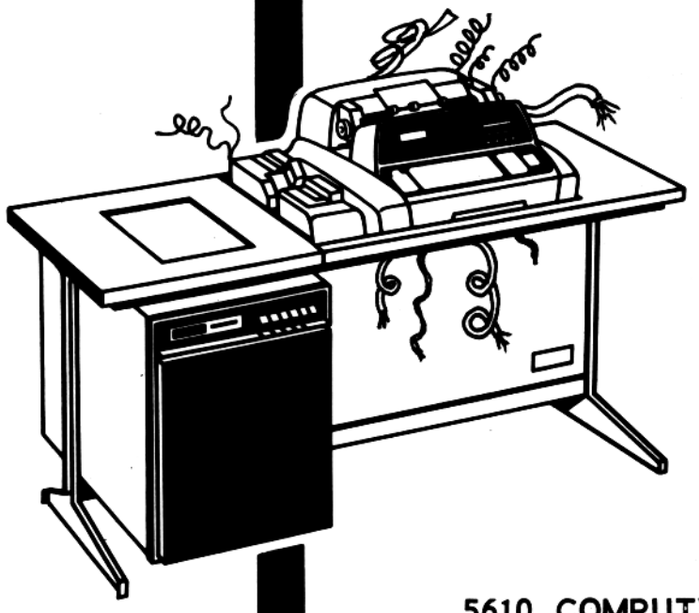
Friden



5610 COMPUTYPER

SERVICE MANUAL

Friden

FRIDEN, INC · DIVISION OF THE SINGER COMPANY

SAN LEANDRO · CALIFORNIA · 94577

TP - 161SL

COPYRIGHT 1968 BY FRIDEN, INC.
A Division of The Singer Company

PREFACE

This manual covers the 5610 COMPUTYPER* data processor by FRIDEN*. The greater part of it is intended to serve as a supplement to the training received at one of the FRIDEN service training centers. It is not meant to be a self-teaching manual. The information contained in the latter part of this manual is essentially helpful to the Customer Service Representative when maintaining or servicing the 5610.

In the preparation of this manual, it is assumed that the trainee has prior knowledge of basic electricity and electronics, basic semiconductor electronics, and basic computer logic, including flip-flops, AND and OR gates, etc. In addition, basic information pertaining to delay lines, semiconductors, logic, etc., is available in the FRIDEN Electronic Course, so it is not duplicated in this manual.

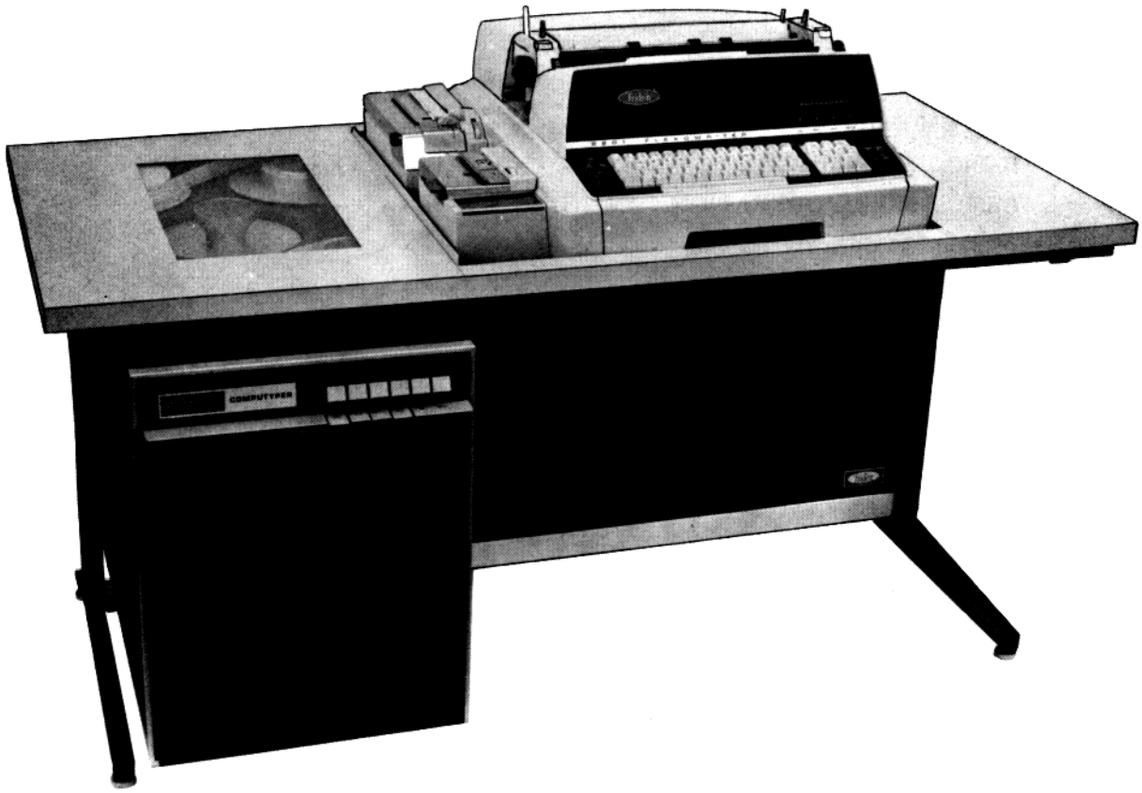
It is also assumed that the trainee has had prior training in the operation and adjustment of the 2201 FLEXOWRITER* automatic writing machine, the 2214 SELECTADATA* selective reader, and other auxiliary units. Only those aspects of these machines that are unique to their operation with the 5610 are covered in this manual.

As with any machine of this type, refinements in circuitry, new optional features, and other changes may be introduced in the future. These changes will first be introduced to the CSReps in the field through 5610 Service Letters, and later, if necessary and applicable, through addendums or revised pages in this manual.

*A Trademark of FRIDEN, Inc.

TABLE OF CONTENTS

	Page
SECTION 1, GENERAL INFORMATION	
SYSTEM STRUCTURE	1-1
BASIC PROGRAMMING CONCEPTS	1-4
BASIC SYSTEM OPERATION	1-4
Load Mode	1-4
Translate Mode	1-5
Search Mode	1-5
Retrieve Mode	1-7
Execute Mode	1-7
Special Flexowriter Functions	1-8
INTRODUCTION TO LOGIC	1-8
True and False	1-9
Flip-flops	1-9
Inverters	1-12
Gates	1-13
Relays and Switches	1-16
COMPONENT LOCATIONS	1-16
Diodes	1-16
Hex Packs	1-17
Other Components	1-18
Pin Numbers	1-18
GLOSSARY OF SIGNAL NAMES	1-19


COMPUTYPER

SECTION 1, GENERAL INFORMATION

The 5610 COMPUTYPER is a highly-flexible stored program data processor which uses the delay line technique for program and data storage. Program storage is completely independent of data storage. Program storage can accommodate up to 1198* program characters; data storage consists of 60 storage registers, each capable of containing 13 digits plus sign.

A. SYSTEM STRUCTURE (See Figure 1-1)

There are five basic units making up the 5610. They are the 2205 FLEXOWRITER automatic writing machine, the electronic unit, the delay line stackup, the power supply, and the desk, which contains a built-in horizontal tape transport system. In addition, certain FRIDEN auxiliary input and auxiliary output devices, which operate in conjunction with the writing machine, can be connected to the system.

The primary input/output device used with the 5610 electronic unit is the 2205 FLEXOWRITER writing machine. The 2205 is basically a model 2201 FLEXOWRITER writing machine with the following changes:

1. Three additional relays (KEB-1 & 2, KKM).
2. Printed circuit board added to right rear side frame. (This board contains a fourth relay, KCL, which is a reed relay.)
3. Additional selector contacts.
4. STC-5 cam-operated contact added to the translator.
5. Tab, Carriage Return, Space, Period, and Dash keylevers not interlocked.
6. All relay coils, clutches, bit magnets, and cam-operated contacts have either diode or RC network arc suppression added.
7. Wiring necessary for the addition of the above items.

Standard features of the 2205 include four selector relays, twelve field switches, edge card punch and reader, and 20" carriage. A single-feed tab card reader, a front-feed posting carriage, and other optional features are also available.

