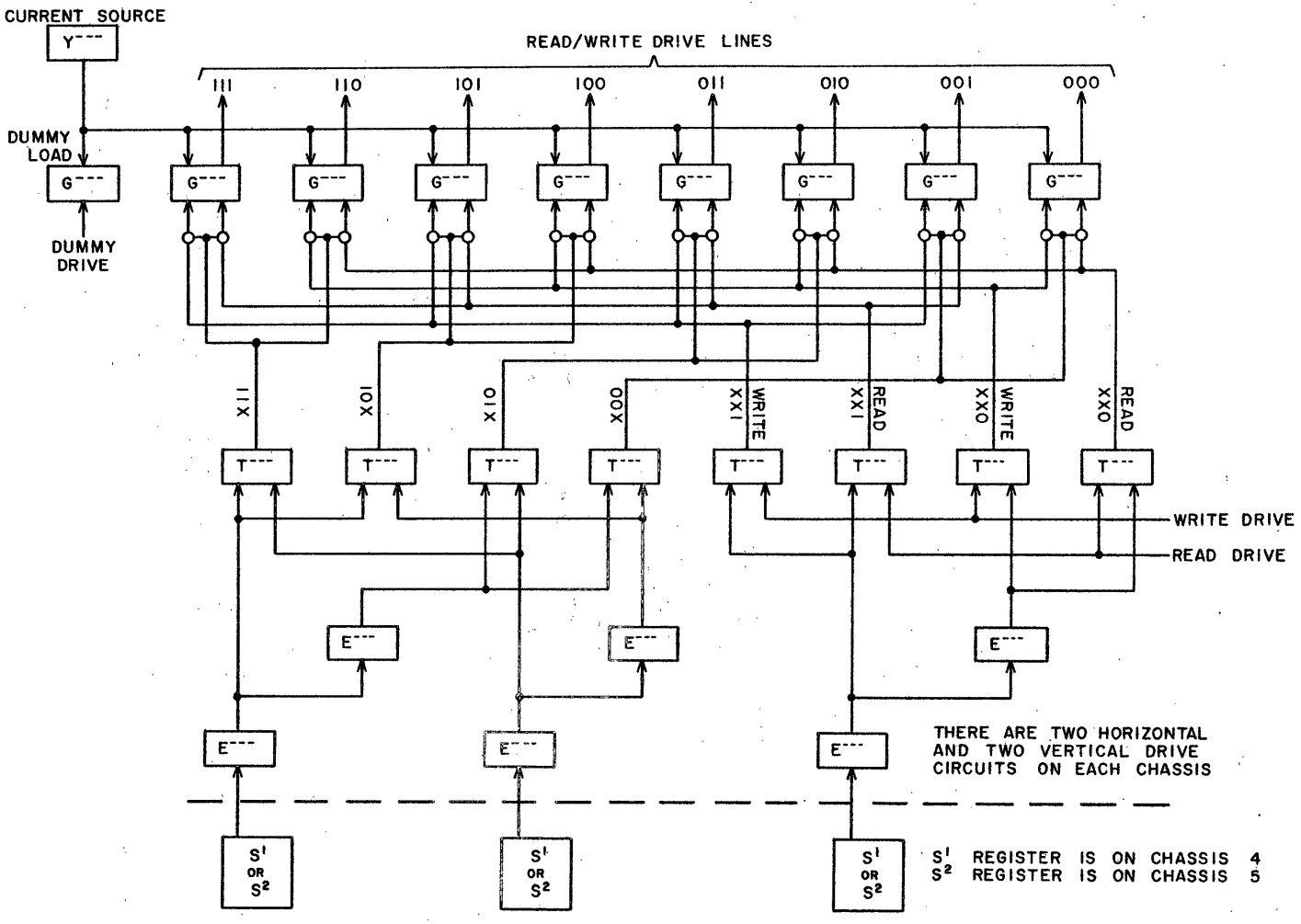



Figure 4-12. Typical Horizontal or Vertical Drive Circuit

The translator associated with each driver circuit selects one of the eight drivers (designated $G^{---}$ in figure 4-12) on the basis of the contents of the three stages of S. The translation is performed by single-inverter cards (designated $E^{---}$) and selector cards (designated $T^{---}$). A pair of $E^{---}$ cards is so connected to each FF that one provides a "O" output when the FF stores "1" and the other a "O" output signal when the FF stores "O". The outputs of the $E^{---}$ cards, along with the signals Read Drive and Write Drive from the storage sequence control, are applied to the $T^{---}$ cards in such combinations that the following output signals are produced by the $T^{---}$ cards: Read "XX0", Write "XX0", Read "XX1", Write "XX1", "00X", "01X", "10X" and "11X". In each case, the letter "X" represents a bit of S which does not affect the output of the $T^{---}$ cards.

The outputs of the $T^{---}$ cards are combined at AND inputs to the $G^{---}$ cards in such a manner that a read output is applied to the one AND of each card and a write output to the other, and each is ANDed with one of the other four outputs. Thus, for any combination of bits in the three stages of S, one of the $G^{---}$ cards is selected, and this card provides both read and write currents to the memory plane stack.

The current source card (designated $Y^{---}$) is the source of d-c from which the selected driver card draws the read-write current. To maintain a constant load on the power supply, a $G^{---}$ card is connected as a dummy load to the current source card. During periods of no storage references, the dummy load is continuously selected; during a storage reference the dummy load is disabled for a period of 6.2 microseconds by the Dummy Drive signal. (Note: The period during which the dummy load is disabled need not coincide exactly with the period during the storage reference when the driver cards are selected, since a-c fluctuations in the load do not adversely affect the power supply.) The Dummy Drive signal originates at the storage sequence control.

## HORIZONTAL AND VERTICAL DIVERTERS

Sixteen horizontal and sixteen vertical diverter circuits are associated with each stack of the odd and even memory units. The diverter circuits are identical; a typical circuit is shown in figure 4-13. The vertical diverter circuits are connected in parallel to stages 03, 04, 05 and 06 of the appropriate S register; the horizontal diverter circuits are connected in parallel to stages 09, 10, 11 and 12.

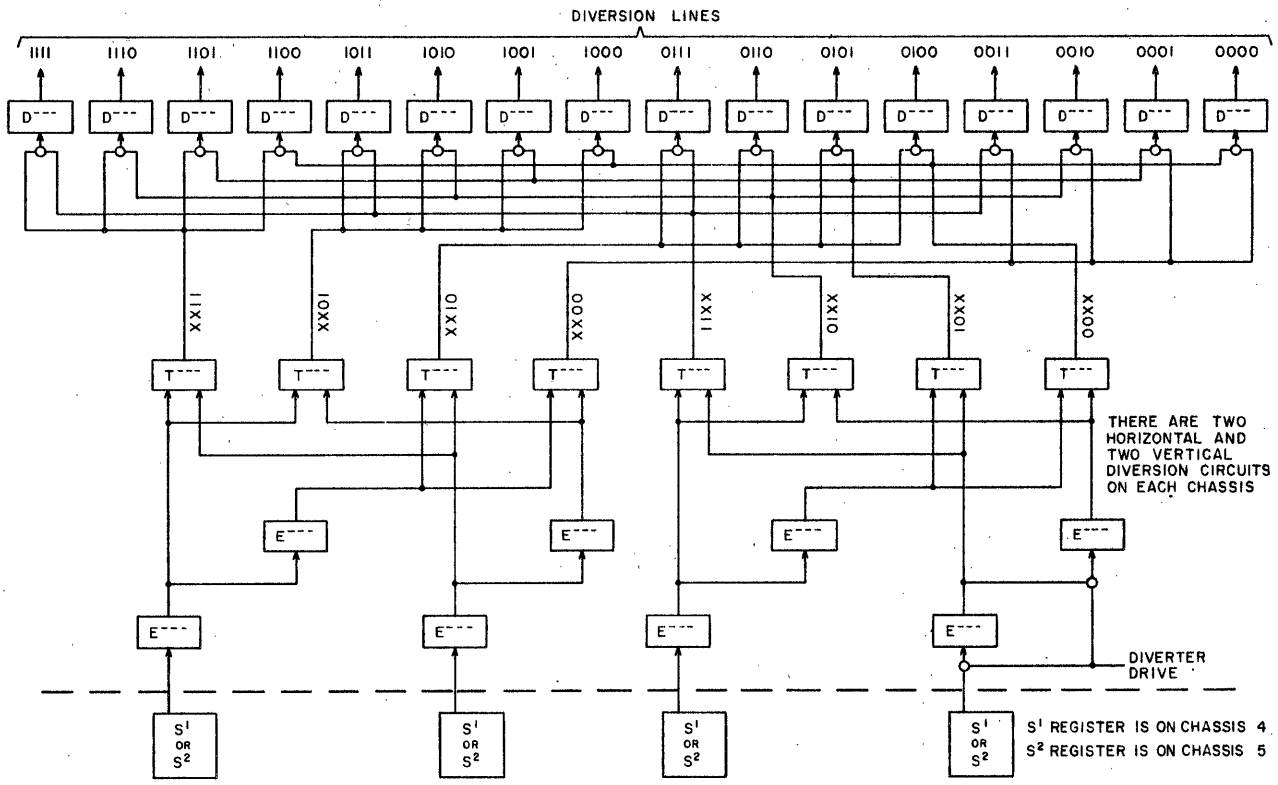
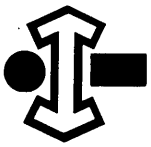



Figure 4-13. Typical Horizontal or Vertical Diversion Circuit

The translator associated with each diverter circuit selects one diverter card (designated $D^{---}$ in figure 4-13) on the basis of the contents of the four stages of S. The translation is similar to that described for the driver circuits. The $E^{---}$ cards provide positive "1" and "O" outputs on the basis of the information stored in the FFs. The outputs from the $E^{---}$ cards are applied to the $T^{---}$ (selector) cards in such combinations that the following negative output signals are produced by the $T^{---}$ cards: "XX00", "XX01", "XX10", "XX11", "00XX", "01XX", "10XX" and "11XX". The outputs of the $T^{---}$ cards are combined at the inputs of $D^{---}$ cards in such a manner that, for any combination of bits in the four stages of S, only one diverter card is selected. The card completes the current path for a single H or V wire within the group of 16 such wires fed by the selected driver.



## BIT PLANE CIRCUITS

The bit plane circuits implement data transfer between storage and the rest of the computer. Data flow out of storage takes place during the first half of the storage cycle. The states of the selected cores are sampled by detecting the voltage induced in the sense windings when the read drive pulse clears the cores to the "O" state. Those cores which stored "1" bits produce pulses on the respective sense windings which are amplified by the sense amplifiers. During read operations, the Read signal, in combination with the appropriate quadrant selection signal, gates the pulses from the sense amplifiers to the $I^5$ (even storage unit) or $I^6$ (odd storage unit) rank of inverters. (During write operations, the Write signal gates the word in $X^2$ into $I^5$ or $I^6$.) The Set Z pulse gates the pulses from $I^5$ or $I^6$ to the Z register during a critical portion of the read drive pulse. At the same time, during read operations, the appropriate pulse $I^5I^6 \rightarrow X$ or $I^5I^6 \rightarrow U$ gates the pulses from $I^5$ or $I^6$ to the X or U registers.

Data flow into storage takes place during the second half of the storage cycle. The quantity to be stored is in Z, as a result of a transfer from X to Z via $I^5$ or $I^6$ or from the sense windings to Z via $I^5$ or $I^6$. The write drive pulse attempts to set the selected cores to the "1" state. Simultaneously, however, the inhibit current generators produce an inhibit pulse corresponding with each "O" bit in Z. The inhibit pulse nullifies the write drive pulse at the core and prevents the latter pulse from setting the core to "1". In this manner, the word in Z is exactly reproduced at the selected storage location.

### SENSING CIRCUIT

For the duration of the read drive pulse, all 48 cores of the selected word receive full-amplitude pulses in the proper direction to switch their flux states to "O". A voltage is induced in the sense line of each core; typical voltages for "1" and "O" outputs, along with the read drive current pulse for comparison of timing, are shown in figure 4-5. The voltages may be of either polarity because of the manner in which the sense line is strung

through the cores.

Since the "1" output is the desired signal, the output from "O" must be regarded as noise and should be as small as possible. This signal arises from the shape of the hysteresis loop but is also dependent on the number of half-write pulses the core has received since it was first set to "O". Since the sense wire is strung through every core in the quadrant, it also links those cores which receive the half-read pulses. Thus, the signals produced on the sense line by those cores also contribute to the noise.

The flux density of those half-selected cores in the "1" condition is reduced slightly, while that of cores in the "O" condition is increased slightly, by the read drive pulse. In both cases, small noise voltages of about 1 to 2 millivolts are produced on the sense winding. The noise voltages are reduced in two ways: (1) by threading the sense line through the cores of a quadrant in such a manner that the noise signals from half-selected cores cancel each other, and (2) by sampling the sense amplifiers at a time when the "1" output voltages are near their peak and the noise voltages have decayed to a small part of their maximum.

The path of the sense wire through a four core matrix is shown in figure 4-14. The sense line follows a similar path through all the cores of a quadrant. If drivers 1 and 3 generate read drive current pulses, core A receives a full field, cores B and C half fields, and core D no field. Core A, which is the selected core, induces the desired "1" signal on the sense line while cores B and C induce noise voltages on the sense line. The noise signals add algebraically on the line and, as a result of the manner in which the lines thread the cores, the noise signals are of opposite polarity. In this manner, most of the noise signals in a quadrant cancel each other if they are nearly equal.

The sensing circuit for a typical memory plane is shown in figure 4-15. A sensing circuit such as this is provided for each memory plane of the odd and even storage units. The signal from the sense line is applied to the appropriate sense amplifier ($Y^{100}$, $Y^{101}$, $Y^{102}$

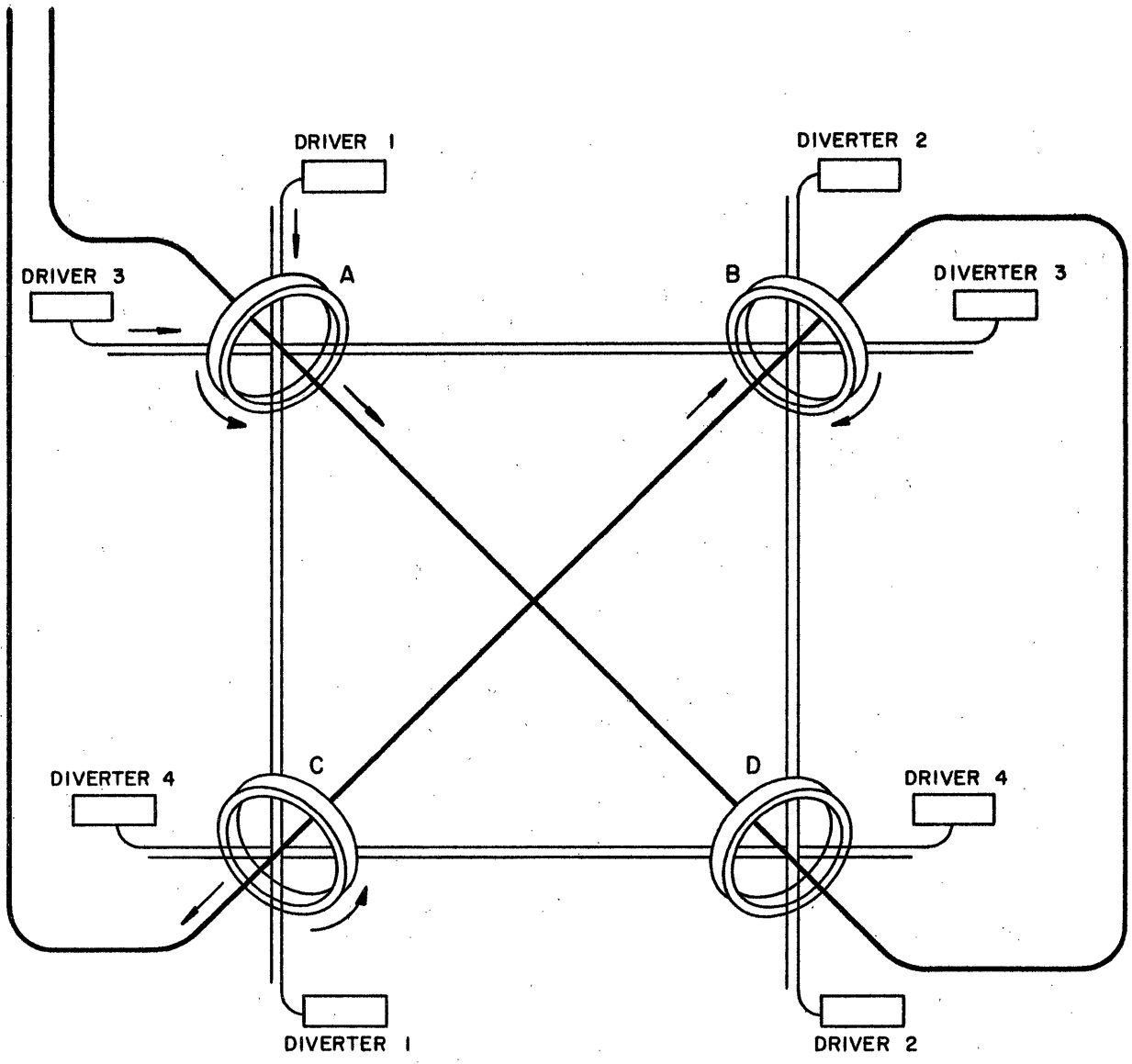



Figure 4-14. Path of Sense Wire Through a Four-Core Matrix.

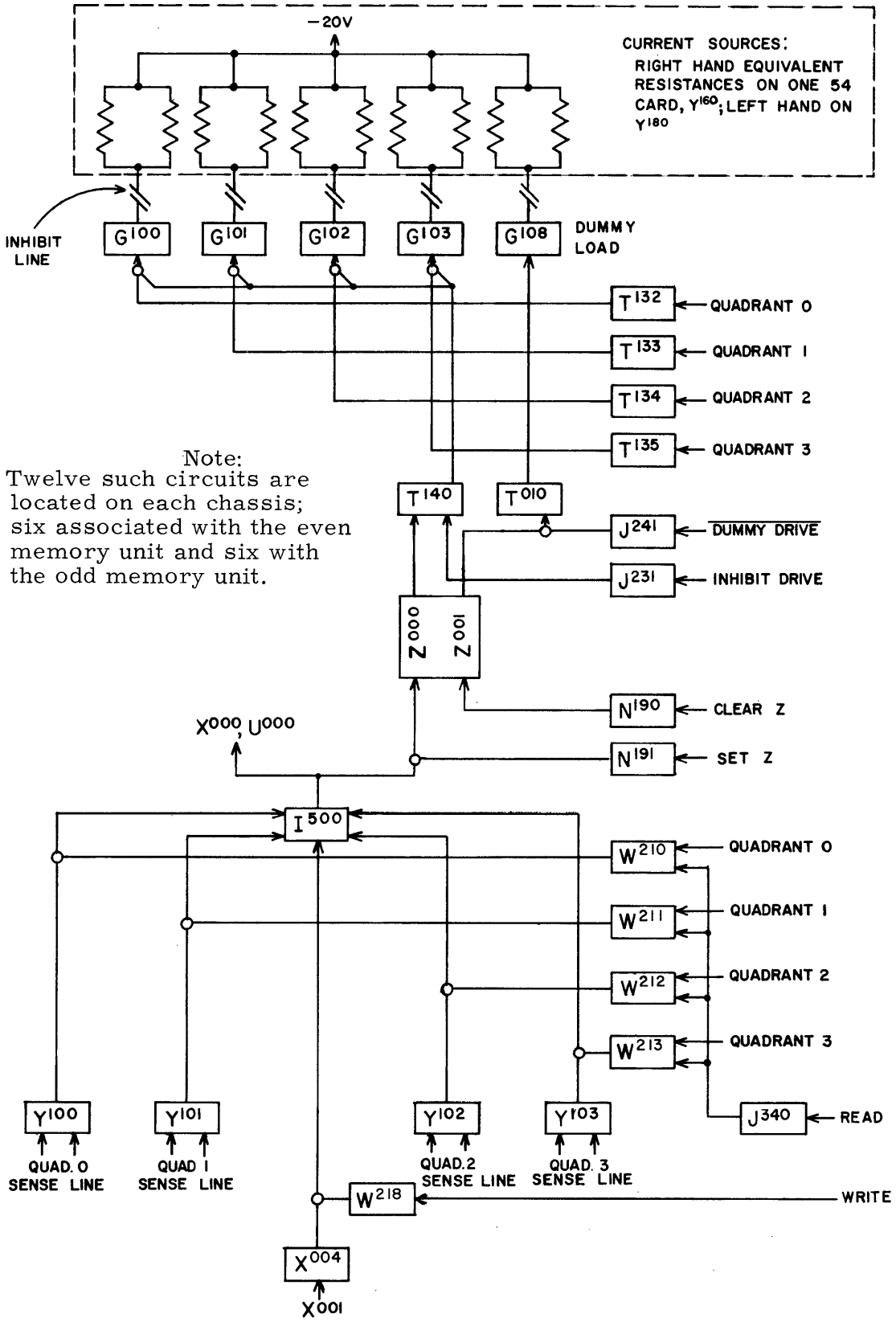
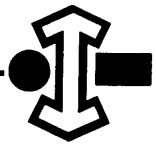



Figure 4-15. Typical Stage of the Bit Plane Control.



or $Y^{103}$ in this case) and the output of the amplifier is gated to $I^5$ by the appropriate Read Quadrant signal. A "1" signal in the sense line results in a "O" signal from the sense amplifier and a "1" signal from $I^5$.

The Set Z signal from the storage sequence control gates the output of $I^5$ into the Z register. The register is initially cleared by the Clear Z signal. A "1" output from a stage of $I^5$ allows the Set Z signal to pass the AND and set the associated stage of Z to "1". In the case of a read reference, either of the signals $I^5I^6 \rightarrow X$ or $I^5I^6 \rightarrow U$ is also generated; these signals set the stages of X and U according to "1's" in $I^5$ or $I^6$.

As shown in figure 4-5, the "1" signal from the sense amplifier reaches its peak after the "O" signal has decayed to about 6 millivolts. By generating the Set Z, $I^5I^6 \rightarrow X$ and $I^5I^6 \rightarrow U$ signals approximately 1.6 microseconds after the start of the read drive pulse, most of the "O" signal is avoided.

INHIBIT CIRCUITRY

Figure 4-16 shows the path of the inhibit wire through a four-core matrix. The inhibit line follows a similar path through all the cores of a quadrant. Note that the inhibit current flows in opposite directions in adjacent lines. This is related to and compatible with the fact that the direction of the write drive current reverses from line to line. For example, in figure 4-16 the write drive current flows down from driver 1, up from driver 2, right from driver 3 and left from driver 4. Following the path of the inhibit line through the matrix, it is apparent that the inhibit current is at all time opposite in direction to the write drive current.

The inhibit circuit for a typical memory plane is shown in figure 4-15. Identical inhibit circuits are provided each memory plane of the odd and even storage units. The write drive pulse, following the read drive pulse, attempts to switch all the cores of the selected memory location to the "1" state. The inhibit drive pulse passes inverter $T^{140}$ if the stage of Z stores a "O" bit and probes the AND inputs to the inhibit current generators ($G^{100}$,

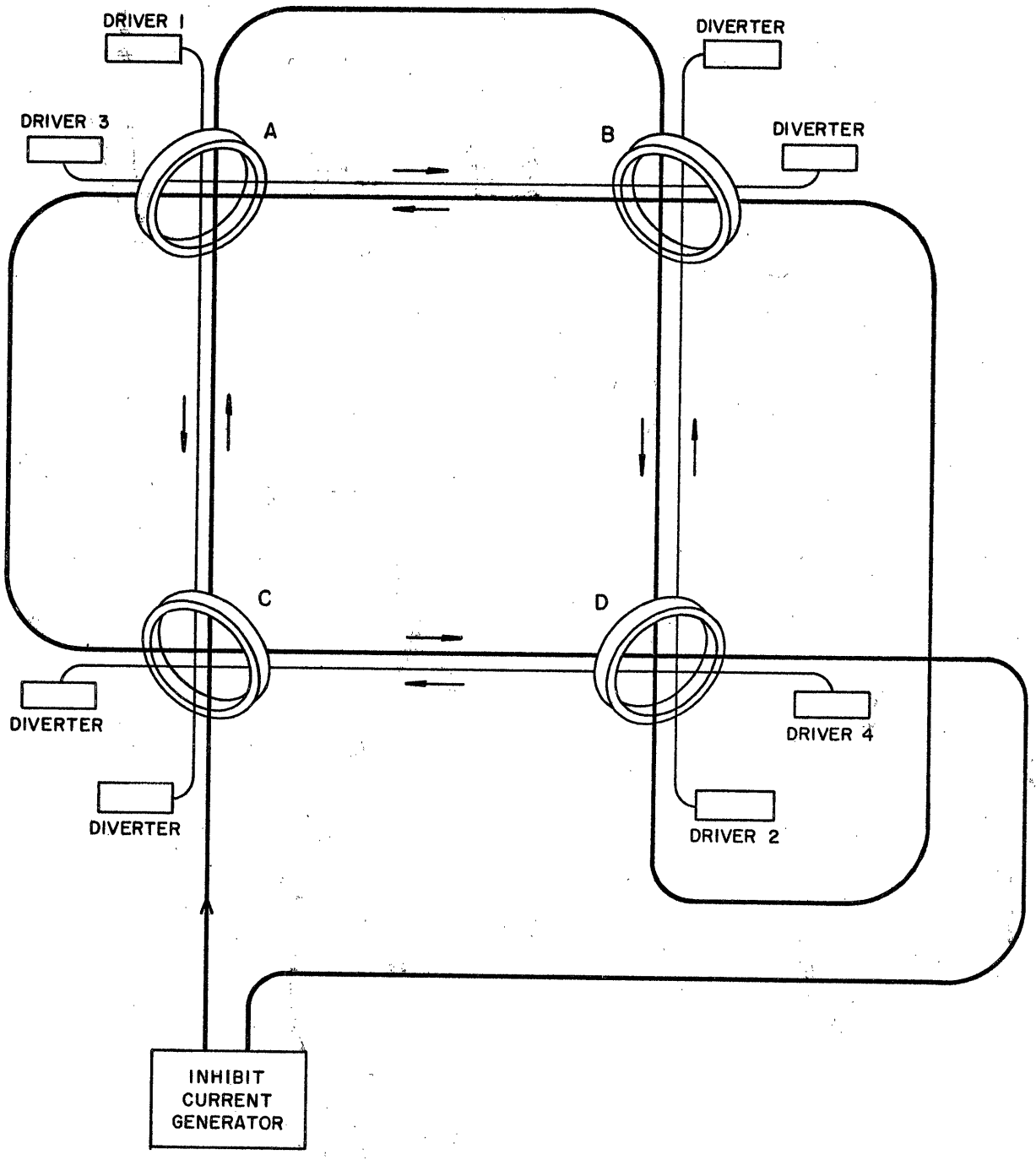
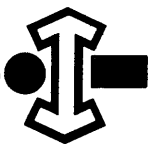



Figure 4-16. Path of Inhibit Wire Through a Four-Core Matrix.



$G^{101}$, $G^{102}$ and $G^{103}$). One of these AND inputs is enabled by the appropriate quadrant selection signal; the output of the AND enables the appropriate inhibit current generator, which then draws inhibit current through the inhibit wire from the current source ($Y^{160}$ and $Y^{180}$). The inhibit current, occurring at the time of the write current but opposite in polarity, cancels the effect of the write current within that quadrant and as a result, the core is not switched to "1".

The dummy load associated with an inhibit circuit is continuously enabled during periods of no storage references and also during those references which do not result in inhibit current generation. In the case of references which do result in inhibit current generation, the "O" output of Z enables the AND input of $T^{010}$. As a result, the Dummy-Drive signal passes $T^{010}$ and disables the dummy load for a period of 6. 2 usec during the reference.

## STORAGE SEQUENCE CONTROL

The storage sequence control, in response to initiating signals from the control section of the computer, executes reading and writing operations by generating the signals which control the address selection and bit plane circuits. The basic pulse sequence for reading and writing is shown in figure 4-17. The first pulse, called the read drive, drives the selected core to its "O" state of magnetization. The second pulse, called the inhibit drive, allows a "O" bit to be retained in the core by inhibiting (cancelling) the effect of the write drive pulse. The third pulse, called the write drive, drives the core to its "1" state of magnetization if the inhibit drive is absent.

The storage sequence control consists of the initiate storage reference circuit and identical sequence controls for the odd and even storage units. The initiate storage reference circuit initiates a reference by entering the address in the appropriate S register and selecting the appropriate odd or even sequence control. The selected sequence control then generates a fixed sequence of control pulses which executes the reference.

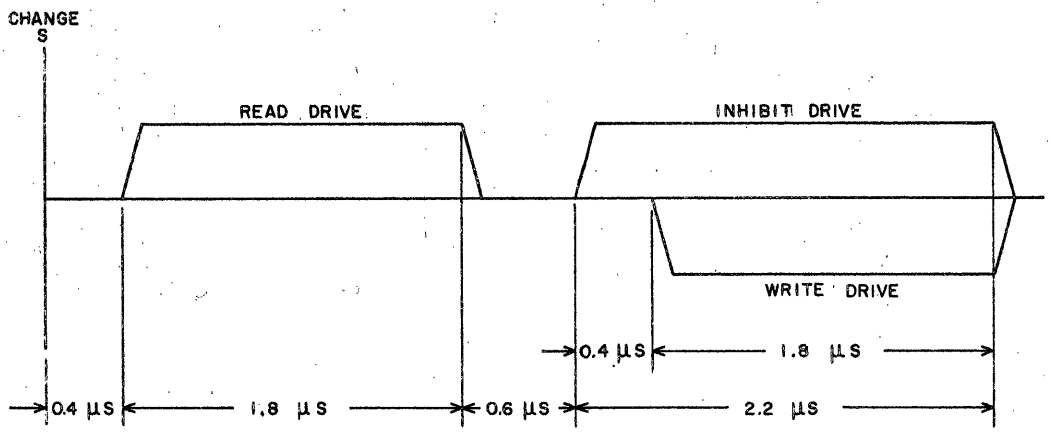
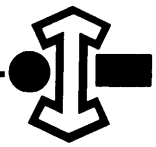



Figure 4-17. Basic Pulse Sequence for Storage Reference

### INITIATE STORAGE REFERENCE CIRCUIT

A sequence chain of the control section orders a storage reference by generating the signal Initiate Storage Reference. This signal performs two functions: (1) it controls the transmission of the storage address involved in the reference to the appropriate S register;



(2) it initiates the basic storage sequence within the storage unit.

The initiate storage reference circuit is shown in figure 4-18. The storage references differ from each other with respect to the source of the address involved in the reference. The address may originate from one of three sources, depending upon the sequence which initiates the reference. Table 4-1 lists the source of the address for each of the 8 sequences which initiate storage references. As shown in figure 4-18, the Initiate Storage Reference signal sets one of five flip-flops, depending upon the address source. These FFs control the AND inputs to eight control delays which generate the control signals that change S and initiate the storage sequences.

TABLE 4-1. SOURCES OF ADDRESSES FOR STORAGE REFERENCES

| Sequence | Source of Address |
|---|---|
| Read Next Instruction | Program Address Register ($P^1$) |
| Read Operand | Program Control Register ($U^2$) |
| Write Operand | |
| Search and Transfer | |
| Iterative | |
| Auxiliary (Buffer Initiates #2) | |
| External Function | |
| Auxiliary (Advance Clock, Interrupt and Buffer Initiates #1 and #3 | Input-Output Control |

## SEQUENCE CONTROLS

The two sequence controls are identical. Each, in response to the Initiate Storage Sequence signal, generates a fixed sequence of control signals. The signals and their time relationships with respect to the Initiate Storage Sequence signal are shown in figure 4-19.

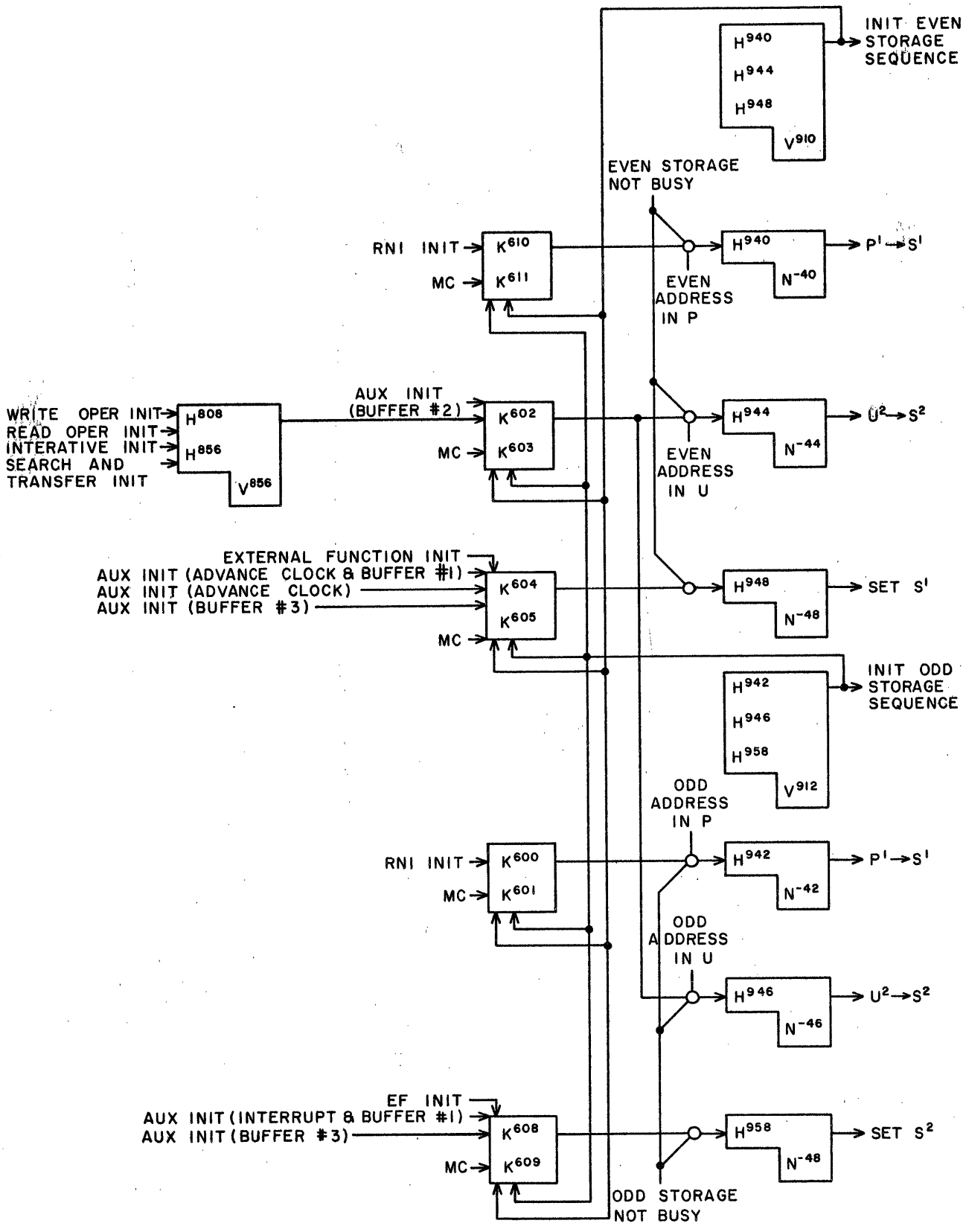



Figure 4-18. Storage Reference Circuit.

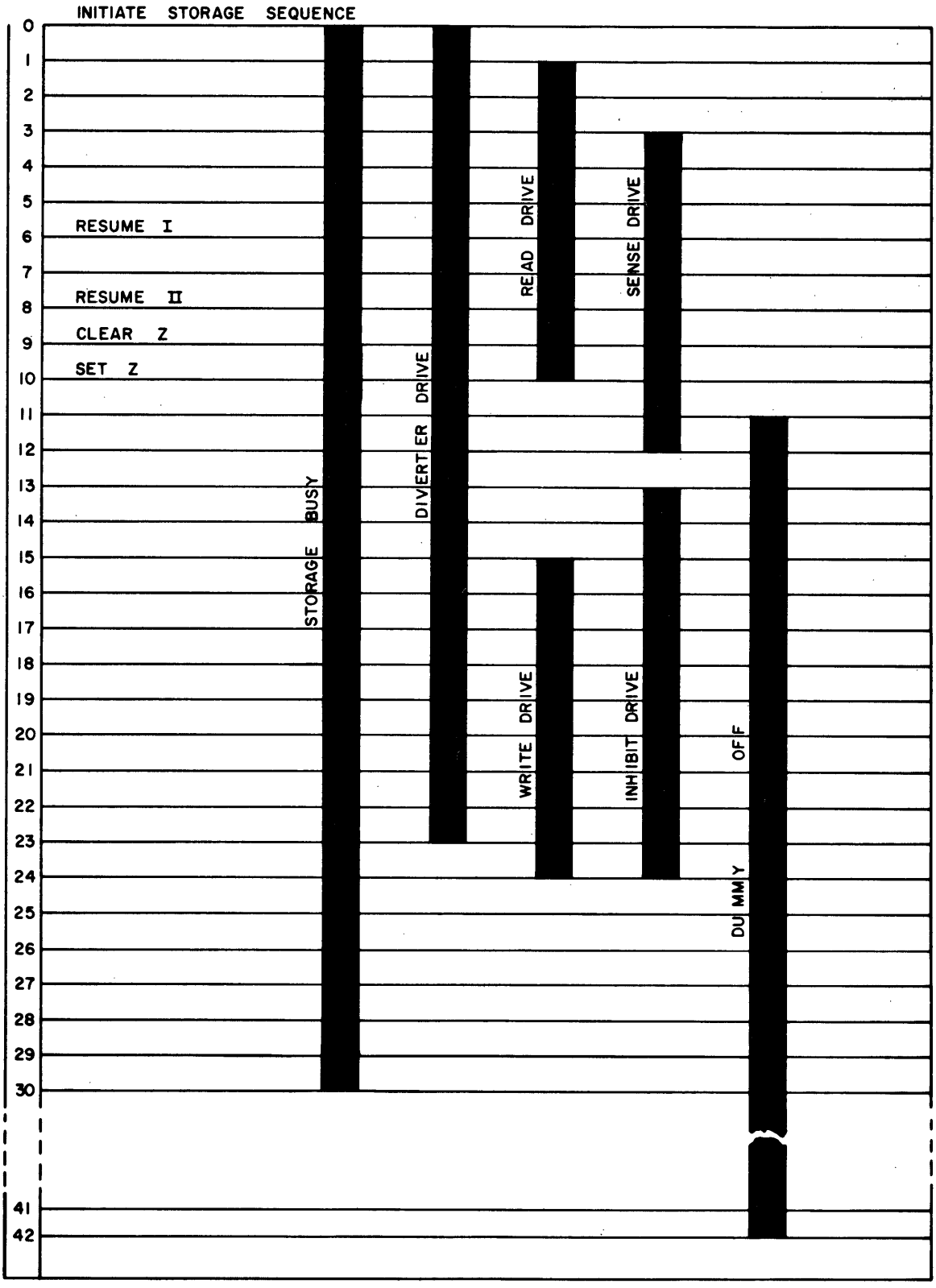



Figure 4-19. Sequence of Pulses Generated by Control Sequence.

TIMING PULSE GENERATOR

A sequence control consists of a timing pulse generator, a group of drive generators, a group of pulse generators and a fault detector. The timing pulse generator generates a sequence of timing pulses during the 8.4 microseconds immediately following the Initiate Storage Sequence signal. The timing pulse generator of the even sequence control is shown in figure 4-20. A loop of eight control delays ($H^{061}$ through $H^{068}$) generates the basic 8-pulse cycle. The pulses are separated by the 0.2 microseconds interval inherent in control delays. The remaining circuitry of the timing pulse generator divides the control delays into two logical groups: group one consists of $H^{061}$ through $H^{064}$; group two consists of $H^{065}$ through $H^{068}$.

A two-rank, two-stage counter is advanced by a signal from each group of control delays. Rank I of the counter consists of FFs $K^{640/641}$ and $K^{644/645}$; rank II consists of FFs $K^{642/643}$ and $K^{646/647}$. During the first half of an 8-pulse cycle, the output of $V^{062}$ transfers the count in rank I to rank II. During the second half of the cycle, the output of $V^{066}$ transfers the count in rank II to rank I, advancing it by one in the process. Thus, the counter is advanced through its cycle of four counts by four successive 8-pulse cycles.

Table 4-2 lists the 32 steps of a complete counter cycle. During the first half of each 8-pulse cycle, rank I holds the count while rank II is being changed and during the second half of the cycle, rank II holds the count while rank I is being changed. A distributor, combining selected outputs of the control delays with the outputs of the counter, takes advantage of this feature. Four inverters ($J^{160}$, $J^{162}$, $J^{164}$ and $J^{166}$) translate the outputs of Rank I to provide negative outputs for counts 00, 01, 10 and 11, respectively. A second group of four inverters ($J^{161}$, $J^{163}$, $J^{165}$ and $J^{167}$) provides the same translation for the outputs of Rank II. The outputs of the first group of inverters are combined with the outputs of the first half of the timing chain; the outputs of the second group of inverters are combined with the outputs of the second half of the timing chain. The combination of outputs from the inverters with outputs from the timing chain and the timing pulses produced by each combination are listed in table 4-3.

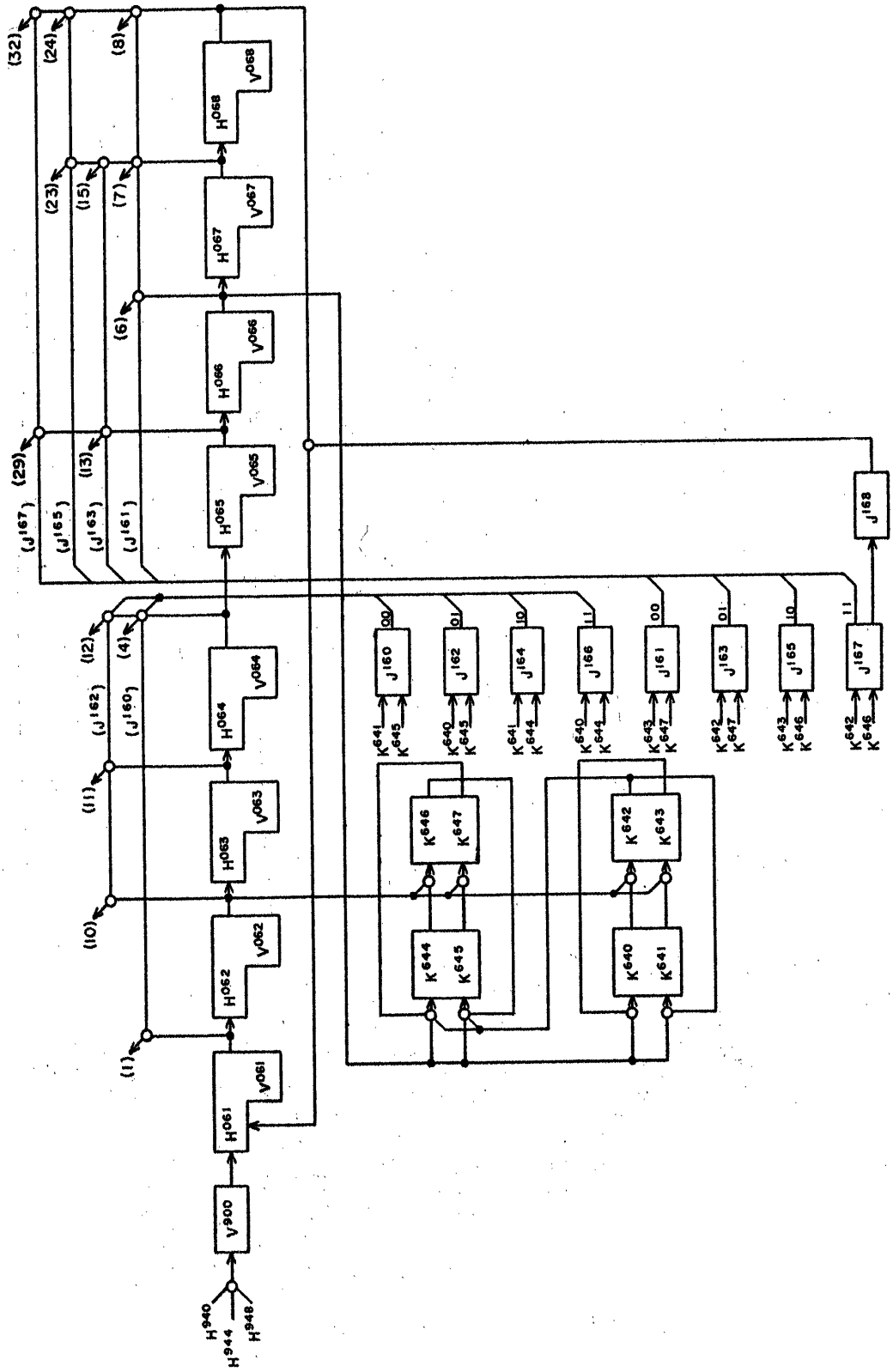



Figure 4-20. Timing Pulse Generator of Even Storage Sequence Control.