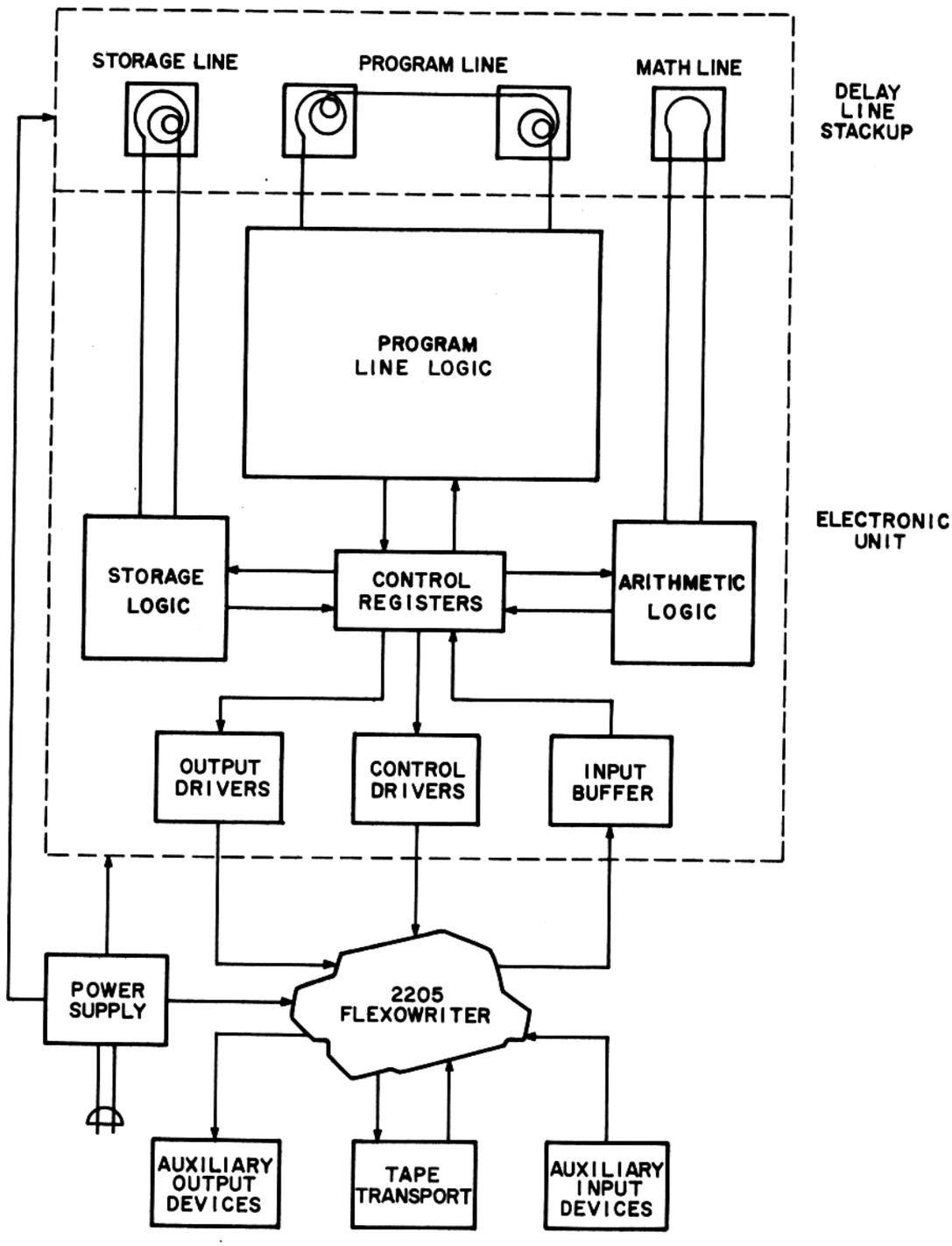
In addition to its primary function as input/output unit for the electronic unit, the 2205 is also used to prepare the program tape. This tape is then placed in the 2205 reader to load the program into the electronic unit.

The electronic unit consists basically of 28 printed circuit cards and their associated wiring. The heart of the electronic unit is the program line logic, which governs the action of all other parts of the system. It is used first to "load" the program onto the program delay line. It then selects the instructions, in sequence, from the program line, and initiates the logic operations and system control necessary to execute the machine function. Once the operator has selected a particular portion of the program, the program line logic is in complete command of the entire system until that part of the program is complete.

Two other sections of the electronic unit also operate in conjunction with delay lines. The arithmetic line logic, when directed so by the program line logic, performs all of the 5610 mathematical functions, with automatic decimal point alignment. The storage line logic selects the proper storage when the instructions "Store", "Recall", or "Accumulate" are called out by the program line logic.

The input buffer accepts data from the reader or the keyboard of the 2205 and decodes it into the proper signal levels for use by the program line logic (during program loading) and by the arithmetic line logic (during entry).

*See Service Letter #60.



5610 SIMPLIFIED SYSTEM BLOCK DIAGRAM

FIGURE 1-1

The output drivers and control drivers convert the electronic logic signals into "power" pulses sufficient to drive the writing machine. The output drivers control the 2205 translator, and the control drivers operate the "special" functions, which control readers, punches, and non-print.

The control registers are a group of shift registers which route data between delay lines. They also develop logic outputs for use in program control, perform the parity check during entry, and provide the outputs to the 2205.

The delay line stackup consists of four electro-mechanical delay lines similar to the type used in the EC-130 and EC-132 electronic calculators. The program line uses two of these delay lines connected end-to-end, effectively making one long delay line. Thus, the four lines actually function as three.

The program line has a total delay time of approximately twelve milliseconds (.012 second), so that a pulse fed into one end will emerge from the other end twelve milliseconds later. This gives the effect of storing the pulse for twelve milliseconds. With the timing arrangement used in the 5610, the program line is capable of storing a total of 1119 program characters. Each alpha-numeric character is made up of a combination of 6 "bits", corresponding to the binary coded decimal system used in FRIDEN systems equipment. The 5-bit and the 8-bit are eliminated in order to conserve space on the program line (however parity is still checked electronically).

A second delay line, called the storage line, contains the sixty numeric storages available for use in the 5610. Each storage contains a maximum of thirteen digits (usually 8 left-of-decimal and 5 to the right) plus the algebraic sign. Each digit is made up of a combination of four bits, again using the BCD system. This delay line is constructed so that it takes about 6 milliseconds to recirculate the data.

The third delay line, termed the arithmetic line, contains the five working registers, called R0, R1, R2, R3, and R4. All of the 5610 arithmetic operations are performed on these registers. Data organization within these registers is the same as in the storage registers, each one having a capacity of thirteen digits plus sign. The arithmetic line is shorter than the others, having a delay time of approximately 350 microseconds.

The power supply, located in back of the electronic unit, operates on 115 volts ac and delivers the following voltages:

1. +50 volts dc, unregulated
2. 1 volt ac, "
3. +24 volts dc, "
4. -24 volts dc, "
5. +6 volts dc, regulated
6. +11 volts dc, "
7. -11 volts dc, "
8. 115 volts ac (2205 Supply)

The 2205 contains its own 100-volt dc power supply, but it must receive its line voltage from the 5610 power supply.

The desk is used to house the power supply, the delay line stackup, and the electronic unit. It also has a horizontal tape transport system built into the top just to the left of the writing machine. This system stores 1000 feet of tape for the punch, and rewinds 500 feet for both the punch and the reader. The reader unwinder has a 500 foot capacity. The tape transport is made accessible by sliding part of the desk top to the left. The electronic unit, delay line stackup, and power supply are located to the left of the writing machine, below the tape transport.

Any auxiliary input or auxiliary output device used on a standard 2201 FLEXOWRITER writing machine, except the 2217 Data Selector, is available for use with the 5610. However, these units must be specially modified. After modification, all auxiliary units except the 2214 SELECTADATA selective reader can still be used with a standard 2201. The 2214 has the capability of high speed (non-printing) entry into the electronic unit.

B. BASIC PROGRAMMING CONCEPTS

Each individual step in the execution of the entire program is referred to as an "instruction". Each instruction is made up of two parts: a "command" and an "address". The command tells the processor what to do (Add, Store, Type, etc.) and the address defines the command further as to its nature or location (Add register 1 to register 2, Store in storage 47, Type a letter "A", etc.).

There are 44 different instructions which the 5610 is able to perform. Each instruction in the program is stored on the program delay line in the form of a "character". For example, when the letter "B" is recognized it will initiate the Subtract function. The letter "F" represents Add, "S" represents Store, and other characters represent the other various commands. A complete list of instructions is presented in Section 2, Programming. The address can contain anywhere from 0 to 15 characters. This will be fully explained in Section 2.

Every instruction in the entire program is not necessarily performed in order. Instead, the program is usually broken up into several sections, called "Sequences". For instance, if a billing application is being programmed, it would not be unusual to have one sequence for the header portion, another for line items, another for discount, sub-total, etc., and still another for net total. The choice as to the number of sequences and the purpose of each is up to the programmer.

Each program sequence on the delay line is headed by a sequence separator code (letter "H"), and is identified by a 1-character title. Sequence "A" would have the characters "HA" on the program line preceding the instructions. When a particular program sequence is to be performed, the 5610 operator merely strikes the 2205 key corresponding to that sequence title, or places the corresponding code in the reader. Once the sequence is selected, the 5610 takes over and executes all of the instructions contained in that sequence, one after the other.

C. BASIC SYSTEM OPERATION

There are six modes of operation in the 5610; Load, Translate, Search, Retrieve, Execute, and Special Flexowriter Functions.

1. LOAD MODE

Load mode (1 mode) is used to load the program into the 5610. Each time the 5610 is turned off, the program and all other data are lost. Since the system cannot operate without a program, one must be loaded each time the machine is turned on. For this reason the 5610 automatically places itself in the Load mode each time it is turned on.

The program can be loaded onto the program line from either the 2205 reader or keyboard. In an actual application the reader is used almost exclusively to insure that the program is loaded exactly the same every time. Since the program in the paper tape is in the format of "machine language" codes, it is usually loaded in Non-Print; the copy produced by reading this tape in the normal manner is of little value.

The program line has a maximum capacity of 1119 program characters. As each character is loaded it increments a counter in the arithmetic line logic. When this counter reaches 1119, it stops the reader, if it had been running, and locks the keyboard so that no further characters can be loaded.

After the program has been loaded, the 5610 operator depresses the Load switch (non-locking), which takes the system out of the Load mode and places it into the Search mode. At this time, the 5610 is ready to operate, but it is waiting for further directions as to which sequence of the program is to be executed.

If it is desired to load another program after the 5610 has been operating, the operator merely depresses the Load switch again. This places the system back into 1 mode but does not destroy data in storage, so that another program may be loaded and executed to further process the data accumulated by the first program.