TABLE 4-2. TIMING PULSE GENERATOR OF EVEN STORAGE SEQUENCE CONTROL: COMPLETE COUNTER CYCLE

| Timing Pulse | Source | Rank I | Rank II |
|---|---|---|---|
| 0 | $V^{900}$ | 00 | 00 |
| 1 | $V^{061}$ | 00 | Rank I → Rank II |
| 2 | $V^{062}$ | 00 | |
| 3 | $V^{063}$ | 00 | |
| 4 | $V^{064}$ | 00 | |
| 5 | $V^{065}$ | Rank II + 1 → Rank I | 00 |
| 6 | $V^{066}$ | | 00 |
| 7 | $V^{067}$ | | 00 |
| 8 | $V^{068}$ | | 00 |
| 9 | $V^{061}$ | 01 | Rank I → Rank II |
| 10 | $V^{062}$ | 01 | |
| 11 | $V^{063}$ | 01 | |
| 12 | $V^{064}$ | 01 | |
| 13 | $V^{065}$ | Rank II + 1 → Rank I | 01 |
| 14 | $V^{066}$ | | 01 |
| 15 | $V^{067}$ | | 01 |
| 16 | $V^{068}$ | | 01 |
| 17 | $V^{061}$ | 10 | Rank I → Rank II |
| 18 | $V^{062}$ | 10 | |
| 19 | $V^{063}$ | 10 | |
| 20 | $V^{064}$ | 10 | |
| 21 | $V^{065}$ | Rank II + 1 → Rank I | 10 |
| 22 | $V^{066}$ | | 10 |
| 23 | $V^{067}$ | | 10 |
| 24 | $V^{068}$ | | 10 |
| 25 | $V^{061}$ | 11 | Rank I → Rank II |
| 26 | $V^{062}$ | 11 | |
| 27 | $V^{063}$ | 11 | |
| 28 | $V^{064}$ | 11 | |
| 29 | $V^{065}$ | Rank II + 1 → Rank I | 11 |
| 30 | $V^{066}$ | | 11 |
| 31 | $V^{067}$ | | 11 |
| 32 | $V^{068}$ | | 11 |

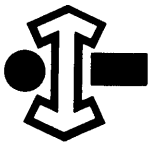


TABLE 4-3. TIMING PULSE GENERATOR; SOURCE OF TIMING PULSES

| Timing Pulse | Source | |
|---|---|---|
| | Inverter | Control Delay |
| 0 | | $V^{900}$ |
| 1 | $J^{160}$ | $V^{061}$ |
| 4 | $J^{160}$ | $V^{064}$ |
| 6 | $J^{161}$ | $V^{060}$ |
| 7 | $J^{161}$ | $V^{067}$ |
| 8 | $J^{161}$ | $V^{068}$ |
| 10 | $J^{162}$ | $V^{062}$ |
| 11 | $J^{162}$ | $V^{063}$ |
| 12 | $J^{162}$ | $V^{064}$ |
| 13 | $J^{163}$ | $V^{065}$ |
| 15 | $J^{163}$ | $V^{067}$ |
| 23 | $J^{165}$ | $V^{067}$ |
| 24 | $J^{165}$ | $V^{068}$ |
| 29 | $J^{167}$ | $V^{065}$ |
| 32 | $J^{167}$ | $V^{068}$ |

## DRIVE GENERATORS AND PULSE GENERATORS

The drive generators and pulse generators of the odd and even sequence controls are identical. Those of the even sequence control are shown in figure 4-21. Each drive generator consists of a flip-flop and two or more inverters. The FF is set and cleared by timing pulses which coincide with the leading and trailing edge, respectively, of the desired drive signal. The inverters serve as slaves providing the multiple output signals required. The inverters are connected in parallel to either the "1" or "O" output of the FF as necessary to provide signals of the desired polarity. The pulse generators are control delays which produce Z register commands and the storage resumes.

The following discussions refer to both the odd and even sequence controls. The logical designations used, however, are those of the even sequence control.

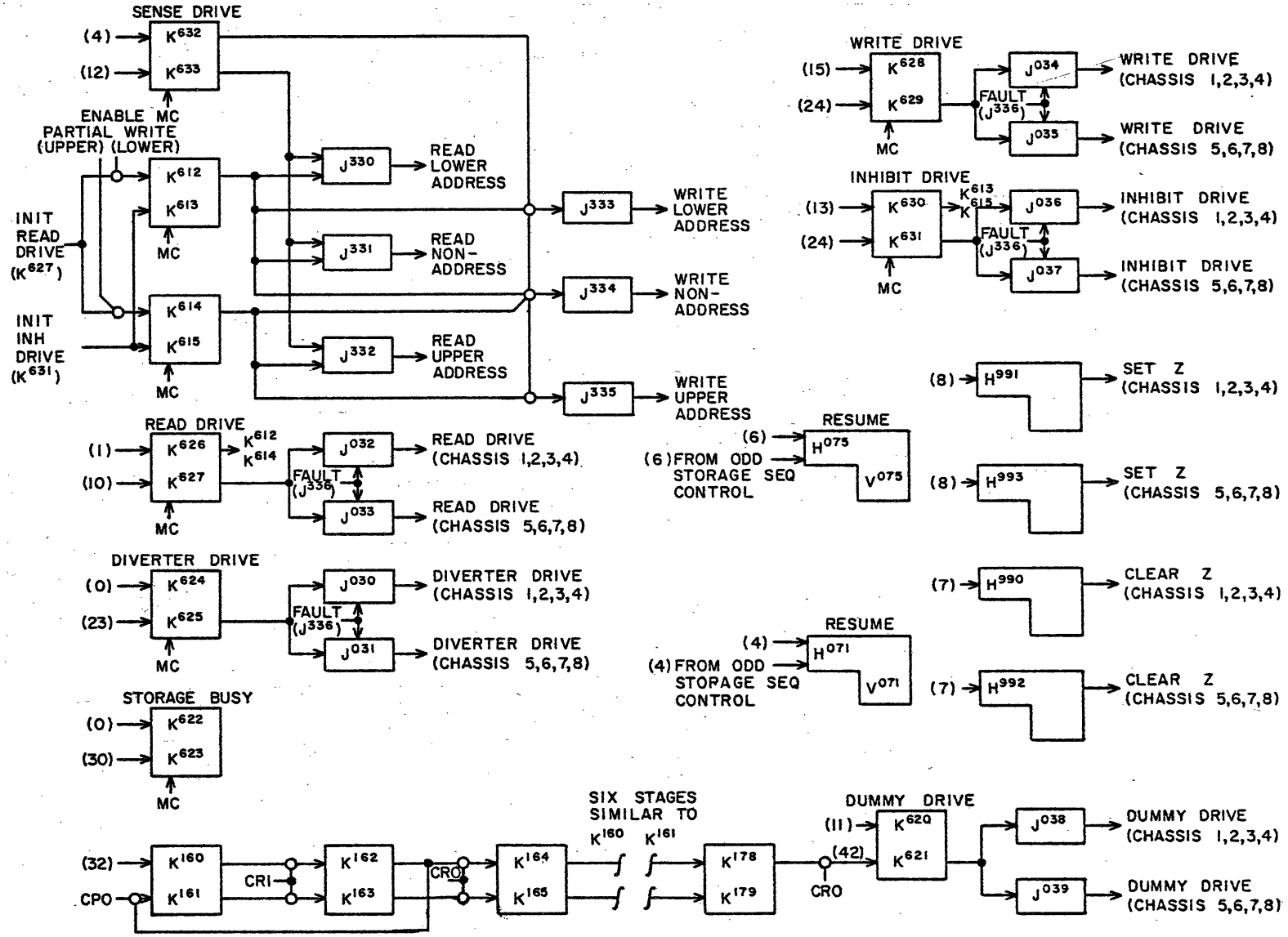



Figure 4-21. Drive and Pulse Generator of Even Storage Sequence Control.

1) Diverter Drive. The Diverter Drive FF ($K^{624/625}$) is set by timing pulse 0 and cleared by timing pulse 23; during this interval, inverters $J^{030}$ and $J^{031}$ provide the positive Diverter Drive Signal to the horizontal and vertical diverters of the address selection circuit.

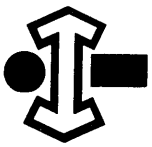
2) Read Drive. The Read Drive FF ($K^{626/627}$) is set by timing pulse 1 and cleared by timing pulse 10; during this interval inverters $J^{032}$ and $J^{033}$ provide the negative Read Drive signal to the horizontal and vertical drivers of the address selection circuit.

3) Write Drive. The Write Drive FF ($K^{628/629}$) is set by timing pulse 15 and cleared by timing pulse 24; during this interval, inverters $J^{034}$ and $J^{035}$ provide the negative Write Drive signal to the horizontal and vertical drivers of the address selection circuit.

4) Inhibit Drive. The Inhibit Drive FF ($K^{630/631}$) is set by timing pulse 13 and cleared by timing pulse 24; during this interval, inverters $J^{036}$ and $J^{037}$ provide the Inhibit Drive signal to the inhibit circuits.

5) Sense Drive. The Sense Drive FF ($K^{632/633}$) is set by timing pulse 4 and cleared by timing pulse 12. During this interval, $K^{632}$ provides a positive Sense Drive signal to inverters $J^{330}$, $J^{331}$ and $J^{332}$, and $K^{633}$ provides a negative Sense Drive signal to inverters $J^{333}$, $J^{334}$ and $J^{335}$.

The signals Enable Partial Write (upper) and Enable Partial Write (lower) from the control section control the generation of signals from the inverters. The signal Enable Partial Write (lower) enables the "Set" input gate of FF $K^{612/613}$ and this FF is set to "1" during the read half of the storage sequence. Similarly, the signal Enable Partial Write (upper) enables the input gate to $K^{614/615}$. The output of $K^{613}$ is applied to the "OR" inputs of inverters $J^{330}$ and $J^{331}$ and to the "AND" inputs of inverters $J^{333}$ and $J^{334}$. The output of $K^{615}$ is applied to the "OR" inputs of inverters $J^{331}$ and $J^{332}$ and to the "AND" inputs of inverters $J^{334}$ and $J^{335}$.



The absence of all of the Partial Write Enables from control results in negative signals from $J^{330}$, $J^{331}$ and $J^{332}$ during the period of the Sense Drive. These signals, designated Read (Lower Address), Read (Non-address) and Read (Upper Address), respectively, effect a read reference by transferring a complete word from the sense lines of the selected quadrant of a memory plane to $I^5$.

If both Partial Write signals are present, inverters $J^{333}$, $J^{334}$ and $J^{335}$ provide negative output signals during the period of the Sense Drive signals. These signals, designated Write (Lower Address), Write (Non-address) and Write (Upper Address), respectively, effect a write reference by transferring a complete word from the X register to $I^5$.

It follows from the above analysis that the presence of the signal Partial Write (Lower) and the absence of the other results in the generation of the signals Write (Lower Address), Read (Non-address) and Read (Upper Address). Similarly, the presence of the signal Partial Write (Upper) and the absence of the other results in the generation of the signals Read (Lower Address), Read (Non-address) and Write (Upper Address). These combinations of Read and Write signals result in partial write operations; the former combination writes the lower address and restores the remainder of the word, and the latter combination writes the upper address and restores the remainder of the word.

6) Dummy Drive. The Dummy Drive FF ($K^{620/621}$) is set by timing pulse 11 and cleared approximately 6.2 usec later by timing pulse 42. A special chain of 10 FFs ($K^{160/161}$) through ($K^{178/179}$) is used to delay timing pulse 32 for 10 periods in order to obtain timing pulse 42. Each FF provides approximately a .0.2 microsecond delay; the precise delay period provided by control delays is not necessary in this application since the timing of the Dummy Drive signal is not critical.

During the interval that the Dummy Drive FF is set, inverters $J^{038}$ and $J^{039}$ provide negative Dummy Drive Off signals which disable the dummy loads of the

horizontal and vertical drivers and selected dummy loads of the inhibit circuits.

7) Clear Z. Control delays $H^{990}$ and $H^{992}$ receive timing pulse 7 and provide Clear Z output pulses to the bit plane control. The output of $H^{990}$ is sent to chassis 1, 2, 3 and 4; the output of $H^{992}$ is sent to chassis 5, 6, 7 and 8.

8) Set Z. Control delays $H^{991}$ and $H^{993}$ receive timing pulse 8 and provide Set Z output pulses to the bit plane control. The output of $H^{991}$ is sent to chassis 1, 2, 3 and 4; the output of $H^{993}$ is sent to chassis 5, 6, 7 and 8.

9) Resume. Two control delays, shared by the odd and even storage sequence controls, generate the Resume I and Resume II pulses. Control delay $H^{071}$ receives timing pulse 4 from both the odd and even storage sequence controls and provides the Resume I pulse to the control section. Control delay $H^{075}$ receives timing pulse 6 from both the odd and even storage sequence controls and provides the Resume II pulse to the control section.

FAULT DETECTOR

The fault detector detects the presence of more than one pulse in the timing loop of a sequence control. Such unwanted pulses cycle the drive generators on and off at a rapid rate and overheat the generators. The first control delay of both the even and odd timing loops is odd-numbered. Thus, only even-phase pulses can enter the loops. The fault detector of the even storage unit (figure 4-22) combines the outputs of control delay $V^{063}$ and $V^{065}$ at the AND input to the set side of FF $K^{634/635}$. If two successive even pulses enter the loop, the FF will be set and the Storage Fault signal generated. If two pulses separated by the six phase periods enter the loop, the FF will be set during the second recirculation of the pulses (when they are separated by two phase periods).

During the period $K^{634/635}$ is set, the AND input of $J^{336}$ is not satisfied. As a result, a "1" is generated by $J^{336}$ as the Storage Fault signal. This signal is applied to the slave inverters of the drive FFs, thus holding off the drive generators. A master clear, performed at the console, clears the drive FFs and also $K^{634/635}$. During the period the

CLEAR lever switch is depressed, the AND input of $J^{336}$ is inhibited by a "0" from $J^{337}$. Thus, the drive generators are held off during the period of the master clear.

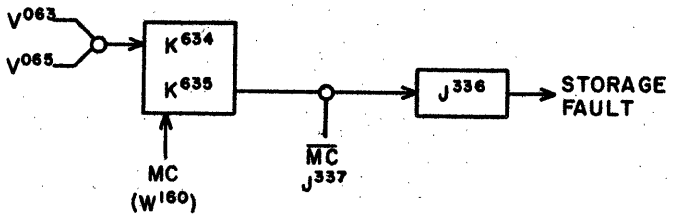



Figure 4-22. Fault Detector of Even Storage Unit

## ELECTRONIC THEORY OF MEMORY CIRCUITS

The storage section uses many non-standard cards in order to perform non-logical functions such as amplification, pulse generation and switching. The non-standard card types are as follows: the drive generator (card type 51), the diverter (card type 52), the selector (card type 53), the current source (card type 54), the inhibit generator (card type 55) and the sense amplifier (card type 56). The following paragraphs discuss the electronic theory of each of these card types and describe the interconnections of the cards to perform specific functions within the storage section.

### DRIVE GENERATOR (Type 51)

The drive generator (figure 4-23) develops the read-write current which is applied to the selected H and V wires. The circuit consists of two identical channels feeding opposite ends of the primary winding of transformer T01. Each channel consists of transistor Q01, connected as an emitter-follower, and transistors Q02 and Q03, connected in parallel as amplifiers. The input signal is received from an AND combination of two selector outputs. A -1v input signal results in approximately $0^{V}$ at the base of Q01. The emitter of Q01 is clamped to ground by CR03 and thus neither Q02 or Q03 conduct. Consequently, there is no current flow in the primary of T01.

A -12v input signal causes Q01 to conduct; however, the conduction is held below saturation by feedback diode CR01. The negative voltage developed across R06 is applied to the bases of Q02 and Q03, causing these transistors to conduct. Current flows from the current sources through the emitters of Q02 and Q03 to the collectors, through the primary of T01, to the current sources. The current pulse from the secondary of T01, amplified by the step-down action of T01, is applied to the H and V wires of the memory plane assembly.

The polarity of the output current from T01 is determined by the direction of the current flow in the primary. The direction of current flow, in turn, is determined by the channel

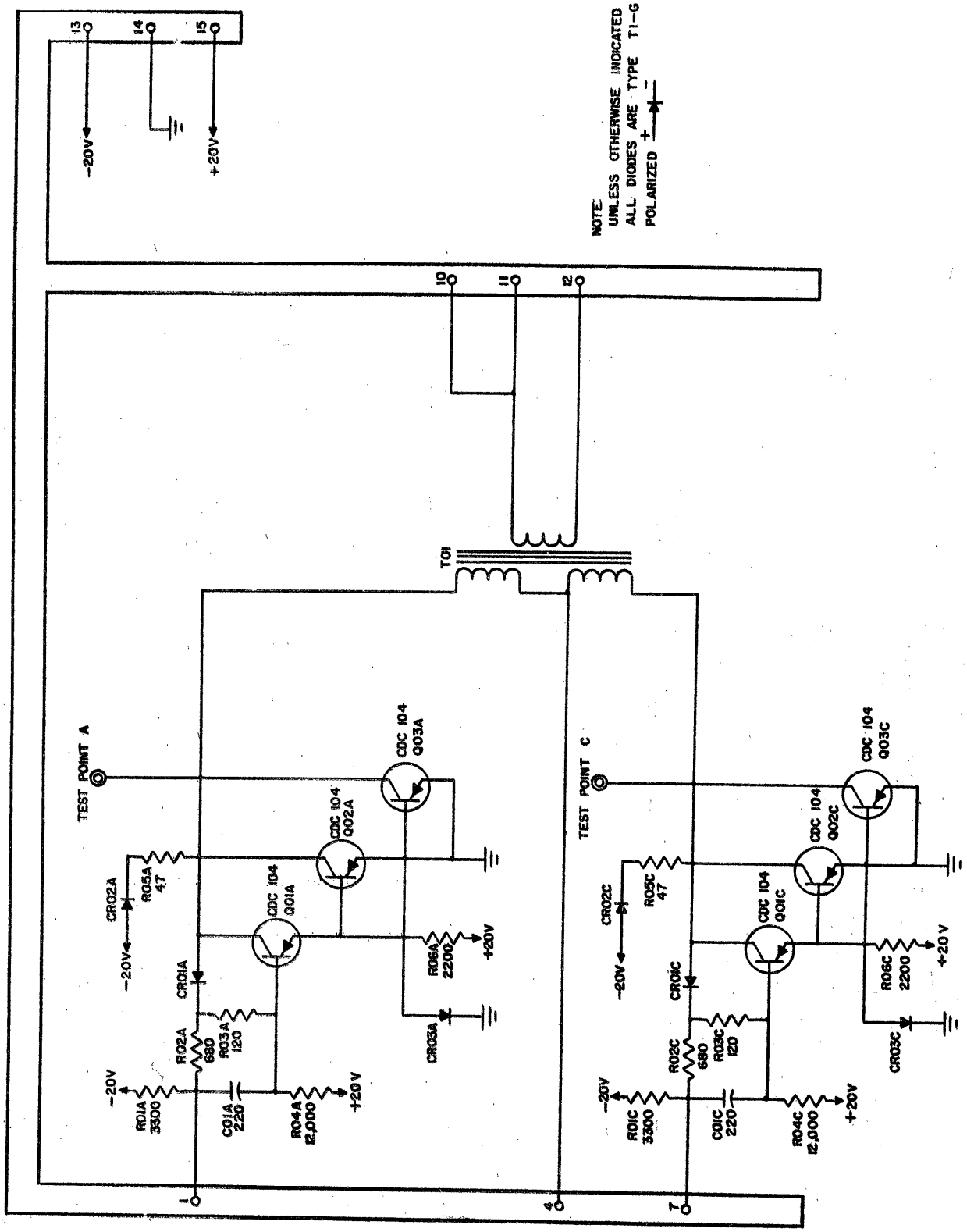



Figure 4-23. Drive Generator (Card Type 51).

which is selected. For example, if channel A receives a -12v input pulse a Read pulse is generated; if channel B receives the pulse, a Write pulse is generated.

DIVERTER (TYPE 52)

The diverter circuit (figure 4-24) serves as an electronic switch in series with an H or V wire of the memory plane assembly. The diverter consists of transistor Q01, connected as an emitter follower, and transistors Q02 and Q03, connected as switches. A -3v input signal causes Q01 to conduct; the negative signal from Q01 enables transistors Q02 and Q03. One or the other of these transistors passes the current pulse on the H or V wire to which the diverter is connected, depending on the polarity of that pulse.

A positive pulse passes one of the pairs of diodes CR03 and CR04, CR07 and CR08, CR11 and CR12, etc., depending upon the driver selection, and passes Q02. A negative pulse passes one of the pairs of diodes CR01 and CR02, CR05 and CR06, etc., and passes Q03. In either case, the pulse is applied to a bleeder network composed of R02, R03 and R04. The bleeder networks of all the diverters are connected in parallel via terminals 11 and 12. This connection equalizes the current flow through the bleeders and thus reduces heating.

SELECTOR (TYPE 53)

Each selector card consists of two identical selector circuits. A selector circuit (figure 4-25) is similar to the standard inverter except that the resistances of R08 and R12 are 10k and 12k ohms, respectively. The larger resistance results in output signal levels of -1v and -12v. Each selector circuit has two input diodes (CR01 and CR02) and four output diodes (CR09, CR10, CR11 and CR12).

CURRENT SOURCE (TYPE 54)

The current source card (figure 4-26) consists of five banks of parallel resistors. The effective resistance of four banks is 150 ohms each; that of the remaining bank is 303 ohms. One end of each bank is connected to the -20v output of the power supply. The 150 ohm

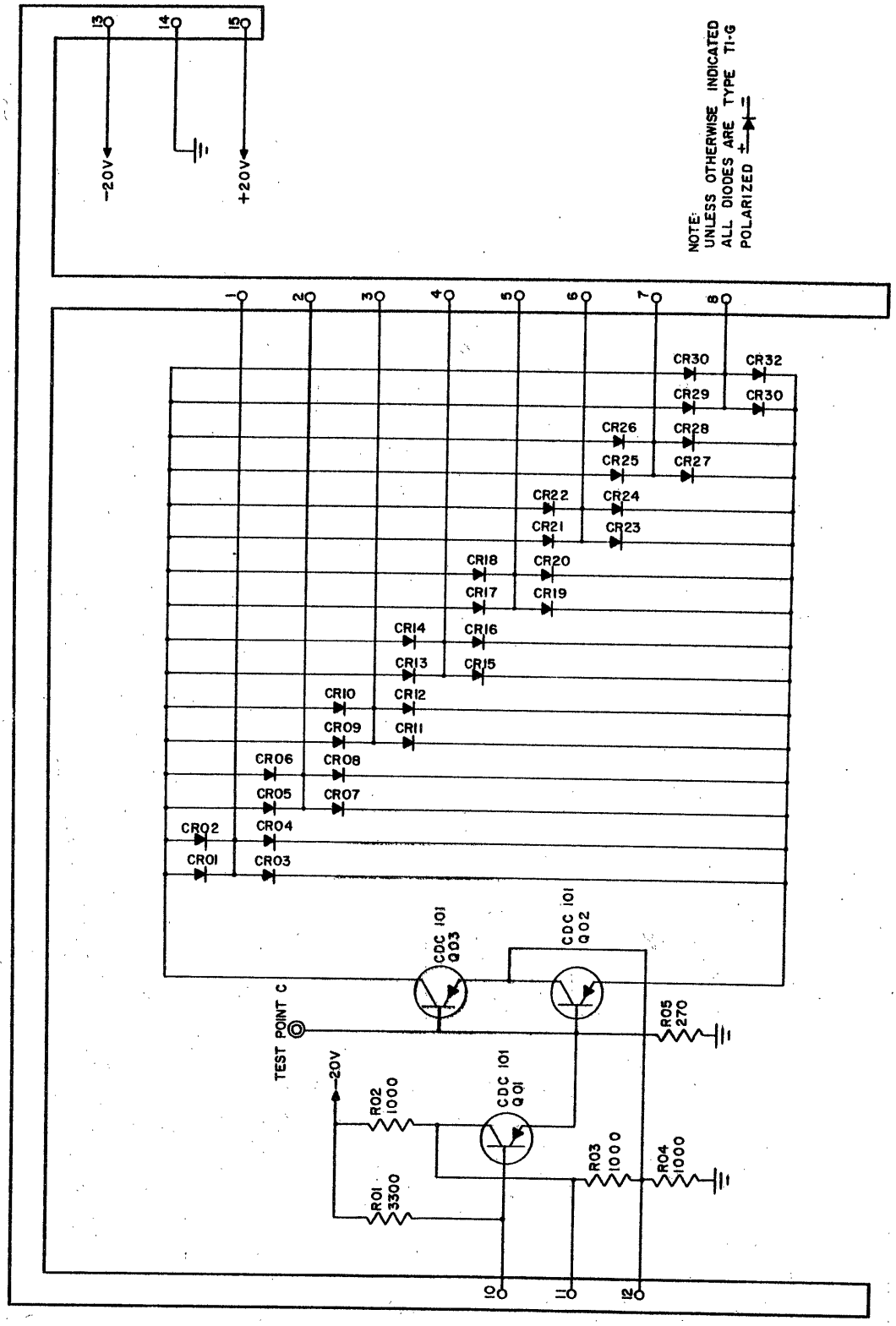



Figure 4-24. Diverter (Card Type 52).

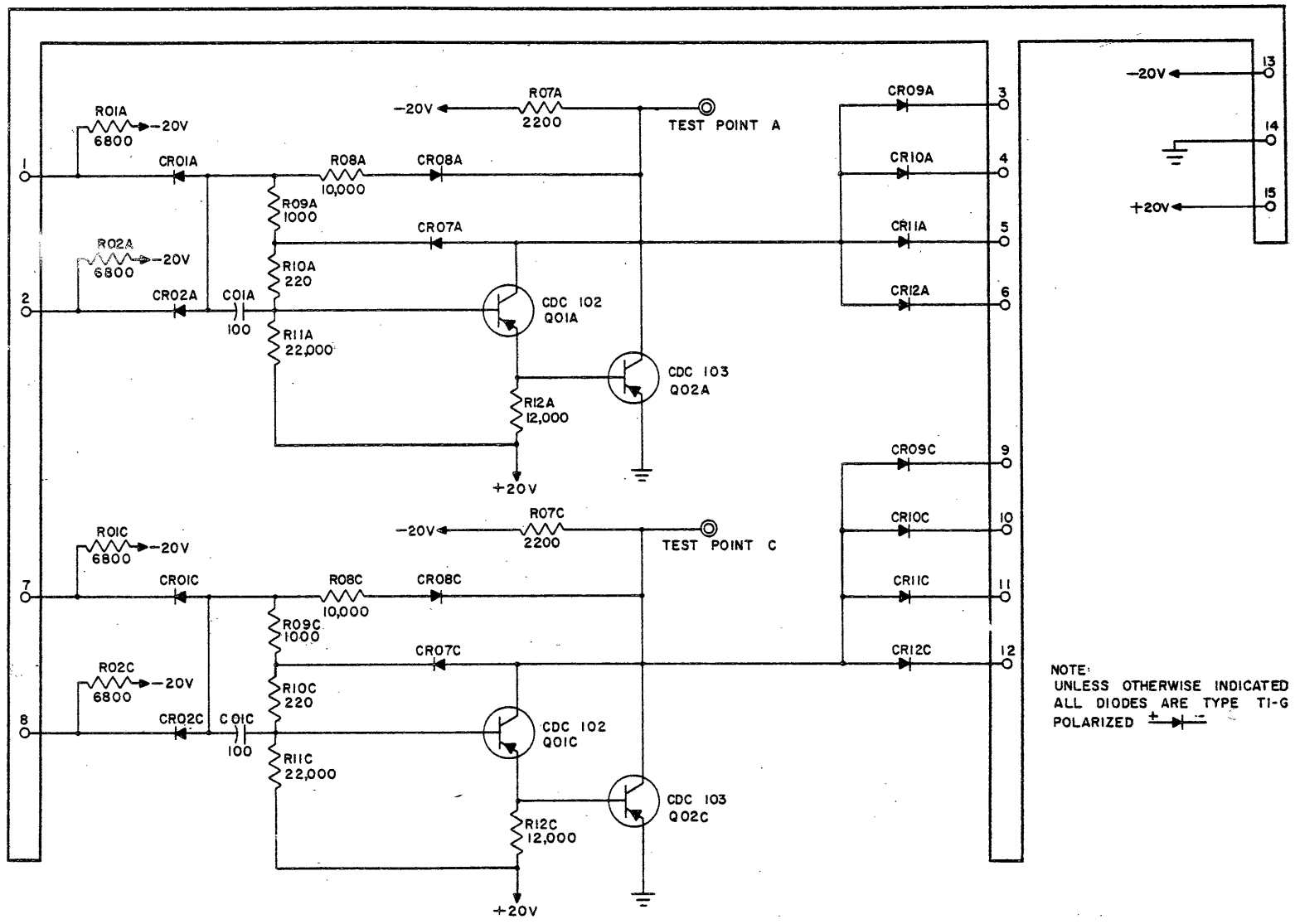



Figure 4-25. Selector (Card Type 53).

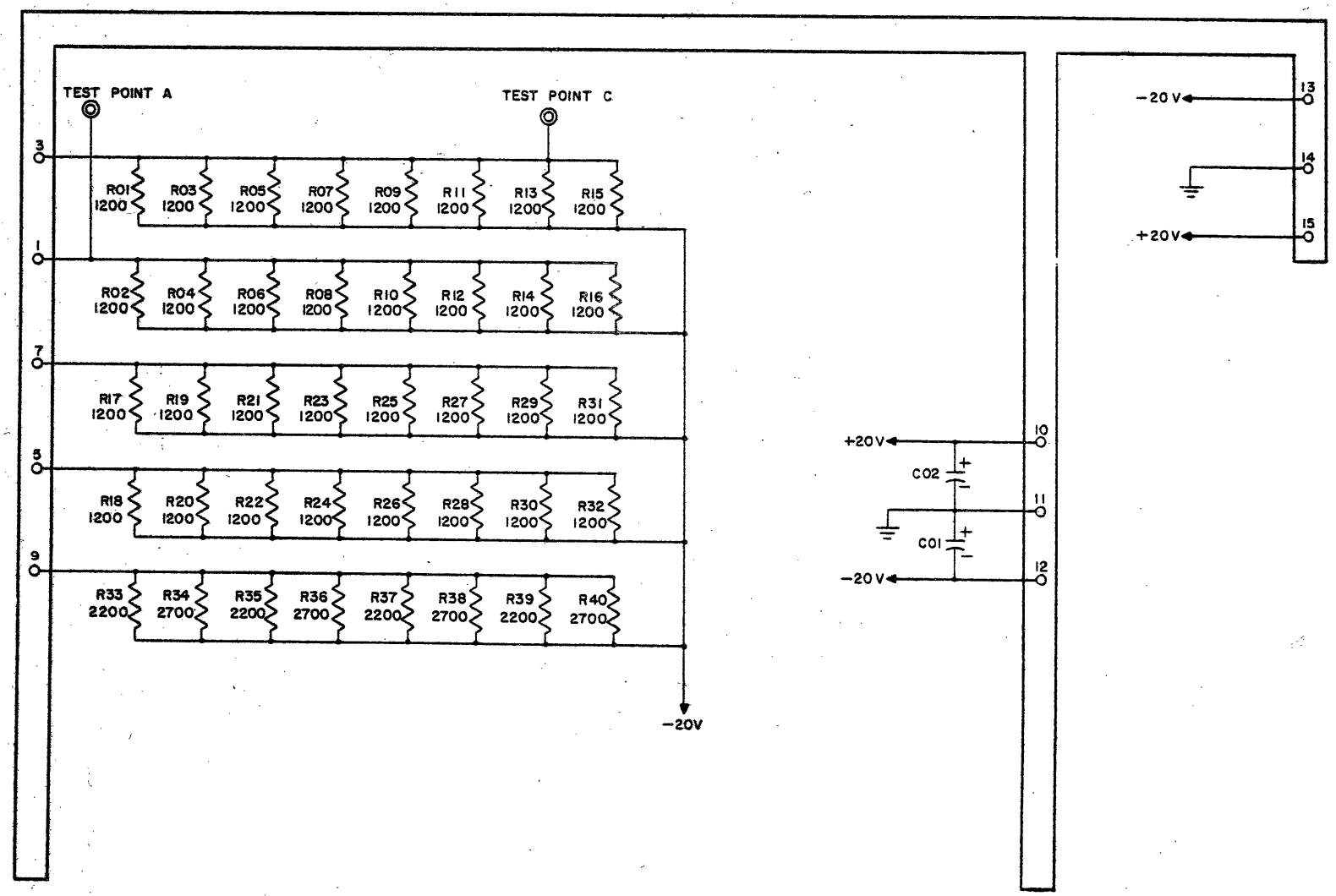



Figure 4-26. Current Source (Card Type 54).

banks supply current to the H, V and inhibit current generators; the 303 ohm banks supply current to the dummy loads.

INHIBIT GENERATOR (TYPE 55)

The inhibit generator (figure 4-27) is used as an inhibit current generator and as a dummy load. Each card has two generator circuits; the circuits are identical to the type 51 drive generator channels except for the absence of an output transformer. The output of each channel is independently connected to a terminal of the card.

A -12v input signal to either drive generator of a type 55 card causes Q01 to conduct and thus enable Q02 and Q03. Current from the external source connects to the drive generator via terminal 6 or 12 and passes through Q02 and Q03 to ground.

SENSE AMPLIFIER (TYPE 56)

The sense amplifier (figure 4-28) amplifies the sense signals from a memory plane quadrant. Transistors Q01, Q02, Q03 and Q04 are connected in a differential amplifier circuit. The signals from either end of the sense windings are applied to Q01 and Q03, respectively. The emitters are held at the difference voltage by a difference network composed of R04 and C01. As a result, noise voltages on the sense line are largely cancelled.

Capacitors C02 and C03, in the collector circuits of Q02 and Q04, respectively, provide d-c stabilization. Diodes CR01 and CR02 pass the negative-going components of the signals from Q02 and Q04, and also serve as clippers. The bias across these diodes, and thus the clipping level, is adjustable by the MARGIN SWITCH on the operator's console. When the MARGIN SWITCH is "up", +20v is applied to the junction of R14 and R15 and as a result, the reference voltage across the input diodes to the last stage is raised. For this condition, the circuit is less sensitive to the signal from the sense line and, weak signals tend to be dropped. When the MARGIN SWITCH is "down", -20v is applied to R14 and R15 and, as a result, the reference voltage across the input diodes of the last stage is raised. For this condition, the circuit is more sensitive to the signal from the sense line and spurious pulses tend to look like "1's".

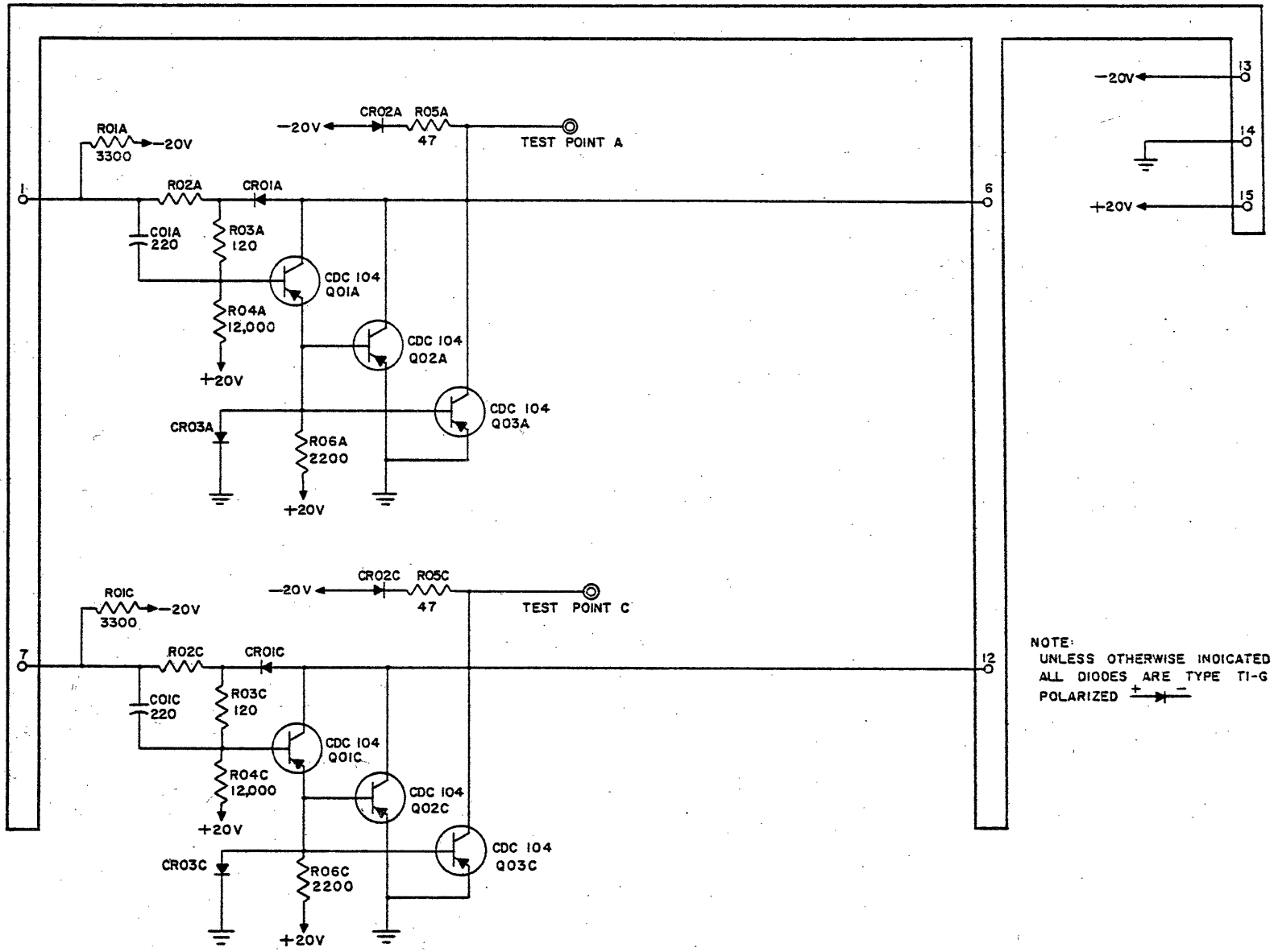



Figure 4-27. Inhibit Generator (Card Type 55).

Transistors Q05 and Q06 are connected in an amplifier-inverter circuit. The output a signal from CR05, as a result of a "1" signal from the sense wire, is -0.5v.

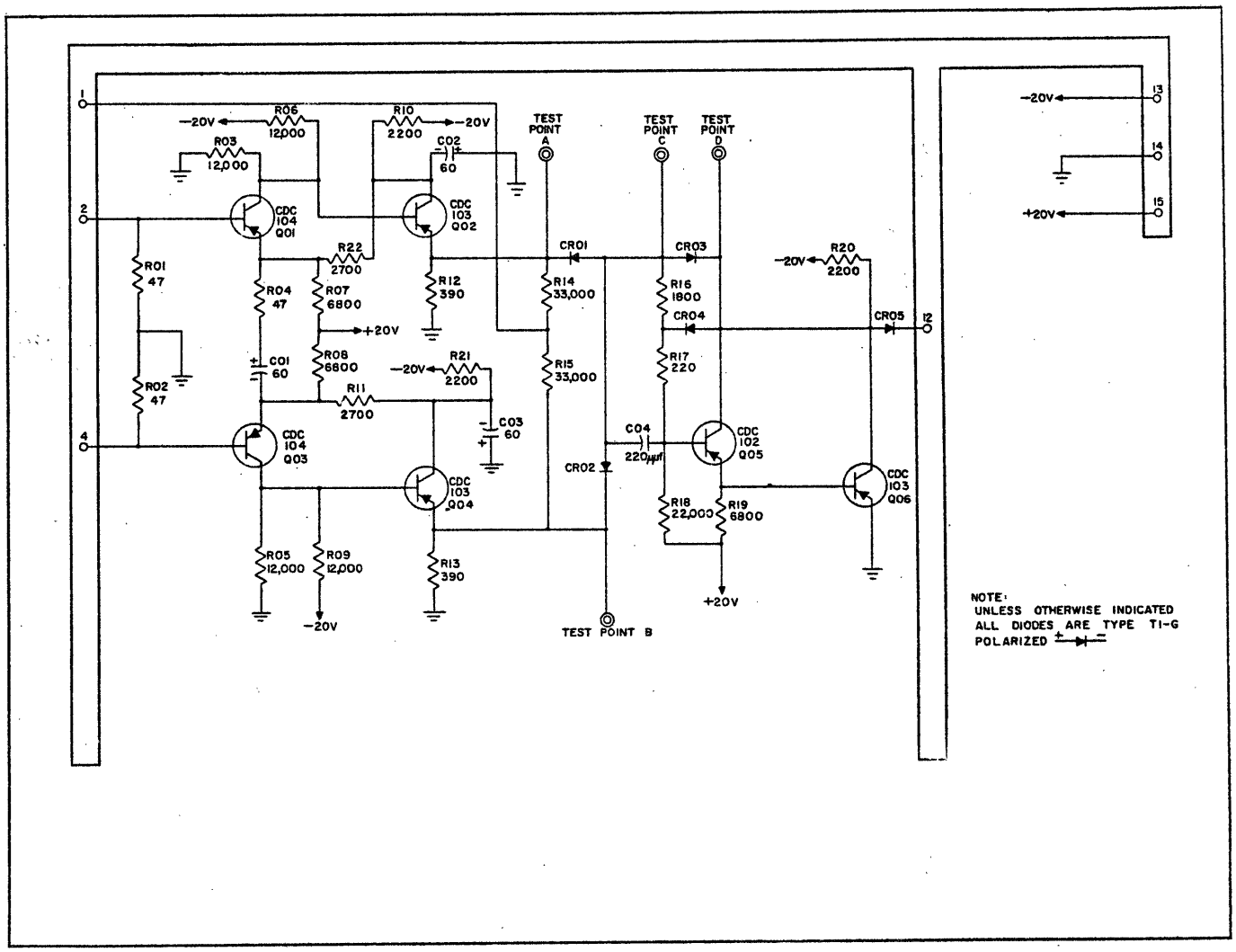
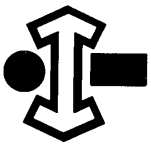



Figure 4-28. Sense Amplifier (Card Type 56).



# CHAPTER 5

# INPUT-OUTPUT SECTION

The input-output section of the computer (figure 5-1) provides the methods for data exchange and for proper control of information transmission between the computer and the various external equipments. These equipments are linked to and communicate with the computer through four cable groups; all information must enter or leave the computer through one of these cable groups.

Information normally is exchanged between the computer and the external equipments in a block of words at a word-by-word rate. The data exchange rate is determined by the speed of the particular equipment in communication with the computer.

The heart of the input-output control circuitry is the auxiliary scanner, the function of which is to sequentially sample each of the six buffer channels, the common interrupt line, and the advance clock circuit to determine whether any of these demand action. The computer responds to an action request from one of these eight sources only when the scanner is at the position associated with that source.

Three basic steps are performed in establishing a communication path and handling transmissions: 1) control of external equipment; 2) activating a channel; 3) the data transmission process itself.

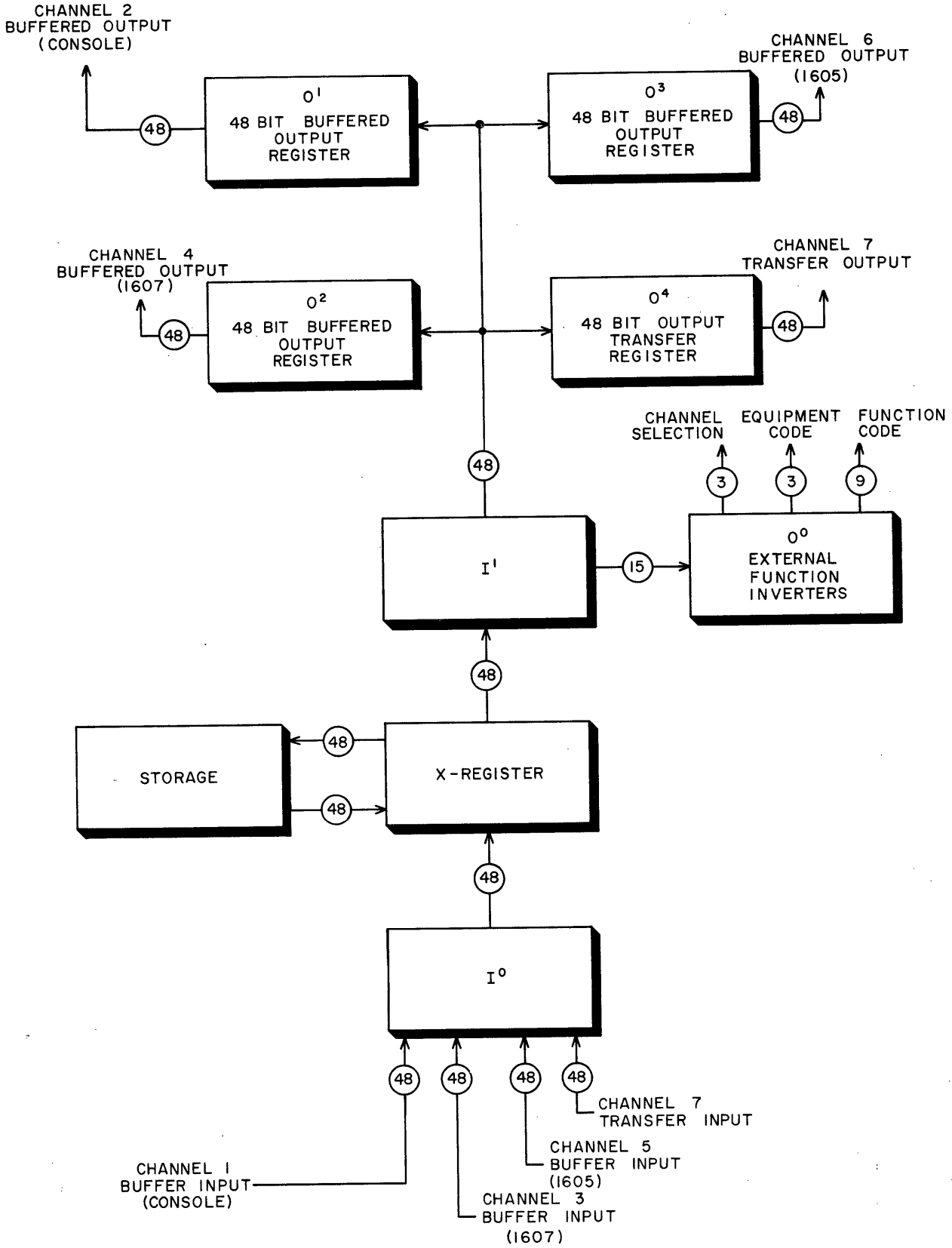
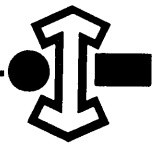



Figure 5-1. Overall Logic Diagram Input-Output Section.



## COMMUNICATION CABLE GROUPS

Three of the cable groups contain buffer channels (one input buffer and one output buffer channel in each group) and the fourth contains a single transfer channel. An external equipment must use only the two channels of the cable group to which it is linked. Low-speed equipments use the three buffer cable groups; high-speed equipments use the transfer cable group.

Cable group 1 is associated with input channel 1 and output channel 2; cable group 2, with input channel 3 and output channel 4; cable group 3, with input channel 5 and output channel 6; cable group 4, with transfer channel 7. Groups 1, 2 and 3 offer a buffer type of communication; group 4, a transfer type.

Each cable group is composed of a set of six cables. Each cable, in turn, is composed of 23 twisted-pair information or control lines and one twisted-pair common ground line; line assignments of a cable group are outlined in table 5-1. The buffer cables connected to external equipments are interchangeable as they are all wired in a similar pattern. Cable group 4 is always used as the transfer channel. Unused wires within the cables transmit and receive "0".

TABLE 5-1a. LINE ASSIGNMENT OF A GIVEN CABLE GROUP - DATA LINES

| | | |
|---|---|---|
| **Input Equipment to Computer** | | |
| Input data (48 lines) | | Comprise two complete cables and two lines of a third cable of a buffer or transfer cable group. |
| Input data ready (1 line) | Function: | Indicates equipment input register contains information which computer may sample. |
| | Operation: | Turned off by input data resume from computer. (Computer resync circuitry orients itself about leading edge of ready signal; auxiliary scanner is stopped and input word is passed to computer.) |
| **Computer to Input Equipment** | | |
| Input data resume (1 line) | Function: | Indicates to equipment that computer has accepted input word. |
| | Operation: | Turned off by input data ready; turned on when computer has accepted and stored input word. |
| **Computer to Output Equipment** | | |
| Output (48 lines) | | Comprise two complete cables and two lines of a third cable of a buffer or transfer cable group. |
| Output data ready (1 line) | Function: | Accompanies output data from computer. |
| | Operation: | Turned on when computer has word of information ready for equipment; off by resume from equipment. |
| **Output Equipment to Computer** | | |
| Output data resume (1 line) | Function: | Indicates that equipment has accepted word. |
| | Operation: | Turned on when equipment has accepted word. (Computer resync circuitry orients itself about the trailing edge of resume; when signal drops auxiliary scanner stops and another word is exchanged. Computer prepares a word for exchange while output data resume signal is up.) |

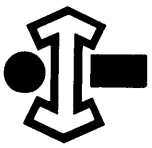


TABLE 5-1b. LINE ASSIGNMENT OF A GIVEN CABLE GROUP - CONTROL LINES

| | | |
|---|---|---|
| **Computer to External Equipment** | | |
| External Function (12 lines) | Function: | Carries external function (EF) codes to select or sense a condition within the equipment. |
| | Operation: | Lines continuously monitored by all equipment. Appropriate function or sense ready alerts equipment on a given channel to sample EF code on lines. |
| External master clear (1 line) | Function: | Clears all equipment on all channels. |
| | Operation: | Occurs when Clear switch at console is in up position. |
| **External Equipment to Computer** | | |
| Sense response (1 line) | Function: | Indicates equipment's reply to sense code. |
| | Operation: | ON indicates presence of condition specified by sense code; OFF indicates absence of condition. |
| Interrupt (1 line) | Operation: | External equipment or internal computer control sends signal when a condition arises that was previously selected by a 74.0. |
| **Computer to Input Equipment** | | |
| Input Function Ready (1 line) | Function: | Accompanies EF select code. |
| | Operation: | Turned on by instruction 74.0. Causes input equipment to translate EF code. |
| Input Sense Ready (1 line) | Function: | Accompanies EF Sense Code. |
| | Operation: | Turned on by instruction 74.7. Causes input equipment to translate EF code and send response back to computer. |
| Input Buffer Active (1 line) | Function: | Indicates computer is prepared to receive a block of data. |
| | Operation: | Turned on when input buffer channel is activated by instruction 74.1, 74.3 or 74.5. Remains on until final word of block is entered in storage. |

TABLE 5-1b. (Cont'd)

| | | |
|---|---|---|
| Computer to Output Equipment | | |
| Output Function Ready (1 line) | Function: | Accompanies EF select code. |
| | Operation: | Turned on by instruction 74.0. Causes output equipment to translate EF code. |
| Output Sense Ready (1 line) | Function: | Accompanies EF sense code. |
| | Operation: | Turned on by instruction 74.7. Causes output equipment to translate EF code and send response back to computer. |
| Output Buffer Active (1 line) | Function: | Indicates computer is prepared to transmit a block of data. |
| | Operation: | Turned on when output buffer channel is activated by instructions 74.2, 74.4, or 74.6. Remains on until final word of block is transmitted to output equipment. |

## DATA AND CONTROL INFORMATION PRESENTATION

The input-output circuitry uses static, direct-coupled, parallel circuits. Data is presented on the communication lines as one of two D-C voltage levels; the binary "1" condition is represented by 0v, the binary "0" condition is represented by -20v. All binary digits of a word are presented simultaneously on the wires of the cable group.

A second set of connectors in each external equipment enables more than one equipment to be connected to the computer via one cable group. Unique select codes provided by the computer determine which equipment attached to a cable is to communicate with the computer.

## EQUIPMENT SELECTION

The External Function instruction (74) selects equipment to communicate with the computer. A unique equipment and its operating condition are selected by 74.0. Instruction 74.7 senses a specific condition within an equipment. The base execution address portion of 74.0 and 74.7 is the external function code, a part of which is sent to the equipment to either select or sense a specified condition. The base execution address portion of these two instructions

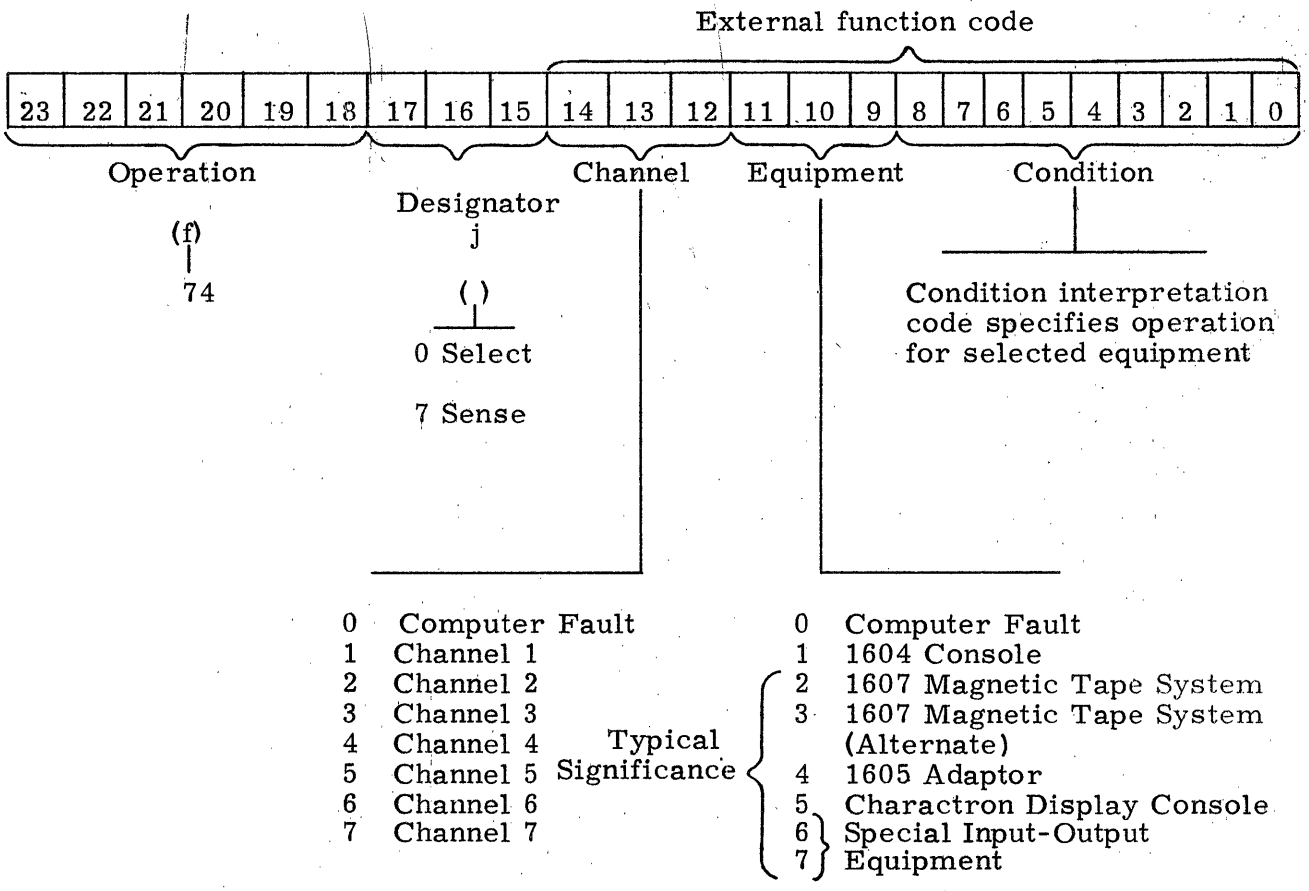



Figure 5-2. Individual Bit Significance of External Function Instructions.

is composed of three distinct elements: 1) channel selection; 2) equipment selection; 3) condition code.

Bits 12 through 14 determine the channel to which the remaining 12 bits of the code are sent. Bits 09 through 11 designate an equipment while bits 00 through 08 specify a condition for that equipment. Although bits 00 through 11 are sent to every equipment on the channel only the one identified by the equipment selection code responds.

The external function code is held by the X register. After the code passes through the external function inverters, it appears on the external function lines within the cable group. One of the following list of ready signals generated within the computer by bits 12 through 14 of instructions 74.0 and 74.7 enables interpretation of the equipment and condition codes by the equipment:

| | |
|---|---|
| input function ready | output function ready |
| input sense ready | output sense ready |

The use of 74.7 is determined by its position within the instruction word. If the instruction is placed in the upper position, it is a skip instruction and the lower instruction of the word is skipped if the sensed condition exists. If the condition does not exist, the lower instruction is performed in the normal manner. If, however, 74.7 is placed in the lower position, the instruction is repeated until the specified condition exists. This is a convenient method by which the computer can be made to await the establishment of a certain condition within the external equipment. The position of 74.0 within the instruction word does not affect its operation.

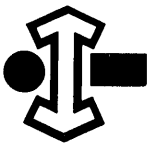




Figure 5-3 shows the path of data input to the least-significant stage of the X register. Each of the 48 stages of the X register has a similar logic arrangement between it and the cable groups. Note that data enter no input register as such but instead go to $X^1$ and then directly to storage.

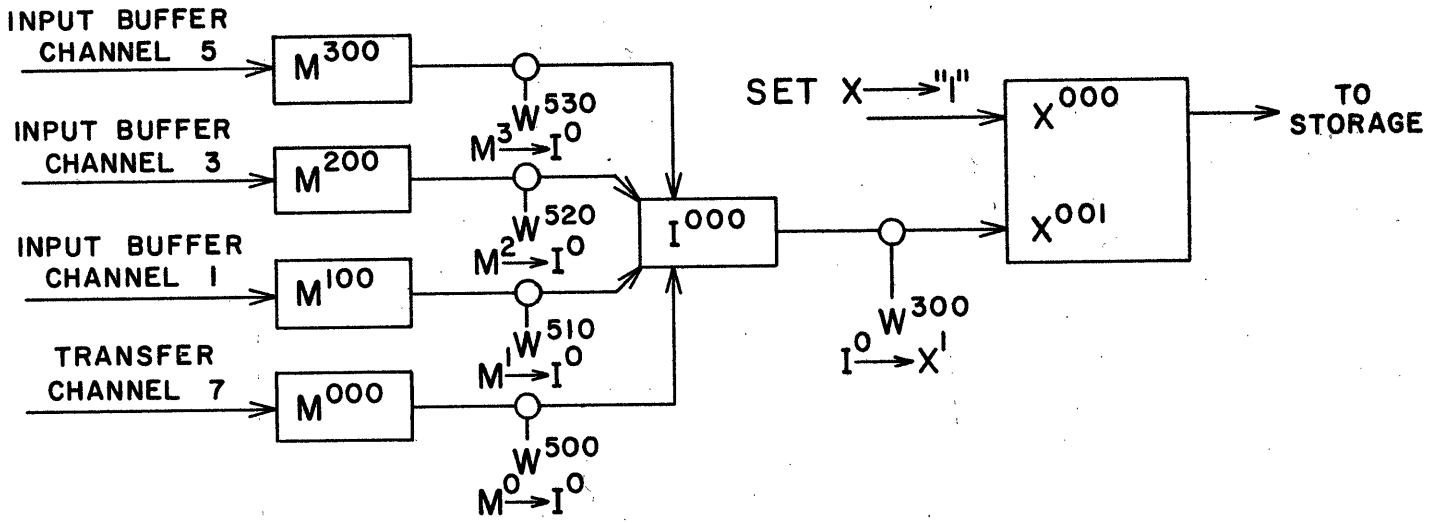



Figure 5-3. Data Input to the Least-Significant Stage of X Register.

Inputs from the four cable groups connect to inverter rank $I^0$. The output of $I^0$ goes to the X register as a single input rather than four. Because of the inversion by $I^0$, X is first set to all "1's" and the $I^0 \rightarrow X^1$ command clears appropriate bits.

## INPUT AMPLIFIERS

Type 61 cards contain the input amplifier circuits. Three identical circuits are located on each card and each is assigned an individual $M^{---}$ symbol. These amplifiers have two functions: they form the input boundary elements between the external equipment and the computer and they convert communication line signals to computer signal levels. Each of these amplifiers has a single input and two outputs.

## COMPUTER OUTPUT

Figure 5-4 shows the path of data output from the least-significant stage of the X register. Each of the 48 stages of the X register has a similar arrangement between it and the four cable groups. All output data is taken from storage prior to being supplied to the external equipment.

In order to supply the necessary number of outputs the X register connects to inverter rank $I^1$, using only one output. $I^1$, in turn, connects to all of the cable groups. Due to the inversion caused by $I^1$, it is connected to the clear side of $X^2$.

### OUTPUT REGISTERS

The computer has one separate output register for each of the four cable groups. The registers, $0^1$, $0^2$, $0^3$ and $0^4$, are each 48 bits long. These registers hold the output data prior to its acceptance by the external equipment. A clear $0^-$ command prepares the output register for the next word of output data; $X^2 \rightarrow 0^-$ command enters the word into the output register.

### OUTPUT AMPLIFIERS

Type 62 cards contain the output amplifier circuits. Three identical circuits are located on each card. Each circuit is assigned an individual $L^{---}$ symbol. Each amplifier has a single input and a single output. The output amplifiers have two functions: they form the boundary elements between the computer and the cable groups and they convert computer signal levels to communication line levels.

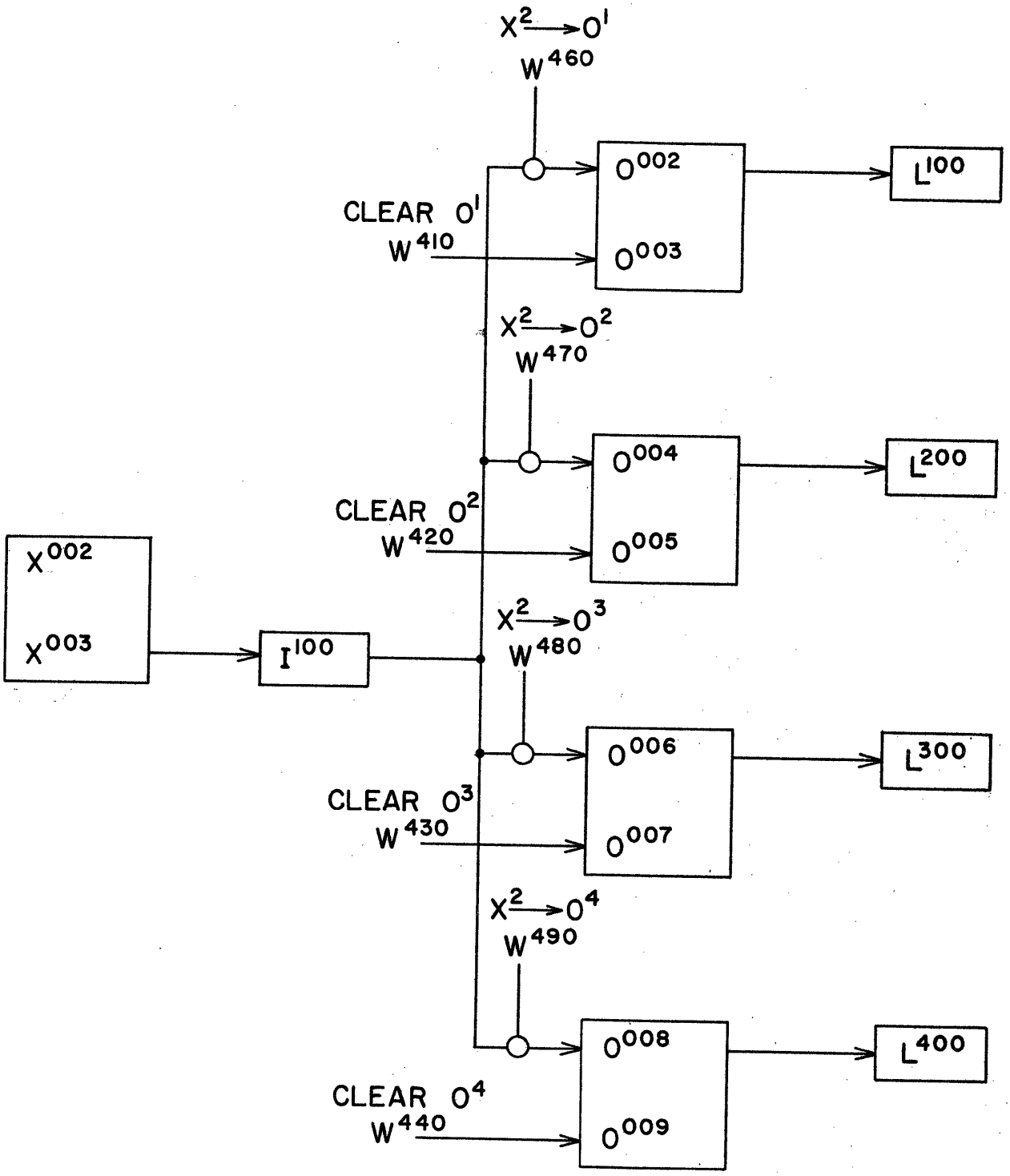



Figure 5-4. Data Output from the Least-Significant Stage of X Register.

EXTERNAL FUNCTION INVERTERS

The external function inverters supply the EF code of instructions 74.0 and 74.7 to the cable groups. All data within the X register automatically appears within the external function inverters; only the absence of the proper ready signal bars the interpretation of their content by the external equipment. Figure 5-5 shows the method by which the lowest bit of the external function code is made to appear on the function lines within the cable groups.

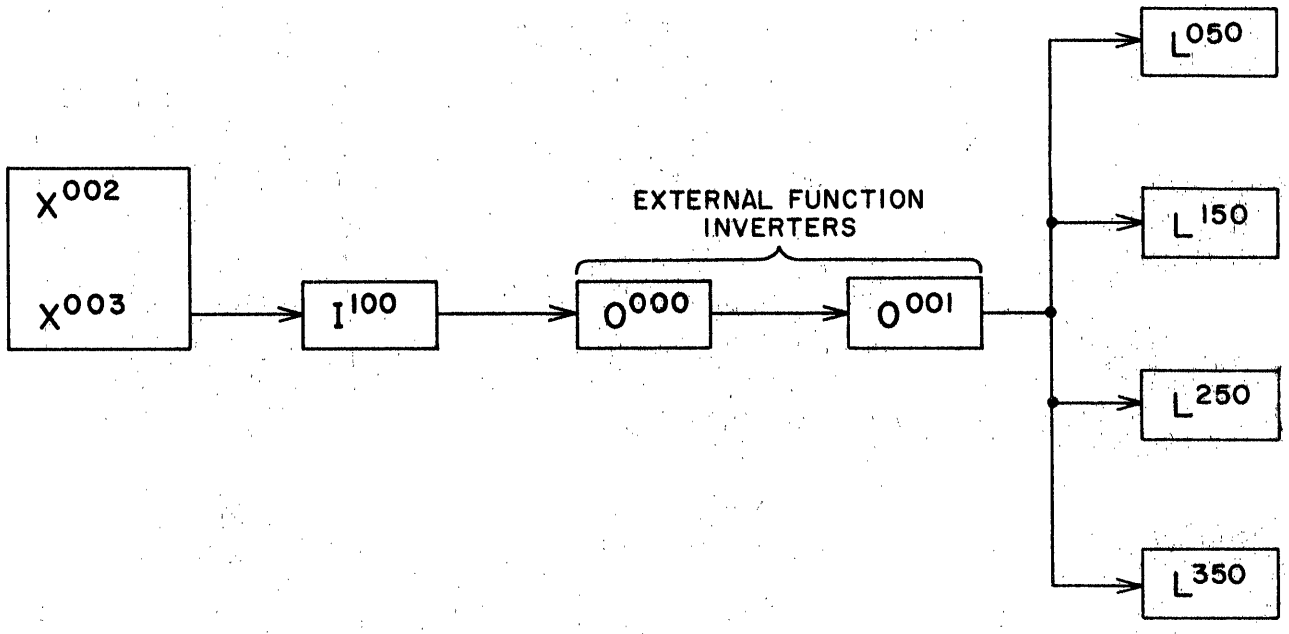
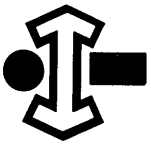



Figure 5-5. External Function Code Output from the Lowest Stage of X Register.



## INPUT-OUTPUT CONTROL

Data input and output are governed by control circuitry which is logically a part of the input-output section. This circuitry produces the signals which: 1) process each buffer channel; 2) respond to interrupt signals; 3) perform the advance clock operation; 4) generate the enables which select the special storage addresses.

Four major areas exist within the control circuitry which contribute toward the execution of these functions. These are scanner circuitry, auxiliary request circuitry, ready-resume circuitry and external function circuitry.

### AUXILIARY SCANNER

The auxiliary scanner guards against the possibility of a single external device monopolizing the computer, and gives proper recognition to interrupt and advance clock requests. It sequentially samples each of the six buffer channels, the interrupt line and the advance clock circuits. In the event that none of these lines demand processing, the scanner completes one entire cycle in 3.2 microseconds.

When an auxiliary operation must be performed, the scanner stops at the line which demanded the action. The computer is forced to halt execution of the main program and to perform the indicated operation by entering the Auxiliary sequence. When the request has been satisfied, the computer returns to the main program and the scanner resumes its operation with the next count. In the slowest case (action demands by all six buffer channels occurring with consecutive 66-microsecond instructions) communication is limited to a 5 KC word rate. During ordinary conditions, the word rate may be expected to increase in the order of approximately 10 times.

Figure 5-6 is a diagram of the scanner, which is essentially a specialized three-stage additive counter, the distinction being that only one FF of each rank changes state as the count is advanced. For each Rank I count there is one FF of Rank II that is set. Each Rank II FF is associated with one of the eight lines. Each Rank I binary count and the auxiliary function (and line) associated with it is as follows:

| AUX Function | Rank I Count | Rank II FF |
|---|---|---|
| Advance Clock | 000 | $K^{806/807}$ |
| Channel 1 | 001 | $K^{808/809}$ |
| Channel 3 | 011 | $K^{812/813}$ |
| Channel 2 | 010 | $K^{810/811}$ |
| Channel 6 | 110 | $K^{818/819}$ |
| Interrupt | 111 | $K^{820/821}$ |
| Channel 5 | 101 | $K^{816/817}$ |
| Channel 4 | 100 | $K^{814/815}$ |

When a Rank II FF is set the associated line is examined by one of the action request FFs ($K^{782}$ - $K^{797}$). The scanner is so constructed that if a line demands processing when it is sampled, the scanner will stop the next time Rank I reaches the count of the line in question.

The scanner continues to cycle as long as no auxiliary operations need to be performed, a condition which is indicated by a "0" output from the eight action request FFs. A "1" output from any of these FFs stops the scanner at the associated position.

A remote possibility exists that all FFs of the scanner may stabilize at "0" after initial turn-on. To remedy this situation, the eight FFs of Rank II are formed into two groups as

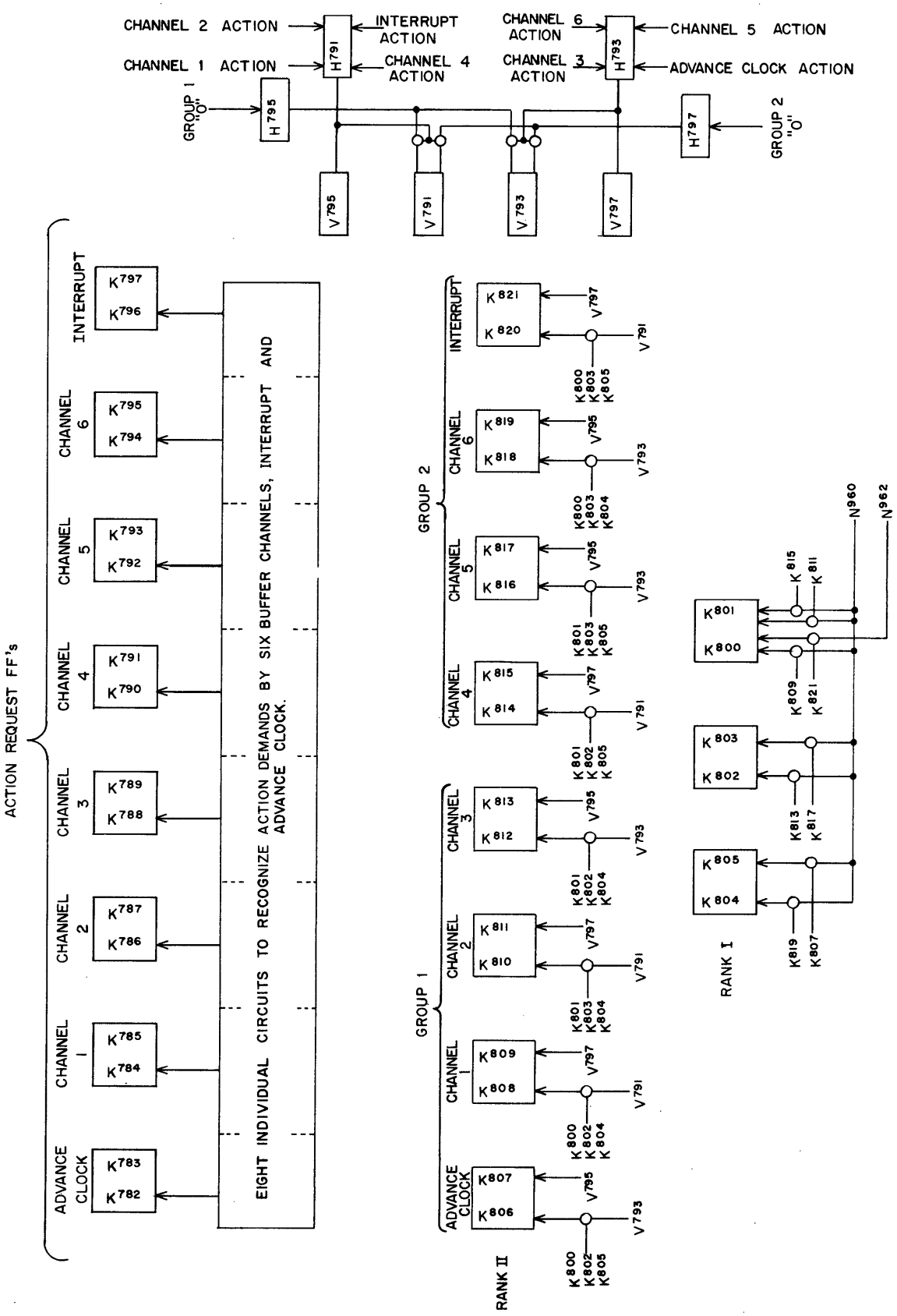



Figure 5-6. Auxiliary Scanner.

inputs to $H^{795}$ and $H^{797}$ to enable the inputs to all of the Rank II FFs.

When the scanner stops, the content of Rank I is transmitted to the auxiliary reference designator. Its state is translated to provide the gating for commands from the Auxiliary sequence. Thus, the position at which the scanner stops determines by means of the designator which commands the sequence will produce. This designator is also used during the execution of subinstructions 74.1 through 74.6.

AUXILIARY REQUEST

Initiation of the Auxiliary sequence is governed by the presence or absence of the auxiliary request signal; figure 5-7 shows how this signal is generated. Each time a line demands action, the scanner stops and this circuit then produces the auxiliary request signal which forces the computer to halt the main program and enter the Auxiliary sequence. Then FF $K^{712}$ $K^{713}$ and its slave $K^{714}$ $K^{715}$ are set to indicate an auxiliary request.

When a line demands action the scanner will stop when Rank I next reaches the count for that line and the auxiliary request circuitry will be activated through $K^{822/823}$. When no line is active, this FF changes state with every clock phase. The stopped scanner causes succeeding pulses of the same phase to enter either the set or clear sides of $K^{822/823}$ until finally the auxiliary request signal is produced.

DATA READY AND DATA RESUME

The signals which accompany the exchange of data between the computer and the external equipment are:

| | |
|---|---|
| input data ready | output data ready |
| input data resume | output data resume |

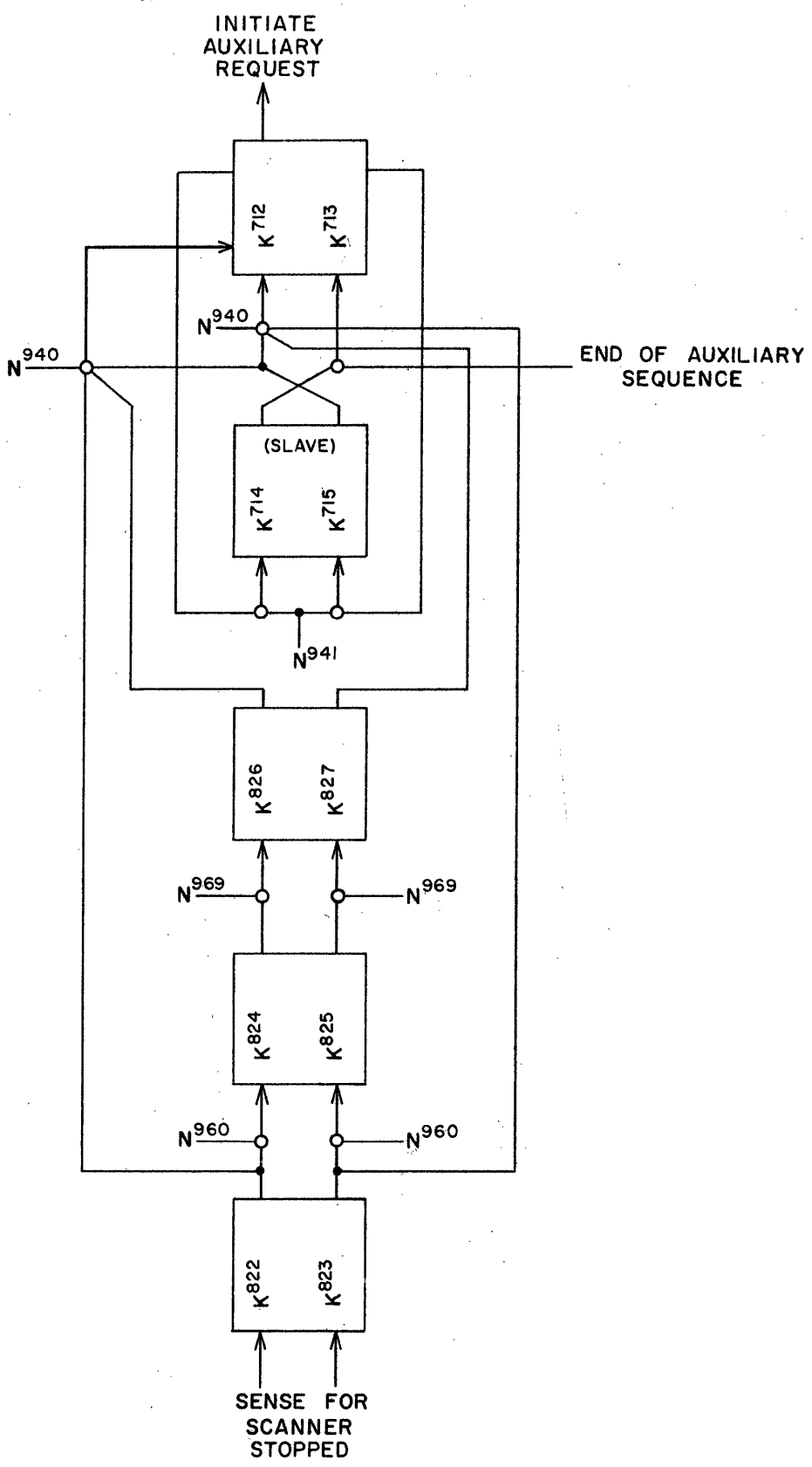



Figure 5-7. Auxiliary Request.

The ready signal accompanies the data while the resume is a return signal which indicates that more information may be transmitted.

Data Resume Signals

Within the computer, generation of the input data resume depends upon a simultaneous fulfillment of the following conditions:

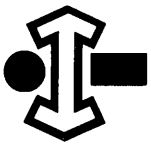
1) An active input channel. Buffer channels 1, 3, 5 are activated by a 74.1, 3, 5 instruction.
2) A resume from the storage section indicating that the previous word has been stored.

Generation of the output data resume which depends upon the operation of the external equipment indicates reception of the previous word and a readiness to receive the next word.

Data Ready Signals

Within the computer, the output data ready depends upon the simultaneous fulfillment of the following conditions:

1) An active output channel. Buffer channels 2, 4, 6 are activated by a 74.2, 4, 6 instruction.
2) One of the output registers ($0^1$ through $0^4$) in the full state.



A delay of 1.8 microseconds is provided between appearance of data on the output data lines and generation of the output data ready. This delay permits the lines to stabilize before the external equipment is allowed to sample them.

Production of the output data ready which depends upon conditions within the equipment indicates to the computer that data is present on the lines in a form which may be sampled.

Control Line Relationships for Input Communication

The exchange of control signals which are required in the transmission of a word to the computer from the external equipment is outlined in figure 5-8 and table 5-2.

Transition of the input data ready line to the on state may take place at any instant with respect to the synchronized (clocked) operations of the computer. To resolve runt pulses and to convert the input data ready signal into a single discrete pulse synchronized with the computer is the job of the input ready resync circuits.

Figure 5-9 shows the Channel 1 ready resync circuit. This circuit is typical also of the circuits for the other two input buffer channels. When instruction 74.1 is programmed, the External Function sequence sets the Buffer Active FF (3). From this point on the computer is considered to be in the buffer mode until the final word of the block is received, at which time this FF is cleared.

As long as no input ready is present, the Input Ready FF is cleared by the even sync pulse. During the odd sync pulse which follows, the Wait Ready FF is set. The state of the circuit with no input ready present is: (1) Buffer Active FF set; (2) Ready FF clear; (3) Wait Ready FF set; (4) Action Request FF clear.

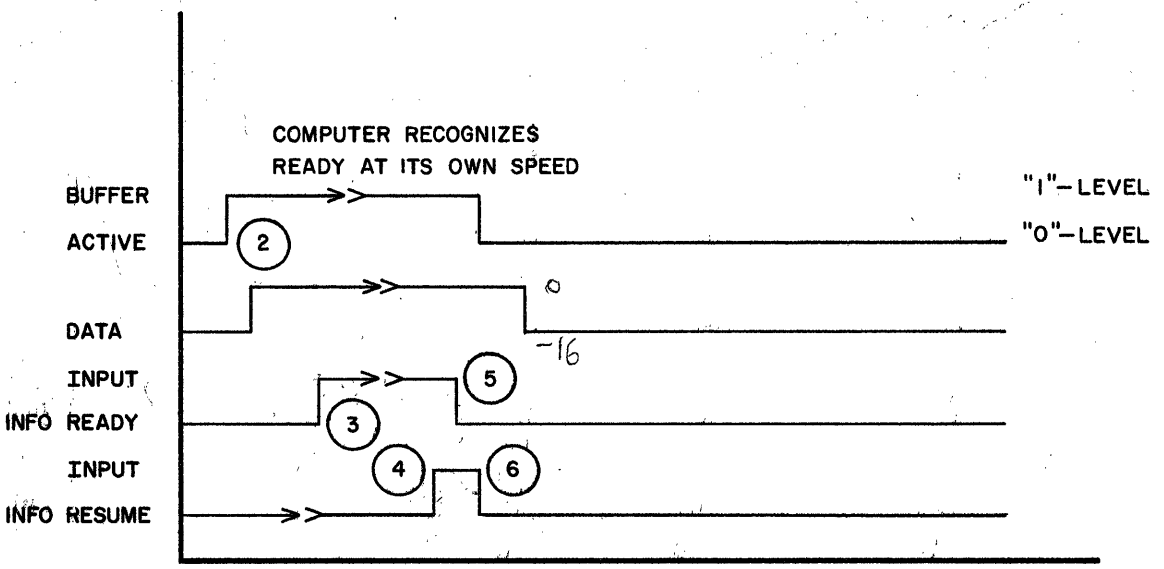



Figure 5-8. Exchange of Control Signals for One Input Word, Timing Diagram.

TABLE 5-2. EXCHANGE OF CONTROL SIGNALS FOR ONE INPUT WORD

| | |
|---|---|
| | 1) Computer, through proper combination of external select codes and external sense codes, establishes equipment from which it is to take information. |
| | 2) Computer activates input buffer line. This line remains up until final word of the block is transmitted. |
| Repeat until entire block is transmitted* | 3) Equipment places a word on the lines and produces input data ready. |
| | 4) Computer, when it has accepted the word, produces input data resume. |
| | 5) Input data resume causes equipment to turn off its input data ready signal and prepare another word of input data for computer. |
| | 6) Removal of input data ready causes computer to turn off resume. |

*When the final word of the block has been transmitted and steps 3 through 6 are completed, the buffer active signal is dropped and the buffer operation is terminated.

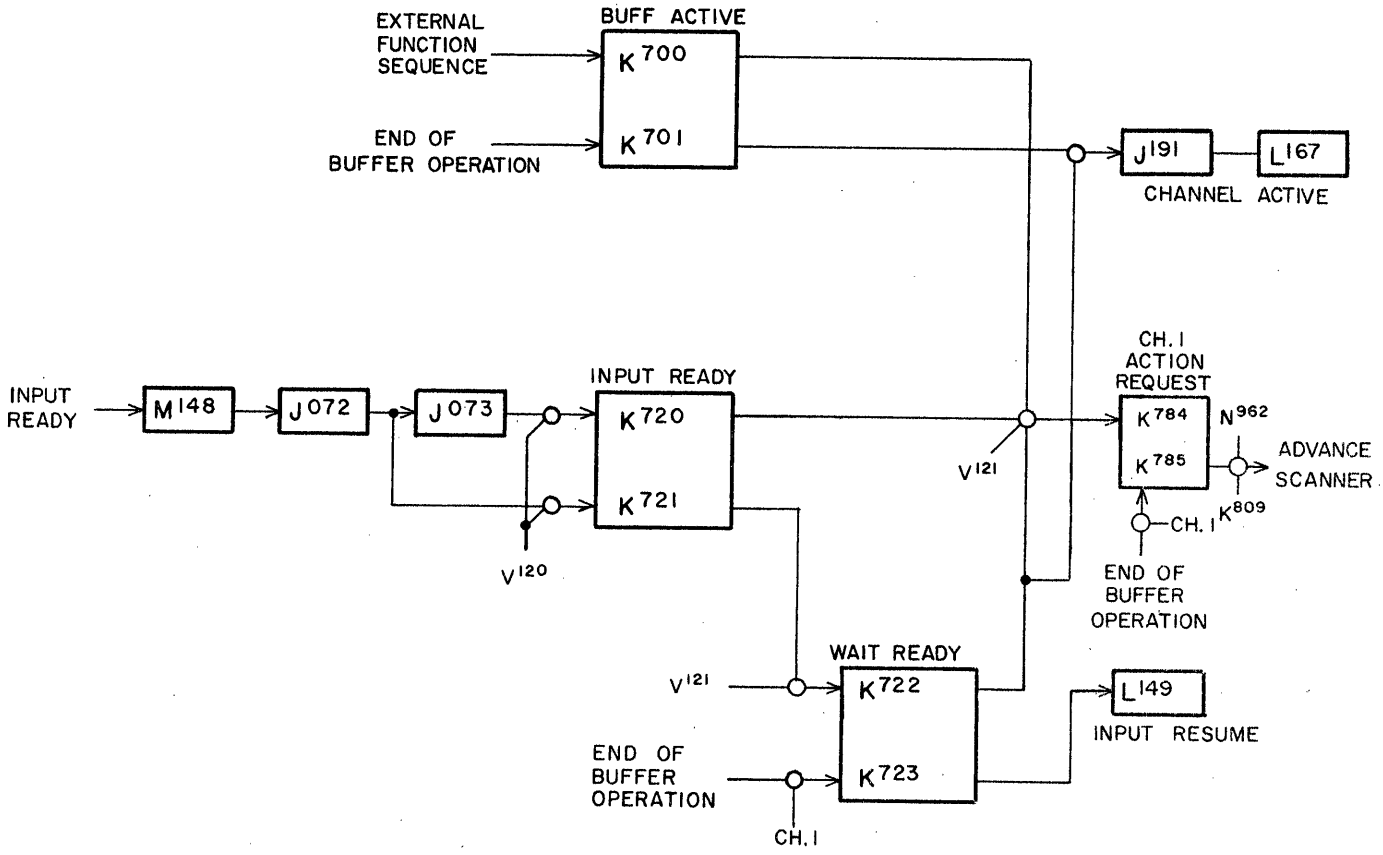



Figure 5-9. Channel 1 Ready Resync Circuit.

The occurrence of the input ready generates the following sequence of actions:

1) even sync pulse samples ready and sets Input Ready FF
2) following odd sync pulse samples Input Ready FF and sets Action Request FF
3) scanner stops at channel 1 ($K^{809}$) because Action Request FF is set

The circuit remains in this state until late in the resulting buffer operation. At this time the Wait Ready FF and Action Request FF are cleared. Clearing the first one sends out the input resume which causes the equipment to drop its ready. This returns the resync circuit to its initial condition. Clearing the Action Request FF allows the scanner to advance to the next position.

Control Line Relationships for Output Communication

The exchange of control signals for the transmission of a word to the external equipment from the computer is outlined in figure 5-10 and table 5-3.

TABLE 5-3. EXCHANGE OF CONTROL SIGNALS FOR ONE OUTPUT WORD

| | |
|---|---|
| | 1) Computer, through proper combinations of external select codes and external sense codes, establishes equipment to which information is to be sent. |
| | 2) Computer activates output buffer active line. This line remains on until final word of the block is transmitted. |
| Repeat until entire block is transmitted* | 3) Computer places a word in proper output register. All data lines are energized in parallel. |
| | 4) When all data lines are stable, computer generates output data ready signal which indicates to equipment that data is available on lines in a stable steady-state form. |
| | 5) Equipment accepts output data ready signal and information at its own rate. When it accepts output data ready it produces output data resume which it returns to computer. |
| | 6) Computer accepts output data resume and turns off output data ready. |
| | 7) Removal of output data ready in computer causes termination of output data resume signal within equipment. |

*When the final word has been transmitted and steps 3 through 7 are completed the output buffer active signal is dropped and the buffer operation is terminated.