2. TRANSLATE MODE

The Translate mode (2 mode) is used to convert English-language statements into machine-language codes punched into paper tape. Saying this another way, the 5610 programmer types the program on the writing machine in English, and a tape is produced which contains the program in machine language. This tape, called the object code tape, is then run through the reader in 1 mode to load the program onto the program line.

a. SWIFT TRANSLATOR. The instrument which enables the 5610 to do this is the SWIFT translator (for SoftWare-Implemented Friden Translator), a standard English-language program tape, developed by the Systems Engineering Department. This tape is loaded into the 5610 in the Load mode. The programmer then places the 5610 into the Translate mode, and as he enters the English-language program, the SWIFT translator converts these statements into machine-language codes (object codes) and sends them to the 2205 punch to prepare the object tape.

b. SOURCE TAPE. In most cases, the English-language program is not typed directly on the keyboard, but is prepared on tape earlier to make sure it is error-free. This tape is referred to as the source tape.

c. TRANSLATE SWITCH. After the SWIFT translator has been loaded, the Load switch must be operated to take the 5610 out of the Load mode. Then the Translate switch may be depressed. The Translate switch locks down and the 5610 stays in this Translate condition until the switch is restored manually. Note that operation of the Translate switch alone will NOT take the 5610 out of Load: the Load switch must be operated in order to disable the Load condition. (The Reset switch will also take the system out of Load.)

d. STEPS IN TRANSLATING. The sequence of steps to be observed in translation is as follows (See Figure 1-2):

(1) Develop the program and prepare the source tape on the 2205 or on an off-line 2201.
(2) Load the SWIFT translator into the 5610.
(3) Depress the Load switch to disable the Load condition, and depress the Translate switch.
(4) Enter the English-language program (source tape) through the reader, to produce the object tape.
(5) Revert to the Load mode (lift up Translate switch and depress Load switch) and load the new object tape.
(6) Disable Load condition (depress Load switch once more) and execute the program.

The customer will not normally operate the 5610 in the Translate mode. Once the programmer has produced the object tape, the customer merely loads the object tape into the processor each morning and proceeds to execute the program.

All translation does not take place in 2 mode. Rather, the translating of each program instruction BEGINS in 2 mode, and then jumps to 3 mode, 4 mode, and often to 5 mode and 6 mode. This jumping through modes is completely automatic, and not under the control of the operator.

3. SEARCH MODE

The basic purpose of Search mode (3 mode) is to find the next instruction to be executed. Other FRIDEN equipment, such as the 5010 and 6010, have the individual instructions wired in a patchboard, and it is a simple process to step from one instruction to the next. In the 5610, however, the program is continually recirculating on a delay line, and an instruction can only be sensed as it comes off of the line. 3 mode monitors the output of the delay line, and when the desired instruction emerges, a jump to 4 mode occurs.

Whenever an instruction to branch is sensed, the portion of the program to which the branch is to be made is also found through a 3 mode search.

a. SEQUENCE SEPARATOR SEARCH. After the operator has loaded the program and disabled the Load mode, the 5610 is automatically placed in 3 mode waiting for a sequence title to be entered. Once the entry is made, a search is conducted to find the selected sequence; each

sequence separator is searched out, and the next code (the sequence title) is compared to the code entered via the writing machine. When the proper sequence is located, a "Marker" (tape feed code) is written onto the program line immediately behind the sequence title.

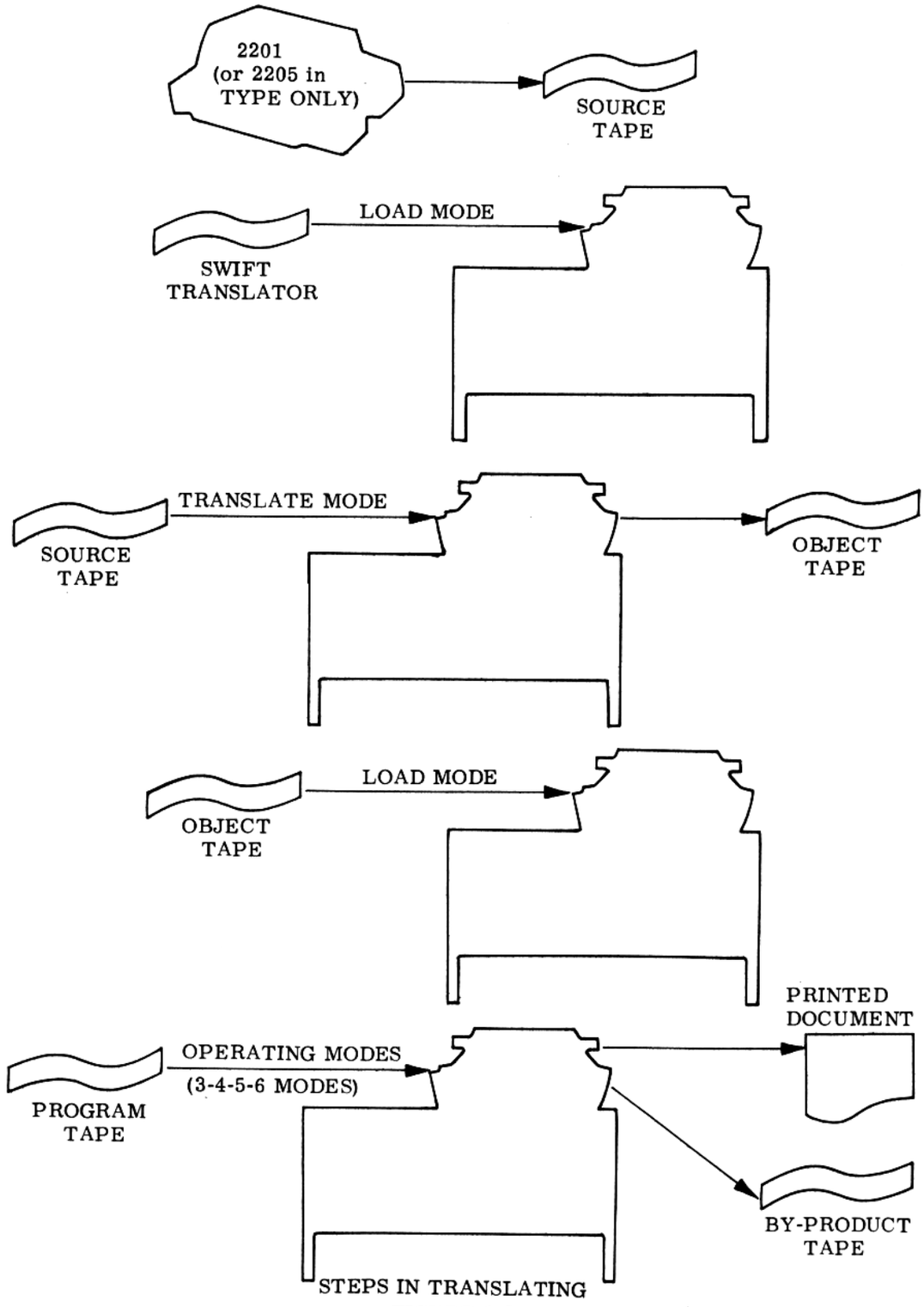


FIGURE 1-2

b. MARKER SEARCH. When the first instruction in the sequence is sensed, the marker is moved to a position behind the first command. After this occurs, the processor automatically leaves 3 mode and goes on to execute the instruction.

When the instruction is completed, another 3 mode search is conducted for the marker, and the instruction following the marker is sensed. The marker is again moved to a position behind the instruction just sensed. This instruction is then executed and the entire process repeats, until the last instruction in the sequence has been performed. When this occurs, the marker is destroyed, and the sequence light turns on, indicating completion of the sequence.

c. RESUME SEARCH. When a conditional branch instruction is executed, the marker is not written onto the line behind the branch command, but behind the corresponding RESUME code instead. If the conditional branch instruction says "branch to Resume C", 3 mode searches out each Resume command and compares the next code (the "address") with the title dictated by the conditional branch instruction. When Resume C is located, the marker is placed on the program line immediately after the "C".

d. VARIATIONS IN SEARCHES. The three types of searches just covered can also take place during Unconditional Branch conditions, at which time operation is slightly modified (although the basic concept does not change). When the Unconditional Branch instruction is executed, another Sequence Separator search is conducted to find the sequence specified by the Branch instruction. When the new sequence is found, the branch marker (2-4-7) is inserted into the program instead of the normal marker. While in the Unconditional Branch state, the normal marker is ignored and the branch marker is moved down the line as the instructions are executed.

4. RETRIEVE MODE

The Retrieve mode (4 mode) takes over where 3 mode ends. After 3 mode locates the marker, 4 mode retrieves the next instruction from the program line. 4 mode then analyzes the command to determine what following steps need to be taken.

a. PROGRAM ALTERATION INSTRUCTIONS. Instructions such as Unconditional Branch, Return, Positive Branch, etc., all move or change the marker without affecting data. These instructions usually are partially executed in 4 mode and then cause a jump back to 3 mode to finalize the instruction.

b. NO-OP INSTRUCTIONS. Instructions such as Negative Branch when conditions are positive, etc., are recognized as "No-Operation" instructions in 4 mode. When 4 mode retrieves a "No-Op", a jump back to 3 mode is initiated to search out the next instruction.

c. 6 MODE INSTRUCTIONS. The Special Flexowriter Functions, listed later in this section, and storage oriented instructions using the Address Modifier feature are executed in 6 mode. When one of these instructions is retrieved, 4 mode provides the signals necessary to achieve 6 mode.

d. 5 MODE INSTRUCTIONS. The great majority of instructions (all except Program Alteration instructions, No-Op's, and Special Flexowriter Functions,) are executed in 5 mode. Different instructions must enter 5 mode in different places, so after retrieving these instructions, 4 mode decodes them and provides the signals needed to cause the jump to the correct phase in 5 mode.

5. EXECUTE MODE

The Execute mode (5 mode) is the mode in which most instructions are finally performed. All instructions dealing with the arithmetic and storage lines, including entry and printout instructions, are executed in 5 mode. All writing machine functions except those to be covered shortly under 6 mode are also carried out in 5 mode. All printout and 2205 functions executed in 5 mode reach the writing machine through the output drivers, and drive the translator.