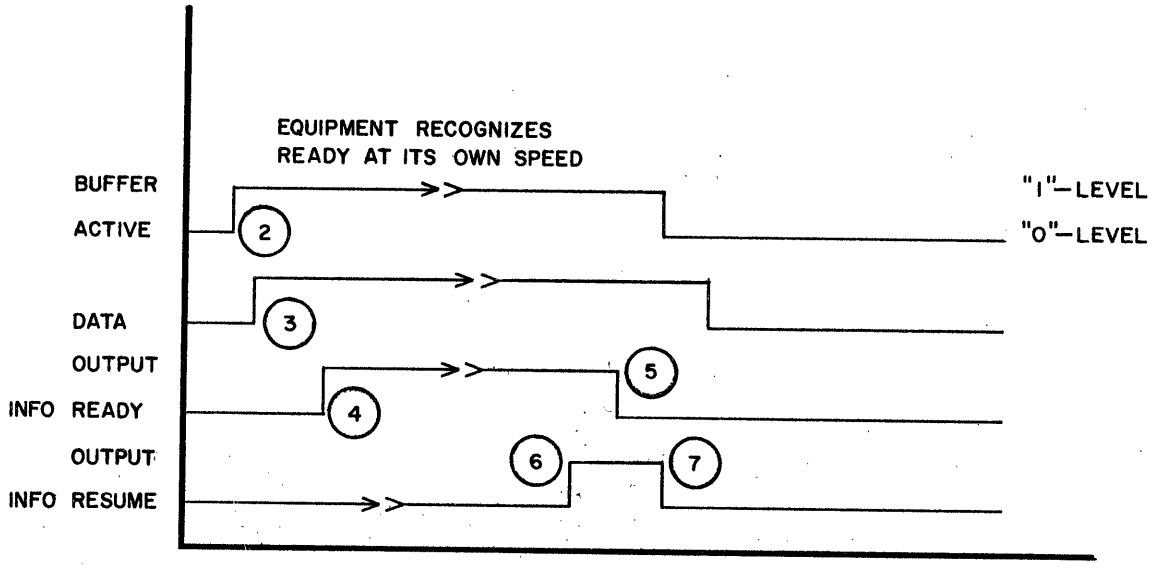



Figure 5-10. Exchange of Control Signals for One Output Word, Timing Diagram.

As is the case with the input data ready, transition of the output data resume line to the on state may take place at any time with respect to the synchronized (clocked) operations of the computer. Figure 5-11 shows the channel 2 resume resync circuit, which is typical also of the corresponding circuits for the other two output buffer channels. When instruction 74.2 is programmed, the External Function sequence sets the Buffer Active FF (3). From this point on the computer is considered to be in the buffer mode until the final word of the block is received, at which time this FF is cleared.

After sending out a ready at the end of a buffer operation, a period of waiting ensues until a resume is received. During the waiting period the state of the circuit is: Buffer Active set, Output Resume clear, Wait Resume set, and Action Request clear.

Later, when the output equipment returns a resume, the following actions occur:

1) Even sync pulse samples resume and sets Output Resume FF.
2) The following odd sync pulse samples Output Resume FF and clears Wait Resume FF to turn off output ready.
3) Turning off output ready causes output resume to be turned off.
4) Even sync pulse samples resume line and clears Output Resume FF.
5) The following odd sync pulse sets Action Request FF.
6) Scanner stops at channel 2 ($K^{811}$) because Action Request FF is set.
7) At the end of the buffer operation that results from stopping the scanner the Action Request FF is cleared to allow the scanner to advance and the Wait Resume FF is set to send the output ready.

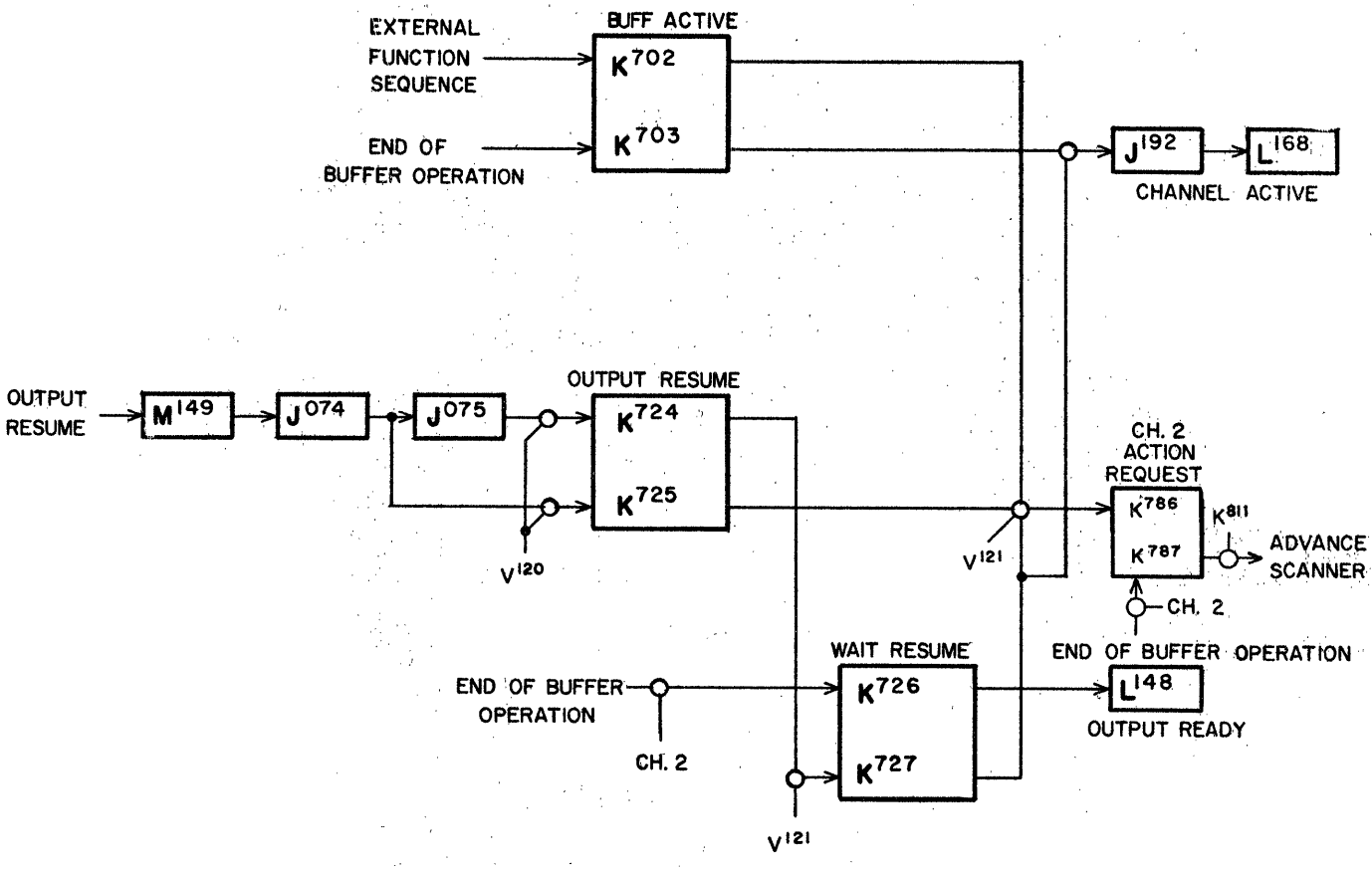
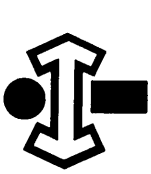



Figure 5-11. Channel 2 Resume Resync Circuit.



# INPUT-OUTPUT INSTRUCTIONS

Information transfer into and out of the computer is the function of three computer instructions: (1) 62 Input Transfer; (2) 63 Output Transfer; (3) 74 External Function.

## TRANSFER INSTRUCTIONS (62-63)

The designator, b, of the instruction chooses the B register whose contents specify the number of words to be transferred. The quantity, $m + B^b - 1$, chooses the storage location of the first word to be transferred. The final word comes from the address given by m. The reduced value of the specified B register, when added to the m designator, indicates the storage locations in which the incoming or outgoing words are stored.

## EXTERNAL FUNCTION INSTRUCTION (74)

Instruction 74 provides the means for selecting and sensing a variety of conditions within the computer and the external equipments. These conditions are determined by the external function code, m, a 15-bit designator which is used to specify the appropriate channel, equipment and condition. The codes for each equipment are given in the books accompanying those equipments, or, in the case of the console equipments, in chapter 6.

### EF Select, 74.0

A 74.0 (external function select) selects a condition within the computer, or chooses an external equipment and places it in a particular operation mode as determined by m. Within the computer, execution of the instruction sends the code to the relevant portion of internal computer control where it produces the specified condition.

Codes for internal conditions (table 5-4) fall into three groups: status of the buffer channels, arithmetic faults and real-time clock. The buffer channel and arithmetic fault select codes (figure 5-12) either clear a specified FF or set a FF which then enables an interrupt to be produced when the specified condition arises.

TABLE 5-4. EXTERNAL FUNCTION CODES

| INTERNAL | |
|---|---|
| 74.0 SELECT | 74.7 SENSE |
| 00010 Interrupt on channel 1 inactive | 00010 Exit on channel 1 active |
| 00011 Remove selection above | 00011 Exit on channel 1 inactive |
| 00020 Interrupt on channel 2 inactive | 00020 Exit on channel 2 active |
| 00021 Remove selection above | 00021 Exit on channel 2 inactive |
| 00030 Interrupt on channel 3 inactive | 00030 Exit on channel 3 active |
| 00031 Remove selection above | 00031 Exit on channel 3 inactive |
| 00040 Interrupt on channel 4 inactive | 00040 Exit on channel 4 active |
| 00041 Remove selection above | 00041 Exit on channel 4 inactive |
| 00050 Interrupt on channel 5 inactive | 00050 Exit on channel 5 active |
| 00051 Remove selection above | 00051 Exit on channel 5 inactive |
| 00060 Interrupt on channel 6 inactive | 00060 Exit on channel 6 active |
| 00061 Remove selection above | 00061 Exit on channel 6 inactive |
| 00070 Clear arithmetic faults | 00110 Exit on divide fault |
| 00100 Interrupt on arithmetic faults | 00111 Exit on no divide fault |
| 00101 Remove selection above | 00120 Exit on shift fault |
| 01000 Start real-time clock | 00121 Exit on no shift fault |
| 02000 Stop real-time clock | 00130 Exit on overflow fault |
| | 00131 Exit on no overflow fault |
| | 00140 Exit on exponent fault |
| | 00141 Exit on no exponent fault |

Two codes are reserved for use with the real-time clock. One, 01000, sets the Clock Disconnect FF (figure 5-18); this code enables the circuits which increment the clock count held in special storage address 00000. The other, 02000, clears the Clock Disconnect FF and inhibits the advance clock circuitry.

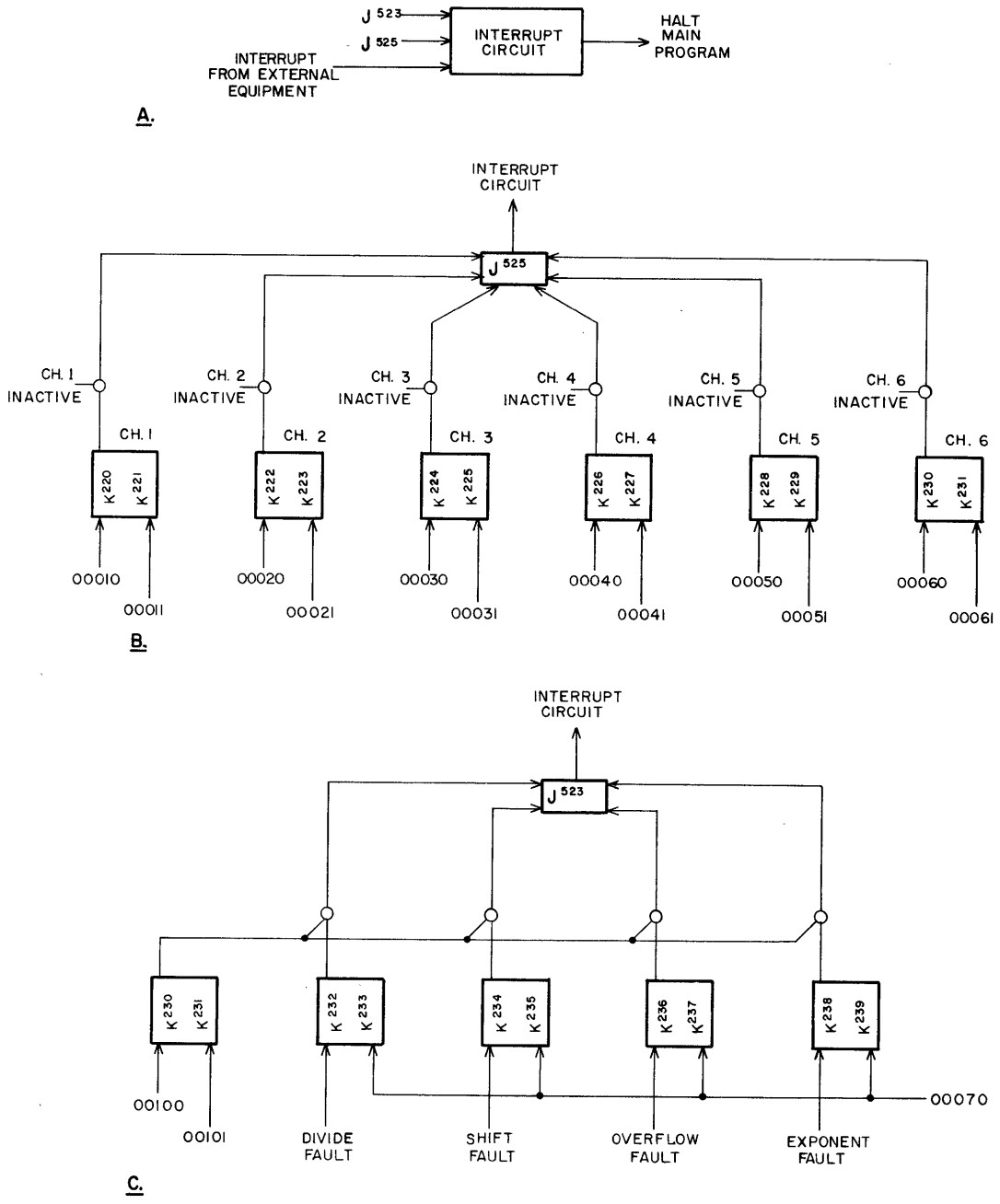



Figure 5-12. Internal Select Codes and Interrupt Generation.

EF Sense, 74.7

A 74.7 (external function sense) senses the presence or absence of conditions within the equipment or the computer. The presence or absence of the condition (depending on the code used) produces a sense response signal which is returned to the input-output section of the computer.

Instruction 74.7 is a skip instruction when in the upper position within an instruction word. The sense response signal determines whether a full or half exit is to be taken to the next instruction.

Two groups of internal conditions may be sensed: buffer channel status and arithmetic faults. Figure 5-13 shows how the sense codes are combined with the specified conditions to produce a sense return signal from $J^{968}$. A pair of sense codes are associated with each condition (table 5-4); codes differ only in the lowest digit. Note that the sense response signal from $J^{968}$ is produced independent of the lowest digit. For example, the sense response from $J^{968}$ is "1" when either code 00010 or code 00011 are used and channel 1 is active. The lowest digit of the sense code is combined within the sense response circuits with the signal from $J^{968}$; this combination determines whether the 74.7 instruction is to make a full or a half exit.

Instruction 74.1-6 chooses one of six buffer channels as determined by the specific value of the designator j. The m designator, which holds the storage address of the initial word of the block of information buffered, is sent to the upper address position of special storage location 0000j. Special storage addresses 00001 through 00006 are buffer control addresses whose function it is to hold the control word for the associated buffer channel. The control word is divided into an upper and lower address: the upper address contains the initial address of the block of information; the lower address contains the terminal address (plus one) of the block. The purpose of the control word is to define the block of storage locations to be used in the buffer operation and to terminate the operation when the initial and terminal addresses are equal.

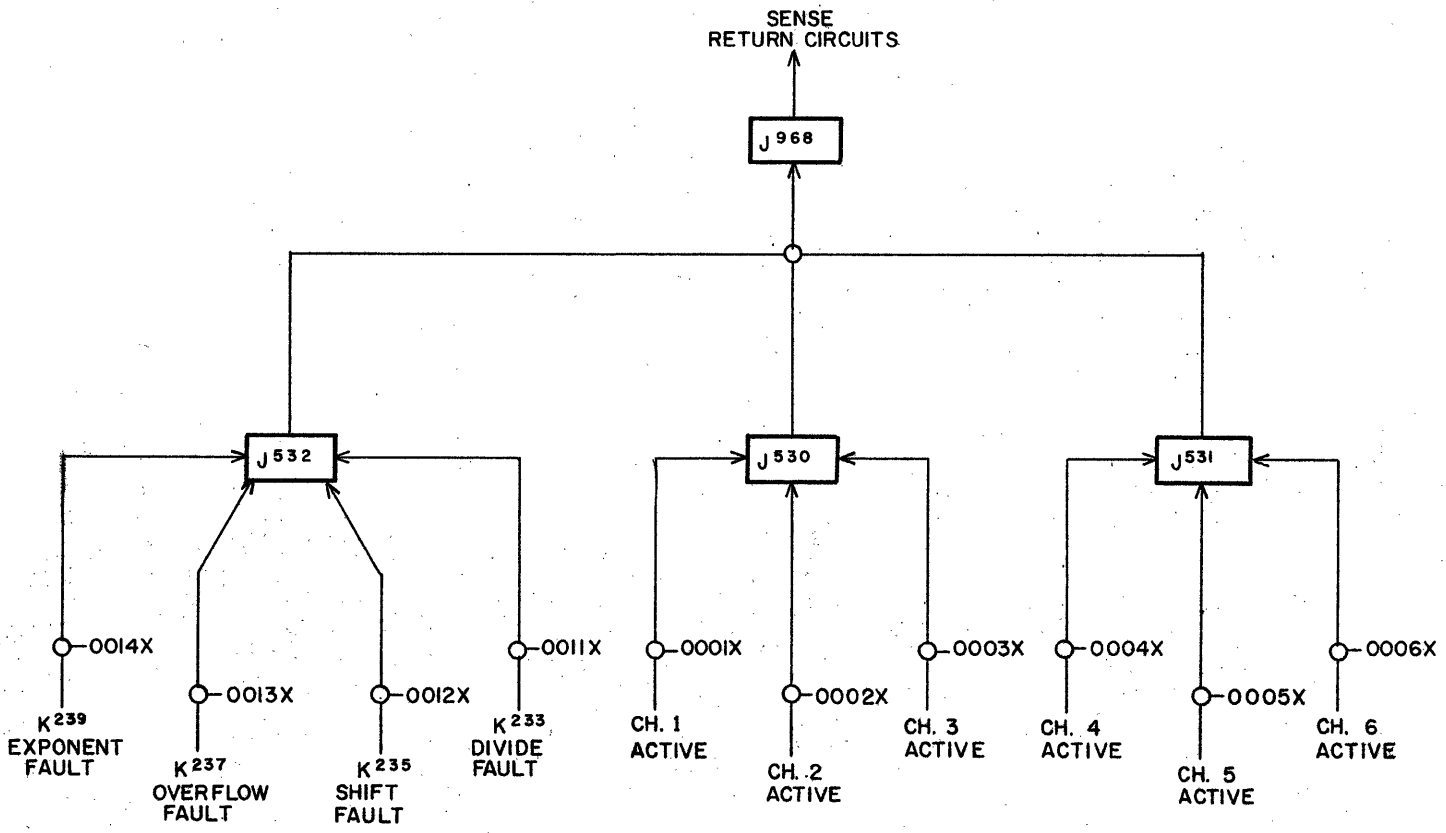



Figure 5-13. Sense Codes and Return Signal.

## EXTERNAL FUNCTION SEQUENCE

The External Function sequence (figure 5-14) executes instruction 74, External Function. The basic duties performed by the sequence are:

1) Select an operating condition for an external equipment or part of computer control (j=0).
2) Sense a condition within an external equipment or a part of computer control (j=7).
3) Activate buffer channel (j=1-6); the duty to be performed is specified by j.

In each case the unmodified execution address is used somewhat differently.

### SELECTING A CONDITION

During a 74.0 instruction the sequence sends the code contained in the execution address to a specified channel (1 through 7) or to computer control. This code establishes the condition. Thus it is through this instruction that the program retains control of external equipment and certain internal matters. For 74.0 the following commands are generated:

Clear $X^1$
$U^2 \rightarrow X^1$
$X^1 \rightarrow X^2$
$X^2 \rightarrow 0^0$ (sends out code)
Set Sense Resync FF if Exit FF = "0"
Wait 8 microseconds (due to EF counter)
Full or half exit (chosen by state of Sense Resync FF)

At the conclusion of these steps the code has been sent to the specified equipment or part of internal control. The purpose of the EF counter is to hold the code on the lines for a long enough period (8 microseconds) to insure its proper sampling.

SENSING A CONDITION

During a 74.7 instruction the sequence sends the code contained in the execution address to a specified channel (1 through 7) or to computer control. This code specifies the condition to be examined; the presence or absence of the condition is then indicated by the sense response signal. In this way the program can determine the state of an equipment or a part of computer control. The internal conditions which may be sensed by 74.7 appear in table 5-4.

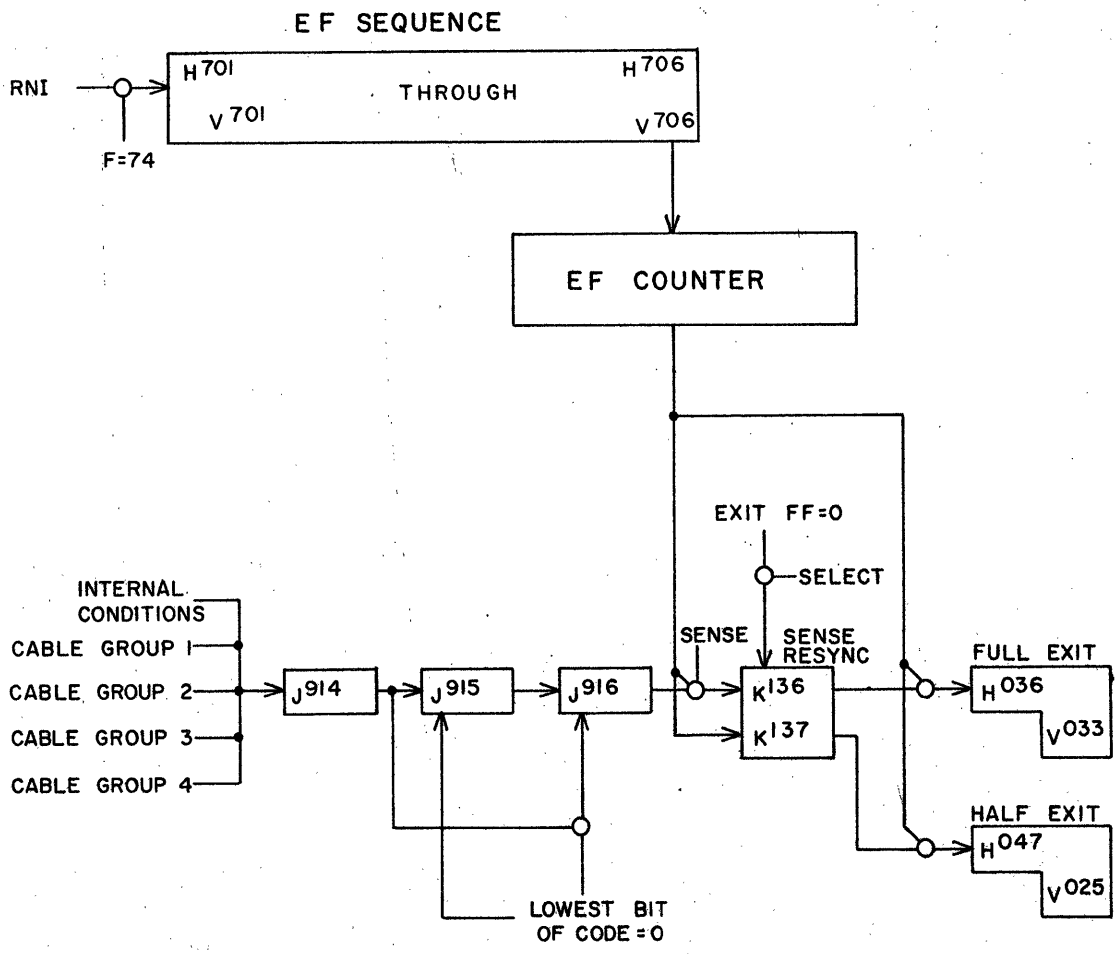



Figure 5-14. External Function Sequence and Associated Circuits.

For instruction 74.7 the following commands are generated:

Clear $X^1$

$U^2 \rightarrow X^1$

$X^1 \rightarrow X^2$

$X^2 \rightarrow 0^0$ (sends out code)

Wait 8 microseconds (due to EF counter)

Combine Sense Response from $J^{914}$ with lowest bit of code

Resynchronize the output of $J^{916}$

Full or half exit as determined by response

A pair of sense codes are associated with each condition (table 5-4). For example: 00010, exit on channel 1 active; 00011, exit on channel 1 inactive. The lowest bit of the code is not sent out but rather is combined with the response received from $J^{914}$. For this example, if channel 1 is active, the response received by $J^{914}$ is a "1" regardless of which code is used; if channel 1 is inactive, $J^{914}$ receives a "0". After this response is combined with the lowest bit, the output of $J^{916}$ is "1" if the code is 00010 and channel 1 is active, or the code is 00011 and channel 1 is inactive. The full exit is taken for these cases. When the output of $J^{916}$ is "0" the half exit is taken. The exit chosen indicates to the program whether the specified condition is present.

When the 74.7 instruction appears in the upper position, taking the full exit skips the lower instruction and taking the half exit leads to execution of the lower instruction. When 74.7 appears in the lower position, the full exit leads to the next instruction when the specified condition exists. When the condition does not exist the half exit causes the 74.7 to be repeated. Repetition of the instruction continues until the condition does exist.

ACTIVATING A BUFFER CHANNEL

The 74 instructions with j = 1-6 activates buffer channel j. The execution address of the instruction is the initial address of the block of words to be buffered on the channel. The following steps are involved in the execution of the instruction:

1) Store m (initial address) in upper address position of buffer control word for channel j (see figure 5-15). Address 0000j holds this word. The terminal address of the buffer is loaded in lower address position of address 0000j by a previous instruction.
2) Compare initial and terminal addresses.
3) Set Buffer Active FF for channel j if addresses are unequal.

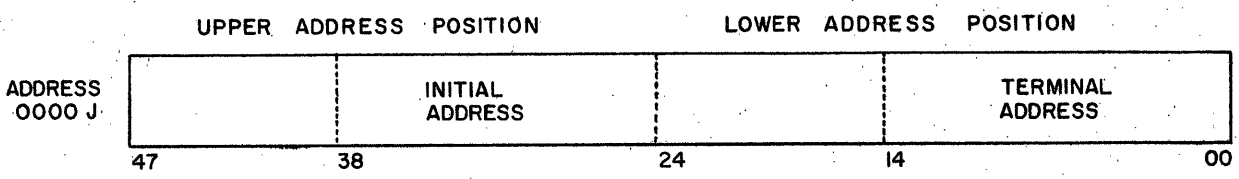



Figure 5-15. Buffer Control Word for Channel j.

Figure 5-16 shows the sequence that performs 74.1-6. It consists of the EF sequence ($H^{70-}$) and a portion of the Auxiliary sequence ($H^{76-}$, $H^{77-}$, $H^{78-}$). The following commands prepare for storing the initial address in the upper address position of the buffer control word.

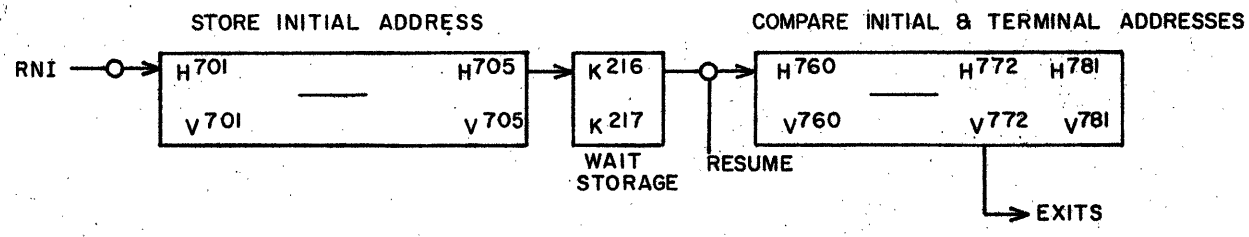



Figure 5-16. External Function and Auxiliary Sequences for Instruction 74.1-6.

$U^1 \rightarrow U^2$

Advance P (prepares for exiting)

Clear $X^1$

Transmit j no auxiliary reference designator

Initiate storage (address 0000j)

$U^2 \rightarrow X^1_{LA}$

$X^1_L \rightarrow X^2_U$

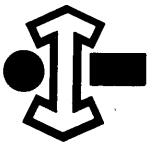
$X^2 \rightarrow X^1$

Complement Exit FF (prepares for exiting)

Wait storage

Following these commands, the initial address is held in $X^1$ ready to be written into storage. This storage reference not only provides for writing the initial address but also for reading out the terminal address, which goes to $X^1$. The following commands store the initial address:

| | |
|---|---|
| $I^5I^6 \rightarrow X^1$ | Reads content of 0000j (except upper address) into $X^1$ |
| Store $X^1$ in 0000j | Stores initial address |
| $X^1_L \rightarrow X^2_U$<br>$X^2 \rightarrow X^1$<br>$X^1_{UA} \rightarrow I^2$<br>Clear R<br>$I^2 \rightarrow R$<br>Partial add $R^1$ to $U^2$<br>$U^2 \rightarrow R^2$<br>Sense R = 0 | Compares initial and terminal addresses |
| Set Buffer Active FF for channel j if $R \neq 0$<br>Clear Buffer Active FF for channel j if R = 0 | Acts on result of comparison |



Note that the terminal address is always set one greater than the address to be used in the last item buffered.

At the conclusion of these commands all the preparation has been made to buffer a block of words to or from the block of addresses specified by the control word. If the Buffer Active FF for channel j is not set, buffering does not begin. The sequence exits to RNI by the half or jump exits, the choice being made by the Exit FF.

After the execution of a 74.1-6 the actual buffer operations are performed independently of the main program by means of the Auxiliary sequence.

## AUXILIARY SEQUENCE

Three specific auxiliary operations are performed by the Auxiliary (AUX) sequence: buffer, advance clock and interrupt operations.

The auxiliary request signal, described previously, causes the computer to suspend performance of the main program while the Auxiliary sequence performs the required operation. When the auxiliary operation is concluded, performance of the main program continues.

An auxiliary operation may be performed whenever any one of these conditions occur: (1) coincidence of an auxiliary request and a sequence exit; (2) coincidence of an auxiliary request and completion of a search or transfer operation; (3) presence of an auxiliary request while the computer is stopped.

BUFFER OPERATION

A buffer operation exchanges an individual word between the computer and an external equipment, that is, one buffer operation is performed for each word exchanged in a block of words. The rate of buffer operations on a given channel is determined by the speed of the external equipment which is in communication with the computer.

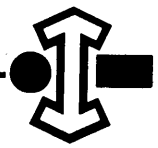
In order for buffer operations to occur the following preparatory steps must have been performed:

1) Select the external equipment and its mode of operation (accomplished by 74.0 instruction with suitable EF codes).
2) Load lower address part of buffer control word (for channel to be used) with terminal address of buffer. Terminal address is one greater than last one to be used.
3) Activate the buffer channel and load upper address of buffer control word with the initial address, that is, the first address to be used. This step is accomplished by 74.1-6, as described earlier.

After these steps have been performed by the main program, a block of words is sent to or from the block of storage locations defined by the initial and terminal addresses. The actual exchange of words proceeds entirely independently of the program. The computer input-output section controls the buffer exchange.

For each individual buffer operation, which exchanges one word, the initial address part of the buffer control word specifies the storage location to be used. Near the end of the operation this quantity is increased by one to provide the address for the next buffer operation.

A buffer operation occurs for input when the external equipment sends an input ready indicating it has a word to be received by the computer; or for output when the external equipment



sends an output resume indicating that it has sampled and stored the last word on its medium and is prepared to receive another word.

In the buffer operation, the buffer control word is first read from storage in order to obtain the initial address portion. This quantity is the address of the location used in the next step, which involves either storing an input word in this address or reading an output word from the address and sending it to the external equipment. If the operation is input, a resume is sent to the equipment; if output, a ready is sent.

The initial address is then increased and stored in a buffer control word. At the same time the incremented initial address is compared with the terminal address. If they are equal the channel is made inactive, since the entire block has been buffered; if they are not equal, the channel remains active.

After these steps are performed by the Auxiliary sequence, an exit is taken to resume execution of the program. The commands that carry out each of the three steps of a buffer operation are listed in table 5-5. These commands are generated by the corresponding segments of AUX shown in figure 5-17.

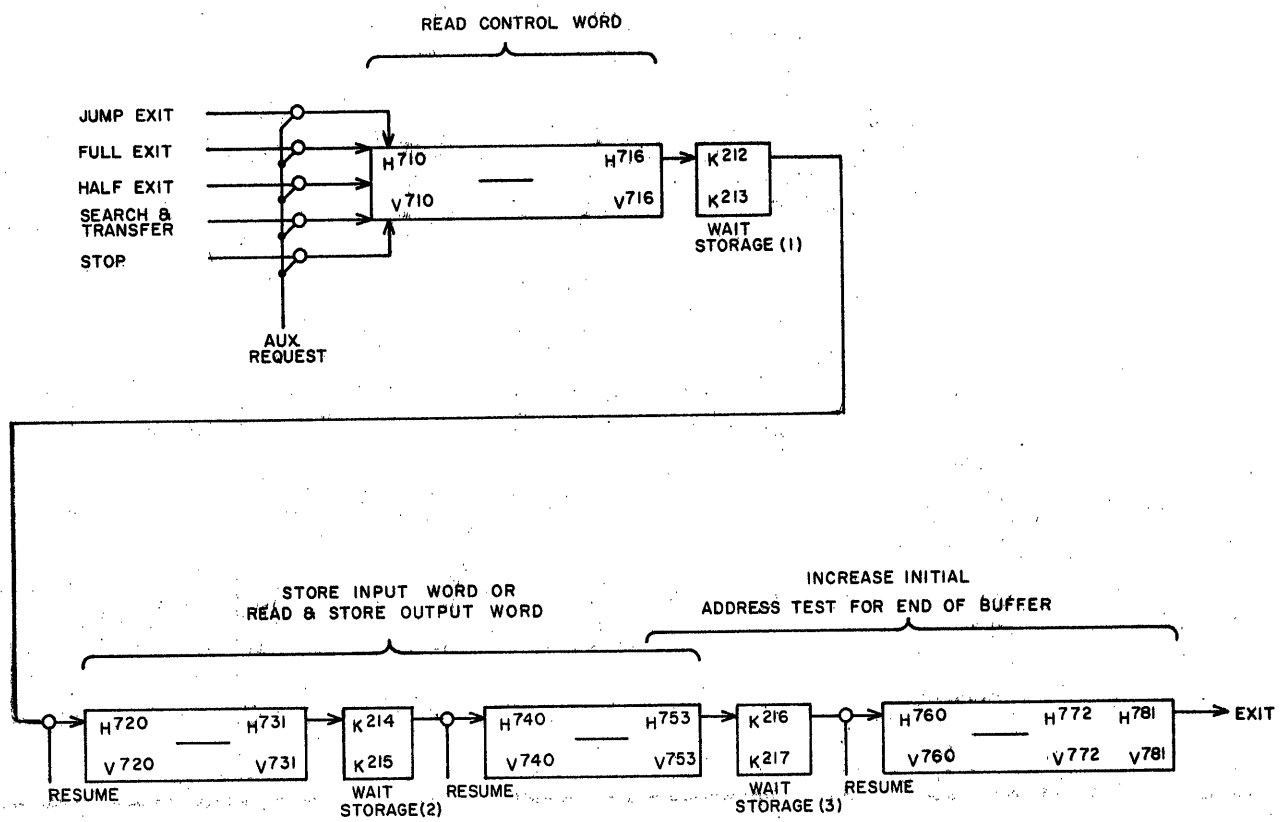



Figure 5-17. Auxiliary Sequence for Buffer Operation.

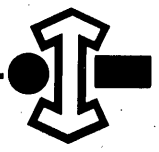


TABLE 5-5. COMMANDS FOR BUFFER OPERATION

| Read Buffer Control Word | | | |
|---|---|---|---|
| Set auxiliary reference designator from scanner* | | | |
| Initiate storage | (address from auxiliary reference designator) | | |
| Clear $U^1_U$ | | | |
| Clear $R^1$ | | | |
| Wait Storage | | | |
| $I^5I^6 \rightarrow U^1_U$ | (upper half control word to $U^1_U$) | | |
| $U^1_{UA} \rightarrow U^2$ | | | |
| $U^2 \rightarrow R^2$ | (save initial address for incrementing) | | |
| **Store Input or Read Output Word** | | | |
| Input | | Output | |
| Clear $X^1$ | | | |
| $I^0 \rightarrow X^1$ | (input word to X) | Clear $X^1$ | |
| Initiate Storage | (address from $U^2$) | Initiate storage | (address from $U^2$) |
| Wait Storage | | Wait storage | |
| $X^1 \rightarrow$ Storage | (stores input word) | $I^5I^6 \rightarrow X^1$ | (obtains output word) |
| Input Resume | | $X^1 \rightarrow X^2$ | |
| | | $X^2 \rightarrow O^-$ | (output register) |
| | | Output ready | |

*The Operation AUX is to execute; is indicated by position at which scanner stopped; designator gives address of control word and conditions commands for the operation.

TABLE 5-5. (Cont'd.)

| Increment Address and Test for End of Buffer | |
|---|---|
| Complement $R^2$ to $R^1$ | |
| $R^1 \rightarrow R^2$ | ($R^2$ holds initial address) |
| Reduce $R^1$ to $R^2$ | adds one to R |
| Complement $R^2$ to $R^1$ | |
| Clear $U^1_U$ | |
| $U^1_{UA} \rightarrow U^2$ | (clears $U^2$) |
| Partial add $R^1$ to $U^2$ | (transmits incremented address to $U^2$) |
| Clear $X^1$ | |
| $U^2 \rightarrow X^1_L$ | Prepare incremented address to be written in control word |
| $X^1_L \rightarrow X^2_U$ | |
| Initiate storage | (address from auxiliary reference designator) |
| Wait storage | |
| Write $X^1_{UA}$ in storage | (enters incremented initial address in control word) |
| $I^5I^6 \rightarrow X^1_{LA}$ | (read out terminal address) |
| $X^1_L \rightarrow X^2_U$ | |
| $X^2 \rightarrow X^1$ | prepare terminal address for comparison |
| $X^1_{UA} \rightarrow I^2$ | |
| $I^2 \rightarrow R^1$ | |
| Partial add $R^1$ to $U^2$ | (compares incremented and terminal addresses by toggling first with second) |
| $U^2 \rightarrow R^2$ | |
| Test $R^2 = 0$ | |
| Clear Buffer Active FF for channel if R = 0 (terminates buffer) | |
| Exit | |

ADVANCE CLOCK OPERATION

Storage address 00000 is assigned a special function as a 48-bit counter to provide an indication of elapsed real time. The count in this address is increased every 1/60 of a second, governed by the 60-cycle line power. By means of this count real-time measurements may be used in computation. Advancing the clock count in address 00000 is done by the AUX sequence. During each scanning cycle the scanner senses whether the clock requires advancement. Sensing occurs with the scanner at 000 position.

The circuits that establish the need for advancing the clock appear in figure 5-18. The Clock Disconnect FF may be set or cleared by a 74.0 instruction with the codes indicated;

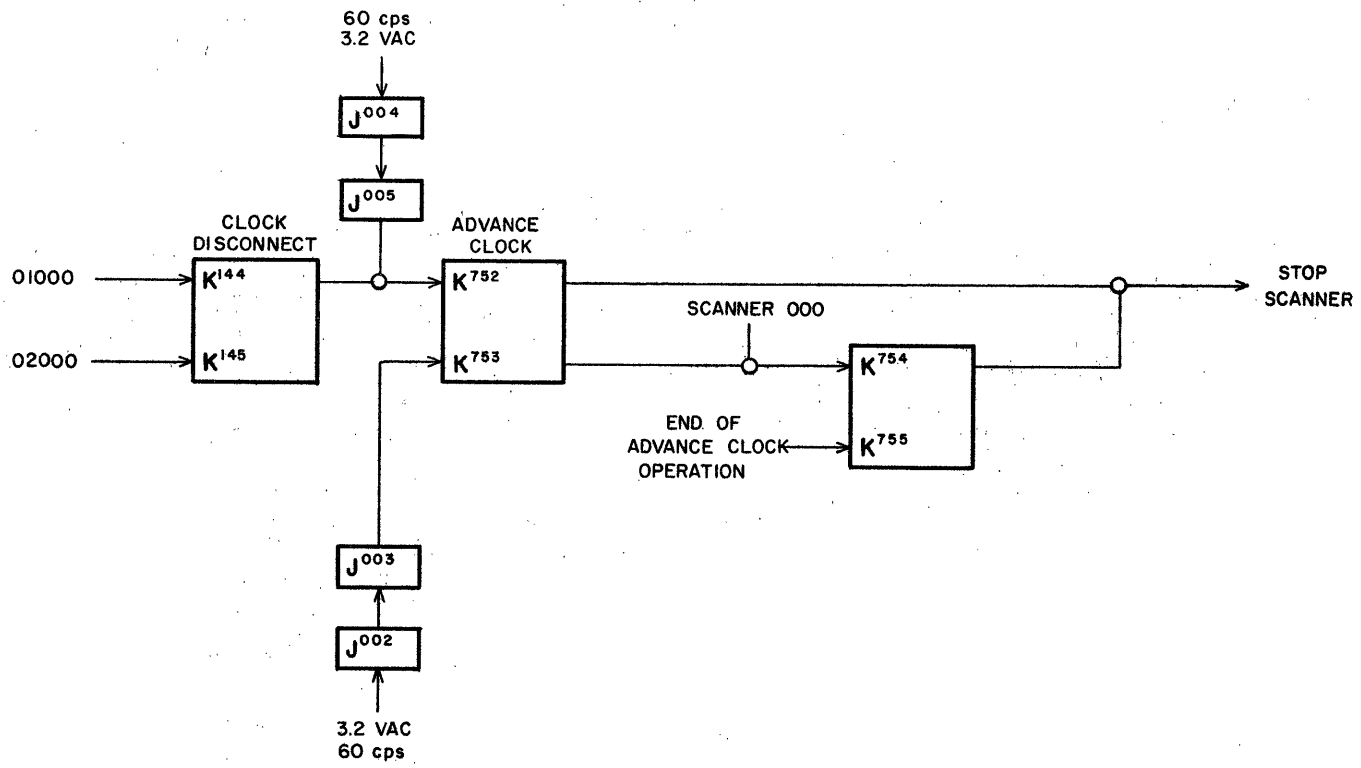



Figure 5-18. Advance Clock Resync Circuit.

advancement is allowed only when this FF is set. The Advance Clock FF monitors the 60-cycle line. Once each cycle it is set if $K^{144}$ $K^{145}$ is set. This stops the scanner when it reaches the advance clock position (count 000), and AUX is then initiated to perform the advance clock operation.

Figure 5-19 shows the portions of AUX used during this operation. A is first temporarily stored in Q. The content of address 00000 is read into A, where one is added to it. A is then stored in address 00000, and the initial content of A is returned from Q to A.

The commands given by AUX to advance the clock are:

| | |
|---|---|
| Clear $X^1$ | |
| $X^1 \rightarrow X^2$ | |
| Initiate storage | (address 00000) |
| Set $X^2$ to 1 | (prepares increment) |
| $Q^1 \rightarrow Q^2$<br>$A^2 \rightarrow Q^1$ | (Store A initial in $Q^1$ while $Q^2$ holds Q initial) |
| Partial add $X^2$ to $A^1$ | (transmits increment to A) |
| Wait storage | |
| $I^5I^6 \rightarrow X^1$ | (old clock count read out) |
| $X^1 \rightarrow X^2$ | |
| Add $X^2$ to $A^1$ | |
| Clear $X^1$ | |
| $A^1 \rightarrow X^1$ | |
| Initiate storage | (address 00000) |
| $Q^2 \rightarrow A^1$<br>$Q^1 \rightarrow Q^2$<br>$A^2 \rightarrow Q^1$<br>$Q^2 \rightarrow A^1$ | Return initial contents of A and Q |
| Wait storage | |
| Exit | |

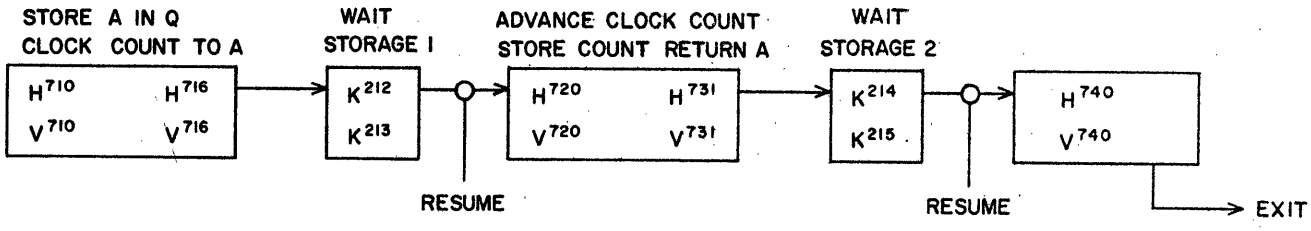



Figure 5-19. Auxiliary Sequence for Advance Clock Operation.