When an instruction is completed in 5 mode, a jump back to 3 mode occurs so that a search for the next instruction may be initiated.

6. SPECIAL FLEXOWRITER FUNCTIONS

Special Flexowriter Functions, also referred to as "direct pick" Flexowriter functions, are executed in 6 mode. This group includes the following eight commands:

(1) Readers Off
(2) Non Print
(3) Print Restore
(4) Punch 1 On
(5) Punch 2 On
(6) Punches Off
(7) Reader 1 On
(8) Reader 2 On

When one of these instructions is executed, the function is accomplished without cycling the 2205 translator, by picking and/or dropping relays directly, and without regard to the program panel patching. As with 5 mode, when an instruction is completed in 6 mode, a jump back to 3 mode occurs.

6 mode is also used by the "Address Modifier" special feature.

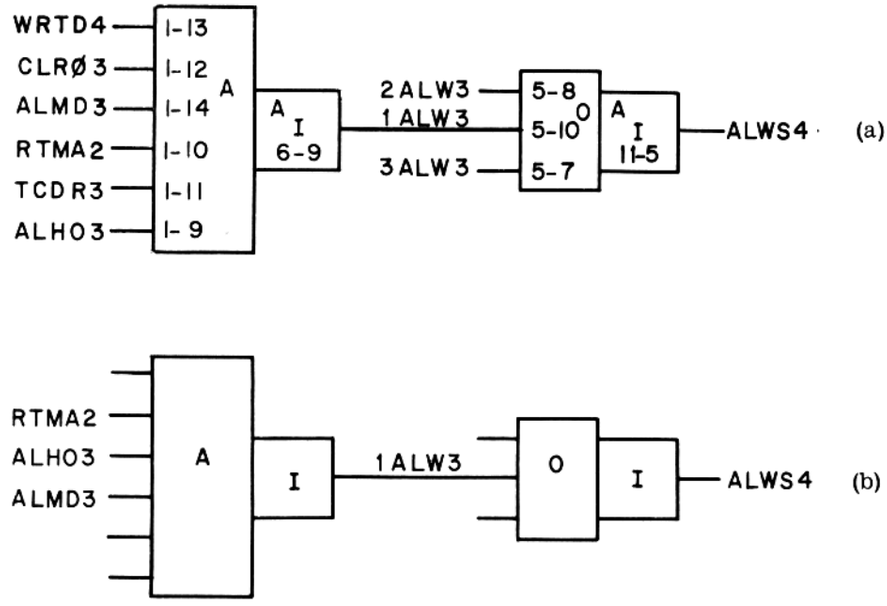
7. TYPE ONLY

With the Type Only switch depressed, the 2205 operates the same as a 2201. No processor functions occur. Operating the Type Only switch does not affect program or data storage.

D. INTRODUCTION TO LOGIC

There are basically two types of study material available to those learning the 5610: the service manual and the logic drawings. The service manual is, for the most part, narrow in scope, dealing only with a specific phase of 5610 operation at a given time. On the other hand, the logic drawings must be all inclusive. They must show every flip-flop, gate, etc., whether it is applicable to the operation being studied or not. They must also show all component locations, terminal points, etc., because these drawings will later be used for troubleshooting. The service manual need not show component locations, redundant signal names, etc., for these would only complicate the learning process. (In certain places, however, logic drawings have been duplicated in this manual to save drafting time.)

Figure 1-3 (a) shows a typical drawing taken directly from the logic prints. Figure 1-3 (b) shows the same logic elements as they are depicted in this manual. In Figure 1-3 (b) all notations not relevant to the subject under discussion are eliminated.



COMPARING LOGIC DRAWINGS TO SERVICE MANUAL DRAWINGS

FIGURE 1-3

In the material that follows, reference will occasionally be made to points on which the logic drawings and the service manual differ; at the same time, the information will be presented which the technician must have in order to make the other material more meaningful.

1. TRUE AND FALSE

In almost all cases the two logic states in the 5610 are represented by either +6 volts or 0 volts. 0 volts is the "True" state, or logic 1: +6 volts is the false logic state, or logic 0. In a few cases the false logic state is represented by +11 or +24 volts. Therefore, we can say that 0 volts, or ground, is the True state, and any other voltage that is definitely not 0 represents the False state.

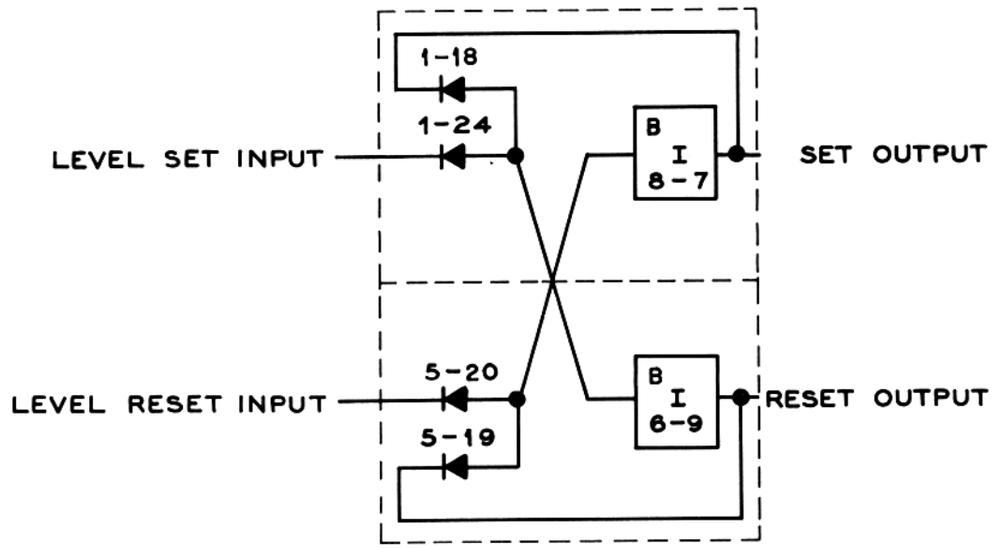
2. FLIP-FLOPS

As noted below, 5610 flip-flops are somewhat different electronically from anything FRIDEN has used in the past. However, from a logic point of view, the operation is the same as any other flip-flop: it will stay in one of two states until some external signal changes it.

a. PHYSICAL CHARACTERISTICS. All 5610 flip-flops are made up of a pair of "cross-coupled" inverters, contained in micro circuit modules. Each module contains six inverters, hence the term "hex pack". The diodes used to couple the inverters are separate components. Therefore, the physical components necessary to make up a flip-flop, exclusive of triggering, are two inverters and two diodes.

b. ELECTRONIC OPERATION. The electronic operation of the "hexinverter" is covered in Section 8, Electronic Circuits. For the present it is sufficient to say that the inverter changes the logic state of a signal: a true signal on the input will cause a false signal on the output, and a false input will cause a true output.

Refer now to Figure 1-4 which schematically illustrates a typical flip-flop. Assume a true signal on the "Level Set" input. This causes a true input to inverter B6-9, whose output, now false, is coupled through diode 5-19 to the input of inverter B8-7. The output of this second inverter, now true, is coupled, through diode 1-18, to the input of the first inverter, B6-9, so that when the true signal on the Level Set input is removed, the flip-flop will remain in this state.



CROSS-COUPLED INVERTER FLIP-FLOP

FIGURE 1-4

The output of inverter B6-9 is considered the reset output, and the output of inverter B8-7 is the set output. In the present state the flip-flop is "set".

Since the input to inverter B6-9 is now clamped at 0 volts (true), any voltage change on the Level Set input will have no effect. However, if the signal on the Level Reset input were to become true, diode 5-20 would become forward-biased and the input to inverter B8-7 would become true. Its output, now false, would cause the output of inverter B6-9 to become true, and the flip-flop would be reset. Any additional signal change on the reset input would now have no effect since the input to inverter B8-7 is now clamped true through diode 5-19.

In Figure 1-4, the "5-19", "1-24", "B8-7", etc., refer to component locations on the printed circuit card. The interpretation of these locations is explained later in this section.

c. NAMES. Most flip-flops in the 5610 have names that are associated with their functions. These names are abbreviated for simplicity. The Data Operation Stop flip-flop, shown in Figure 1-5, is abbreviated "DOST". In all cases a flip-flop will be set when its named function is occurring. In this case, the DOST flip-flop will be set whenever an arithmetic function (data operation) is not occurring, for whenever DOST is set it stops data operation.

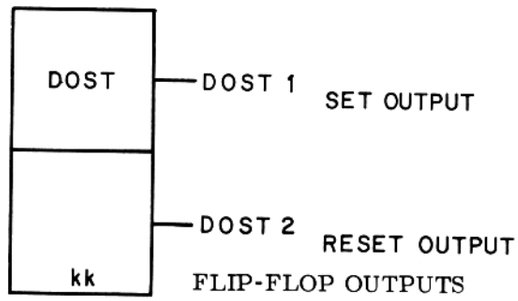
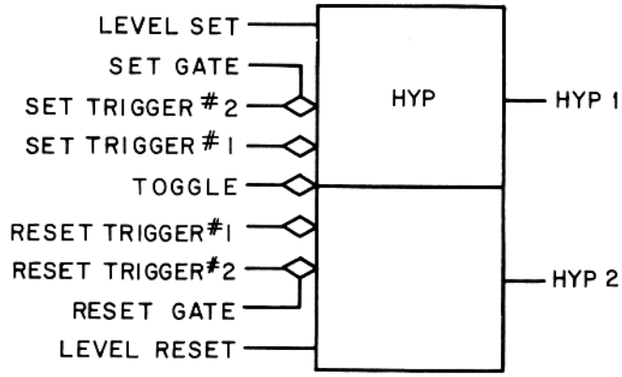


FIGURE 1-5

d. SYMBOLS. Figure 1-5 shows the symbol used to depict a 5610 flip-flop. The small letters at the bottom denote the printed circuit card on which the particular flip-flop is located.