## INTERRUPT OPERATION

In each piece of external equipment, as well as in parts of the internal computer control, certain conditions may arise which make it necessary that the main program be notified of their presence. When these conditions occur, the main program is halted and a routine of instructions is performed which determines the cause of the interruption and takes the actions appropriate to the condition causing the interrupt. After these two operations have been completed, the main program is resumed at the exact point from which the entrance to the interrupt routine was made.

The signal which notifies the computer of the presence of the specified condition is called an interrupt. The program has control of and determines whether the occurrence of a given condition within an external equipment or a part of computer control is to be indicated by an interrupt. An interrupt is produced by the presence of a condition only if a 74.0 has previously selected an interrupt on that condition. Unless such selection is made, no interrupt is produced when the condition arises.

The occurrence of an interrupt in no way indicates to the computer the specific source of the signal. The interrupt subroutine examines each source of interrupts by means of the 74.7 instruction; after the source is determined the necessary action is taken.

Sources of the interrupt are cable groups 1, 2, 3 and 4 and internal computer control. Figure 5-20 shows how these sources are combined into a single line by an OR ($J^{526}$). The interrupts from arithmetic faults are combined by $J^{523}$, those from inactive buffer channels by $J^{525}$, and the interrupt lines from the external equipment by $J^{524}$. Any individual interrupt will produce an output from $J^{526}$, which then causes the main program to halt.

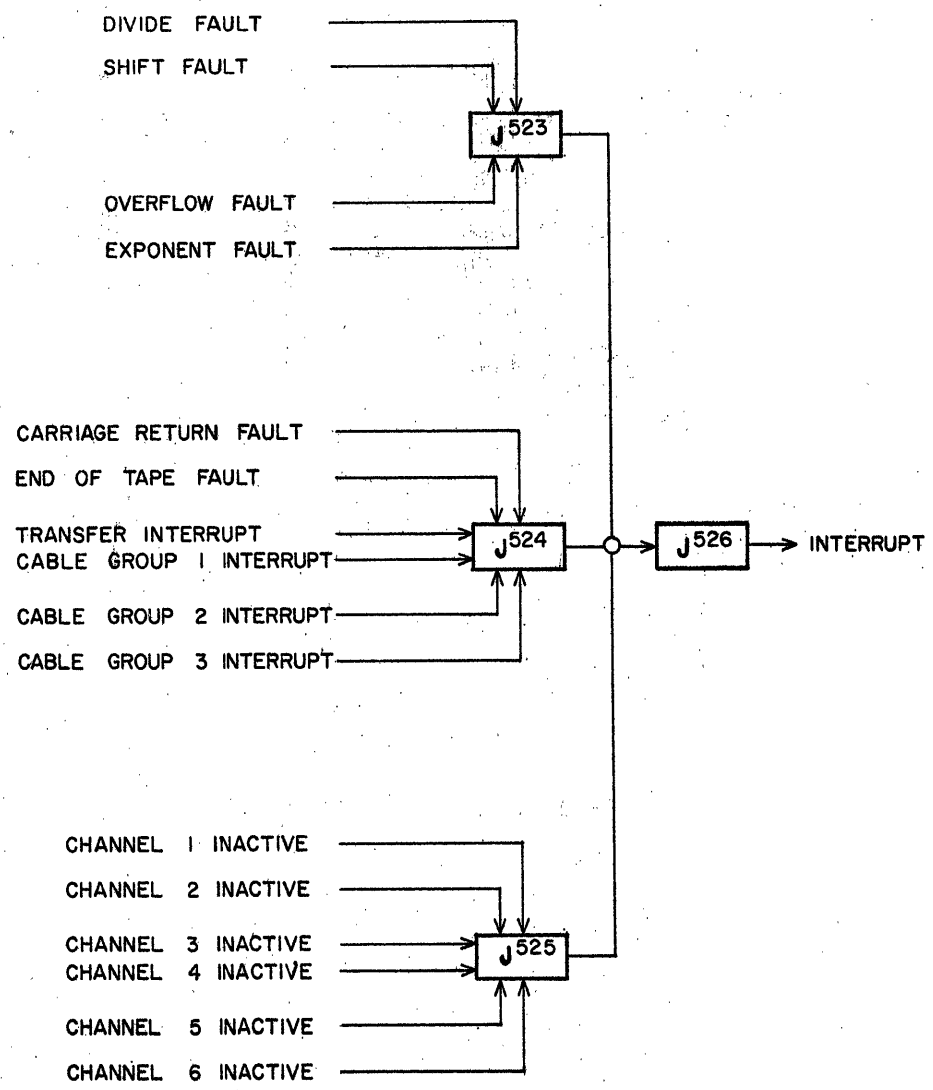
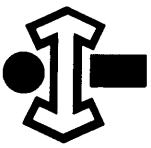



Figure 5-20. Interrupt Source.



Recognition of Interrupt

One position of the auxiliary scanner is associated with interrupt. The interrupt bus is sampled only when the scanner is at the interrupt position. If an interrupt signal is present when the scanner is at this position and if the computer is not already performing the interrupt routine to respond to a previous interrupt signal, then the scanner is stopped and the AUX sequence initiated.

A special form of AUX accomplishes the entrance to the interrupt routine. This involves performing something very much like a return jump on address 00007. This is a special address allocated for use as the entrance point to the interrupt routine and for the return from this routine to the main program.

Typically address 00007 contains two unconditional jump instructions. For example:

75 0 XXXXX 75 0 YYYYY

The upper instruction provides for the return to the main program upon completion of the interrupt routine. To accomplish this the upper address portion (XXXXX) is loaded with the contents of the P register when the interrupt routine is entered. The lower instruction jumps to the interrupt routine which begins with an instruction whose address is indicated in the lower address part of 00007, that is, it has been inserted in place of YYYYY.

The functions of the interrupt form of AUX are to:

1) Store P in upper address part of 00007.
2) Enter lower instruction at 00007 in $U^1_U$ so that it may be executed next to begin the interrupt routine.
3) Set the Interrupt Lockout FF in order to prevent recognition of subsequent interrupts until the response to the present one has been completed.

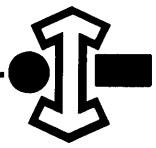
When AUX has performed the interrupt operation consisting of these functions, execution of the main program ceases and the computer begins executing the interrupt routine instead.

Determining Source of Interrupt

Of crucial importance in determining the specific source of the interrupt is the fact, noted above, that the computer must select by a 74.0 instruction the condition for interrupting (see table 5-5 for examples of codes used for such selections). For example, if an interrupt is to occur when buffer channel 1 becomes inactive, a 74 0 00010 instruction must have been programmed. Typically, after executing such an instruction the program loads a suitable sense instruction in the interrupt routine; in this case the instruction would be 74 7 00011. This procedure is followed each time an instruction is performed to select a condition for interruption. As a result the interrupt routine always contains a series of sense instructions (74.7) such that there is one for each active source of interrupt.

To determine the source of the interrupt, the routine successively executes the sense instructions in the series described above. When the sense return line indicates the presence of the condition (for example, channel 1 inactive) the interrupt routine knows the source of the interrupt. Ordinarily the routine would branch to a subroutine designed specifically to handle interrupts from this source.

This method of locating the interrupt source shows that no priority distinction is made between two or more interrupts. The first source that is sensed as being in the condition for producing an interrupt is the one that is responded to. Should a second source (which is sensed later in the series of sense instructions of the interrupt routine) be producing an interrupt, this source must wait until the response to the first is complete. From this it becomes apparent that each interrupt source must hold its interrupt signal on until it is sensed, which indicates that the computer will then respond. One of the actions of the interrupt program is, therefore, to clear the interrupt condition immediately after sensing it.



For the cable group interrupt sources the signal is produced by external equipments connected to the cable group. The internal source (from within the computer itself) handles interrupts due to buffer channels becoming inactive and interrupts due to the occurrence of an arithmetic fault.

Return to Main Program Following an Interrupt Operation

The address of the exact point at which the main program was interrupted must be stored so that when the interrupt operation is completed the computer will return to the next unexecuted instruction. The contents of P are stored in the upper address portion of the control word. The upper instruction of 00007, the last instruction of the interrupt routine, is an unconditional jump instruction which restores the contents of P. The next to the last instruction of the interrupt routine is a normal jump to address 00007.

Analysis of the Interrupt Operation

Figure 5-21 shows the portion of AUX used to perform the interrupt operation. The commands generated by the sequence are:

| | |
|---|---|
| Initiate storage | (address 00007) |
| $P^1 \rightarrow X^2_{LA}$ | |
| $X^2 \rightarrow X^1$ | |
| Clear $U^1$ | |
| Set Interrupt Exit FF | (Stores Exit FF) |
| Set Interrupt Lockout FF | |
| Set P to 00007 | |
| Wait storage | |
| $X^1_L \rightarrow X^2_U$ | |
| $X^2 \rightarrow X^1$ | |
| $I^5I^6 \rightarrow U^1$ | |
| Half exit | |

The single storage reference provides for storing P in the upper address position of 00007 and for reading the content of 00007 into $U^1$.

Since the first instruction of the interrupt program is located in the lower position of 00007, the interrupt operation takes the half exit from AUX. (For this reason also it does not matter that the unaltered contents of 00007 are placed in $U^1_U$ by the storage reference.) When the interrupt program is completed, a jump is made to the upper instruction of 00007. This instruction returns the computer to the correct position in the main program since it causes a jump to the address defined by the original contents of P.

The Interrupt Lockout FF inhibits the recognition of other interrupt signals until the present interrupt request has been honored. The Interrupt Exit FF stores the state of the Exit FF, indicating whether the succeeding instruction in the main program is in the upper or lower position within the instruction word. (See discussion of the Read Next Instruction sequence in Chapter 2.) At the time the Exit FF is sampled during an interrupt operation, this FF (Exit) has already been reversed by the exit which leads to the execution of AUX; therefore the Interrupt Exit FF is interpreted in a manner opposite to that of the Exit FF.

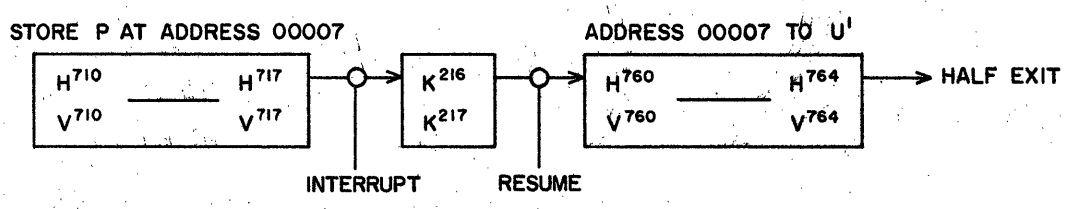
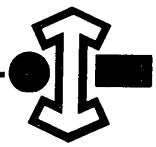



Figure 5-21. Auxiliary Sequence for Interrupt Operation.



## SEARCH AND TRANSFER SEQUENCE

The search and transfer sequence (ST) performs six instructions:

| | |
|---|---|
| 62 Input Transfer | 65 Threshold Search |
| 63 Output Transfer | 66 Masked Equality Search |
| 64 Equality Search | 67 Masked Threshold Search |

Although search and transfer instructions accomplish very different functions, a common basic procedure is used to carry them out. For both types of operation the designated index register, $B^b$, specifies the number of words to be searched or transferred. These words are located in a block of consecutive storage locations. The first word of a block is located at address $m + B^b - 1$, the second word at address $m + B^b - 2$, and so forth. The last word is located at address m.

A sequence of commands is performed repeatedly, once for each word searched or transferred. After each repetition, $B^b$ is reduced by one. Thus $B^b$ holds at all times a count of the number of words remaining in the block. Reducing $B^b$ for each repetition also prepares the address of the word for the next operation, since the address for the current word is the sum of m and the current count in $B^b$.

Figure 5-22 shows the basic chain of control delays that generate the ST commands. The first segment, address preparation, produces the same commands for each of the six ST instructions; these commands, which yield the first address, are:

$U^1 \rightarrow U^2$
$B^b \rightarrow I^2\ I^3$
$I^2\ I^3 \rightarrow R^1$
$R^1 \rightarrow R^2$
} done by RNI

Reduce $R^1$ to $R^2$ ($B^b - 1$)
$R^2 \rightarrow R^1$
Exits ($b \neq 0$ and $R = 0$ before reduce)
Add $R^1$ to $U^2$ (first address; $m + B^b - 1$)

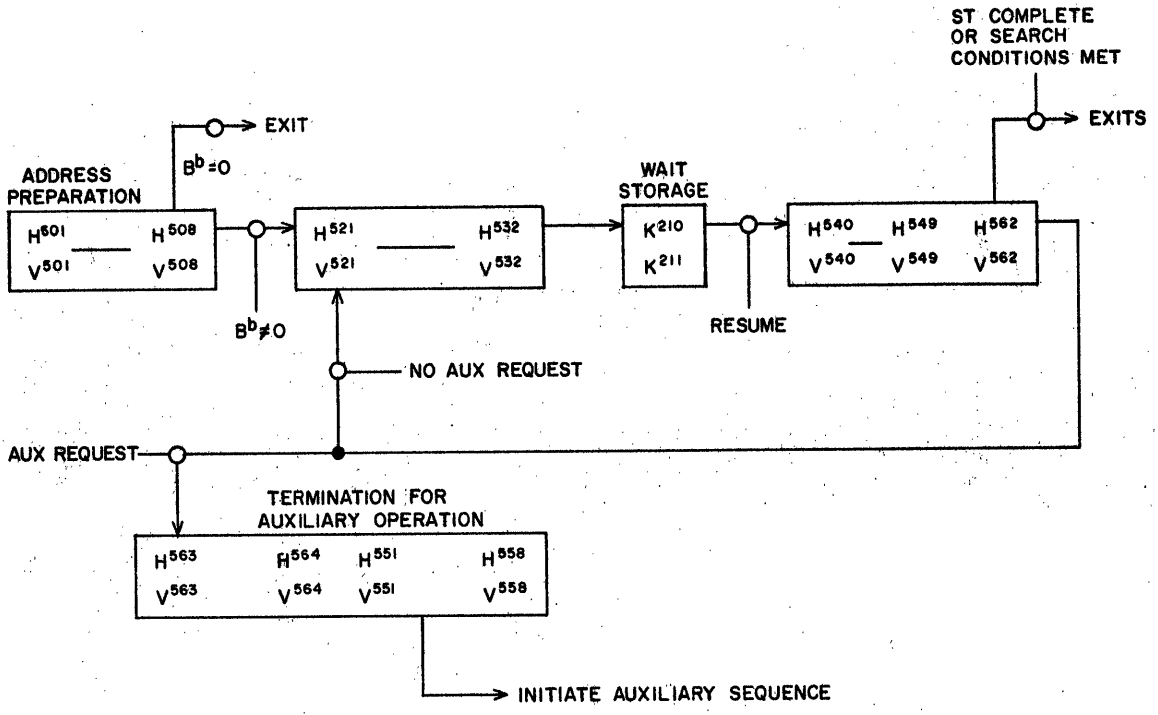
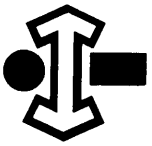



Figure 5-22. Search and Transfer Sequence.

The second and all succeeding addresses are also obtained by a method common to the six instructions. Each execution of the loop performs the following commands to generate the address of the word for the next operation:

| | |
|---|---|
| Clear $B^b$ | |
| $R^2 \rightarrow B^b$ | (current count in $B^b$) |
| $U^1 \rightarrow U^2$ | (m to $U^2$ for next address) |
| Reduce $R^1$ to $R^2$ | (prepare count for next operation) |
| $R^2 \rightarrow R^1$ | |
| Add $R^1$ to $U^2$ | (address of next word) |

The storing of the count in $B^b$ provides for the situation when an auxiliary operation must be performed. In this case the series of ST operations is suspended after completing the current one. The AUX operation occurs next. Following it the ST operations are resumed and $B^b$ indicates where to begin.



TRANSFER OPERATION

Input and output transfer instructions provide for rapidly moving a block of data from external equipment to storage or from storage to external equipment. Transfer operations may be performed at the maximum computer rate of one 48-bit word every 4 microseconds.

Input Transfer

The actual receiving and storing of an input word by the loop segment of the ST sequence (figure 5-22) occurs only when the input equipment sends a ready. This indicates that a word is available for sampling on channel 7. Thus it is the input ready which initiates each execution of the loop. The commands listed below are produced to store the word and prepare the next address.

Input ready
Clear $X^1$
Clear $B^b$
Initiate storage (address from $U^2$)
$R^2 \rightarrow B^b$
Complement $X^1$ to $X^2$ (Sets X to al 1's)
$X^2 \rightarrow X^1$
$U^1 \rightarrow U^2$
Reduce $R^1$ to $R^2$
$R^2 \rightarrow R^1$
$I^0 \rightarrow X^1$ (Clear X for 0's in input word)
Add $R^1$ to $U^2$
Wait storage
$X^1 \rightarrow$ storage
Initiate AUX sequence (if AUX request)
Exits ($R = 0$ and $b \neq 0$) or $b = 0$
Wait for next ready

Output Transfer

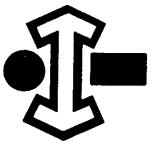
The loop segment of the sequence sends a word out on channel 7 only when the output equipment sends a resume. This signifies to the computer that the preceding word has been accepted by the equipment and that it is prepared to receive the next word. Each resume initiates the loop. The commands listed below are produced to read the word from storage, send it out on channel 7 and prepare the address of the next word.

```
                    Output resume
loop entry      )   X2 → O4 send output ready
here for        ) → Clear Bb
first word      )   Exits if (R = 0 and b ≠ 0) or b = 0
                    Initiate storage            (address from U2)
                    R2 → Bb
                    U1 → U2
                    Reduce R1 to R2
                    R2 → R1
                    Add R1 to U2
                    Wait storage
                    I5I6 → X1                   (acquires next word from storage)
                    X1 → X2
                    Initiate AUX sequence       (if AUX request)
                    Wait for next resume
```

Examination of these commands reveals that the loop anticipates the output equipment by acquiring the next word from storage while the current one (in $O^4$) is being sampled. The resume causes this word that was acquired in advance to be transmitted from $X^2$ to $O^4$.

SEARCH OPERATION

The purpose of the search instructions is to rapidly examine a block of operands for one that meets the search condition, which may be either equality to A or greater than A. The process involves successively acquiring each operand from storage and comparing it with the contents of the A register. When an operand is found that meets the condition, the search is terminated with a full exit. If no operand in the entire block meets the condition, the instruction half exits.



The loop segment of the ST sequence (see figure 5-22) performs the search operation for each operand by the following commands:

Clear $X^1$

Clear $B^b$

Initiate storage (address from $U^2$)

$R^2 \rightarrow B^b$

$U^1 \rightarrow U^2$

$R^1 \rightarrow R^2$

Reduce $R^1$ to $R^2$

Wait storage

Add $R^1$ to $U^2$

$I^5I^6 \rightarrow X^1$

Complement $X^1$ to $X^2$ (64, 65)

LQX (66, 67)

Complement $X^1$ to $X^2$ (66, 67)

Sense X=A (64, 66)

Sense X>A (65, 67)

Half exit if R = 0 (search exhausted)

Full exit search condition met

Repeat loop if $R \neq 0$ and search condition not met

Initiate AUX sequence if AUX request

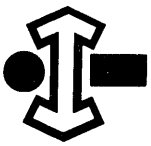
The LQX command for 66 and 67 masks out a portion of the operand. This mask quantity is loaded in Q by a prior instruction. The complement commands prepare the operand for comparison with A.

TEMPORARY TERMINATION FOR AUX OPERATIONS

The final segment of the ST sequence shown in figure 5-22 is used only when the execution of a ST instruction must be temporarily suspended in order to permit one of the AUX operations to be performed. The occurrence of an AUX request prevents the next repetition of the loop part of the sequence. It also initiates the temporary termination part of the ST sequence.

For all ST instructions except 63 this segment simply initiates the AUX sequence. For 63 (output transfer) the function of the temporary termination segment is to prepare the count in $B^b$ so that when the AUX operation is completed the execution of this instruction begins again with the last word for which a resume has not been received. During this instruction $B^b$ holds the count associated with the output word currently in $O^4$. Since no resume has been received for this word, it must be placed in $O^4$ again when the execution of the instruction is resumed. This requires that the count in $B^b$ must be increased by one to allow for the reduction that will take place in the address preparation segment, which is initiated in resuming execution of 63. The following commands from the termination segment accomplish this.

| | |
|---|---|
| Complement $R^1$ to $R^2$<br>$R^1 \rightarrow R^2$ | Adds one to R to make it equal current count in $B^b$ |
| Reduce $R^1$ to $R^2$<br>$R^2 \rightarrow R^1$<br>Reduce $R^1$ to $R^2$<br>Complement $R^2$ to $R^1$ | Prepares incremented count for $B^b$ |
| $R^1 \rightarrow R^2$<br>Clear $B^b$<br>$R^2 \rightarrow B^b$ | Enters incremented count in $B^b$ |



# CHAPTER 6

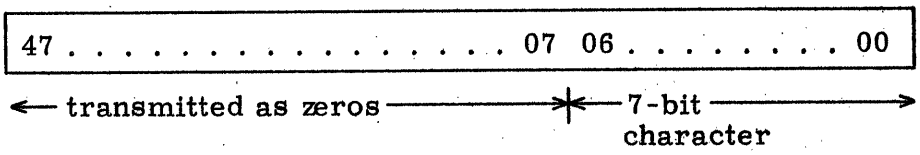
# CONSOLE INPUT-OUTPUT EQUIPMENT

Three units of input-output equipment, mounted on the console, are an integral part of the 1604 computer. A Teletype High Speed Tape Punch and a Ferranti High Speed Tape Reader provide for the processing of perforated paper tape. An IBM Typewriter, modified by Soroban Engineering, Inc., provides for the direct keyboard entry of data and for printed copy output. The console input-output units communicate with the central computer via buffer channel 1 (input) and 2 (output). Other input-output units may share these channels but the console input-output units cannot use any of the other channels.

The control and data-handling circuits associated with these console-mounted units are contained within the main computer cabinet on chassis 7 and 8. This chapter primarily treats these circuits; information concerning the operation of the units themselves appears only incidentally. For more complete information about the units refer to the manufacturers' manuals and diagrams.

# MODES FOR HANDLING DATA

Data transmission between the console equipments and the computer may take place in either the character or the assembly modes. In the character mode a 7-bit character is buffered to or from the computer one bit at a time. As shown below, the 7-bit character occupies the lowest bit positions of a 48-bit word; "0's" are always transmitted in the upper 41 bits of the word.

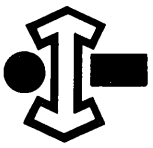



In the assembly mode the 48-bit word being buffered consists of eight 6-bit characters.

| Char. 7 | Char. 6 | Char. 5 | Char. 4 | Char. 3 | Char. 2 | Char. 1 | Char. 0 |
|---|---|---|---|---|---|---|---|
| 47...42 | 41...36 | 35...30 | 29...24 | 23...18 | 17...12 | 11...06 | 05...00 |

During an input buffer in the assembly mode (with the reader, for example) eight successive characters are assembled into a 48-bit word, beginning with character 7. After such a word is assembled it is sent to the computer.

For an output buffer in the assembly mode (with the punch, for example) a 48-bit word from the computer is disassembled into eight characters. These are punched successively beginning with character 7.



# EXTERNAL FUNCTIONS

Program control of the operation of the console input-output equipments is accomplished by means of instruction 74 External Function. The execution address of a 74.0 or 74.7 instruction is an EF code. Console equipments are specified by the following codes:

| | |
|---|---|
| 112XX | reader |
| 212XX | punch |
| 111XX | typewriter input, referred to as keyboard |
| 211XX | typewriter output, referred to as type |

A complete EF code consists of 15 bits; however, only the lower 12 bits are sent out to the equipment. The upper three bits, translated within the computer, select the channel to which the remainder of the code is sent.

## SELECT CODES

The console equipments and their operating mode are selected by means of the 74.0 instruction with one of the select codes listed in table 6-1. The select codes are translated and stored by the external function translator (figure 6-1). The combination of a select ready and a select code causes one or more FFs of the translator to be set. Four of the FFs store equipment selection as follows: Reader ($K^{910/911}$), Punch ($K^{912/913}$), Keyboard ($K^{902/903}$), and Type ($K^{900/901}$).

The select code that sets one of these FFs also clears the FF for the other equipment on that channel. Thus, selecting the typewriter (output) clears the Select Punch FF; selecting the reader clears the Select Keyboard FF. The Select Reader FF is also set by an external master clear. Thus, the reader is automatically selected by an external clear; if a manual load operation follows the clear, the steps necessary to select the reader are eliminated.

The select codes may also set or clear other FFs of the translator, depending upon the operating conditions specified for the selected equipment. These FFs and the codes which set them are listed below:

| | | |
|---|---|---|
| Interrupt on CR | ($K^{988/989}$) | 1114X |
| Reader End-of-Tape | ($K^{904/905}$) | 1121X |
| Interrupt on End-of-Tape | ($K^{906/907}$) | 1122X |
| Assembly Mode | ($K^{914/915}$) | 2XX0X |
| Punch Motor | ($K^{916/917}$) | 2120X, 2121X, 2122X, 2123X |

TABLE 6-1. SELECT CODES FOR CONSOLE EQUIPMENT

| | Channel 1 |
|---|---|
| 11100 | Keyboard entry and no interrupt on CR. |
| 11140 | Keyboard entry and interrupt on CR. |
| 11200 | PT Reader and interrupt on end-of-tape. |
| 11210 | PT Reader and set end-of-tape indicator. |
| 11220 | PT Reader and no interrupt on end-of-tape. |

| | Channel 2 |
|---|---|
| 21100 | Type assembly mode. |
| 21110 | Type character mode. |
| 21200 | Punch assembly mode. |
| 21210 | Punch character mode. |
| 21240 | Turn punch motor off. |

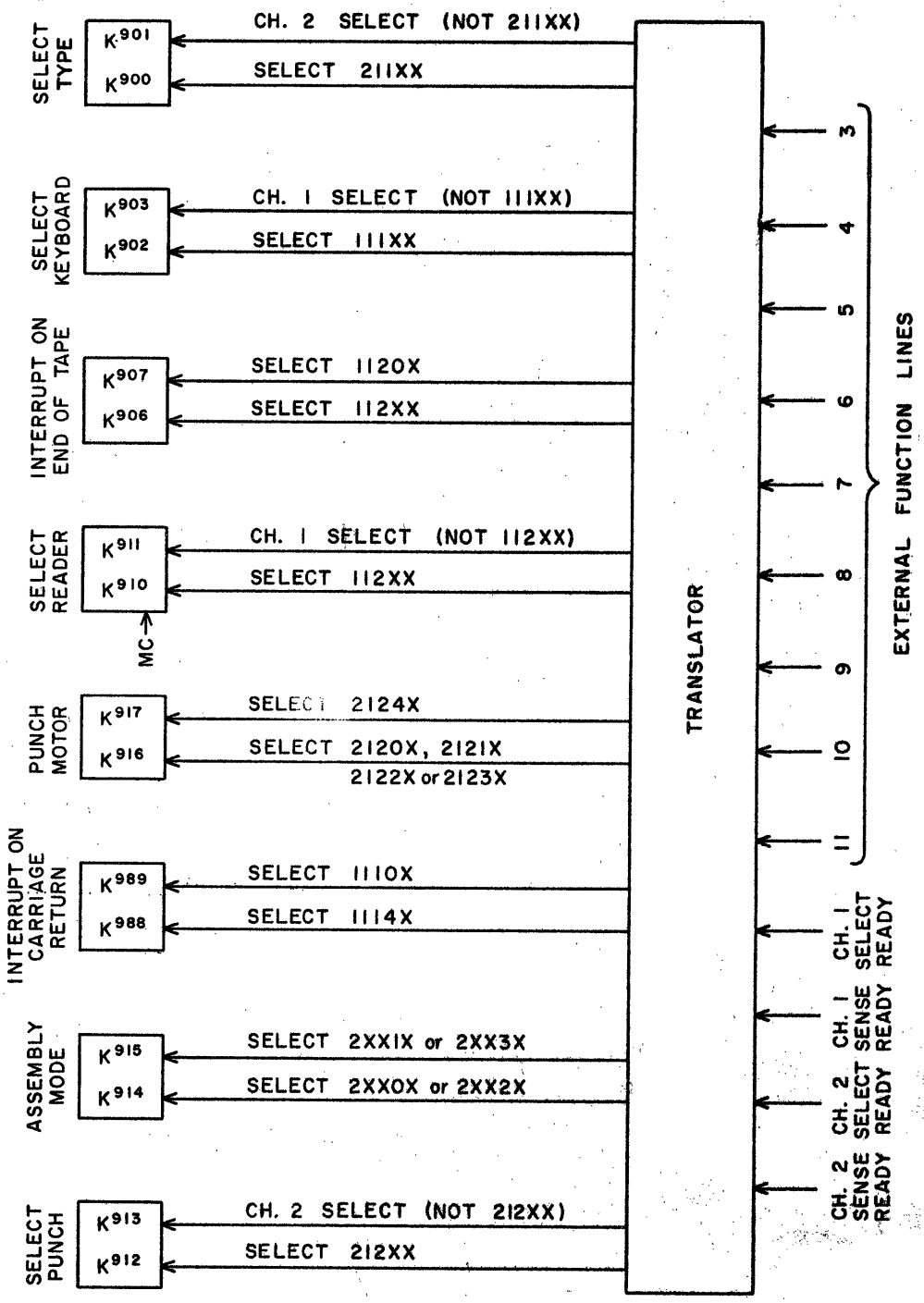



Figure 6-1. External Function Translator.

SENSE CODES

Five different conditions within the console equipments are sensed by means of the 74.7 instruction with the sense codes listed in table 6-2. As shown in figure 6-2, the translation of the code is combined with a signal indicating the specified condition. The output of this sense circuit is the sense return sent to the computer. The sense return from $J^{968}$ is a "1" when the specified condition exists.

The sense circuit of the console equipment ignores the lowest three bits of the sense code; the sense return signal sent to the computer is therefore the same for either of the codes in the pair associated with a condition (table 6-2). The computer sense circuit combines the ignored bits with the signal from $J^{968}$, thereby distinguishing between the two codes in the pair.

TABLE 6-2. SENSE CODES FOR CONSOLE EQUIPMENT

| | Channel 1 |
|---|---|
| 11100 | Exit on keyboard CR. |
| 11101 | Exit on no keyboard CR. |
| 11140 | Exit on keyboard lower case. |
| 11141 | Exit on keyboard upper case. |
| 11200 | Exit on reader end-of-tape. |
| 11201 | Exit on no reader end-of-tape. |
| 11210 | Exit on reader assembly mode. |
| 11211 | Exit on reader character mode. |

| | Channel 2 |
|---|---|
| 21200 | Exit on punch out-of-tape. |
| 21201 | Exit on no punch out-of-tape. |

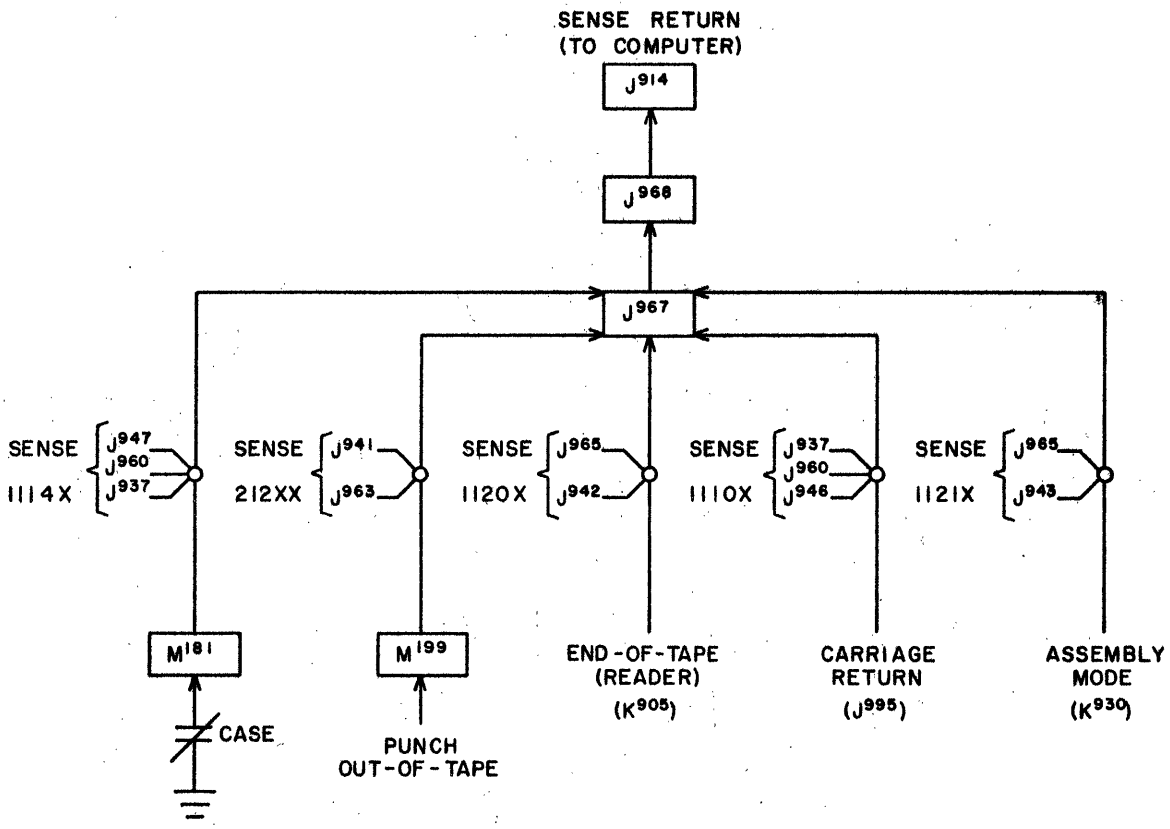



Figure 6-2. Sense Circuit Console Equipment.

## INPUT DISTRIBUTOR

The input distributor (figure 6-3), which consists of the Input Assembly Register (IAR) and the assembly counter, receives characters from the typewriter keyboard and the paper tape reader and forms 48-bit words from these characters for the computer. The 48-bit word is formed in IAR under the control of the assembly counter, which determines the position of IAR into which each character is entered.

For reader input operations in the assembly mode, figure 6-4b shows how the counter gates the characters from paper tape into IAR. After a character is gated into IAR the counter is advanced to prepare for gating the next character.

During an input operation from the reader in the character mode the assembly counter is forced to the count that produces the signal gate character 0. Each 7-bit character from the reader is entered in the lowest seven stages of IAR. The remaining stages hold "0". This 48-bit word is sent to the computer.

Input from the typewriter keyboard always takes place in the character mode. Six-bit characters from the keyboard go directly to the lowest stages of IAR independently of the assembly counter. After a keyboard character enters IAR the entire 48-bit word (consisting of "0's" in the upper 42 stages and the 6-bit character in the lower stages) is transmitted to the computer.

The transmission of IAR to the X register of the computer is via the $I^0$ inverters. When neither the reader nor the keyboard is selected IAR is held clear.

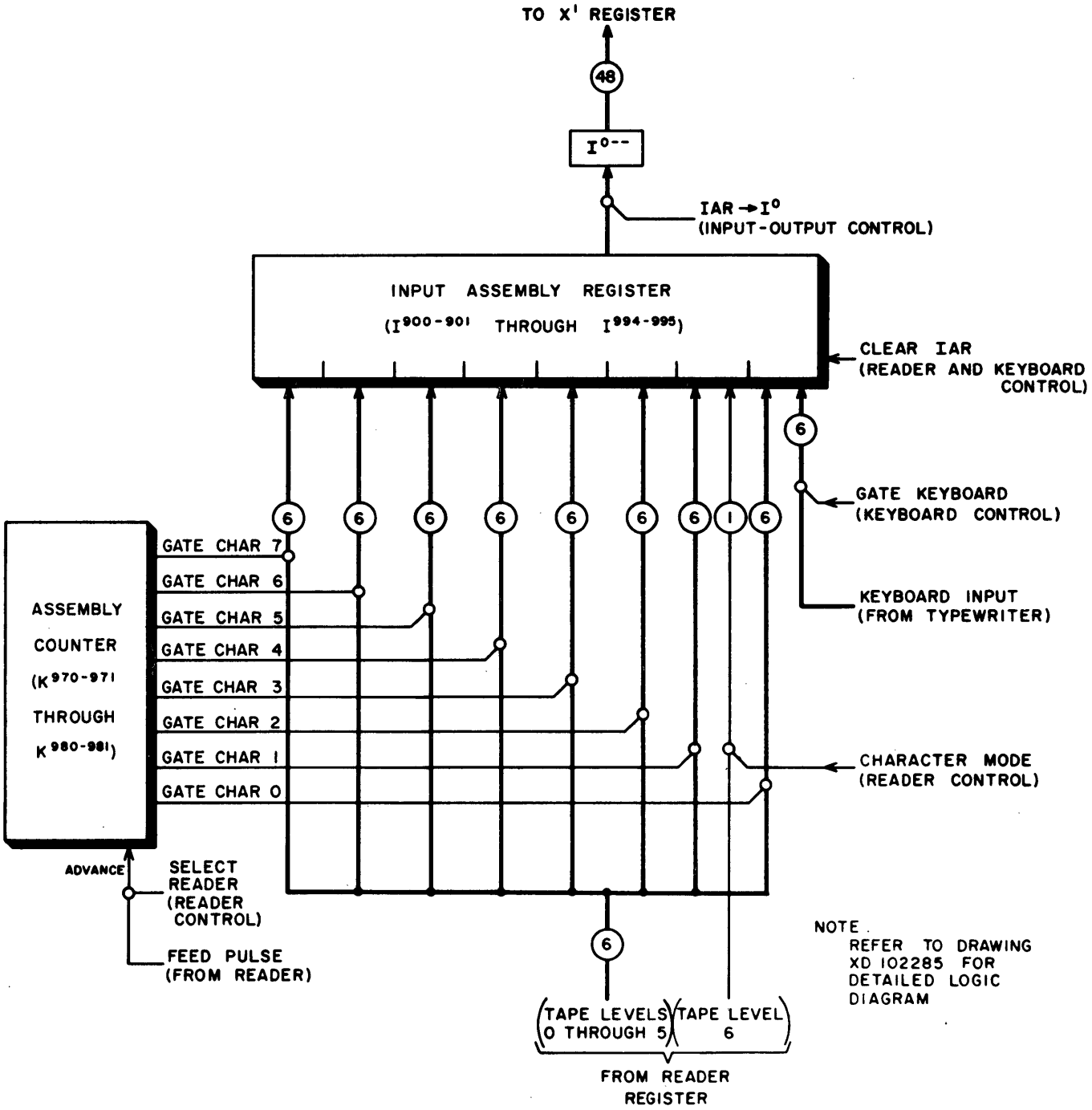



Figure 6-3. Input Distributor, Simplified Diagram.

## OUTPUT DISTRIBUTOR

The output distributor (figure 6-5), which consists of output register $0^1$ and the disassembly counter, receives 48-bit words from the computer and from them extracts characters which are sent to the punch or typewriter.

During output operations in the assembly mode the distributor disassembles each 48-bit word held in $0^1$ into eight 6-bit characters. These characters are sent successively to the punch or typewriter. Figure 6-4a illustrates the order of disassembly, which process is controlled by the counter. After each character is extracted, the counter is advanced to prepare for extracting the next one.

During output operations in the character mode the counter is forced to the count which produces the gate character 0 signal. Thus only the character in the lowest stages of $0^1$ is sent to the punch or typewriter. The upper bits of $0^1$ are ignored.

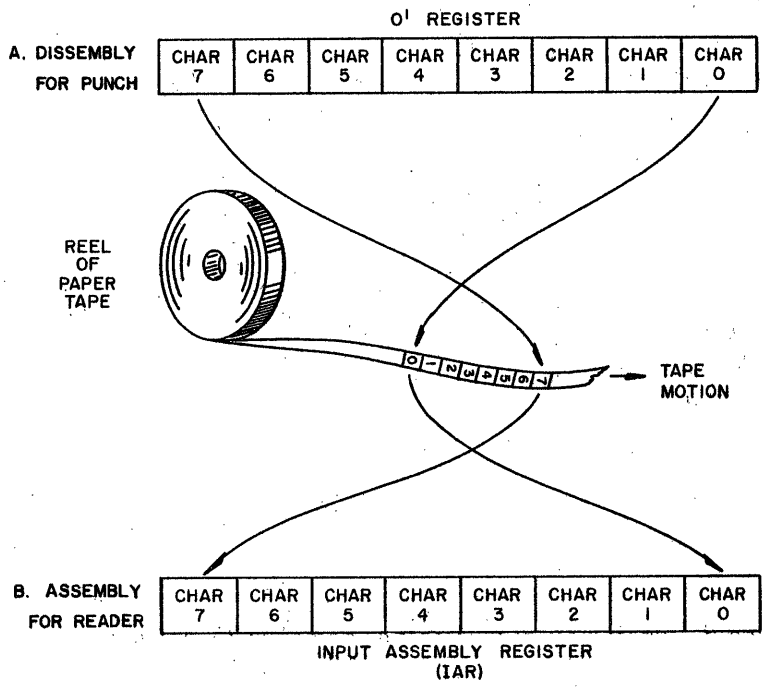



Figure 6-4. Character Orientation in Registers and on Paper Tape.

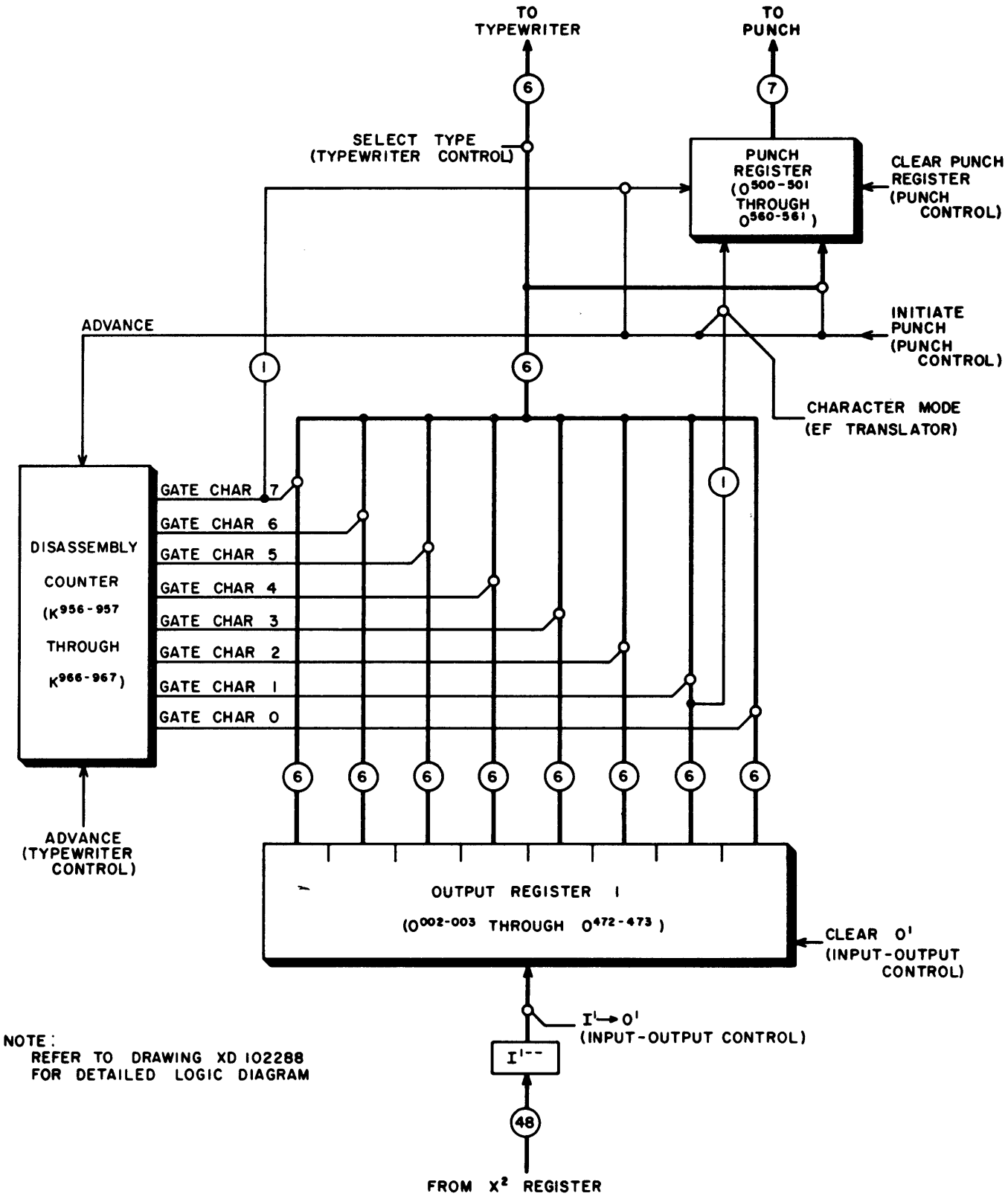



Figure 6-5. Output Distributor, Simplified Diagram.

# PAPER TAPE READER

The Ferranti Type TR 5 punched paper tape reader is used for entering information stored on punched paper tape into the computer. The reader is always connected to channel 1. It operates at a maximum rate of 350 characters (lines) per second; thus, the time interval between successive characters from the reader is 3.3 milliseconds. The complete tape reader is composed of a tape feed mechanism, an optical projection system, the reading and location photocells, the power supply, and the brake and clutch drive circuits.

## MANUAL CONTROLS

Manual controls for the reader are two lever switches on the punch and reader control panel of the console. The Reader Motor switch provides for manually turning on and off the reader motor, and must be set to the up (motor on) position whenever the reader is used. This action sets the Reader Motor FF ($K^{918/919}$); the "1" output of this FF energizes the reader motor.

The Reader Mode switch selects either the character or assembly mode of operation for a reader input operation: the up position (ASSEMBLY) positions the tape at the first frame of the first word (load point); the down position (CHARACTER) moves the tape ahead one frame.

## PAPER TAPE

Information is stored on paper tape in seven levels. A frame, which is across the width of the tape, can store seven bits. As shown in figure 6-6, the levels are designated 6, 5, 4, 3, 2, 1, and 0 from top to bottom. The sprocket or feed holes between levels 2 and 3 generate signals to time and control the reading of the tape.

Figure 6-6 shows the location of the various bits of a 48-bit word that is to be read in the assembly mode. In this mode, level 6 is used as a control rather than an information level. The first of the eight characters in a word is indicated by a hole in the control level.

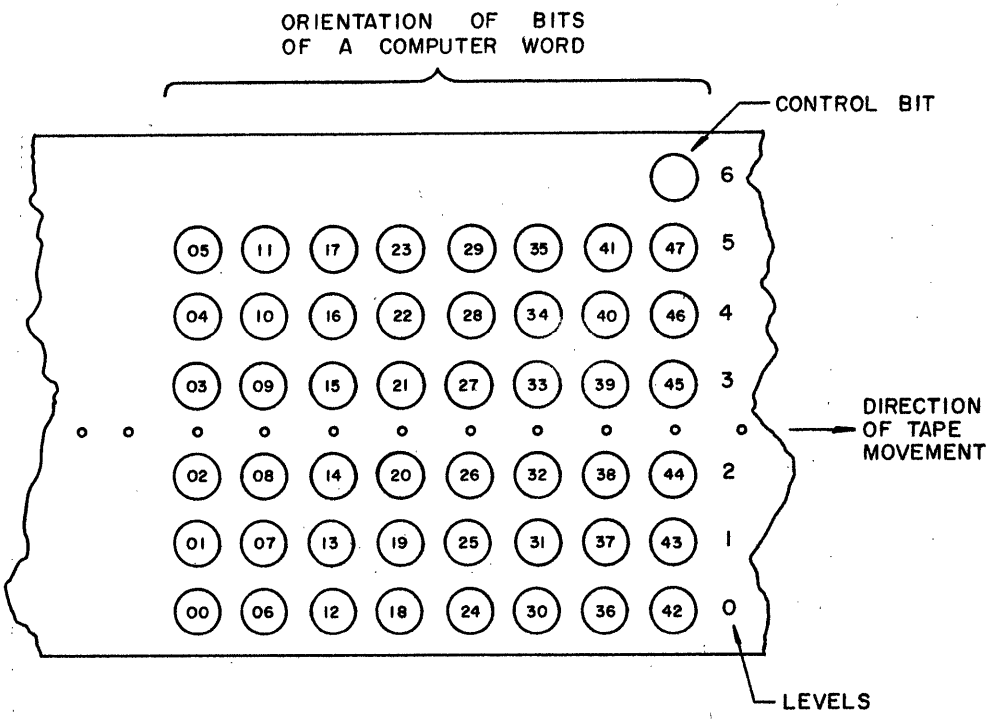



Figure 6-6. Seven Level Punched Paper Tape

READER OPERATION

In the assembly mode, the transmission of each character to its proper position in IAR is effected by the feed pulse (generated by the feed hole in the frame) under the control of the assembly counter. After each transmission of a character, the counter is advanced one count. When a 48-bit word is assembled in IAR a ready is sent to the computer. See figure 6-7 for reader control circuit and table 6-3 for sequence of actions.

In the character mode, (table 6-4) each consecutive character from the reader is entered into the lowest-order seven stages of IAR. After a character is loaded into IAR the ready signal is sent to the computer. The computer thus samples IAR after each character has been read.

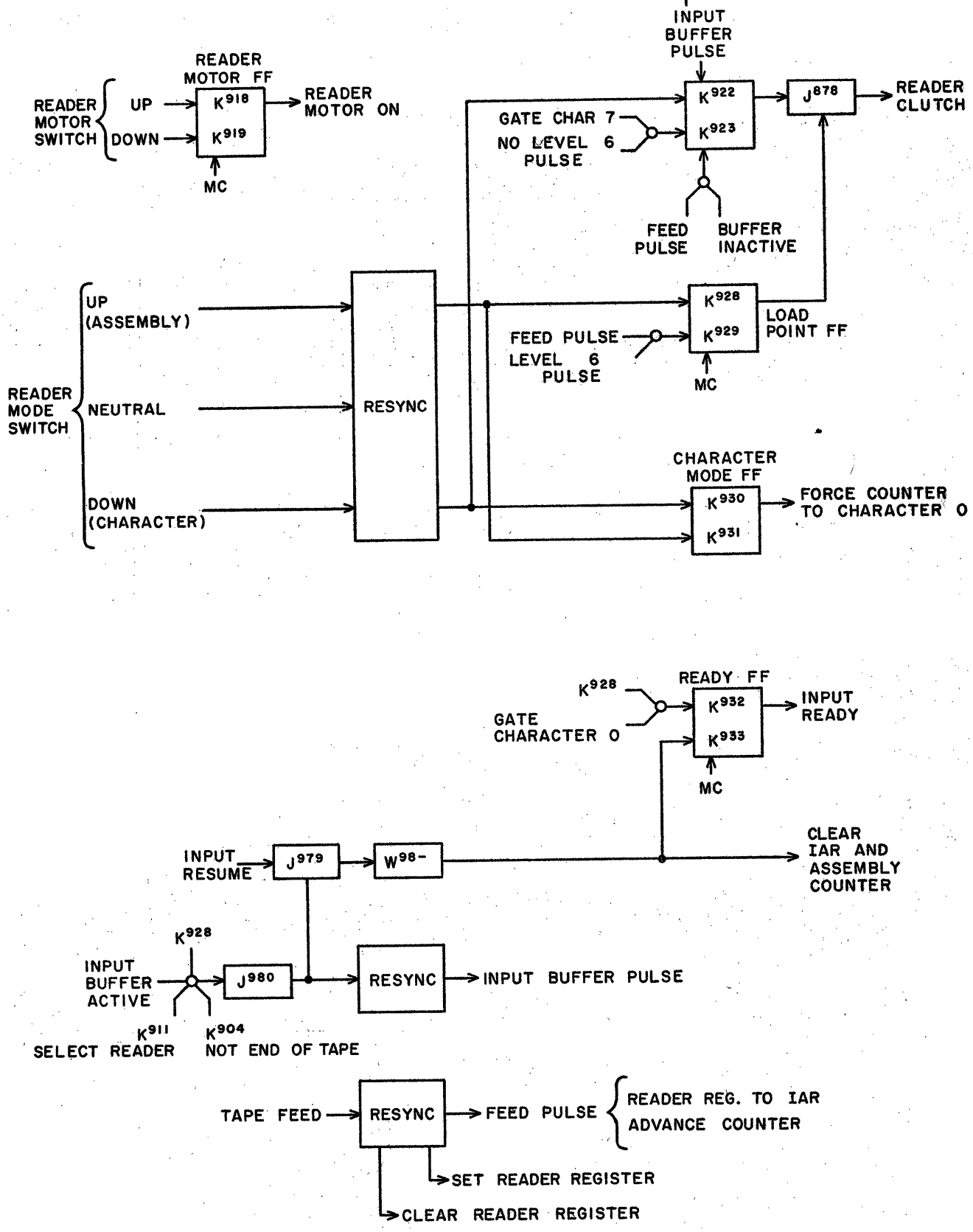



Figure 6-7. Reader Control Circuit.

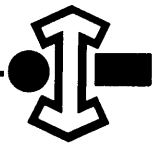


TABLE 6-3. ASSEMBLY MODE OF READER OPERATION

| ACTION | RESULT |
|---|---|
| 1) Select assembly mode (switch up) | Set Load Point FF (energize reader clutch).<br>Clear Character Mode FF.<br>Tape moves through leader until it reaches first hole in level 6 (indicates load point); this is first frame with information.<br>Clear Load Point FF (de-energizes clutch). |
| 2) Buffer Active Signal | Energize clutch.<br>Remove clear on IAR, assbly. ctr., and Ready FF.<br>Gate first character into IAR. |
| 3) Feed pulse | Transmit character from reader photocells to reader register.<br>Transmit reader register to IAR.<br>Advance assembly counter.<br>Wait for next feed pulse; repeat 3.<br>On 8th feed pulse (ctr. at character 0) set Reader Ready FF to send ready to computer. |
| 4) Computer samples IAR and sends resume | Clear IAR.<br>Clear Reader Ready FF.<br>Wait for next feed pulse; repeat 3. |
| If no level 6 hole for character 7 | Clear Clutch FF. |
| If Buffer terminates (made inactive) | Clear Clutch FF. |

TABLE 6-4. CHARACTER MODE OF READER OPERATION

| ACTION | RESULT |
|---|---|
| 1) Select character mode (switch down) | Set Character Mode FF.<br>Force assembly counter to character 0.<br>Set Clutch FF (energizes reader clutch).<br>Move tape one frame.<br>Clear Clutch FF since buffer inactive. |
| 2) Buffer active signal occurs | Energize clutch.<br>Remove clear on IAR, assbly. ctr., and Ready FF.<br>Gate first character into IAR. |
| 3) Feed pulse occurs | Transmit character to reader register.<br>Transmit reader register to IAR.<br>Set Ready FF to send ready to computer. |
| 4) Computer samples IAR, responds with resume | Clear IAR.<br>Clear Ready FF.<br>Wait for next feed pulse; repeat 3. |
| If buffer terminates (made inactive) | Clear Clutch FF. |

The tape motion stops on either of two conditions: (1) if the input buffer active line drops, signaling the end of the buffer operation; (2) if, in the assembly mode, a hole in level six does not accompany the first character of a word on the tape (this condition is interpreted as end-of-tape).

The end-of-tape circuit is shown in figure 6-8. The End-of-Tape FF ($K^{904/905}$) is cleared when a mode selection is made; this FF is set to indicate the end of a tape on either of two conditions:

1) By extemal function code 1121X. With this code, the end-of-tape FF can be set at the end of a character mode operation. This is necessary when program steps are conditioned by the result of a 74.7 instruction that senses end-of-tape.
2) In the assembly mode of operation, by the absence of a level 6 signal (from $I^{806}$) when the assembly counter is at count 000.

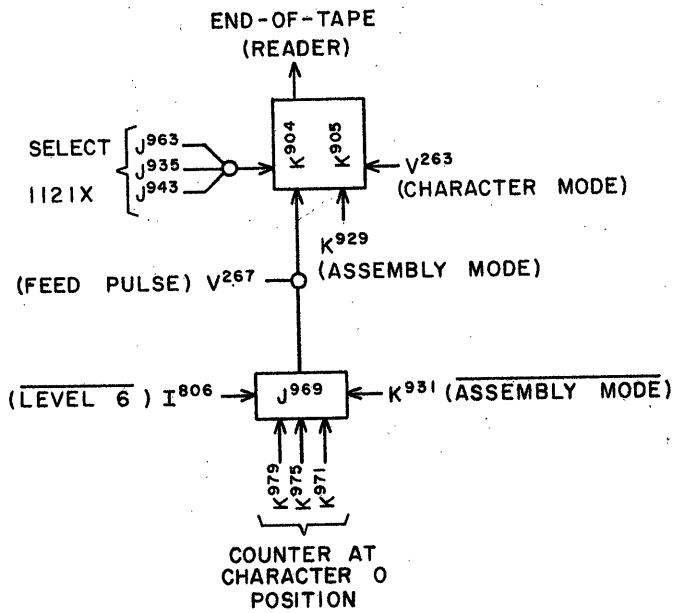
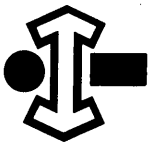



Figure 6-8. End of Tape Circuit.



# PAPER TAPE PUNCH

The Teletype BRPE paper tape punch is used to prepare paper tape output, and is always connected to buffer channel 2. Its nominal operating rate is 60 characters per second. Seven-bit characters are punched when the character mode is selected, and six-bit characters are punched when the assembly mode is selected.

## PUNCH CONTROLS

Punch manual controls are two lever switches on the punch and reader control panel of the console. The punch motor may be manually turned on by placing the Punch Motor switch in the up position or off by placing the switch in the down position. External function codes which control the punch motor are:

2120X Select Punch. assembly mode

2121X Select Punch, character mode

2124X Turn punch motor off

The Punch Mode switch selects the mode of operation of the punch. The switch must be placed in the up position to operate the punch with the computer. The down position provides for tape feed, that is, for punching out leader.

On the punch itself, the feedout lever also provides for punching out leader. (A microswitch is mounted near the roll of paper tape that supplies the punch. When the supply is low, the switch opens and provides an out-of-tape indication which may be sensed.)

## PUNCH OPERATION

When the punch motor is on, the punch mechanism rotates. However, advancement of the tape does not occur unless the punch feed magnets are energized. Once each revolution of the punch mechanism, the punch provides four timing signals to the punch control circuit, figure 6-9. These four signals, provided by two contactors designated No. 1 contactor and

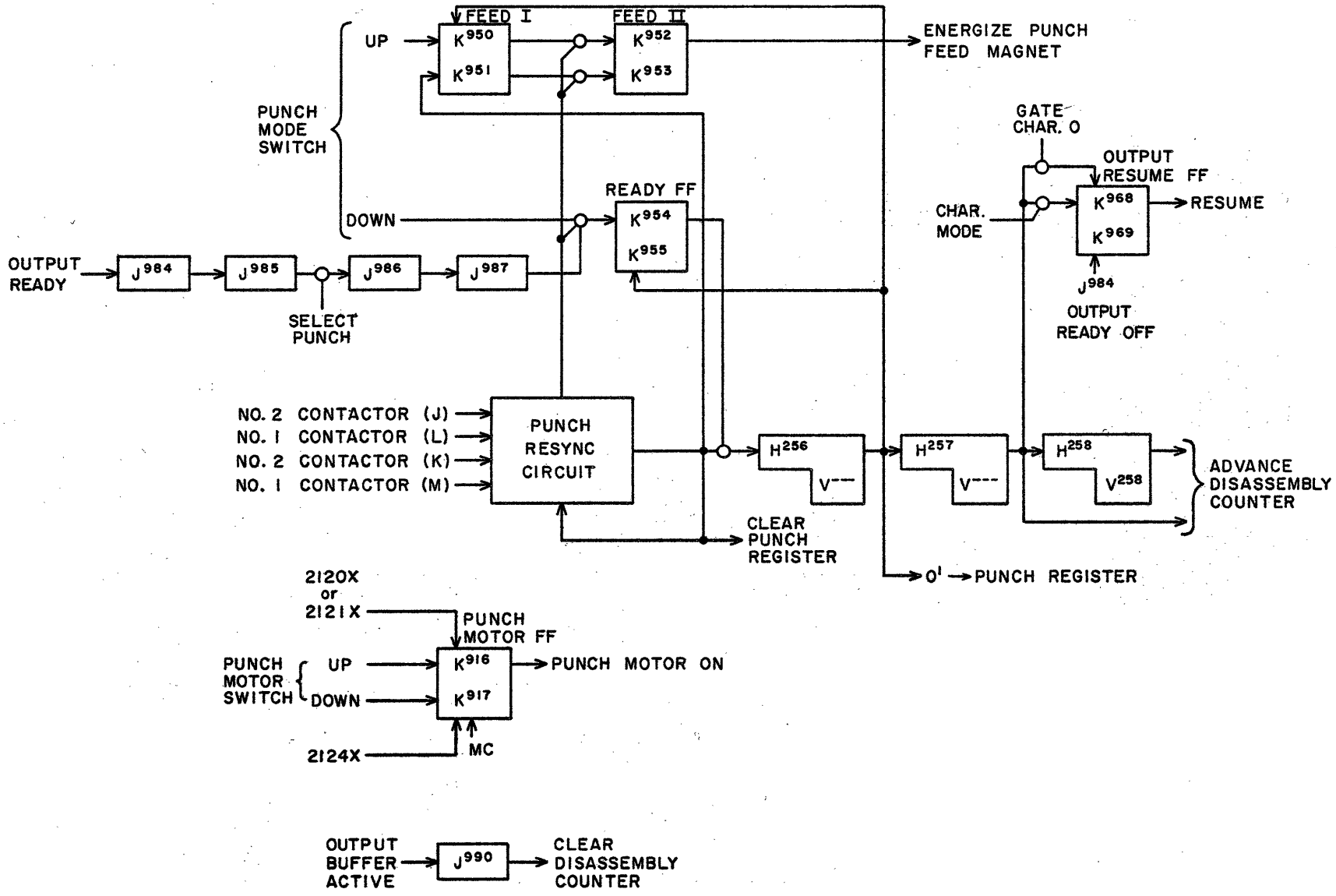
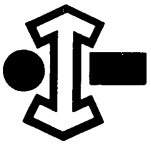



Figure 6-9. Punch Control Circuit.



No. 2 contactor, are resynchronized to produce a timing signal that initiates the steps of the operation causing the characters to be punched.

During the period the Punch Mode key is down, the Feed I and Feed II FFs remain set. The output of the Feed II FF energizes the feed magnet of the punch. As a result, tape is advanced and feed holes punched out.

Once the punch motor is turned on and the Punch Mode switch up, actual punching can begin when the computer: (1) Selects the punch, (2) sends output buffer active signal, and (3) loads $0^1$ and sends accompanying ready. The sequence of events in punching a character appears in table 6-5.

TABLE 6-5. PUNCH OPERATION

| | |
|---|---|
| 1) | Set Ready FF during absence of punch timing signal. |
| 2) | Punch timing signal occurs (initiates remaining steps). |
| 3) | Clear punch register. |
| 4) | Transmit character (determined by counter) from $0^1$ to punch register (this energizes punch magnets and feed magnet). |
| 5) | Clear Ready FF. |
| 6) | Set Resume FF if counter at character 0 (sends resume to computer). |
| 7) | Advance disassembly counter. |
| 8) | Return to step 1 if counter is not at character 0 before step 7. |
| 9) | Computer responds to resume (step 6) by turning off ready. |
| 10) | Clear Resume FF. |
| 11) | Computer loads $0^1$ and sends ready. |
| 12) | Return to step 1. |

The disassembly counter controls the distribution of characters from $0^1$ to the punch register. In the assembly mode, the counter disassembles each word into characters in the order shown in figure 6-4; the highest-order character is recorded first, the lowest-order character last. A resume signal is generated by $K^{968/969}$ each time the lowest-order character is entered into the punch register; this signal notifies the computer that another word may be entered into $0^1$.

In the character mode, the disassembly counter selects only the lowest-order 7-bit character of each word in $0^1$ for recording, ignoring all the other bits. The resume signal is generated after each character has been punched; thus a new word is received from the computer each punch cycle.

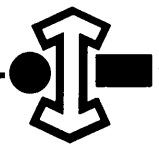
## TYPEWRITER

The typewriter is an IBM Electric, modified by Soroban Engineering, Inc., by the addition of a mechanical coder and decoder. The typewriter may be used as either a keyboard input device or as an output device for producing printed copy; for output the typewriter can operate at a rate of approximately 10 characters per second.

### TYPEWRITER CODES

All of the typewriter characters and functions are represented by unique combinations of six bits. The complete set of codes is given in table 6-6, where the characters and functions are listed along with their associated octal codes.

During a keyboard input operation the depression of a character key causes the coder to produce as an output the code for that key. Space is the only typewriter function for which a code is formed. For other functions (tab, back space, upper case, etc.) no code is formed during input operations. The coded character is then sent to the computer.



For a typewriter output operation a six-bit character consisting of one of the codes in table 6-6 is sent to the decoder. This causes the decoder to actuate the appropriate key for printing a character or closing one of the six function contacts to perform the function.

TABLE 6-6. TYPEWRITER CODES

| a. Characters UC | LC | Code | a. Characters UC | LC | Code | a. Characters UC | LC | Code |
|---|---|---|---|---|---|---|---|---|
| A | a | 30 | P | p | 15 | # | 3 | 64 |
| B | b | 23 | Q | q | 35 | $ | 4 | 62 |
| C | c | 16 | R | r | 12 | % | 5 | 66 |
| D | d | 22 | S | s | 24 | ¢ | 6 | 72 |
| E | e | 20 | T | t | 01 | & | 7 | 60 |
| F | f | 26 | U | u | 34 | ½ | 8 | 33 |
| G | g | 13 | V | v | 17 | ( | 9 | 37 |
| H | h | 05 | W | w | 31 | _ | - | 52 |
| I | i | 14 | X | x | 27 | ? | / | 44 |
| J | j | 32 | Y | y | 25 | " | ' | 54 |
| K | k | 36 | Z | z | 21 | ° | + | 46 |
| L | l | 11 | | | | . | . | 42 |
| M | m | 07 | ) | 0 | 56 | : | ; | 50 |
| N | n | 06 | * | 1 | 74 | , | , | 40 |
| O | o | 03 | @ | 2 | 70 | ÷ | = | 02 |

| b. Functions | Code | b. Functions | Code |
|---|---|---|---|
| Tab | 51 | Carriage Return | 45 |
| Space | 04 | Lower Case | 57 |
| Back Space | 61 | | |
| Upper Case | 47 | | |

KEYBOARD OPERATION

The keyboard control circuit shown in figure 6-10 provides for loading characters from the contacts of the typewriter coder into IAR. Selecting the keyboard allows the circuit to accomplish this by: 1) enabling the sampling of the keyboard common contacts and 2) removing the clear signal to IAR.

Each depression of a key lever actuates the coder which, in turn, closes some combination of the six contacts TC1 through TC6. These contacts provide output signals that form the 6-bit code for the character associated with the key. At the same time that one of the contacts TC1 through TC6 is actuated the common contacts of TCC are actuated. This signals to the control circuit that a character from TC1 through TC6 can be gated to IAR. The sequence of steps or events involved for one character is given in table 6-7.

TABLE 6-7. KEYBOARD OPERATION

| | |
|---|---|
| 1) | Depression of character key. |
| 2) | Key actuates coder which closes appropriate contacts TC1 through TC6 to generate code. |
| 3) | Key actuates common contacts of TCC. |
| 4) | Resynchronized signal from TCC contacts:<br>Gates character from TC1 through TC6 to IAR<br>Sets Ready FF (sends ready to computer)<br>Clears $K^{984/985}$ |
| 5) | TCC contacts return to normal state later in typewriter cycle. |
| 6) | Computer responds to ready with a resume which:<br>Clears IAR<br>Clears Ready FF |
| 7) | Return to step 1. |

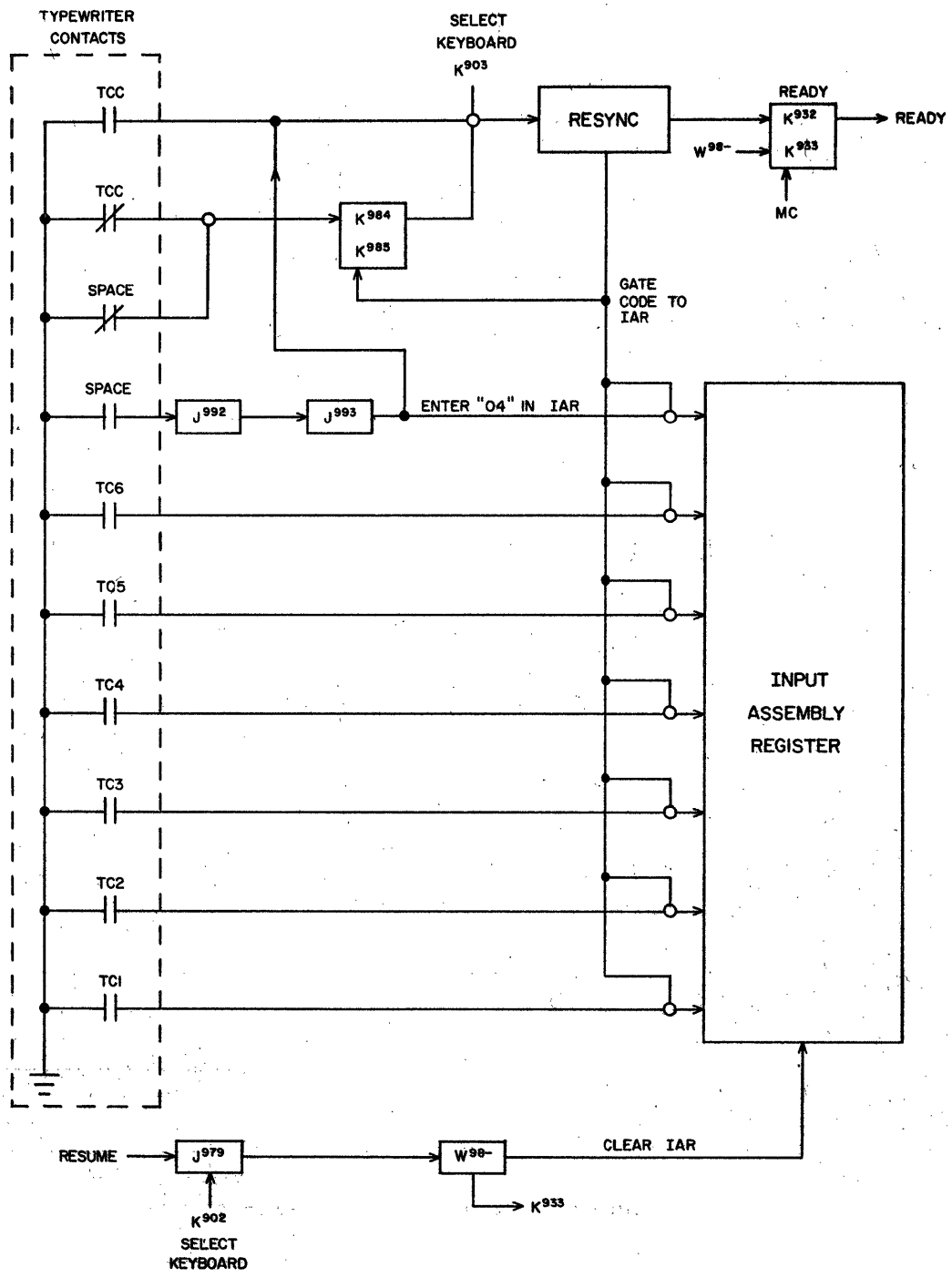



Figure 6-10. Keyboard Control Circuit.

The levers for typewriter functions do not actuate the coder. The space contacts do, however, cause the code 04 to be entered into IAR.

If the keyboard is selected by code 1114X, the interrupt signal is generated each time the carriage return (CR) or tab buttons are depressed. This feature allows the operator to gain the attention of the computer so that subsequently he may enter information via the keyboard.

The carriage return interrupt circuit is shown in figure 6-11. After selecting the keyboard with interrupt on CR, the state of the circuit is: $K^{988/989}$ set, and $K^{994/995}$, $K^{986/987}$ and $K^{996/997}$ clear.

The sequence of events that result from a CR or tab appears in table 6-8. The signal from CR or tab is resynchronized and stored by $K^{996/997}$ to produce the interrupt signal.

TABLE 6-8. CARRIAGE RETURN OR TAB INTERRUPT

| | |
|---|---|
| 1) | Depression of CR or tab<br>sets $K^{994/995}$ which<br>sets $K^{996/997}$ which<br>sets $K^{986/987}$ to prevent contacts from being<br>sampled again during typewriter cycle. |
| 2) | Computer interrupt program<br>determines by sense code 1110X that typewriter is<br>source of interrupt because $K^{996/997}$ is set.<br>does select 1110X which turns off interrupt and removes<br>interrupt selection by clearing $K^{988/989}$. |
| 3) | Several milliseconds later CR or tab contacts return to normal state<br>which clears $K^{994/995}$ and $K^{986/987}$. |

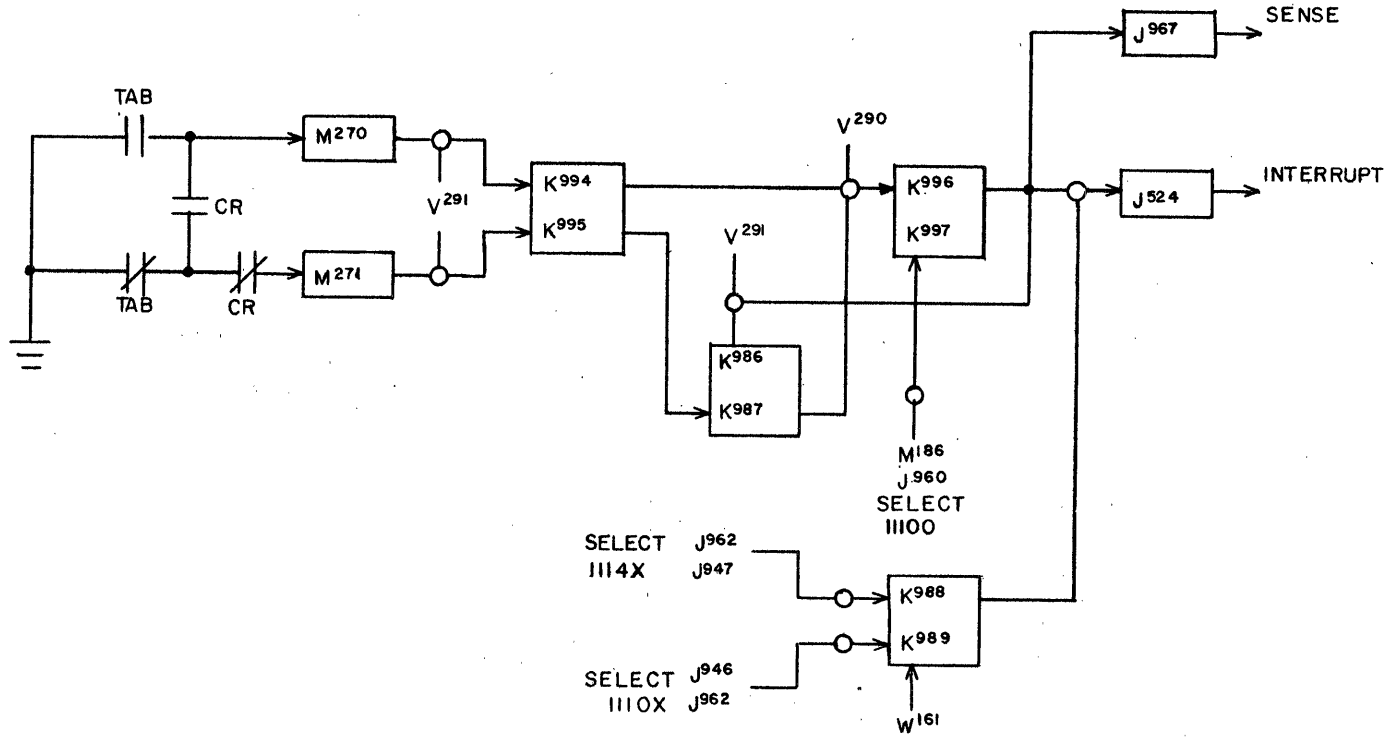



Figure 6-11. Carriage Return Interrupt Circuit.

## TYPEWRITER OUTPUT OPERATION

The output control circuit (figure 6-12) provides for accepting a character code from the $O^1$ register and sending it to the translator magnets (TM1 through TM6) of the typewriter decoder. Later the translator cam magnet (TCM) is energized by the circuit to: (1) type the character; or (2) close a function contact and perform the function.

The sequence of events in typing a character is given in table 6-9. Operation is allowed to begin when the typewriter is selected and the computer signals by a ready that $O^1$ is loaded. The control circuit gates the code to TM1 through TM6 and thereby starts the mechanical cycle of operation in the typewriter. Later, this cycle closes ribbon feed or one of the function contacts. The closing of one of these contacts signals to the control circuit that: (1) TCM can be energized; and (2) the code has energized some combination of TM1 through TM6. This signal after resynchronizing is used to prepare the next code.

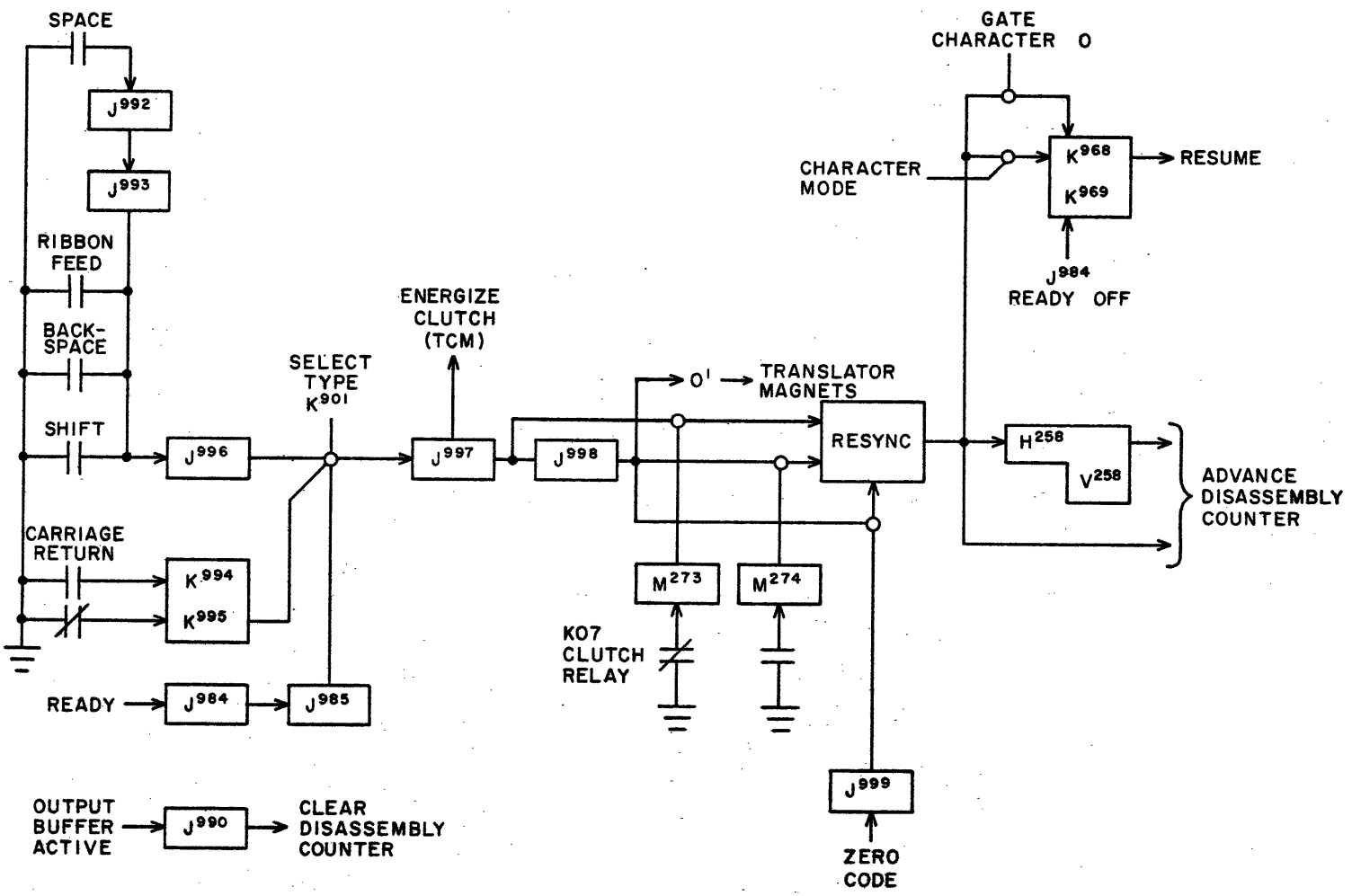
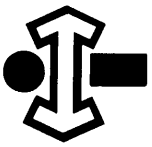



Figure 6-12. Typewriter Control Circuit (Output).

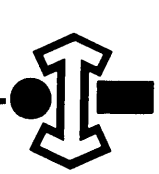


During character mode operation the disassembly counter is held at the count for character 0. Thus a resume occurs after each operation. When an illegal code (one not defined in table 6-6) is gated to TM1 through TM6 the decoder does not respond. As a result neither ribbon feed nor any function contact closes, and operation "hangs up" after step 2 of table 6-9. Depressing the CR, backspace or shift buttons will allow operation to be resumed.

A zero code (consisting of all "0" bits) constitutes a "do nothing " code. It is used, for example, to fill out a 48-bit word in the assembly mode. Since a zero code does not energize TM1 through TM6, the code is sensed by $J^{999}$ which initiates the necessary sequence of control signals.

TABLE 6-9. TYPEWRITER OPERATION (OUTPUT)

| | |
|---|---|
| 1) | Computer sends ready after loading $O^1$ register. |
| 2) | A character in $O^1$ (determined by disassembly counter) is transferred to TM1 through TM6. |
| 3) | TCM energizes to begin typewriter cycle that prints the character or performs the function. |
| 4) | If code is for typewriter character, ribbon feed contact closes. |
| 5) | If character mode or if in assembly mode, counter is at character 0, then set Resume FF. This sends resume to computer. |
| 6) | Advance disassembly counter (ineffective in character mode). |
| 7) | Computer drops ready because of typewriter resume; this clears typewriter Resume FF. |
| 8) | Near the end of the typewriter cycle the contacts open. |
| 9) | Return to step 1. |



# CHAPTER 7

# POWER AND COOLING SYSTEM

The computer system operates on two basic power inputs which are 120 vac, 60 cps and the 208 vac, 400 cps, three-phase input. The 120 vac, 60 cps power input is taken directly from the line to operate such equipment as the utility outlets, blower motors, punch motor, etc. The 400 cps power is produced by a motor-generator set which is used because its output current is relatively free from power surges occurring on the input to the motor, and is more easily filtered after rectification. The power output of the motor-generator (MG) is regulated to within $\pm$ 2 per cent of 208 vac and distributed to various d-c supplies.

The cabinets are provided with centrifugal blowers (when required) which cool the chassis by forcing room air around them. The room air, in turn, should be air-conditioned. To protect the components in the cabinet from physical damage due to excessive ambient temperatures, a high temperature thermostat and interlock system is employed.

## MOTOR-GENERATOR

The motor-generator set (figure 7-1) supplies the basic 208 vac, 400 cps power to the computer and external equipment cabinets. The motor operates from a three-phase, four wire 208 vac input which is controlled by the MG contactor (see figure 7-4). The contactor, in turn, is enabled by the power switch on the console. When the Power On switch is momentarily depressed, the MG contactor is energized and it closes the four normally-open contacts, thus applying input power to the motor. A holding contact keeps the contactor energized until the Power Off switch is depressed. The shaft of the motor is connected directly to the generator shaft.

Note that the 60 cycle circuit breaker for the computer cabinet and the 60 cycle contactor must be turned on before the 400 cycle contactor can be energized by the depression of the

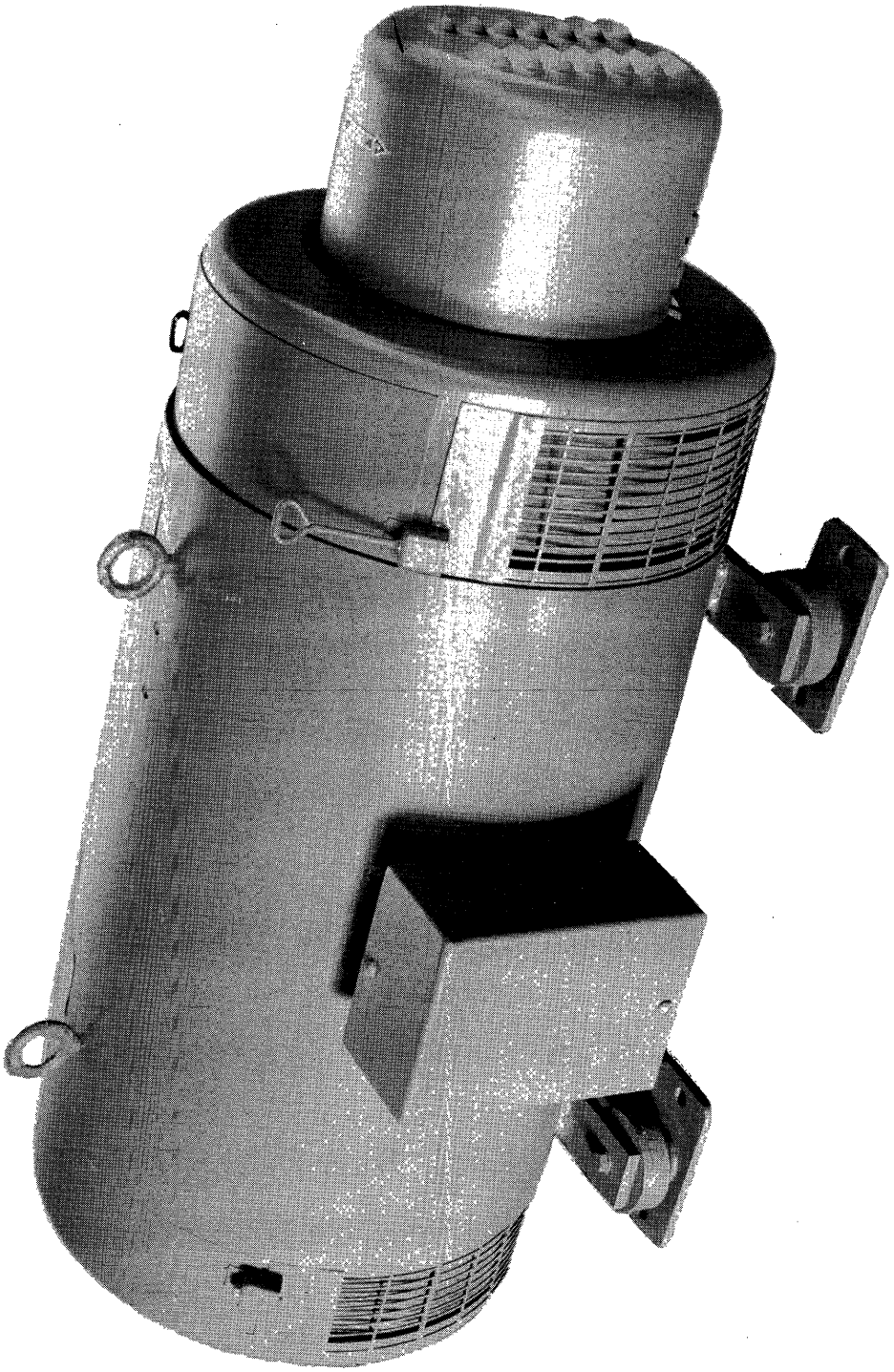
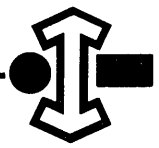


Figure 7-1. Motor Generator Set.



Power On switch. This is due to the fact that the 120 vac line that activates the MG contactor is taken from the 1604 circuit breaker, which is on the 60 cycle switch panel.

The generator output voltage is controlled by a voltage regulator circuit, which employs two magnetic amplifier reactors, as shown on Electric Machinery Manufacturing Co. 's diagram B-57652. The output voltage of the generator is sampled by this circuit, which then supplies a d-c voltage to the field of the brushless exciter, according to the amount of deviation of the generator output voltage.

The brushless exciter functions as a separate d-c generator that governs the field voltage of the main generator. As a result, a deviation of generator output voltage will cause an increase or decrease in the generator field voltage. The regulated output voltage of the generator is brought out to the control cabinet where it is used for the meters, indicators, etc. The 208 vac, 400 cps power output is brought through the air circuit breaker (ACB) which is located on the control cabinet.

Figures 7-2 and 7-3 show the front outside and inside views, respectively, of the control cabinet, which contains most of the control units for the motor-generator set. The various manual controls and indicators for the motor-generator are located on the cabinet door and are visible from the outside of the cabinet. A standard ammeter and voltmeter display the current and voltage values of the generator output voltage, while a vibrating reed type of frequency meter indicates the frequency of the generator output. Both the ammeter and the voltmeter are associated with a switch that must be turned on before the respective readings can be obtained.