(1) Outputs. Still referring to Figure 1-5, note that the set output of a flip-flop is labeled with the abbreviated flip-flop name followed by the digit 1. The reset output is labeled with a 2. This enables the signal to be easily identified when it is used on another drawing where the flip-flop is not shown.

(2) Inputs. Figure 1-6 shows a hypothetical flip-flop having all possible types of inputs. More than one of any type of input is possible. A true signal on the Level Set input will set the flip-flop, causing HYP1 to become true and HYP2 to become false. A true signal on the Level Reset input will reset the flip-flop, causing HYP1 to become false and HYP2 to become true.



FLIP-FLOP INPUTS

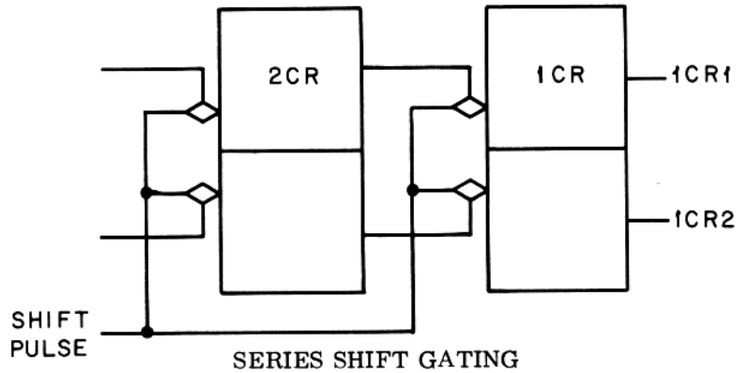
FIGURE 1-6

All other inputs require a changing signal to affect the flip-flop. In all cases the signal must change from false to true. A true-going signal on Set Trigger #1 will set the flip-flop: a true-going signal on Reset Trigger #1 will reset it. A true-going signal on the Toggle input will change the flip-flop from its prior state: if it was set it will become reset, and vice versa.

A true-going signal on Set Trigger #2 will set the flip-flop if the signal on the Set Gate is true. The signal on the Set Gate must be true prior to the time the transition from false to true takes place on Set Trigger #2. If the Set Gate signal is false the changing signal on Set Trigger #2 will have no effect. The same rules apply to Reset Trigger #2 and the Reset Gate.

In all cases except that of the Toggle input, a second signal will have no effect. For instance, if the flip-flop is already set, another signal which would ordinarily set it will now be ignored.

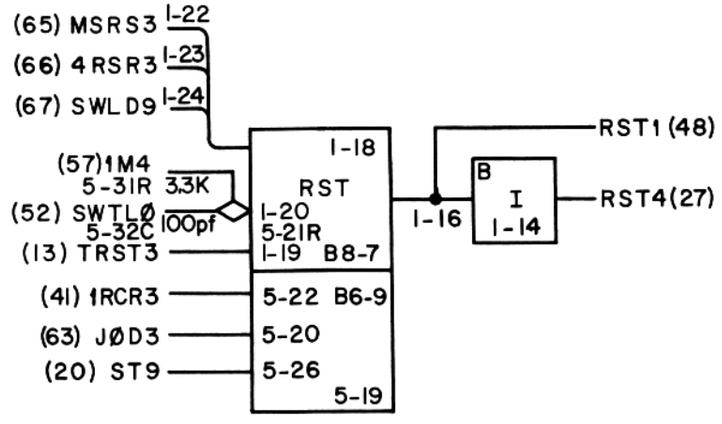
A modified version of these gatings is shown in Figure 1-7. This type of gating is used extensively in series shift registers and series shift counters, but it can also be used on individual flip-flops. If a true-going shift pulse should occur, the 1CR flip-flop will assume the state previously held by 2CR, and 2CR will assume the state determined by the signals on its gates. If 2CR had been set before the shift pulse, 1CR would become set, if 2CR had been reset, 1CR would become reset. This type of gating is designed so that each flip-flop will always have one gate signal true and the other one false. It is possible for any of these flip-flops to have other inputs, as previously described, in addition to those shown in Figure 1-7.



SERIES SHIFT GATING

FIGURE 1-7

A few flip-flops in the 5610 have so many inputs that it would be difficult to draw the flip-flop large enough to have all inputs shown in the same manner as they are shown above. In these cases the inputs are drawn "cabled" as shown in Figure 1-8.



"CABLED" INPUTS

FIGURE 1-8

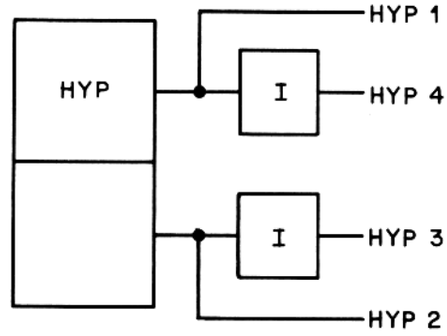
TP-161SL

Figure 1-8 also illustrates the method used to denote component locations on the logic drawings. Compare Figure 1-8 to Figure 1-4 and note how the inverter and cross-over diode locations are transferred from the schematic to the logic drawing. For the sake of simplicity, only one set input was shown in Figure 1-4 the other input diodes are actually connected (anode-to anode) to diode 1-24. The numbers in parentheses denote plug pin locations between the printed circuit card and the mother board.

3. INVERTERS

In addition to their use in flip-flops, hexinverters are also used in conjunction with diode gates and to invert the logic states of individual signals.

a. SYMBOLS. When an inverter is not a part of a flip-flop, its output signals are always assigned either a 3 or a 4. Figure 1-9 again shows the hypothetical flip-flop, this time with its outputs applied to inverters. When the set side of a flip-flop is inverted, it is assigned the digit 4 to distinguish it from the other signals. When the reset side is inverted it is assigned the number 3. Note that when all odd-numbered signals have one logic state, all even-numbered signals will have the opposite logic state.

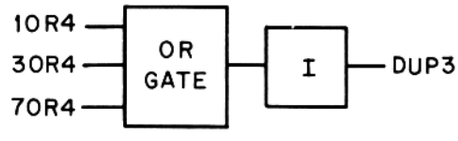


INVERTER OUTPUTS

FIGURE 1-9

If the HYP4 signal were further inverted, it would be assigned the number 3 again. If this were to happen, however, the name would be changed to HYPX3, or something similar, so that it would not be confused with the other HYP3 signal.

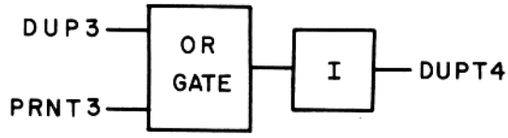
Inverter inputs are also developed by combining two or more flip-flop signals in a gate. In this case the inverter output signal will be assigned the number 3 or 4, but its name will usually be derived from what it does. This is illustrated in Figure 1-10. The first three signals represent inverted flip-flop signals. If the operations register contains the 1-3-7 code, all three inputs will be false and the input to the inverter will be false, causing its output to be true. This shows how the DUPlicate signal, DUP3, is developed when the Duplicate command is in the operation register. If the operation register did not have the 1, 3, and 7 flip-flops set, one of the inputs would be true. This would apply a true signal to the input of the inverter and DUP3 would be false.



DEVELOPMENT OF DUP3

FIGURE 1-10

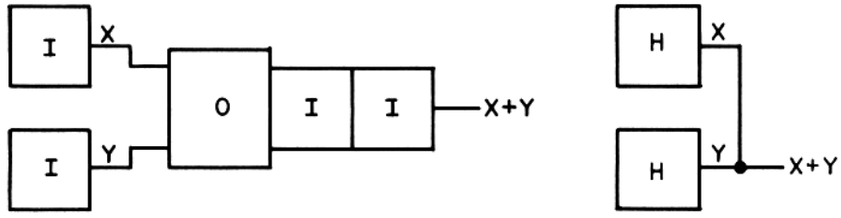
This signal may then be combined with other signals in still another gate, the output of which may have a different name but still have either the number 3 or 4. Figure 1-11 shows DUP3 and PRNT3 (PRiNT) as inputs to an OR gate producing DUplicate or PrinT, DUPT4. If either DUP3 or PRNT3 are true, the input to the inverter will be true and its output will be false. The DUPT4 signal will then be used to control operations common to both Duplicate and Print. DUPT4 could be further inverted to produce DUPT3 if necessary.



DEVELOPMENT OF DUPT4

FIGURE 1-11

b. TYPES OF INVERTERS. There are two types of inverters used in the 5610; "I" inverters and "H" inverters. These two types are described in detail in Section 8, Electronic Circuits. From the logic standpoint, the only difference between the two types is that the outputs of two or more "H" inverters can be joined directly together, whereas the outputs of "I" inverters must be isolated. Figure 1-12 illustrates two equivalent circuits using both types of inverters. It is apparent that the use of "H" inverters can result in a definite savings in component usage.



"I" INVERTERS vs "H" INVERTERS

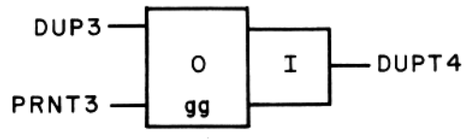
FIGURE 1-12

4. GATES

As explained in Section 8, when the output of a gate is applied to an inverter, it is returned to +6 volts through a resistor which is actually part of the inverter hex pack. For this reason the gate and the inverter are drawn together.

a. "TRUE-OR" GATES. All of the gates in the preceding figures have been shown as OR gates. The electronic operation of these gates is explained in Section 8 of this manual. All gates in the 5610 are physically the same: the diodes are all inserted in the same direction polarity-wise and all are returned through a resistor to +6 volts. Those familiar with the EC-130 would consider these to be AND gates; those familiar with the 5010 or 6010 would think of them as OR gates. In the 5610, a true signal on any of the inputs will cause a true signal on the input to the inverter. All 5610 gates are OR gates for true signals.

Figure 1-11 is reproduced here again as Figure 1-13 to show an example of an OR gate. If either DUP3 OR PRNT3 are true, DUPT4 will be false. Note the small letter inside the gate denoting the card on which the gate is located.

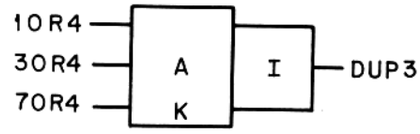


SAMPLE "OR" GATE

FIGURE 1-13

Since all gates feed inverters, a true output from a gate will produce a false output from its inverter. As was noted earlier, if a DUPT3 signal were desired, it would be necessary to further invert DUPT4. This is the only way to obtain DUPT3.

b. "FALSE-AND" GATES. If all gates are OR gates, one might ask how, then, can an "AND" function be accomplished? For instance, how can DUP3 be produced true, using only OR gates, when it is necessary to have the flip-flops 1OR, 3OR, and 7OR all set? The answer is to use the false (inverted) outputs of these flip-flops, as shown previously in Figure 1-10 and again here in Figure 1-14. If 1OR, 3OR, and 7OR are all set, the signals 1OR4, 3OR4, and 7OR4 will all be false. If all three inputs are false, the input to the inverter will be false, and its output will be true.



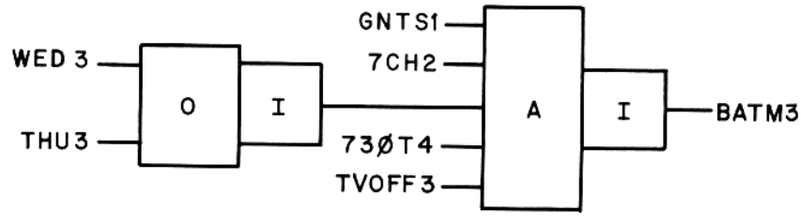
"FALSE-AND" GATE

FIGURE 1-14

With this in mind, it can be stated that any "true-OR" gate can also be considered a "false-AND" gate, meaning that if all inputs are false, the output to the inverter will be false, and the inverter output will be true.

Since all of the gates in the 5610 are of the same type, it would serve no useful purpose to call them all OR gates. Instead, as an aid to understanding and troubleshooting, the 5610 logic drawings have the gates labeled as either true "OR" gates or false "AND" gates.

In all of the examples seen previously the inputs to the gates have been either all 3's or all 4's. So as not to give the wrong impression, Figure 1-15 is presented here to show that any combination of flip-flop and inverter outputs can be used as inputs to a gate.

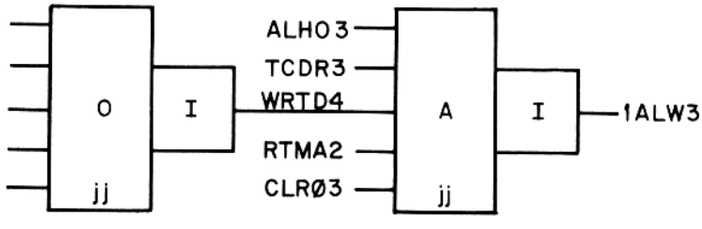


VARIED GATE INPUTS

FIGURE 1-15

Still another aspect of gating should receive some attention before going on. This is the fact that it is not necessarily a requirement that a true signal be developed in order to accomplish a given function. Figure 1-16 shows that if any of five different signals becomes true WRTD4 will be false. When WRTD4 is false it means that the function "Write Direct" is true. It is not even necessary to produce a WRTD3 signal to say that Write Direct is true. For instance, by applying WRTD4 to another gate (see Figure 1-16) which produces 1ALW3 (Arithmetic Line Write), writing onto the arithmetic line in a direct manner can be accomplished merely by making WRTD4 false. WRTD3 is not even necessary.

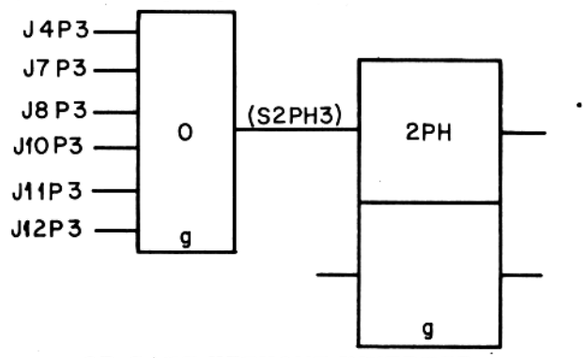
In other words, a function can be made true merely by making its "4" signal false. Again, if WRTD3 were needed for some other reason, it would be developed by further inverting WRTD4.



DEVELOPMENT OF 1ALW3

FIGURE 1-16

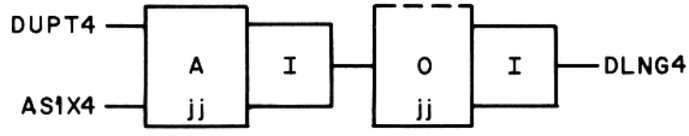
c. GATES WITHOUT INVERTERS. In a few cases, OR gate outputs are used without being inverted. When this occurs the gate will simply be drawn with the inverter eliminated. The example in Figure 1-17 shows that if any of the inputs to the OR gate (Jump to 4 Phase, Jump to 7 Phase, etc.) are true, the 2PH flip-flop will be set.



OR GATE WITHOUT INVERTER

FIGURE 1-17

d. SPECIAL FORMS OF GATE REPRESENTATION. A special form of gate representation used in this manual is illustrated in Figure 1-18 (a). If DUPT4 and AS1X4 (Action Step 1 eXtra) are both false, the inverter output will be true. This will cause DLNG4 to become false, thus making the function "Data LoNG" true. The dotted line shows that there are other inputs to the OR gate which can make DLNG4 false. However, at this time we are only concerned with that input produced by DuPlicate or PrinT during Action Step 1, so the others are not shown.



SPECIAL FORMS OF GATE REPRESENTATION

FIGURE 1-18 (a)

An alternate method of representing a similar situation is shown in Figure 1-18 (b). This time all inputs are shown, but the only ones labeled are those which might change during the operation under consideration. All un-named inputs will be false for the time being.

This same configuration is also used in discussing AND gates, as shown in Figure 1-18 (c). Here again, all inputs not named will remain false during the operation being discussed.

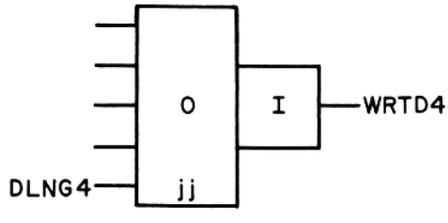


FIGURE 1-18 (b)

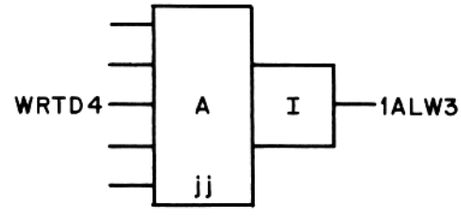
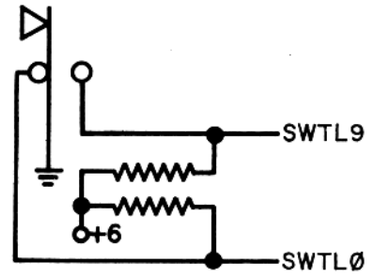


FIGURE 1-18 (c)

TP-161SL

5. RELAYS AND SWITCHES

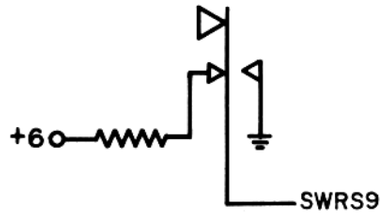
Relay and switch contacts in the 5610 produce signals which are named after the switch function. These signals are assigned the number 0 or 9, depending upon their logic state during the time the switch is operated. If the signal is true while the switch is operated, it is labeled with a 9; if it is false, it is labeled with a 0. For instance, the Translate switch, shown in Figure 1-19, has two outputs, SWTL0 and SWTL9. When the switch is operated SWTL0 is false and SWTL9 is true.



TRANSLATE SWITCH OUTPUTS

FIGURE 1-19

The 9 signal is not always produced by the N/O contact, nor is the 0 signal always produced by the N/C contact. Compare the Reset switch, shown in Figure 1-20, to the Translate switch, shown in Figure 1-19. In the case of the Reset switch, an SWRS0 signal was not needed, and it was more convenient to wire the switch in the manner shown. As a result, no definite rule can be stated governing the numbering of signals by contact type.