A rheostat allows the manual adjustment of the output voltage; however, this control should not be used when the generator output voltage is at an improper level due to a

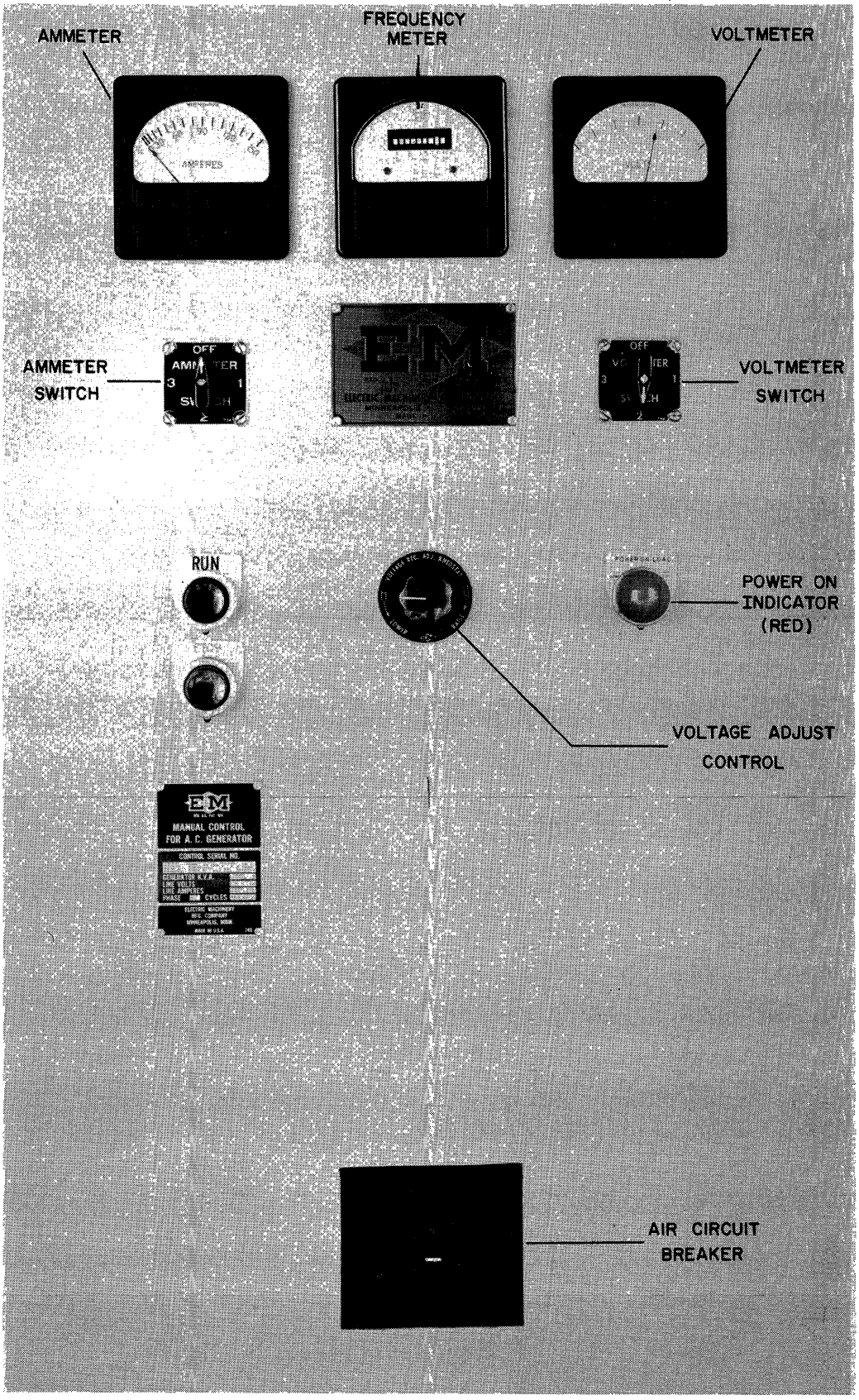



Figure 7-2. Power Control Cabinet, Front Outside View.

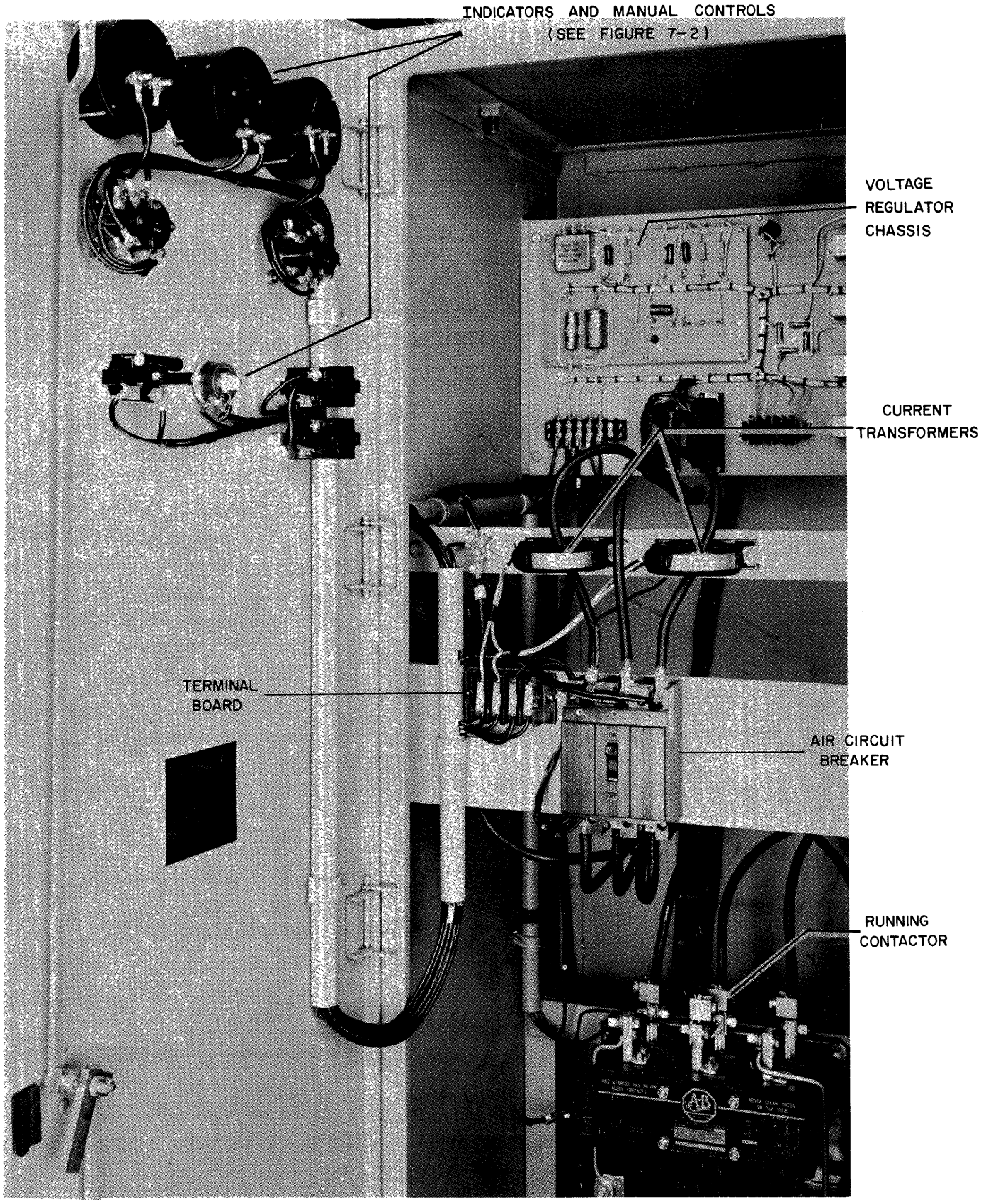



Figure 7-3. Power Control Cabinet, Front Inside View.

malfunction of the generator.

MAIN POWER DISTRIBUTION

Figure 7-4 shows the main power distribution for the computer and the associated external equipment cabinets. The 208 vac, 400 cps power is brought directly from the MG control cabinet through a junction box to the 400 cycle switch panel, which contains the circuit breakers. Each of the circuit breakers on this panel protects the 400 cycle power input to the associated cabinet and also allows manual disconnection of the 400 cycle power to these cabinets, should it become necessary. However, the usual procedure is to leave these circuit breakers on and to control the power from the console.

The 120 vac, 60 cps three-phase input is brought in directly from the line to the normally open (NO) contacts of the contactor in the 60 cycle switch panel. The contactor is energized by the depression of the Power On switch. A set of holding contacts maintains the contactor energized through the normally closed (NC) contacts of the Power Off switch. When this switch is depressed, the contactor is de-energized and all 60 cycle power is disconnected from the equipment.

The distribution of power to the cabinets is accomplished by cables originating at the two switch panels. All of the necessary connections are made in the junction box under the switch panels.

MAIN CABINET POWER DISTRIBUTION

The distribution of power in the main computer cabinet is shown in figure 7-5. The power cables bring 208 vac, 400 cps, three-phase and 120 vac, 60 cps, three-phase power into the cabinet.

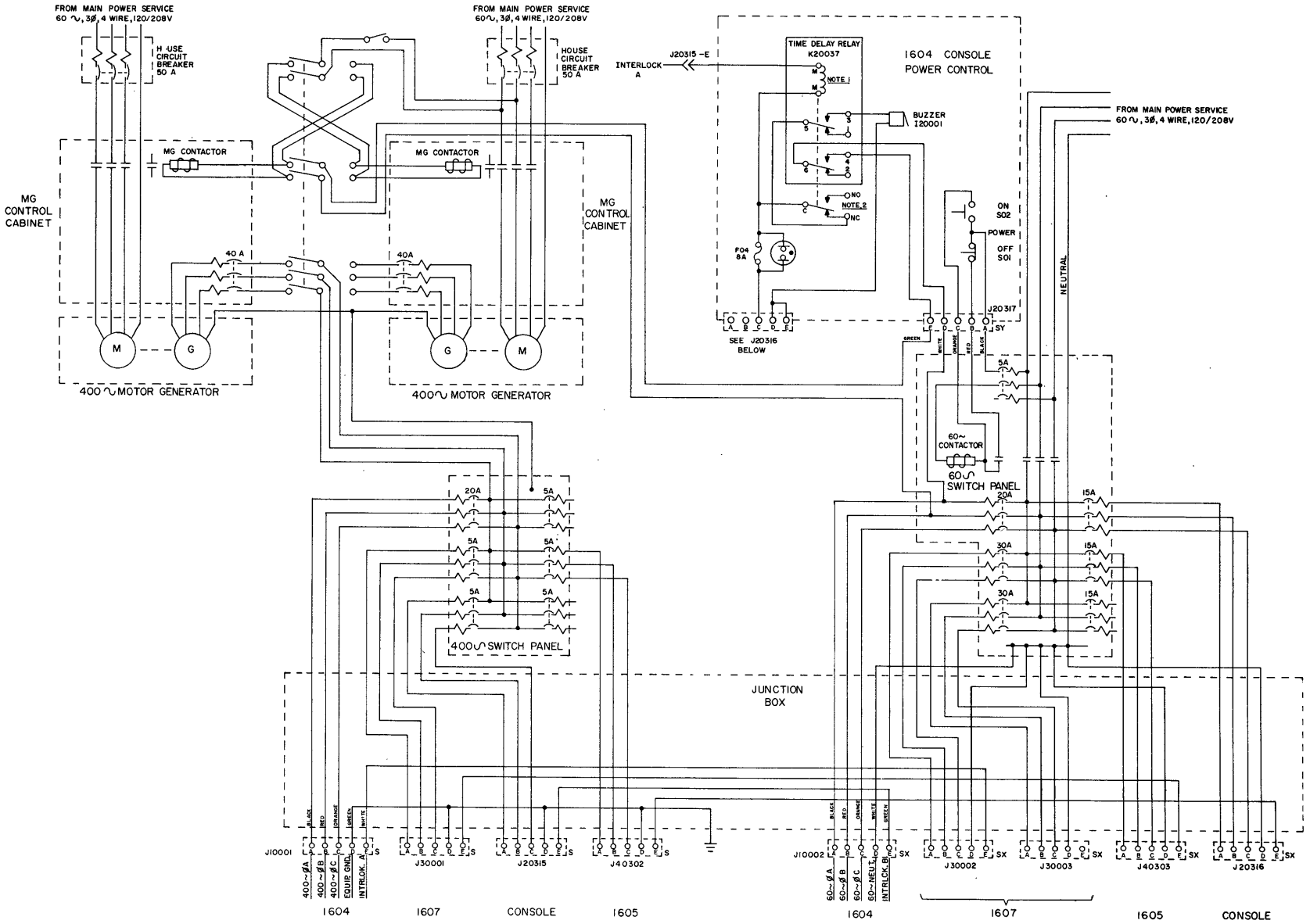



Figure 7-4. Main Power Distribution.

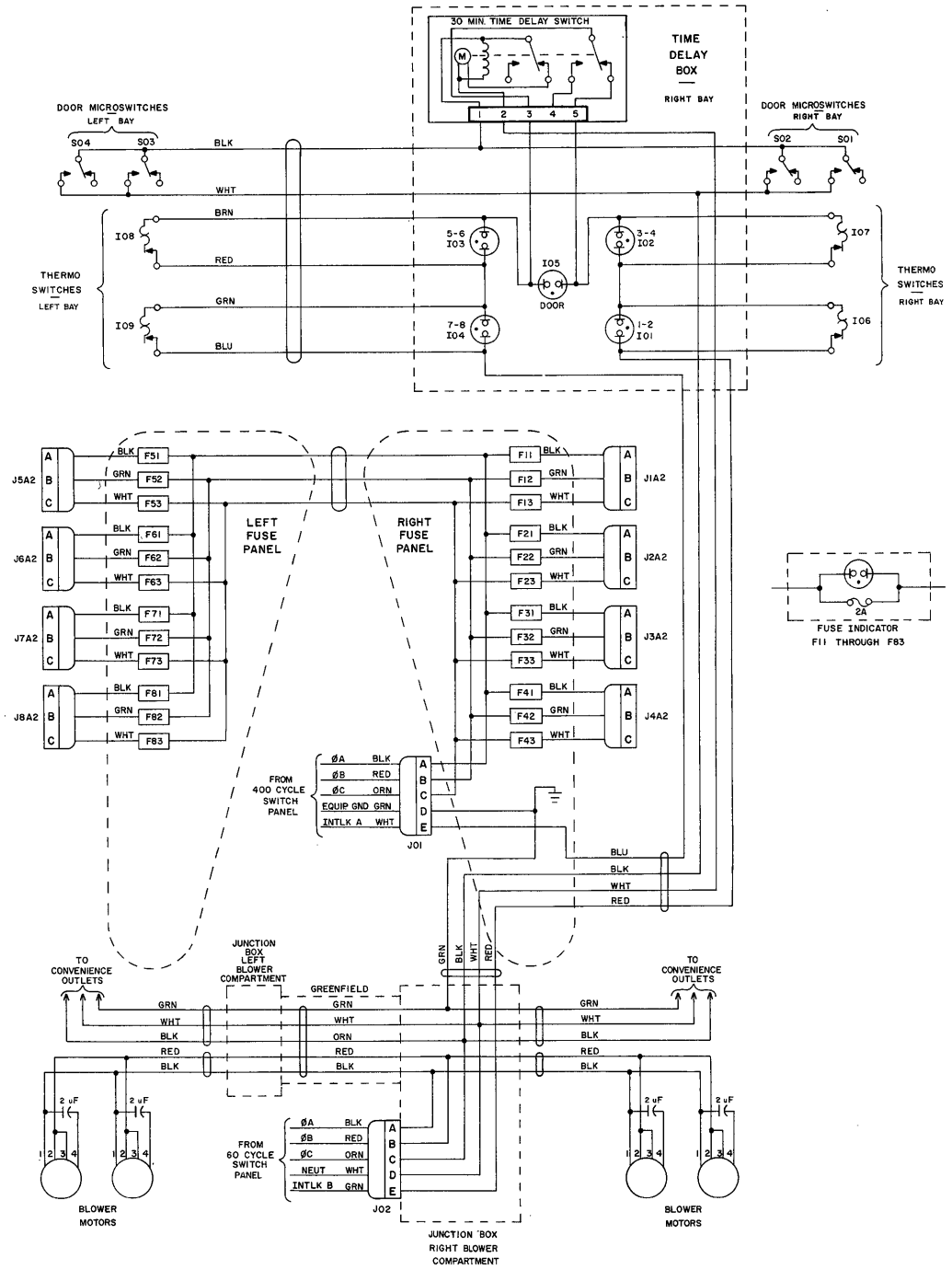
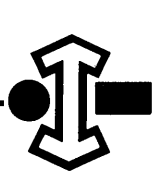



Figure 7-5. Main Cabinet Power Distribution.



The 208 vac, 400 cps power is connected in parallel to each of the eight chassis. Each of the three-phase inputs to a chassis is protected by a 2 amp fuse. The 400 cps power is applied to the primaries of the four delta-star transformers which are located on each chassis and furnish positive and negative 20 volt power to the printed circuit cards on the chassis (see figure 7-6). The secondary outputs of the transformers are rectified by a full-wave rectifier circuit which is composed of silicon rectifiers. These rectifiers are located on the lip of the corresponding chassis. The capacitors, which are contained on the 32 type 54 cards on each chassis, filter the output voltage.

CONSOLE POWER DISTRIBUTION

The console uses 208 vac, 400 cps, three-phase and 120 vac, 60 cps single phase power. The 400 cps power is applied to the -15 volt power supplies on relay chassis 20100 and 20200. These supplies, in turn, furnish -15 volts to the typewriter relays, the punch magnets and the relays associated with the display modules. The 60 cps power operates the lamps in the display modules, the convenience outlets, and the three external equipments in the console. A console power distribution diagram is shown on page 79 of Volume 5.

The 208 vac, 400 cps power input to the console is connected in parallel to the primaries of the delta-star transformer on relay chassis 20100 and 20200 respectively. These transformers in conjunction with the rectifiers and filter capacitors make up the -15 volt supply for their respective relay chassis. The -15 volt supplies are identical to the -20 volt supplies in the main cabinet chassis, with the exception that: (1) the transformers are wound to produce -15 volts instead of -20 volts after rectification; and (2) the capacitors are 2000 microfarads instead of the 10 microfarads on each of the type 54 cards. The -15 volt outputs of the supplies are connected to pin 1 of each relay on their respective chassis as well as the relays located on the brackets behind the chassis. The -15 volt output is also used to supply the speaker coil.

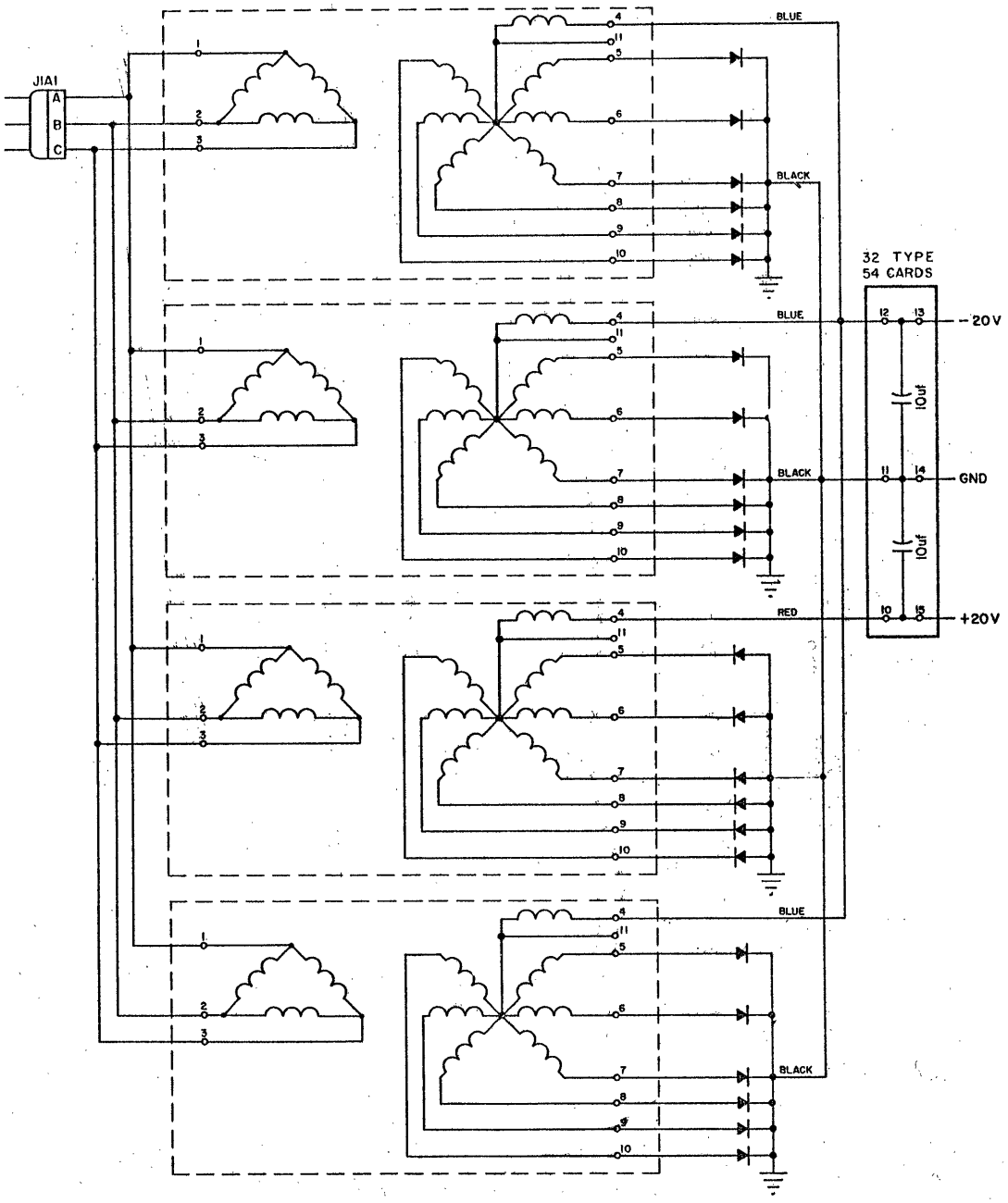
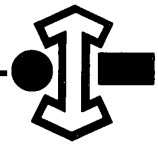



Figure 7-6. Typical DC Power Supply on Chassis.



Two transformers, T20001 and T20002, step the 60 cps power down to 6.3 vac for use by the lamps in the display modules. One side of the 6.3 vac supply connects to pin 6 of the relay associated with the uppermost bit in each octal group of the digit display translator. The other side of the supply is wired directly to one side of the lamps in the display modules. As a result, the translation of a particular octal digit closes the 6.3 vac circuit to the corresponding lamp in the associated module.

When the digit light disconnect relay 2D04 (lites) is de-energized (computer running), the 6.3 vac is opened to the relays associated with the digit lamp modules as shown on figure 7-6. Thus, the digit display is deactivated while the computer is running. However, the 6.3 vac circuit to the relays associated with the background indicator lights remains activated during the time the computer is operating.

COOLING SYSTEM

The computer cabinet is cooled by centrifugal blowers which force room air around the chassis. The room air temperature should be maintained at approximately $72^{o}$F. Blowers are mounted on the framework beneath each half of the cabinet floor. The blowers pull the air through the bottom section of the cabinet, through a diagonally mounted filter and through the two screen grills in each half of the cabinet floor. The air is guided through the card spaces on the chassis and exits through the grill at the top.

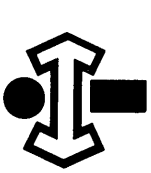
In installations where refrigerated air is supplied through ducts in the false floor, the air intake through the bottom of the computer cabinet is connected directly to the duct. Thus, the refrigerated air from the ducts is forced through the chassis card spaces.

PROTECTIVE INTERLOCK SYSTEM

A protective interlock system, consisting of high temperature thermostats and door interlocks, protects the printed circuit components against excessive ambient temperatures. A high temperature thermostat is mounted above the two chassis in each quarter section of the computer cabinet. These four thermostats are then connected in series with a time delay relay which is normally energized. (See figure 7-5.) A red indicator light is connected in parallel with the NC contacts of each of these thermostats; the lights are mounted on a bracket which is revealed by swinging out chassis 1 and 2.

Thus, if the ambient temperature in a particular quarter section becomes 85°F, the corresponding thermostat opens the circuit which turns on the associated indicator light and de-energizes the time delay relay which is mounted in the console (figure 7-4). The de-energizing of this relay immediately causes the warning buzzer to sound. (There is no delay in the closing of the NC contacts of this relay.) After a three minute delay, NO contacts 6-4 and 3-5 open which turns off the buzzer and opens the circuit to the MG contactor. As a result, all 400-cycle power is disconnected from the equipment. The thermostats automatically reset when the temperature falls to the proper level. However, the Power On switch must be depressed before 400-cycle power can be reapplied to the equipment.

The door interlock system consists of four normally open switches (one on each door). These switches are connected in parallel with each other and in series with the time delay motor. Thus, when one of the cabinet doors is opened, the corresponding switch is closed and the time delay motor is started. If the door is not closed within 30 minutes, the timer NC contacts open which disconnects the motor from the line and turns on the door indicator light. In addition, the circuit to the time delay relay is opened and this relay is de-energized. The de-energizing of the time delay relay causes the 400-cycle power to be turned off after three minutes as previously explained.



The closing of the cabinet door before the 30 minute time interval is up resets the timer to zero. However, the door should not be reponened for at least 10 minutes after it has been closed.

In summary, the thermostat and interlock system may be considered to be a simple series circuit from the 60 cycle neutral line, through all the thermostats, interlock contacts and the three minute time delay relay, to the 60 cycle hot line. (Figure 7-7 shows a simplified version of this series circuit for a typical installation, including the 1604 main cabinet, the console, two 1607 cabinets and one 1605 cabinet.)

As long as this series circuit remains closed, the three minute time delay relay is energized. Thus, the circuit to the MG contactors is kept closed until one of the NC contacts in a particular cabinet opens, which de-energizes the three minute time delay relay This action causes the warning buzzer to sound. After a three minute delay the two NO contacts of this relay open, which turns off the warning buzzer and de-energizes the MG contactor. The de-energizing of the contactor drops 400 cycle power from the equipment.

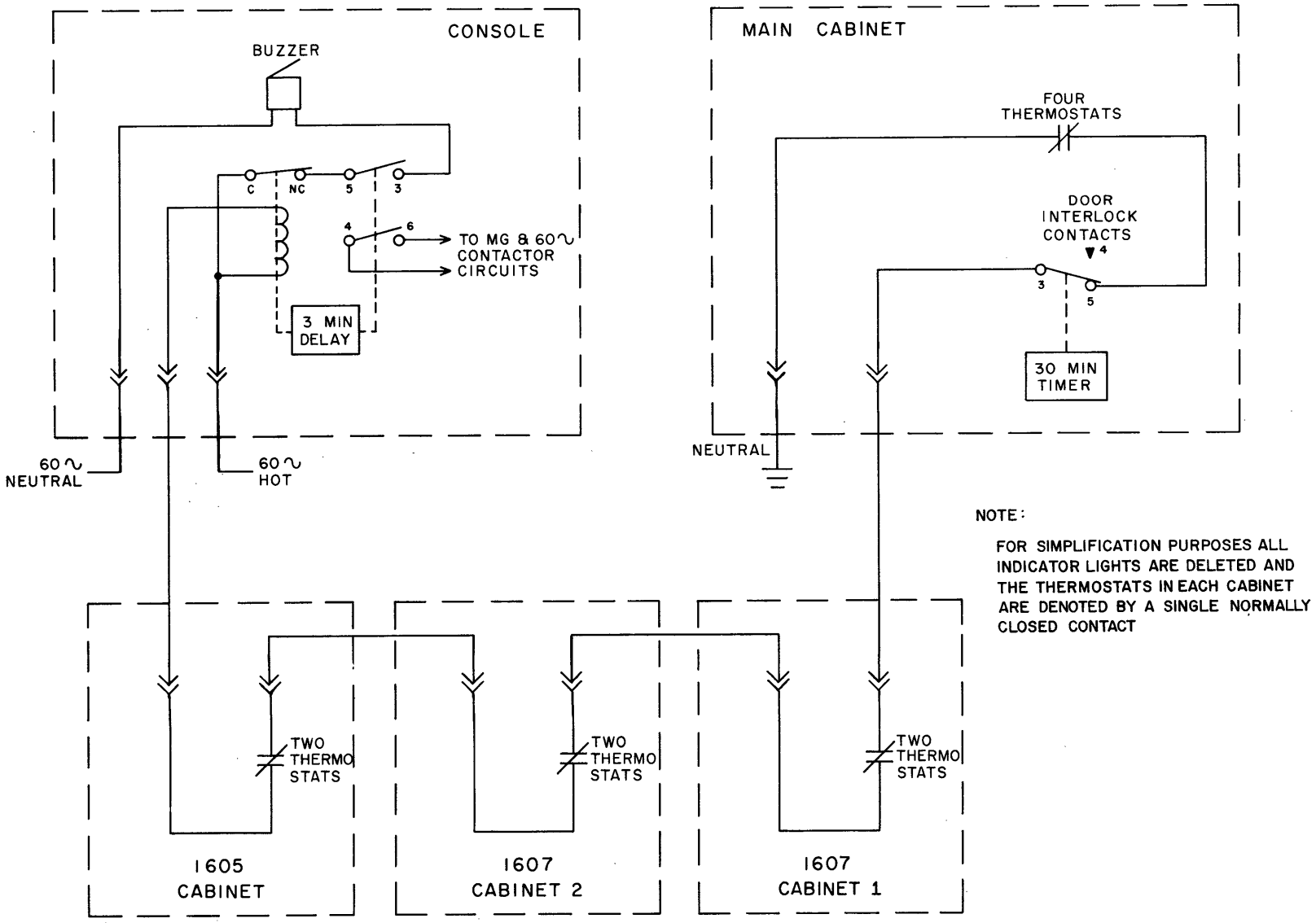
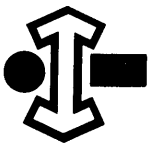



Figure 7-7. Simplified Thermostat and Interlock Circuit



# APPENDIX A

## BORROW PYRAMID

The borrow pyramid of the accumulator is discussed in detail in this appendix. Since the pyramid associated with the $U^2$ register differs only in the number of bits, this discussion applies to it as well. Figure A-6 (end of appendix) shows the general structure of the pyramids and gives the state of the output of each inverter during the performance of an actual addition. The operands used for illustration in figure A-6 are:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Addend ($A_i$) = | 000 | 000 | 000 | 000 | 011 | 110 | 110 | 000 | 111 | 110 | 000 | 010 | 110 | 101 | 001 | 111 |
| Augend ($X_i$) = | 000 | 000 | 000 | 000 | 001 | 010 | 000 | 111 | 111 | 100 | 111 | 101 | 110 | 110 | 001 | 011 |
| Sum ($A_f$) = | 000 | 000 | 000 | 000 | 101 | 000 | 111 | 000 | 111 | 011 | 000 | 000 | 101 | 011 | 011 | 010 |

The output of each inverter is indicated by the "0" or "1" in its associated block symbol. The derivation of the output given for any block can be proved by examing the figures which accompany the discussion of the various levels.

Each level consists of single inverters which are associated with a class of symbols; for example, the bit borrow enables are identified by the symbol $A^{--0}$. The name of each level is descriptive of the function it performs, or, more precisely, of the condition it senses. For any level except toggle control ($A^{--7}$) the following rules can be used to determine whether it is a "1" or a "0" output from the inverters that indicates the presence of the condition sensed by that level:

1) If the last digit of the superscript is even, a "1" output represents the presence of the condition denoted by the name of the level.
2) If the last digit is odd, a "0" output represents the presence of the condition denoted by the name of the level.

In the case of $A^{--7}$, a "1" output indicates that the stage is to be toggled.

The term enable refers to the condition in a stage, group, or section when a borrow request cannot be satisfied and it becomes necessary to pass to the next higher stage, group, or section. The term generation refers to the condition in a stage, group, or section when a borrow originating within it requires action from a higher stage, group or section. The borrow enable and borrow generation conditions in a given group or section are mutually exclusive. For example, if the enable condition exists in a given group, this group cannot generate an intergroup borrow. If the generation exists, then an intergroup borrow can be completed in the group.

Interpretation of Equation Symbols in Pyramid. All symbols of inverters in the pyramid have superscripts that lie within the range 500 to 977. The last digit indicates the level or function of a class of inverters. The first two digits indicate the stage, group or section with which an inverter is associated. Thus, stage 00 is denoted by 50 while stage 47 is denoted by 97. For inverters associated with groups or sections, the first two digits are the same as those of the first stage borrow enable in the group or section. However, for the group and section borrow enables the first two digits correspond to those of the second stage in the group or section.

1) Stage Borrow Enable. Each $A^{--0}$ inverter forms the logical difference of $A_n$ and $X_n$. This information about the relation of $A_n$ and $X_n$ is used to determine if: (1) a borrow is generated in that stage; (2) a borrow can be made from that stage without propagating further borrows; or (3) a borrow from that stage will not be completed but propagated to higher-order stages.

That each $A^{--0}$ forms the logical difference of $A_n$ and $X_n$ can be seen more easily by the following transformation performed on the equation for $A^{500}$, which forms the logical difference of $A_{00}$ and $X_{00}$. The horizontal bar represents the "not" function of the single inverter.

$$A^{500} = \overline{X^{003}\ A^{003} + A^{002}\ X^{002}} \qquad 1$$

$$= \overline{(X^{003}\ A^{003})}\ \overline{(A^{002}\ X^{002})} \quad \text{from 1 by De Morgan's theorem} \qquad 2$$

$$= \overline{(X^{003} + A^{003})}\ \overline{(A^{002} + X^{002})} \quad \text{from 2 by De Morgan's theorem} \qquad 3$$

$$= A^{003}\ \overline{X^{002}} + A^{002}\ \overline{X^{003}} \quad \text{Multiply out 3} \qquad 4$$

$A_{00}$ = "0" and $X_{00}$ = "1" $\qquad$ $A_{00}$ = "1" and $X_{00}$ = "0"

The output of $A^{--0}$ is "1" when the logical difference of $A_n$ and $X_n$ is "1". Such an output from an $A^{--0}$ indicates a stage borrow enable; that is, a borrow from this stage will in turn require a borrow from the next higher stage. A "0" output from an $A^{--0}$ indicates that either a borrow is generated in this stage or a borrow can be completed in this stage.

2) Group Borrow Enable. Each of the $A^{--1}$ inverters in the group borrow enable level senses whether an intergroup borrow into the group can be completed or if it will necessitate an intergroup borrow from the next group. A "0" output from a group borrow enable indicates that the borrow cannot be completed. This condition is sensed by combining in an AND function the outputs of the three stage borrow enables in the group.

3) Group Borrow Generation. Each of the $A^{--1}$ inverters in the group borrow generation level senses whether an intergroup borrow is generated within its group. When this is the case, the output of the inverter is "0". A "1" output indicates that no intergroup borrow is generated.

In figure A-1 the three inputs to $A^{501}$ illustrate how a group borrow generation senses the presence of an intergroup borrow. The input consisting of $X^{022} \cdot A^{022}$ specifies that $X_{02}$ and $A_{02}$, the last stages in the group, are both "0". Since it is the complement of X that is subtracted from A, and this case involves subtracting "1" from "0" in the last stage of a group, an intergroup borrow is necessary.

The input consisting of $X^{012} \cdot A^{012} \cdot A^{520}$ indicates that $A_{01}$ generated a borrow from $A_{02}$ and that in the latter stage a borrow cannot be completed. Thus a borrow is again required from $A_{03}$. Finally, the $A^{002} \cdot X^{002} \cdot A^{510} \cdot A^{520}$ input indicates that $A_{00}$ generated a borrow, but a borrow cannot be completed in either $A_{01}$ or $A_{02}$. An intergroup borrow is therefore required from $A_{03}$.

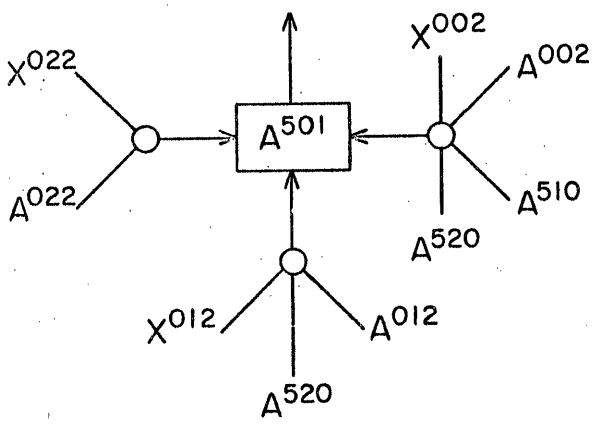



Figure A-1. Typical Group Borrow Generation Inverter.

4 Section Borrow Enables. Each of the $A^{--2}$ inverters in the section borrow enable level senses whether a borrow into its section can be completed or whether it will necessitate a borrow from the next section to the left. A "1" output shows that a borrow cannot be completed in that section. A section borrow enable is sensed by combining the output of the four group borrow enables in the section at the OR inputs of the inverter. Since the signals are represented by "0's" in this case, the circuit performs a logical AND function.

5) Section Borrow Generation. Each of the $A^{--2}$ inverters in the section borrow generation level senses whether a borrow is generated within its section. When this is the case the output of the inverter is "1".

The inputs of a section borrow generation actually sense for the four cases when no intersection borrow is generated. Thus for $A^{502}$ in figure A-2, the input consisting of

$A^{501} \cdot A^{531} \cdot A^{561} \cdot A^{591}$ provides for the case of no intergroup borrows being generated in the section.

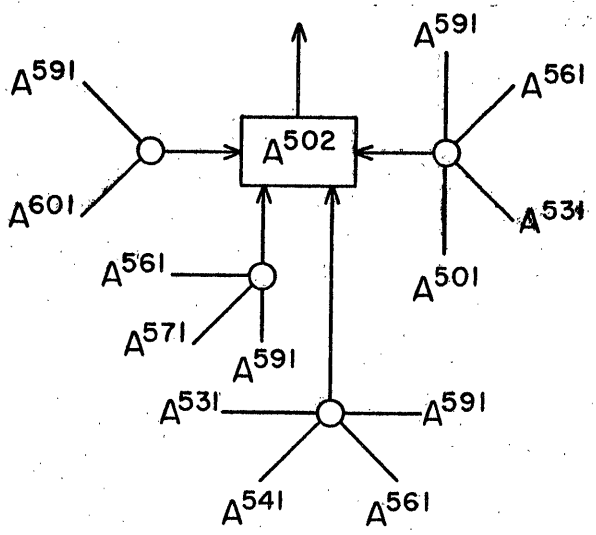



Figure A-2. Typical Section Borrow Generation Inverter.

The $A^{531} \cdot A^{541} \cdot A^{561} \cdot A^{591}$ input senses the case when no intergroup borrows are generated in groups 3 and 4 ($A^{561}$ and $A^{591}$) and a borrow can be completed in group 2 ($A^{531}$ and $A^{541}$). The case of no borrow generated in group 4 and the possibility of completing a borrow in group 3 is sensed by $A^{591} \cdot A^{571} \cdot A^{561}$. Finally, the $A^{591} \cdot A^{601}$ input handles the case when a borrow can be completed in group 4.

6) Section Borrow Input. Each of the four sections has an $A^{--3}$ lead which senses when an intersection borrow is to be made from its section. This condition, which is shown by a "0" output from the $A^{--3}$, is sensed by means of the output of the section borrow enables and generations.

A detailed examination of the four inputs to $A^{503}$ (figure A-3) will make clear the function of the section borrow input inverters. Each of the inputs indicates an intersection borrow; these are also end-around borrows since they come from section 4. The inputs are mutually exclusive.

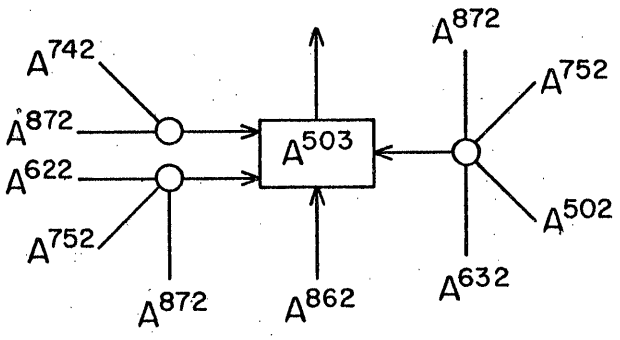



Figure A-3. Typical Section Borrow Input Inverter.

The input from $A^{862}$, the section borrow generation inverter in section 4, provides for the simple case when an intersection borrow is generated in section 4. The AND input consisting of $A^{742} \cdot A^{872}$ provides for an intersection borrow generated in section 3 ($A^{742}$) which cannot be completed in section 4($A^{872}$). The AND input consisting of $A^{622} \cdot A^{752} A^{872}$ provides for an intersection borrow generated in section 2($A^{622}$) which cannot be completed in either section 3($A^{752}$) or section 4($A^{872}$). Finally, the AND input consisting of $A^{502} \cdot A^{632} \cdot A^{752} \cdot A^{872}$ provides for an intersection borrow generated in a higher-order stage of section 1 which cannot be completed in sections 2, 3, or 4 and must, therefore, be made from a lower stage of section 1.

7) Group Borrow Input. Each group has an $A^{--4}$, the purpose of which is to determine whether an intergroup borrow is required from the group. When one is required all the inputs to $A^{--4}$ are "0" and, therefore, its output is "1".

Inputs to $A^{564}$ are shown in figure A-4. Because of the inversion in $A^{--4}$ its inputs may be said to sense the cases when no intergroup borrow is required. The input consisting of $A^{503}$ . $A^{531}$ . $A^{501}$ senses that there is no intersection borrow ($A^{503}$) and there are no intergroup borrows generated in either of the two lower groups ($A^{531}$ and $A^{501}$). The $A^{531}$ . $A^{511}$ . $A^{501}$ input shows that an intergroup borrow is not generated in the second group ($A^{531}$) and that an intergroup borrow can be completed in the first group ($A^{511}$ and $A^{501}$).

The latter condition is determined by the information that an intergroup borrow is neither generated nor enabled in the first group. Finally, the $A^{541}$ . $A^{531}$ input indicates a borrow can be accomplished in the second group and therefore an intergroup borrow will not be required of the third group.

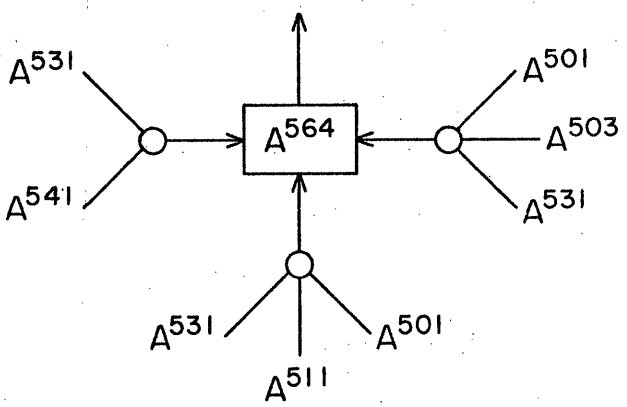



Figure A-4. A Group Borrow Input Inverter.

8) Stage Borrow Input. Each stage has an $A^{--5}$, the purpose of which is to sense when a borrow is required from that stage. The inputs to the three $A^{--5}$ inverters of the second group (figure A-5) are typical of the $A^{--5}$ inverters of other groups which have similar inputs.

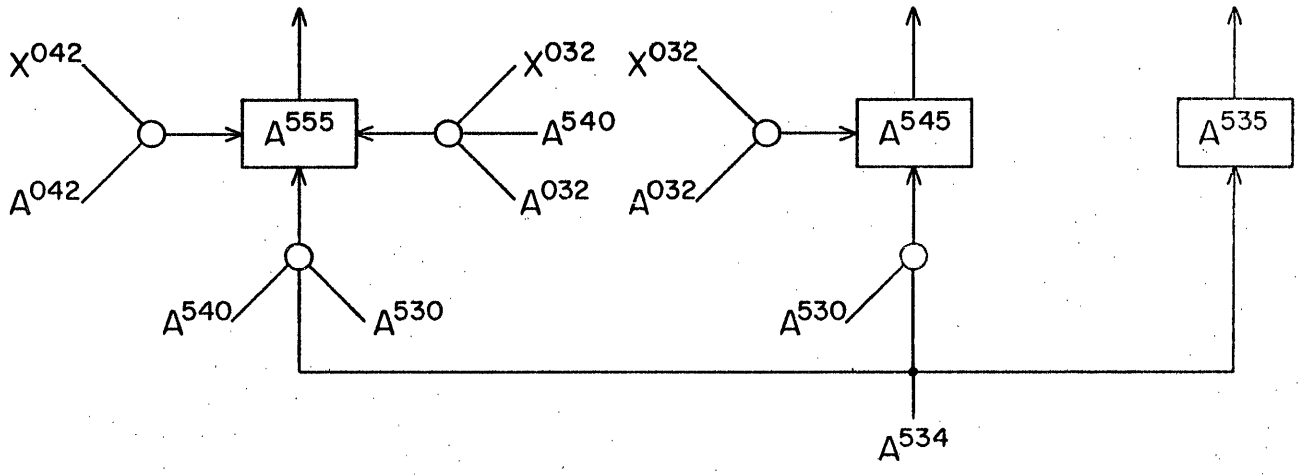
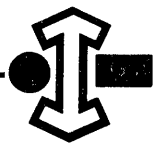



Figure A-5. Typical Stage Borrow Input Inverter.

A borrow is required from $A^{535}$, the lowest stage of the group, only when an intergroup borrow is made from the group. The input from $A^{534}$ handles this.

There are two cases when a borrow must be made from $A^{545}$, the middle stage. The first instance, when a borrow is generated in the lowest stage of the group, is sensed by the $A^{032} \cdot X^{032}$ input, which indicates that both $A_{03}$ and $X_{03}$ are "0". This denotes a "0 minus 1" operation since it is the complement of X that is subtracted. The second case exists when an intergroup borrow ($A^{534}$) must be made from the group and a bit borrow enable is present in the first stage ($A^{530}$).