RESET SWITCH OUTPUT

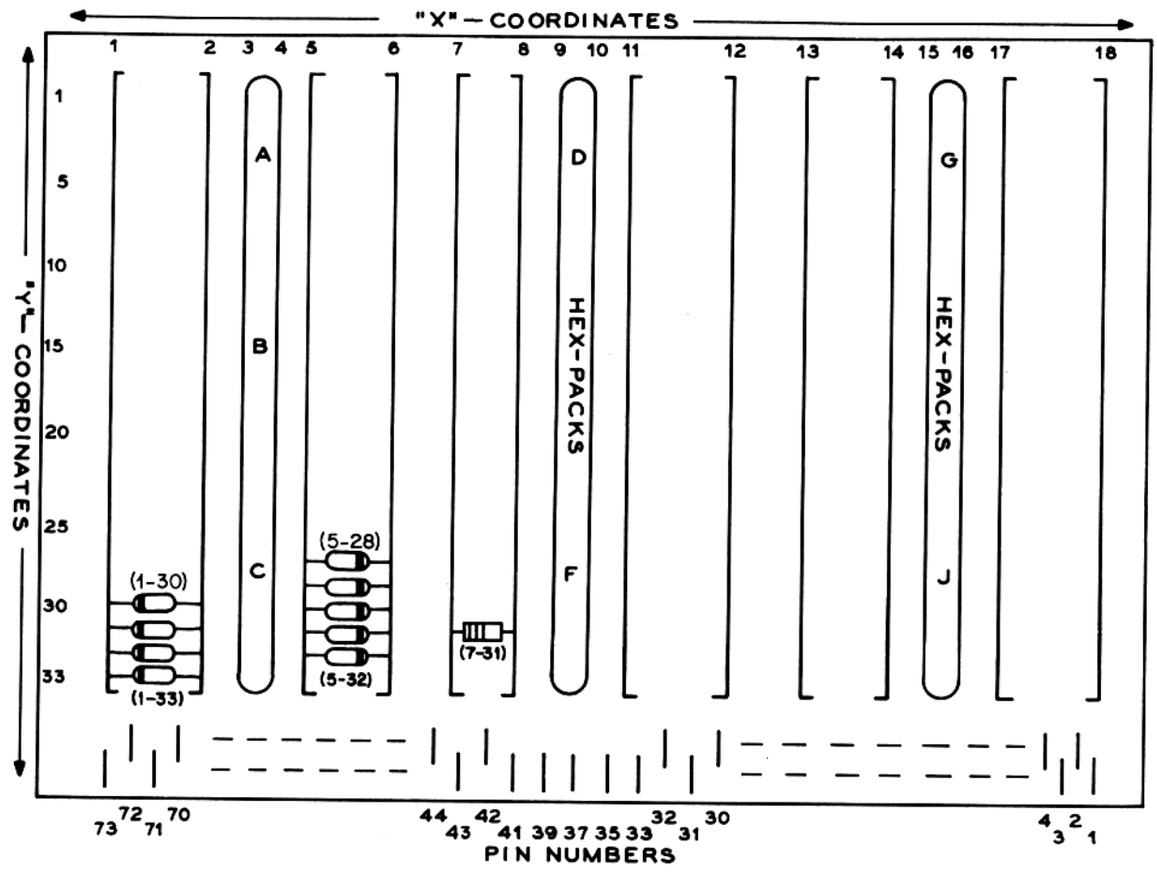
FIGURE 1-20

E. COMPONENT LOCATIONS

Figure 1-21 illustrates a typical printed circuit card layout from the component side. There are 6 vertical rows of components and 3 vertical rows of hex packs.

1. DIODES

When diode locations are given, the "X" coordinate is always listed first, and the "Y" coordinate last. Location 1-33, then, refers to column 1, 33 rows down (the lower left-hand diode on the card). Figure 1-22 illustrates a typical circuit appearing on the logic drawings. All of the diodes used in Figure 1-22 are illustrated in their correct locations in Figure 1-21. The X coordinate for any diode will be either a 1, 5, 7, 11, 13 or 17. The Y coordinate can be any digit from 1 to 33. The resistor (7-31) is a "pull-up" resistor needed when "H" type inverters are used.

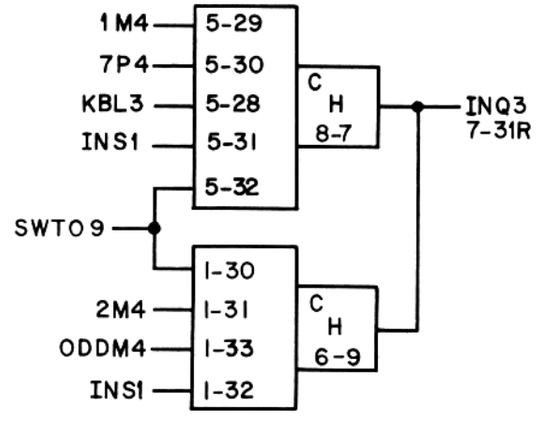


TYPICAL PC CARD LAYOUT

FIGURE 1-21

2. HEX PACKS

The actual hex pack location is given only by the letter designated on the logic drawing. In the case of inverter C6-9, only the "C" refers to the hex pack location. The 6-9 tells that the inverter input is on pin #6 and the output is on pin #9.



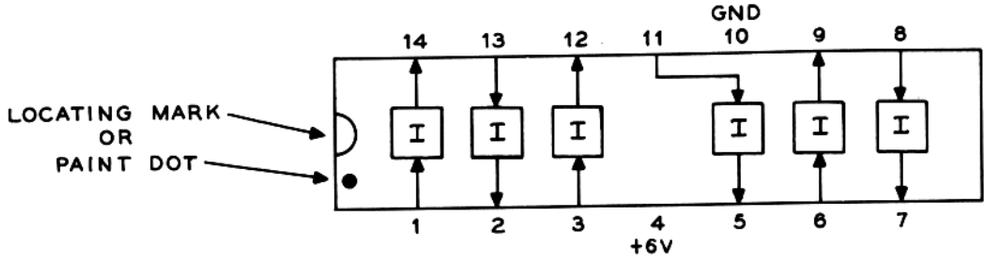
SAMPLE GATES SHOWING DIODE LOCATIONS

FIGURE 1-22

TP-161SL

Hex pack "C" will usually be found in the lower left corner of the card, as illustrated in Figure 1-21. The general locations of hex packs A, B, C, D, F, G, and J are also shown in Figure 1-21. However, it is important to note that these locations are not constant. The hex pack locations in Figure 1-21 are only a general guide. In all cases, the letter location of a hex pack will be printed on the card next to the hex pack.

Pin numbering of the hex packs is illustrated in Figure 1-23.



PIN NUMBERING ON HEX PACKS

FIGURE 1-23

3. OTHER COMPONENTS

Whenever components other than diodes and hex packs are used (resistors and capacitors for flip-flop triggers, etc.) they will be found in one of two places: either in columns, like the diodes, or on "piggyback" printed circuit cards. The "piggybacks" are small cards (containing resistors and capacitors) mounted on top of the hex packs.

The logic drawings do not list the locations of components on the "piggybacks", but in almost all cases these components will be found in the "piggyback" mounted over the hex pack to which they are connected. All resistors mounted on the "piggybacks" are 3900 ohms and all capacitors are 68 pf.

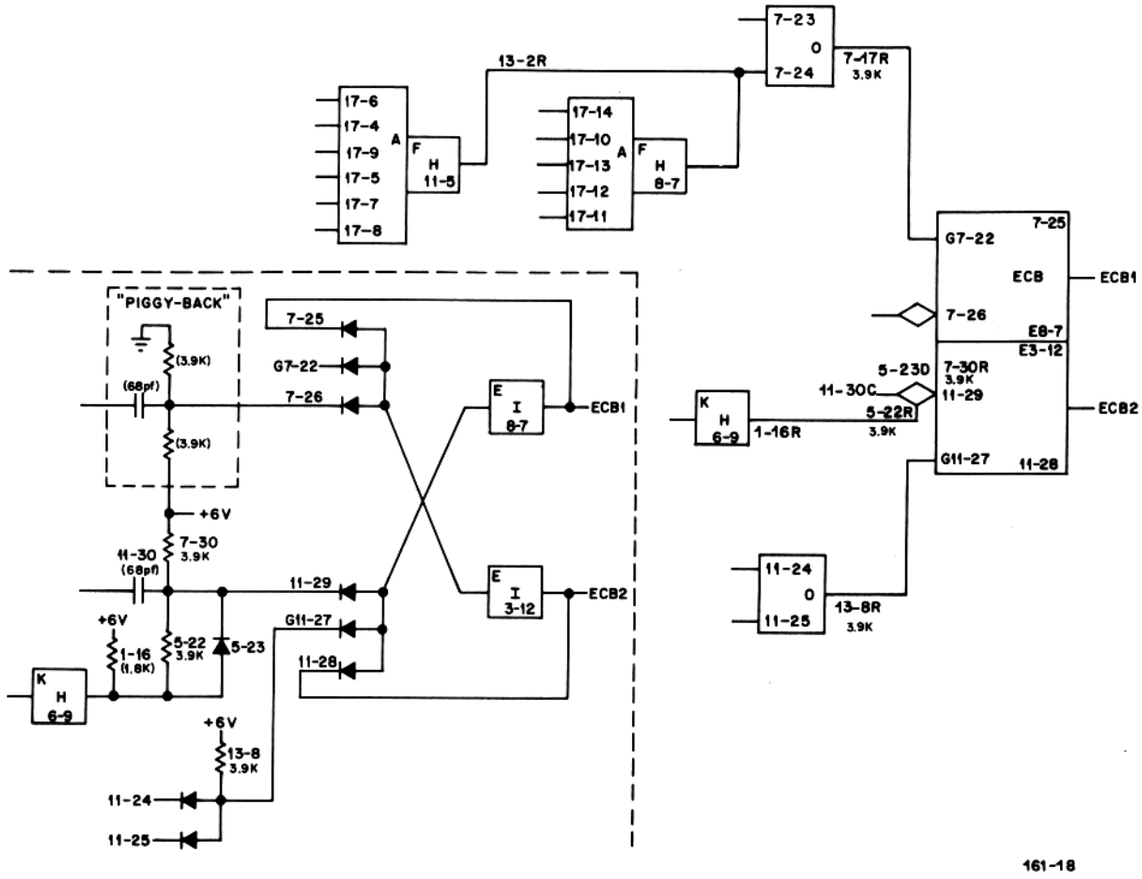
All other components mounted on the printed circuit card have their locations noted on the logic drawings. Unless otherwise specified, resistors are 1800 ohms and capacitors are 68 pf. Figure 1-24 illustrates the method used to depict some of the various components. The schematic relates the locations in the logic drawings to the actual circuit locations of the components.