A borrow is required from the third stage ($A^{555}$) in three cases: (1) when a borrow is generated in the second stage of the group ($A^{042}$ . $X^{042}$); (2) when a borrow is generated in the first stage ($A^{032}$ . $X^{032}$) and a bit borrow exists in the second ($A^{540}$); (3) ($A^{534}$ . $A^{540}$ . $A^{530}$), when this group receives an intergroup borrow ($A^{534}$) and there are bit borrow enables in the first ($A^{530}$) and second ($A^{540}$) stages.

If a borrow is required from a stage, the output of the $A^{--5}$ associated with that stage is "0". Every $A^{--5}$ has an input from a $W^{00-}$. During the addition of $X^2$ to $A^1$ the input to $A^{--5}$ from $W^{00-}$ is "0". But when $A^2$ is to be complemented on the basis of "1's" in $X^1$, then $W^{00-}$ is used to disable the borrow sensing portion of the pyramid by providing a "1" input to every $A^{--5}$. Thus the output of every $A^{--5}$ is "0" regardless of whether a borrow is required from its stage.

9) Stage No-Borrow Input. Each stage has an $A^{--6}$ which inverts the borrow information obtained from the $A^{--5}$ of the stage. The output of an $A^{--6}$ is "1" when a borrow is required from the stage. The function of $A^{--6}$ will be shown in the analysis of the $A^{--7}$ inverters.

10) Stage Toggle Control. The $A^{--7}$ of each stage senses whether or not that stage is to be toggled. At this point it will be well to have clearly in mind the two cases when the addition of X to A requires a given stage of A to be toggled. The cases are:

1) $X_n$ = "0" and no borrow is required from $A_n$
2) $X_n$ = "1" and a borrow is required from $A_n$

In addition, a "1" out of an $A^{--7}$ indicates the presence of one or the other of these two cases.

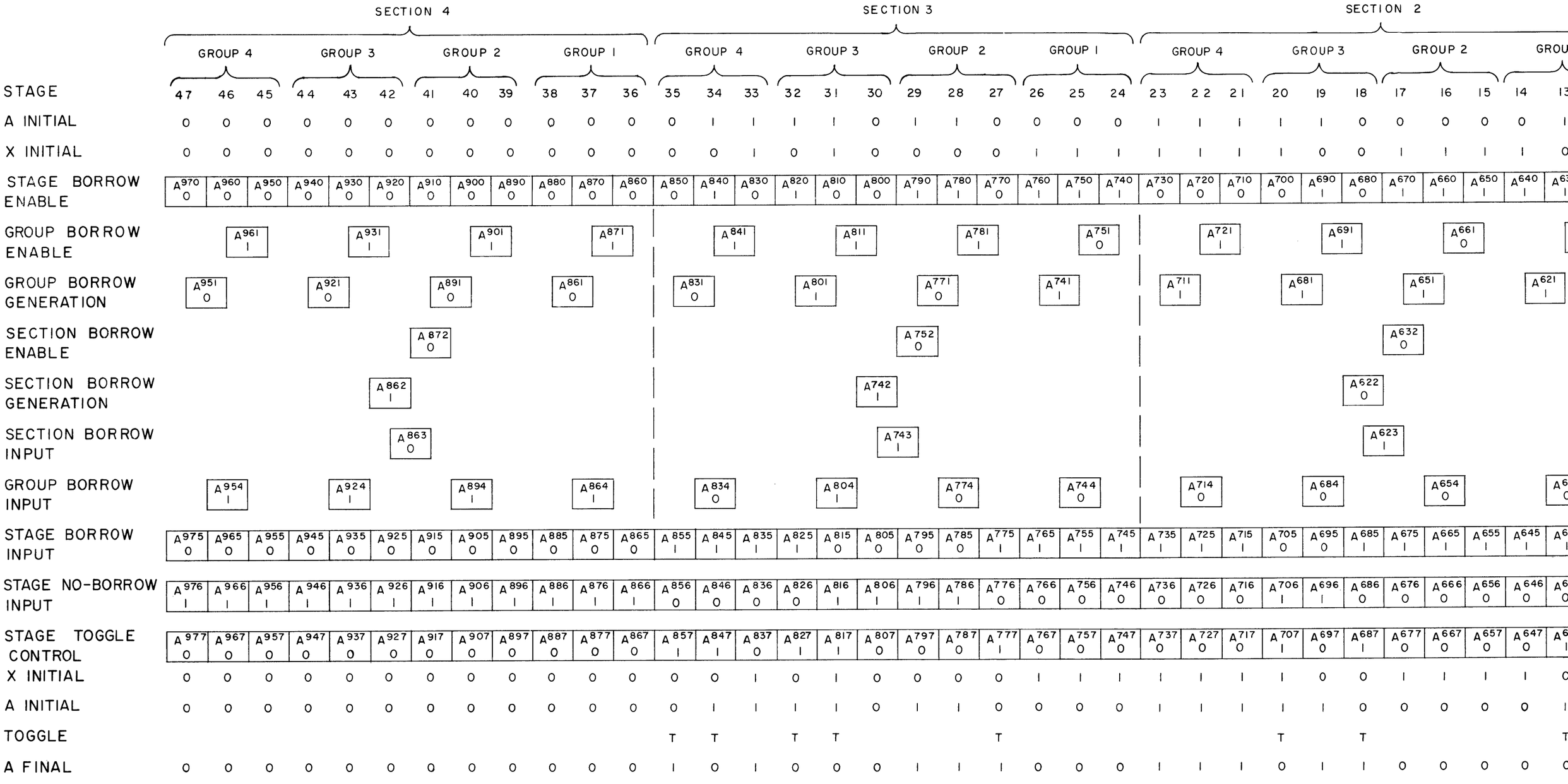
Because of the inversion of $A^{--7}$ their inputs may be considered as sensing the two cases when the stage is not to be toggled. These cases are:

1) $X_n$ = "0" and a borrow is required from $A_n$

2) $X_n$ = "1" and no borrow is required from $A_n$

Using stage $A_{00}$ as an example, the first case is sensed by the $X^{002}$ . $A^{506}$ input to $A^{507}$ while the second is sensed by the $X^{003}$ . $A^{505}$ input. In the first case, $X^{002}$ indicates $X_{00}$ = "0" and $A^{506}$ indicates a borrow is required. In the second case, $X_{00}$ indicates $X^{003}$ = "1" and $A^{505}$ indicates no borrow is required.

When this part of the pyramid is used in complementing $A^1$ on the basis of "1's" in $X^2$, the Partial Add in A FF ($K^{520/521}$) is set. It in turn causes the $W^{00-}$ inverters to enter a "1" in each $A^{--5}$. Therefore, the $A^{--5}$ . $X^{--3}$ AND input to each $A^{--7}$ is not satisfied. The other input, $A^{--6}$ . $X^{002}$, is satisfied if X is "0". If $X_n$ = "0" then the $A^{--7}$ receives a "1" and its output is "0". Consequently, $A_n$ is not toggled. However, when $X_n$ = "1" neither input is satisfied and a "1" output is provided, causing $A_n$ to be toggled.


SECTION 4
SECTION 3
SECTION 2
GROUP 4
GROUP 3
GROUP 2
GROUP 1
STAGE
47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14
A INITIAL
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0
X INITIAL
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1
STAGE BORROW ENABLE
A970 0 A960 0 A950 0 A940 0 A930 0 A920 0 A910 0 A900 0 A890 0 A880 0 A870 0 A860 0 A850 0 A840 1 A830 0 A820 1 A810 0 A800 0 A790 1 A780 1 A770 0 A760 1 A750 1 A740 1 A730 0 A720 0 A710 0 A700 0 A690 1 A680 0 A670 1 A660 1 A650 1 A640 1
GROUP BORROW ENABLE
A961 1 A931 1 A901 1 A871 1 A841 1 A811 1 A781 1 A751 0 A721 1 A691 1 A661 0
GROUP BORROW GENERATION
A951 0 A921 0 A891 0 A861 0 A831 0 A801 1 A771 0 A741 1 A711 1 A681 1 A651 1 A621 1
SECTION BORROW ENABLE
A872 0 A752 0 A632 0
SECTION BORROW GENERATION
A862 1 A742 1 A622 0
SECTION BORROW INPUT
A863 0 A743 1 A623 1
GROUP BORROW INPUT
A954 1 A924 1 A894 1 A864 1 A834 0 A804 1 A774 0 A744 0 A714 0 A684 0 A654 0
STAGE BORROW INPUT
A975 0 A965 0 A955 0 A945 0 A935 0 A925 0 A915 0 A905 0 A895 0 A885 0 A875 0 A865 0 A855 1 A845 1 A835 1 A825 1 A815 0 A805 0 A795 0 A785 0 A775 1 A765 1 A755 1 A745 1 A735 1 A725 1 A715 1 A705 0 A695 0 A685 1 A675 1 A665 1 A655 1 A645 1
STAGE NO-BORROW INPUT
A976 1 A966 1 A956 1 A946 1 A936 1 A926 1 A916 1 A906 1 A896 1 A886 1 A876 1 A866 1 A856 0 A846 0 A836 0 A826 0 A816 1 A806 1 A796 1 A786 1 A776 0 A766 0 A756 0 A746 0 A736 0 A726 0 A716 0 A706 1 A696 1 A686 0 A676 0 A666 0 A656 0 A646 0
STAGE TOGGLE CONTROL
A977 0 A967 0 A957 0 A947 0 A937 0 A927 0 A917 0 A907 0 A897 0 A887 0 A877 0 A867 0 A857 1 A847 1 A837 0 A827 1 A817 1 A807 0 A797 0 A787 0 A777 1 A767 0 A757 0 A747 0 A737 0 A727 0 A717 0 A707 1 A697 0 A687 1 A677 0 A667 0 A657 0 A647 0
X INITIAL
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1
A INITIAL
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0
TOGGLE
T T T T T T T T
A FINAL
0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 0 0 0 1 1 1 0 1 1 0 0 0 0

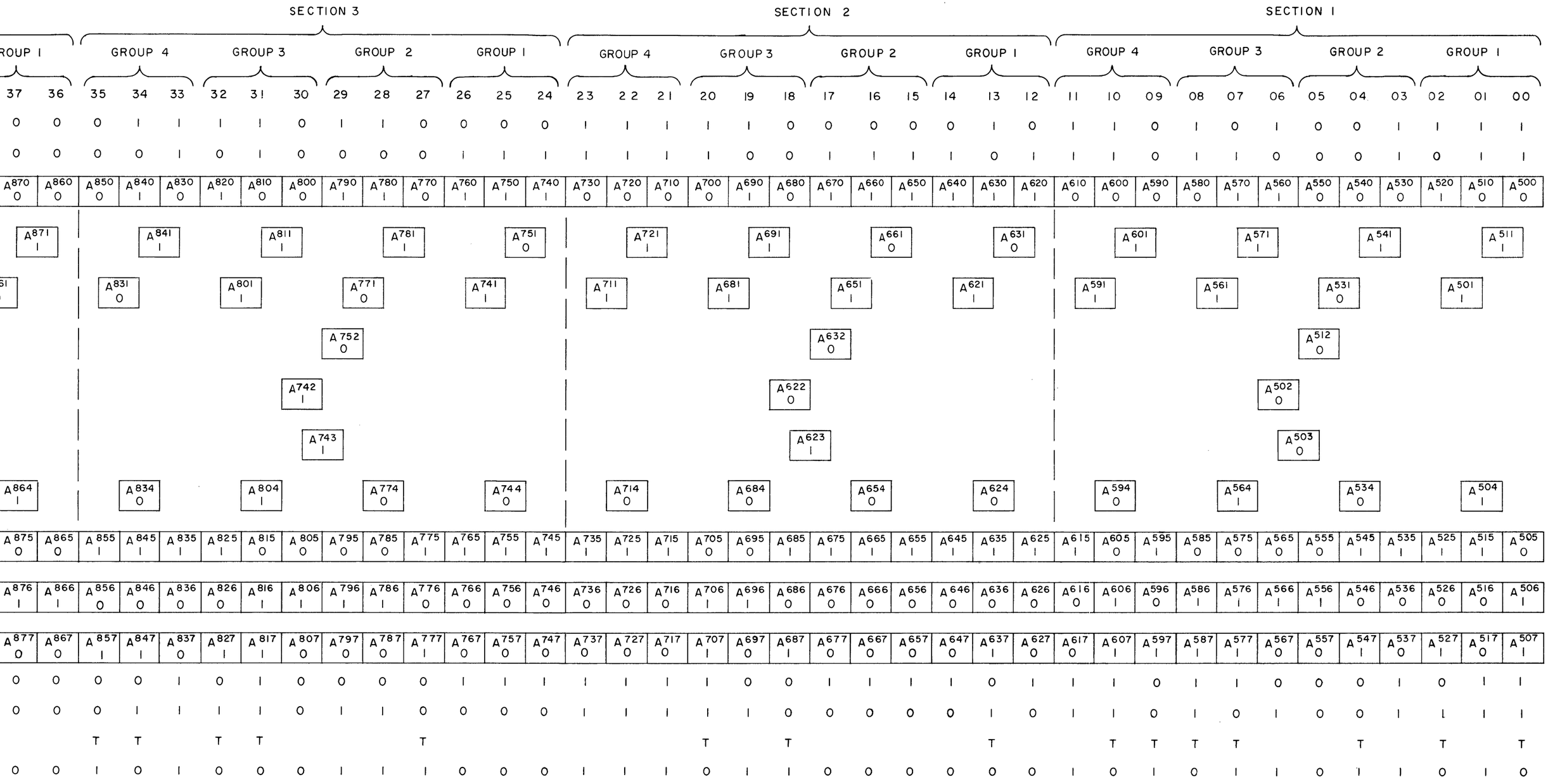
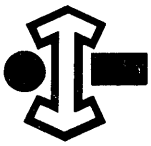



Figure A-6. General Structure of the Pyramid



# APPENDIX B

## GLOSSARY OF TERMS

The following glossary gives the meaning of terms that are used in a relatively specialized sense in this manual. In most cases, no attempt has been made to offer general definitions of the terms.

ACCUMULATOR - In general, a register which has provisions for the addition of another quantity to its content. Used as a proper noun, it refers to the principal arithmetic register, A.

ADDRESS - The number assigned as a designation for a memory location; also used to refer to the memory location itself.

AND FUNCTION - A logical function in Boolean algebra that is satisfied (and thus has the value "1") only when all of its terms have the value "1". For any other combination of the values of the terms it is not satisfied, and hence its value is "0".

BIT - Binary digit; may be either "1" or "0".

BORROW - In a subtractive counter or accumulator, a signal indicating that in stage n, a "1" was subtracted from a "0". The signal is sent to stage n + 1, which it complements.

BUFFER - In general, as a noun, a device in which data are stored temporarily in the course of their transmission from one point to another. As a verb, to store data temporarily. Specifically, the operation in an "active" buffer mode in which either a word from storage is sent to an external equipment via an output channel (output buffer), or a word is sent from an external equipment via an input channel to storage (input buffer).

CARD - Used to refer to any etched wiring board and attached components which is mounted on a chassis by means of a 15-pin connector.

**CARRY -** In an additive counter or accumulator, a signal indicating that in stage n, a "1" was added to a "1". The signal is sent to stage n + 1, which it complements.

**CHAIN -** A group of control delays connected serially and used for timing commands and signals. The sequences consist of such series of control delays.

**CHANNEL -** A transmission path (implemented by cables and connectors) that connects the computer to a given external equipment.

**CHARACTER -** Used in two senses to describe information handled by the computer:
1) A group of six bits which represent a digit, letter or symbol (such as appear on the keys of a typewriter). In the assembly mode, eight 6-bit characters make up a computer word.
2) A group of seven bits which represent an item of information. In the character mode, one 7-bit character and "0's" in the remaining (upper) 41 bits.

**CLEAR -** A command that removes a quantity from a register by placing every stage of the latter in the "0" state.

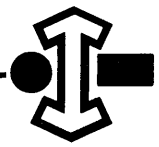
**CLOCK PHASE -** One of the two outputs from the master clock. One of the outputs is called the "even" phase and the other the "odd" phase. A given control delay has its inputs gated by one clock phase and its outputs gated by the other.

**COMMAND -** A signal (generally initiated by a control sequence) that performs a unit operation, such as the transmission of the content of one register to another, the shifting of a register one place to the left or the setting of a FF.

**COMPLEMENT -** Noun: see One's Complement or Two's Complement. Verb: a command which produces the one's complement of a given quantity.

**CONTENT -** The quantity or word held in a register or storage location.

**CONTROL DELAY -** A device capable of receiving a signal pulse and "holding" it for a period determined by the basic clock frequency (0. 2 microsecond) before allowing it to be released. Thus, a "controlled delay" device.



CORE - A small ferromagnetic toroid (doughnut-shaped) that is used as the bistable device for storing a bit in a memory plane.

COUNTER - A register with provisions for increasing (if additive) or decreasing (if subtractive) its content by 1 upon receiving the appropriate command.

ENABLE - In general, to satisfy one of the conditions required for the occurrence of a command.

END-AROUND BORROW - A borrow that is generated in the highest-order stage of an accumulator or counter, and is sent directly to the lowest-order stage. Provision for such a borrow makes the accumulator or counter "closed".

ENTER - To place a quantity, not from storage, in a register. For example: in instruction 10, Enter A, the execution address is placed in the A register. See Load and Store.

EVEN STORAGE - That storage unit which contains the 16, 384 even addresses.

EXECUTION ADDRESS - The lower 15 bits of a 24-bit instruction. Most often used to specify the storage address of an instruction operand. Sometimes used as the operand itself.

EXIT - Verb: the initiation of a second control sequence by the first, occurring when the first is near completion. Specifically, the control sequences exit to the Read Next Instruction or to the Auxiliary sequence.
Noun: refers to the circuitry involved in exiting.

FAULT - A type of operational difficulty which: 1) causes operation to stop, such as inadmissable operation codes 00 and 77; or 2) does not cause operation to stop, but sets an indicator; for example, divide fault.

FLIP-FLOP (FF) - A storage device with two stable states, one called "1" and the other, "0". A "1" input to the set side puts the FF in the "1" state, while a "1" input to the clear side puts the FF in the "0" state. After the input is gone, the FF remains in a state indicative of its last "1" input. A stage of a register consists of a FF.

FUNCTION CODE - The upper nine bits of a 24-bit instruction. It consists of the operation and index codes.

GATE - Verb: to satisfy one of the conditions required for the occurrence of a signal. Noun: may denote either the AND function or the OR function, but more commonly the former.

INDEX CODE - A three-bit quantity consisting of bits 15, 16, and 17 of an instruction. It is most frequently used to specify an index register whose contents are added to the execution address. In some instructions the index code simply specifies the conditions for carrying out the execution of the instruction.

INSTRUCTION - A 24-bit quantity consisting of a function code and an execution address. The instruction directs the computer to take a certain action.

INTERRUPT REQUEST - A signal received from an external equipment that may cause a special sequence of instructions to be executed in order to respond to the request.

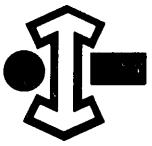
INVERTER - A circuit, which provides as an output a signal that is opposite to its input. If the input is "1", the output of an inverter is "0". An inverter output is "1" only if all the separate OR inputs are "0".

LOAD - To take a quantity from storage and place it in a register. For example: in instruction 12, Load A, a word is taken from storage and placed in A. See Enter and Store.

LOGICAL PRODUCT - In Boolean algebra, the AND function of several terms. The product is "1" only when all the terms are "1"; otherwise it is "0". The logical product of two quantities consists of bits, each of which is the AND function of corresponding bits of the two quantities. Sometimes referred to as the result of "bit-by-bit" multiplication.

LOGICAL SUM- In Boolean algebra, the OR function of several terms. The sum is "1" when any or all of the terms are "1"; it is "0" only when all are "0".

LOWER ADDRESS - Refers to the execution address portion of a lower instruction;



specifically, to bit positions 0 through 14 of a 48-bit register or storage location.

LOWER INSTRUCTION - See Program Step.

MASK - In the formation of the logical products of two quantities, one of them may be used as a mask for the other. The mask determines what part of the other quantity is to be considered. Wherever the mask is "0" that part of the other quantity is cleared, but wherever the mask is a "1", the other quantity is left unaltered.

MASTER CLEAR (MC) - A general command produced by placing the CLEAR switch up (external MC) or down (computer MC). A MC clears all of the crucial registers and control FFs to prepare for a new mode of operation.

MODULUS - An integer which describes certain arithmetic characteristics of registers, especially counters and accumulators, within a digital computer. The modulus of a device is defined by $r^n$ for an open-ended device and $r^n - 1$ for a closed (end-around) device, where r is the base of the number system used and n is the number of digit positions (stages) in the device. Generally, devices with modulus $r^n$ use two's complement arithmetic procedure while devices with modulus $r^n - 1$ use one's complement procedures.

NORMAL JUMP - An instruction that jumps from one sequence of instructions to a second, and makes no preparation for returning to the first sequence.

ODD STORAGE - That storage unit which contains the 16, 384 odd addresses.

ONE'S COMPLEMENT - With reference to a binary number, that number which results from subtracting each bit of the given number from the bit "1". The one's complement of a number is formed by complementing each bit of it individually, that is, changing a "1" to "0" and a "0" to a "1". A negative number is expressed by the one's complement of the corresponding positive number.

OPERAND - Usually refers to the quantity specified by the execution address. This quantity is operated upon in the execution of the instruction.

OPERATION CODE - The upper six bits of a 24-bit instruction which identifies the instruction. After the code is translated, it conditions the computer for execution of the specified instruction. The letter f is used to designate this code, which is expressed by two octal digits. For example, the Add instruction this code has the octal value 14.

OR FUNCTION - A logical function in Boolean algebra that is satisfied (and thus has the value "1") when any of its terms has the value "1". It is not satisfied only when all terms are "0". Often called the 'inclusive' OR function.

OVERFLOW - The condition in which the capacity of a register is exceeded. If the largest number which can be held in a register is $2^{48}-1$, then an overflow condition exists if an attempt is made to enter a number greater than this in it.

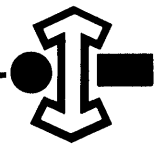
PARTIAL ADD - An addition without carries. Accomplished by toggling each bit of the augend where the corresponding bit of the addend is a "1".

PROGRAM - A precise sequence of instructions that accomplishes a computer routine; a plan for the solution of a problem.

PROGRAM STEP Consists of two 24-bit instructions contained in one 48-bit storage address. The instruction consisting of the higher-order 24 bits is called the "upper instruction"; the lower-order 24 bits make up the "lower instruction". Such a pair of instructions are read from storage together, and the upper instruction is executed first. The lower one is then executed, except when the upper one provides for skipping the lower one.

PYRAMID - For the A or $U^2$ registers, denotes a network of inverters that senses borrow conditions and produces appropriate borrow signals.

RANK - Registers composed of a pair of flip-flops per stage consist of two ranks, each containing one FF from the pair for each stage. Inverters arranged for parallel transmissions with one inverter per bit are also called ranks.



READ - With reference to storage, to remove a quantity from a storage location.

READY - The input-output control signal sent by either the computer or an external equipment to alert the device that is to receive a transmission. The ready signal indicates that the word or character has been transmitted.

REPLACE - In the title of an instruction, refers to the fact that the result of the execution of the instruction is stored in the same location from which the initial operand was obtained.

RESUME - The input-output control signal sent by either the computer or an external equipment to indicate that it is prepared to receive another word (48 bits) or character (usually 6 bits). The resume signal is thus a request for data.

RETURN JUMP - An instruction that jumps from a sequence of instruction to initiate a second sequence of instructions and, in addition, prepares for continuing the first sequence after the second is completed.

ROUTINE - The sequence of operations which the computer performs under the direction of a program.

SHIFT- To move the bits of a quantity columnwise right or left.

SIGN BIT - In registers where a quantity is treated as signed by use of one's complement notation, the bit in the highest-order stage of the register. If the bit is "1", the quantity is negative; if the bit is "0", the quantity is positive.

SIGN EXTENSION - The duplication of the sign bit in the higher-order stages of a register.

SKIP - To omit the execution of an instruction in a program. Only lower instructions may be skipped, and only if the upper instruction provides for skipping on a specified condition, and the condition is met.

SLAVE - Usually an inverter which receives an unconditional input from another circuit and is used simply to provide more outputs than are available from the first.

STAGE - The flip-flops and inverters associated with a given bit position of a register.

SUBINSTRUCTION - While the operation code specifies the instruction, the index code specifies one of eight different forms of that instruction. Such forms are called "subinstructions". Thus, 74. 0 is a subinstruction of instruction 74.

TOGGLE - Verb: To complement each individual bit of a quantity as a result of an individual condition.

TRANSMISSION, CLEARED - A transmission where both "1" and "0" are transferred into a register which has not been cleared previously.

TRANSLATION - An indication of the content of a group of bit registers. A complete translation gives the exact content, while a partial translation indicates only that the content is within certain limits.

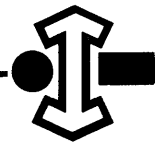
TWO'S COMPLEMENT - With respect to a number, that number which results from subtracting each bit of the given number from the bit "0". The two's complement of a number may be formed by complementing each bit of the given number and then adding one to the result, performing the required carries.

UPPER ADDRESS - The execution address portion of an upper instruction; specifically, bit positions 24 through 38 of a 48-bit register or storage address.

UPPER INSTRUCTION - See Program Step.

WORD - A unit of information which has been coded for use in the computer as a series of bits. The normal word length is 48 bits.

WRITE - With reference to storage, to enter a quantity into a storage location.



# APPENDIX C

## LIST OF INSTRUCTIONS

| | | | |
|---|---|---|---|
| 00 | ZRO | (not used) | Fault |
| 01 | ARS | A Right Shift | Shift (A) Right by K |
| 02 | QRS | Q Right Shift | Shift (Q) Right by K |
| 03 | LRS | AQ Right Shift | Shift (AQ) Right by K |
| 04 | ENQ | Enter Q | Y → Q, Extend Sign Y |
| 05 | ALS | A Left Shift | Shift (A) Left by K |
| 06 | QLS | Q Left Shift | Shift (Q) Left by K |
| 07 | LLS | AQ Left Shift | Shift (AQ) Left by K |
| 10 | ENA | Enter A | Y → A, Extend Sign Y |
| 11 | INA | Increase A | $[Y + (A)] \rightarrow A$, Extend Sign Y |
| 12 | LDA | Load A | (M) → A |
| 13 | LAC | Load A, Complement | (M)' → A |
| 14 | ADD | Add | $[(A) + (M)] \rightarrow A$ |
| 15 | SUB | Subtract | $[(A) - (M)] \rightarrow A$ |
| 16 | LDQ | Load Q | (M) → Q |
| 17 | LQC | Load Q, Complement | (M)' → Q |
| 20 | STA | Store A | (A) → M |
| 21 | STQ | Store Q | (Q) → M |
| 22 | AJP | A Jump * | Jump to m on condition j |
| 23 | QJP | Q Jump * | Jump to m on condition j |
| 24 | MUI | Multiply Integer | (M) (A)→AQ |
| 25 | DVI | Divide Integer | (QA)/(M) → A; Remainder=$Q_f$ |
| 26 | MUF | Multiply Fractional | (M) (A)→QA |
| 27 | DVF | Divide Fractional | (AQ)/(M) → A; Remainder=$Q_f$ |
| 30 | FAD | Floating Add | $[(A) + (M)] \rightarrow A$ |
| 31 | FSB | Floating Subtract | $[(A) - (M)] \rightarrow A$. |
| 32 | FMU | Floating Multiply | (M) (A) → A |
| 33 | FDV | Floating Divide | (A)/(M) → A |
| 34 | SCA | Scale A | A left until (A) ≥ .5 or k=0, k-No. of Shifts→$B^b$ |
| 35 | SCQ | Scale AQ | AQ left until (AQ)≥ .5 or k=0, k-No. of Shifts→$B^b$ |
| 36 | SSK | Storage Skip # | ($M_{47}$) Neg: EXIT; ($M_{47}$) Pos: Half EXIT |
| 37 | SSH | Storage Shift # | ($M_{47}$) Neg: EXIT, left 1; ($M_{47}$) Pos: Half EXIT, left 1 |
| 40 | SST | Selective Set | Set ($A_n$) for ($M_n$) = 1 |
| 41 | SCL | Selective Clear | Clear ($A_n$) for ($M_n$) = 1 |
| 42 | SCM | Selective Complement | Complement ($A_n$) for ($M_n$) = 1 |
| 43 | SSU | Selective Substitute | ($M_n$)→($A_n$) for ($Q_n$) = 1 |
| 44 | LDL | Load Logical | L(Q) (M) → A |
| 45 | ADL | Add Logical | [(A) + L(Q) (M)] → A |

**Legend**

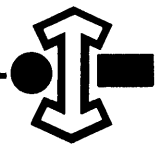
b - desig. for indexing
j - desig. for 22, 23, 74-76
k - exec. add. as shift cnt.
K - k+($B^b$)
m - exec. add. as op. add.
M - m+($B^b$)
y - exec. add. as operand
Y - y+($B^b$)
# - skip inst. (up. posit.)

| | | |
|---|---|---|
| 46 | SBL Subtract Logical | $[(A) - L(Q)\ (M)] \rightarrow A$ |
| 47 | STL Store Logical | $L(Q)\ (A) \rightarrow M$ |
| 50 | ENI Enter Index | $y \rightarrow B^b$; b=0: pass |
| 51 | INI Increase Index | $y + (B^b) \rightarrow B^b$ |
| 52 | LIU Load Index (upper) | $(m_{UA}) \rightarrow B^b$ |
| 53 | LIL Load Index (lower) | $(m_{LA}) \rightarrow B^b$ |
| 54 | ISK Index Skip # | $(B^b) \neq y$: $B^b + 1 \rightarrow B^b$, NI; $(B^b)=y$: $0 \rightarrow B^b$ Skip NI |
| 55 | IJP Index Jump | $(B^b) \neq 0$: $B^b - 1 \rightarrow B^b$, Jump to m; $(B^b)=0$: NI |
| 56 | SIU Store Index (upper) | $(B^b) \rightarrow m_{UA}$ |
| 57 | SIL Store Index (lower) | $(B^b) \rightarrow m_{LA}$ |
| 60 | SAU Substitute Address (up.) | $(A_{00-14}) \rightarrow M_{UA}$ |
| 61 | SAL Substitute Address (lwr.) | $(A_{00-14}) \rightarrow M_{LA}$ |
| 62 | INT Input Transfer | $(B^b)$ words to memory start at M-1 |
| 63 | OUT Output Transfer | $(B^b)$ words to memory start at M-1 |
| 64 | EQS Equality Search # | Search $(B^b)$ words, if (M-1), (M-2), etc. = (A) Skip NI |
| 65 | THS Threshold Search # | Search $(B^b)$ words if (M-1), (M-2), etc.> (A) Skip NI |
| 66 | MEQ Masked Equality # | Search $(B^b)$ words if L(Q) (M-1), (M-2), etc. = (A) Skip NI |
| 67 | MTH Masked Threshold # | Search $(B^b)$ words if L(Q) (M-1), (M-2), etc.>(A) Skip NI |
| 70 | RAD Replace Add | $[(M) + (A)] \rightarrow M$ & A |
| 71 | RSB Replace Subtract | $[(M) - (A)] \rightarrow M$ & A |
| 72 | RAO Replace Add One | $[(M) + 1)] \rightarrow M$ & A |
| 73 | RSO Replace Subtract One | $[(M) - 1] \rightarrow M$ & A |
| 74 | EXF External Function | j=1 - 6; activate ch. j, j=0: sel. ext. equip. m, j=7#: skip on cond. m |
| 75 | SLJ Selective Jump * | Jump to m on condition j |
| 76 | SLS Selective Stop * | Stop on j, and Jump to m* |
| 77 | SEV (not used) | Fault |

## DESIGNATOR FOR * INSTRUCTIONS

| | 22 | 23 | 75 | 76 |
|---|---|---|---|---|
| 0 | (A) = 0: Jump | (Q) = 0: Jump | Jump | Jump, Stop |
| 1 | (A) ≠ 0: Jump | (Q) ≠ 0: Jump | Key 1: Jump | Jump; Key 1: Stop |
| 2 | (A) Pos: Jump | (Q) Pos: Jump | Key 2: Jump | Jump; Key 2: Stop |
| 3 | (A) Neg: Jump | (Q) Neg: Jump | Key 3: Jump | Jump; Key 3: Stop |
| 4 | (A) = 0: Ret. Jump | (Q) = 0: Ret. Jump | Ret. Jump | Ret. Jump, Stop |
| 5 | (A) ≠ 0: Ret. Jump | (Q) ≠ 0: Ret. Jump | Key 1: Ret. Jump | Ret. Jump; Key 1: Stop |
| 6 | (A) Pos: Ret. Jump | (Q) Pos: Ret. Jump | Key 2: Ret. Jump | Ret. Jump; Key 2: Stop |
| 7 | (A) Neg: Ret. Jump | (Q) Neg: Ret. Jump | Key 3: Ret. Jump | Ret. Jump; Key 3: Stop |

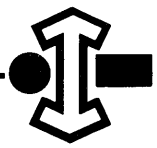
1, 2, & 3 refer to Selective Jump or Stop Key Switches



## LIST OF ILLUSTRATIONS


| Figure | Title | Page |
|---|---|---|
| 1-1 | Typical 1604 System | 1-3 |
| 1-2 | Simplified Diagram of the Computer | 1-5 |
| 1-3 | Typical Printed Circuit Card | 1-14 |
| 1-4 | Schematic Diagram of Standard Inverter Circuits | 1-15 |
| 1-5 | Interconnection of Inverters to Form a Flip-Flop | 1-17 |
| 1-6 | Control Delay | 1-19 |
| 1-7 | AND Circuit | 1-21 |
| 1-8 | Typical Designation for a Building Block Used in a Register | 1-23 |
| 1-9 | Logic Diagram Symbols | 1-25 |
| 1-10 | Three-Stage Single-Rank Register | 1-28 |
| 1-11 | Three-Stage Double-Rank Register with Shifting Properties | 1-29 |
| 1-12 | Three-Stage Counter | 1-31 |
| 1-13 | Interconnection of Three-Stage Counters to Form Nine-Stage Counters | 1-34 |
| 2-1 | Instruction Format | 2-2 |
| 2-2 | Relation of $U^2$ to $U^1$ | 2-3 |
| 2-3 | Structure of Operation Code Translator | 2-5 |
| 2-4 | Typical Translation of Operation Code | 2-6 |
| 2-5 | Adding in $U^2$ Accumulator | 2-10 |
| 2-6 | R Register Counting Structure | 2-14 |
| 2-7 | Parallel Transmission Inverter Ranks | 2-19 |
| 2-8 | Example of a Sequence | 2-22 |
| 2-9 | Over-all Sequence Control | 2-25 |
| 2-10 | Relation of Control and Storage Sequences | 2-27 |
| 2-11 | Form of RNI for Acquiring Instructions | 2-29 |
| 2-12 | RNI for Start and Stop | 2-31 |
| 2-13 | RNI for Interrupt Termination | 2-34 |
| 2-14 | Indirect Addressing Part of RNI | 2-38 |
| 2-15 | Normal Jump Sequence | 2-39 |
| 2-16 | Basic Chain of Control Delays in Zero Address Sequence | 2-40 |
| 2-17 | Basic Chain of Control Delays in RO Sequence | 2-43 |
| 2-18 | Write Operand Sequence | 2-47 |
| 2-19 | Search and Transfer Sequence | 2-51 |
| 2-20 | Iterative Sequence | 2-56 |
| 2-21 | Basic Chain of Control Delays in EF Sequence | 2-65 |
| 2-22 | Auxiliary Sequence | 2-67 |
| 2-23 | Relation of Auxiliary to Other Sequences | 2-69 |
| 2-24 | Console Display | 2-73 |
| 2-25 | Typical Digit Display, Lowest Octal Digit of A | 2-74 |
| 2-26 | Typical Resynchronizing Circuit | 2-77 |
| 2-27 | Resync Counter and Pulse Distribution | 2-79 |
| 2-28 | Connection of Start-Step Switch to RNI | 2-85 |
| 2-29 | Sampling Selective Jump Conditions | 2-88 |
| 2-30 | Master Clock Oscillation Waveforms | 2-91 |

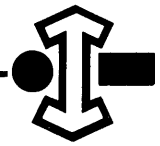
| Figure | Title | Page |
|---|---|---|
| 3-1 | Over-all Block Diagram of Arithmetic Section | 3-2 |
| 3-2 | Typical Stage of A Register | 3-4 |
| 3-3 | Relation of Borrow Pyramid to A and X Registers | 3-14 |
| 3-4 | Shift Control | 3-17 |
| 3-5 | Iterative Sequence | 3-20 |
| 3-6 | Multiply Step | 3-25 |
| 3-7 | Divide Step | 3-29 |
| 3-8 | Typical Relation of Coefficients in Addition | 3-37 |
| 3-9 | Location of Double-Length Product for Floating Multiply | 3-39 |
| 3-10 | Formation of the Logical Product | 3-42 |
| 3-11 | Register Sensing Networks | 3-44 |
| 3-12 | Arithmetic Faults | 3-46 |
| 4-1 | Logical Divisions of the Storage Section | 4-2 |
| 4-2 | Magnetic Core Matrix | 4-4 |
| 4-3 | Memory Plane Stack | 4-6 |
| 4-4 | Typical Hysteresis Diagram | 4-7 |
| 4-5 | Voltage on Sense Winding as a Result of Read Drive | 4-8 |
| 4-6 | Memory Board | 4-10 |
| 4-7 | Distribution of Odd and Even Memory Plane Assemblies Among Chassis of Computer | 4-11 |
| 4-8 | Intersection of H and V Wires in a Simple Plane Assembly | 4-12 |
| 4-9 | Connection of Drive Lines and Diversion Lines to the Odd or Even Portion of the Stacks | 4-14 |
| 4-10 | Selection of Drivers and Diverters by $S^1$ or $S^2$ Registers | 4-16 |
| 4-11 | S-1 and S-2 Registers | 4-18 |
| 4-12 | Typical Horizontal or Vertical Drive Circuit | 4-19 |
| 4-13 | Typical Horizontal or Vertical Diversion Circuit | 4-21 |
| 4-14 | Path of Sense Wire Through a Four-Core Matrix | 4-25 |
| 4-15 | Typical Stage of the Bit Plane Control | 4-26 |
| 4-16 | Path of Inhibit Wire Through a Four-Core Matrix | 4-28 |
| 4-17 | Basic Pulse Sequence for Storage Reference | 4-30 |
| 4-18 | Storage Reference Circuit | 4-32 |
| 4-19 | Sequence of Pulses Generated by Control Sequence | 4-33 |
| 4-20 | Timing Pulse Generator of Even Storage Sequence Control | 4-35 |
| 4-21 | Drive and Pulse Generator of Even Storage Sequence | 4-39 |
| 4-22 | Fault Detector of Even Storage Unit | 4-43 |
| 4-23 | Drive Generator (Card Type 51) | 4-45 |
| 4-24 | Diverter (Card Type 52) | 4-47 |
| 4-25 | Selector (Card Type 53) | 4-48 |
| 4-26 | Current Source (Card Type 54) | 4-49 |
| 4-27 | Inhibit Generator (Card Type 55) | 4-51 |
| 4-28 | Sense Amplifier (Card Type 56) | 4-52 |




| Figure | Title | Page |
|---|---|---|
| 5-1 | Over-all Logic Diagram | 5-2 |
| 5-2 | Individual Bit Significance of the External Function Instruction | 5-7 |
| 5-3 | Data Input to the Least-Significant Stage of X Register | 5-9 |
| 5-4 | Data Output from the Least-Significant Stage of X Register | 5-11 |
| 5-5 | External Function Code Output from the Lowest Stage of X Register | 5-12 |
| 5-6 | Auxiliary Scanner | 5-15 |
| 5-7 | Auxiliary Request | 5-17 |
| 5-8 | Timing Diagram - Exchange of Control Signals for One Input Word | 5-20 |
| 5-9 | Channel 1 Ready Resync Circuit | 5-21 |
| 5-10 | Timing Diagram - Exchange of Control Signals for One Output Word | 5-23 |
| 5-11 | Channel 2 Resume Resync Circuit | 5-24 |
| 5-12 | Buffer Channel and Arithmetic Fault Interrupt Generation | 5-27 |
| 5-13 | Sense Return Signal | 5-29 |
| 5-14 | External Function Sequence and Associated Circuits | 5-31 |
| 5-15 | Buffer Control Word for Channel J | 5-33 |
| 5-16 | External Function and Auxiliary Sequences for Instruction 74. 1-6 | 5-33 |
| 5-17 | Auxiliary Sequence for Buffer Operation | 5-38 |
| 5-18 | Advance Clock Resync Circuit | 5-41 |
| 5-19 | Auxiliary Sequence for Advance Clock Operation | 5-43 |
| 5-20 | Interrupt Sources | 5-44 |
| 5-21 | Auxiliary Sequence for Interrupt Operation | 5-48 |
| 5-22 | Search and Transfer Sequence | 5-50 |
| 6-1 | External Function Translator | 6-5 |
| 6-2 | Sense Circuit Console Equipment | 6-7 |
| 6-3 | Input Distributor, Simplified Diagram | 6-9 |
| 6-4 | Character Orientation in Registers and on Paper Tape | 6-10 |
| 6-5 | Output Distributor, Simplified Diagram | 6-11 |
| 6-6 | Seven-Level Paper Tapes | 6-13 |
| 6-7 | Reader Control Circuit | 6-14 |
| 6-8 | End of Tape Circuit | 6-16 |
| 6-9 | Punch Control Circuit | 6-18 |
| 6-10 | Keyboard Control Circuit | 6-23 |
| 6-11 | Carriage Return Interrupt Circuit | 6-25 |
| 6-12 | Typewriter Control Circuit (Output) | 6-26 |
| 7-1 | Motor Generator Set | 7-2 |
| 7-2 | Power Control Cabinet, Front Outside View | 7-4 |
| 7-3 | Power Control Cabinet, Front Inside View | 7-5 |
| 7-4 | Main Power Distribution | 7-7 |
| 7-5 | Main Cabinet Power Distribution | 7-8 |
| 7-6 | Typical DC Power Supply on Chassis | 7-10 |
| 7-7 | Simplified Thermostat and Interlock Circuit | 7-14 |

LIST OF TABLES


| Table | Title | Page |
|---|---|---|
| 1-1 | Arithmetic Properties of Register | 1-12 |
| 1-2 | Counting Sequence for Three-Stage Counter | 1-32 |
| 1-3 | Description of Standard Card Types | 1-36 |
| 1-4 | Glossary of Boolean Symbols | 1-39 |
| 1-5 | Boolean Identities | 1-40 |
| 1-6 | Proof of Identity 6 | 1-41 |
| 2-1 | Designation of Base Execution Address | 2-8 |
| 2-2 | Control Sequences | 2-20 |
| 2-3 | Examples of Indirect Addressing | 2-37 |
| 2-4 | Commands for Zero Address Sequence | 2-41 |
| 2-5 | Commands for Read Operand Sequence | 2-44 |
| 2-6 | Commands for Write Operand Sequence | 2-48 |
| 2-7 | Commands for Search and Transfer Sequences | 2-53 |
| 2-8a | Iterative Sequence, Main Sequence | 2-57 |
| 2-8b | Iterative Sequence, Execute Multiply Step | 2-59 |
| 2-8c | Iterative Sequence, Execute Divide Step | 2-59 |
| 2-8d | Iterative Sequence, Execute End Correction | 2-60 |
| 2-8e | Iterative Sequence, Execute Floating Point | 2-61 |
| 2-8f | Iterative Sequence, Execute Round, Normalize, Final Assembly | 2-63 |
| 2-9 | External Function Sequence | 2-65 |
| 2-10a | Auxiliary Sequence, Buffer | 2-70 |
| 2-10b | Auxiliary Sequence, Interrupt | 2-71 |
| 2-10c | Auxiliary Sequence, Advance Clock | 2-71 |
| 2-11 | Typical Use of Return Jump Instruction | 2-82 |
| 3-1 | Decimal and Binary Equivalents | 3-7 |
| 3-2 | Binary Subtraction | 3-10 |
| 3-3 | Basic Steps in Floating Add | 3-36 |
| 3-4 | Basic Steps in Floating Subtract | 3-38 |
| 3-5 | Basic Steps in Floating Multiply | 3-39 |
| 3-6 | Basic Steps in Floating Divide | 3-40 |
| 4-1 | Sources of Addresses for Storage References | 4-31 |
| 4-2 | Timing Pulse Generator of Even Storage Sequence Controls: Complete Counter Cycle | 4-36 |
| 4-3 | Timing Pulse Generator; Source of Timing Pulses | 4-37 |




| Table | Title | Page |
|---|---|---|
| 5-1a | Line Assignment of a Given Cable Group - Data Lines | 5-4 |
| 5-1b | Line Assignment of a Given Cable Group - Control Lines | 5-5 |
| 5-2 | Exchange of Control Signals for One Output Word | 5-20 |
| 5-3 | Exchange of Control Signals for One Input Word | 5-22 |
| 5-4 | External Function Codes | 5-26 |
| 5-5 | Commands for Buffer Operation | 5-39 |
| 6-1 | Select Codes for Console Equipment | 6-4 |
| 6-2 | Sense Codes for Console Equipment | 6-6 |
| 6-3 | Assembly Mode of Reader Operation | 6-15 |
| 6-4 | Character Mode of Reader Operation | 6-15 |
| 6-5 | Punch Operation | 6-19 |
| 6-6 | Typewriter Codes | 6-21 |
| 6-7 | Keyboard Operation | 6-22 |
| 6-8 | Carriage Return Interrupt | 6-24 |
| 6-9 | Typewriter Operation (Output) | 6-27 |