"H" inverters require an external collector load resistor (one resistor for all "H" inverters having common outputs), usually 1800 ohms. Also, OR gates that are not followed by inverters require a separate biasing resistor, usually 3900 ohms. The locations of these resistors are depicted as shown in Figure 1-24.

When an OR gate output is connected directly to a flip-flop, it is coupled through a germanium diode, also shown in Figure 1-24.

4. PIN NUMBERS

Each signal that enters or leaves a card has the associated pin number listed in parentheses next to the signal name. Those signals that have no pin number reference are produced on that card and do not leave the card. Note that it is possible for the same signal name to appear on two cards, but never leave either one. For example, PLH3 can go to two different cards and be inverted, producing PLH4 on both cards, even though PLH4 does not leave either card.



SPECIAL DIODE/RESISTOR LOCATIONS

FIGURE 1-24

F. GLOSSARY OF SIGNAL NAMES

The following list explains the derivation of the many signal names used in the 5610. It is included mainly for the student: the signal name always gives some hint as to the manner in which the signal is produced or used.

In the following list, the word "flip-flop", when written out, denotes the fact that the signal being discussed affects that flip-flop in some way. The letters "ff" mean that the signal being discussed is a flip-flop output.

A	Fractional bit time "A", (program line timing chain)
AAA	Fractional bit counter "A" ff, (Arithmetic line timing chain)
AAB	Fractional bit counter "B" ff, (Arithmetic line timing chain)
AAG	Arithmetic column counter "G" ff (binary 1)
AAH	Arithmetic column counter "H" ff (binary 2)
AAJ	Arithmetic column counter "J" ff (binary 4)
AAK	Arithmetic column counter "K" ff (binary 8)
ACLK	Arithmetic timing chain CLocK
ADKK	ADvance trigger ff (test panel)
ADMT	Add in MulTiply

ADPR	ADvance PRime ff (test panel)
AL1M	Arithmetic Line home, not 1 Mode
ALH	Arithmetic Line Home ff
ALHO	Arithmetic Line Half Oscillator ff
ALMD	Arithmetic Line MiD oscillator ff
ALR	Arithmetic Line Read ff
ALRS	Arithmetic Line Read Signal
ALMX	Arithmetic Line Mid oscillator eXtra
ALWS	Arithmetic Line Write Signal
AMA	Address Modifier "A" ff
AMB	Address Modifier "B" ff
AOS	Add Or Subtract
AOST	Add Or Subtract Time
ARCR	Arithmetic Reset Common Register
ASSP	Action Step Shift Pulse
ASYN	Auxiliary SYNc (test panel)
AS0T	Action Step 0 Time
AS1L	Align Shift register 1 Left
AS1T	Action Step 1 Time
AS1X	Action Step 1 eXtra
AS2T	Action Step 2 Time
AS3T	Action Step 3 Time
AS4T	Action Step 4 Time
AS5T	Action Step 5 Time
AX5M	Arithmetic eXtra 5 Mode
AZOF	Arithmetic column Zero, One, or Fifteen
A0T	Arithmetic column 0 Time
A1T	Arithmetic column 1 Time
A2T	Arithmetic column 2 Time
A15T	Arithmetic column 15 Time
A15X	Arithmetic column 15 time eXtra
BASC	Block Action Step Count
BCDR	Back(space) Countdown Decimal Register
BECB	Block Enable Carry/Borrow flip-flop
BKSP	BacKSPace
BZY	"Busy" ff
B0T	Bit 0 Time (arithmetic line timing chain)
C	Fractional bit time "C" (program line timing chain)
CB	Carry/Borrow ff
CCG	Column Counter "G" ff (storage line timing chain)
CCH	Column Counter "H" ff (storage line timing chain)
CCJ	Column Counter "J" ff (storage line timing chain)
CCK	Column Counter "K" ff (storage line timing chain)
CDDR	Count Down Decimal Register
CKPT	CheckK ProtecT
CLOK	Program line timing chain CLOcK
CLR0	CLear Register 0

CMLT	Carry in MuLTiply
CMPT	CoMPuTe
COF	Carry OverFlow ff
COM	COMpare ff
COND	Carry Overflow Not in Divide
COUT	Common OUTput
CRCR	Common Register-to-Common Register
CRCT	CoRreCTion ff
CRDD	Common Register Detect Digit ff (address modifier)
CRDR	Common Register-to-Decimal Register
CRPL	Common Register-to-Program Line
CRR	Common Register Reset ff
CRSP	Common Register Shift Pulse ff
CRSR	Common Register-to-Search Register
CSGN	Change SiGN
C1T	Column 1 Time (storage line timing chain)
C15T	Column 15 Time (storage line timing chain)
DCMP	Decimal CoMPare
DEC	DECimal (in common register)
DEMP	Divide, Enter, Multiply, Print ff
DENT	Decimal ENTry
DIV	DIVide
DOST	Data Operation STop ff
DPS	Decimal Point Storage ff
DRZ	Decimal Register at Zero
DSUB	Division SUBtract
DUP	DUPlicate
DUPT	DUplicate or PrinT
EBZY	Enable BusY
ECB	Enable Carry/Borrow ff
ECTR	Enable CounTeR
EDCM	Enable Decimal CoMpare
EDPS	Enable Decimal Point Storage ff
EDR	Enable DRivers
EEOP	End of Entry or OutPut
EJ5P	Enable Jump to 5 Phase
ENTR	ENTeR
EPD	End of Program Detector
EPLW	Enable Program Line Write
EROD	Enable Right Of Decimal
ERST	Enable ReSeT flip-flop
ESAS	Enable Shift of Action Step counter
ESFF	Enable Special Flexowriter Functions
ESSL	Enable Store Shift Left
ESDN	Enable Shift DowN
ESUP	Enable Shift UP
ETR	Enable TRansfer
E1D	Enable bit 1 Driver

E2D	Enable bit 2 Driver
E3D	Enable bit 3 Driver
E4D	Enable bit 4 Driver
E5D	Enable bit 5 Driver
E6D	Enable bit 6 Driver
E7D	Enable bit 7 Driver
E8D	Enable bit 8 Driver
FBZD	Flexowriter Busy Delayed
FBZDD	Flexowriter Busy Delayed Twice
FBZY	Flexowriter BusY
FCB	Flexowriter Common Bit
FPOC	Five Phase Op Code
F1B	Flexowriter 1 Bit
F2B	Flexowriter 2 Bit
F3B	Flexowriter 3 Bit
F4B	Flexowriter 4 Bit
F5B	Flexowriter 5 Bit
F6B	Flexowriter 6 Bit
F7B	Flexowriter 7 Bit
F8B	Flexowriter 8 Bit
HSIH	High Speed Input inHibit
IETR	Inhibit Enable TRansfer
INEC	INput Enable Counter
INHP	INHibit Phase counter
INQ	INput Qualification
INS	INput Signal ff
INT	INput Trigger ff
IOQ	Inhibit Op Q (Halt)
ISL	Inhibit Shift Left
ISLSL	Inhibit Storage Line-to-Storage Line
IST	Initial STart
JPC	Jump Phase Counter ff
J0D	Jump 0 Decimal
J1D	Jump 1 Decimal
J1P	Jump 1 Phase
J2M	Jump 2 Mode
J3D	Jump 3 Decimal
J3M	Jump 3 Mode
J3P	Jump 3 Phase
J4M	Jump 4 Mode
J4P	Jump 4 Phase
J5D	Jump 5 Decimal
J5M	Jump 5 Mode
J5P	Jump 5 Phase
J6D	Jump 6 Decimal
J6M	Jump 6 Mode

J6P	Jump 6 Phase
J7P	Jump 7 Phase
J8P	Jump 8 Phase
J9P	Jump 9 Phase
J10P	Jump 10 Phase
J11P	Jump 11 Phase
J12P	Jump 12 Phase
J13P	Jump 13 Phase
J14P	Jump 14 Phase
J15D	Jump 15 Decimal
J15P	Jump 15 Phase
KBL	KeyBoard Lock
KRC	Relay Read Control (or non print)
LASP	Launch Arithmetic Sync Pulse
LD	LoaD ff
LLDR	Load Lamp DRiver
LMPC	LaMP Common (test panel)
LPEL	Light Parity Error Lamp
MAOS	Machine Add Or Subtract
MATH	MATHematics
MCRDR	Modifier Common Register-to-Decimal Register (address modifier)
MCRSR	Modifier Common Register-to-Search Register (address modifier)
MDSN	Multiply or Divide SigN addition
MDW	Modifier Delay line Write (address modifier)
MIN	MINus sign (in common register)
MJ6M	Modifier Jump 6 Mode (address modifier)
MJ15D	Modifier Jump 15 Decimal (address modifier)
MLA	Math LAtch ff
MOD	Multiply Or Divide
MROD	Multiply Right Of Decimal
MRR	Modified Reader Run (ten key)
MSH	Modifier fractional bit time SHift (address modifier)
MSRS	Modifier Set ReSet flip-flop (address modifier)
MSUB	Machine SUBtract
MTTF	Math Ten To Fifteen
MULT	MULTiply
NDPT	Non Decimal PrinT
NIN	Number INput
NMKS	Normal MarKer Set
NZR	Non Zero Register 1 ff
OA	Op equals "A" (Resume)
OCBS	Op Conditional Branch Signal
ODDM	ODD nuMbered phases (in 2 mode)
OE	Op equals "E" (Branch)
OFF	Op equals Flexowriter Function
OH	Op equals "H" (normal sequence separator)
ONOM	Op equals NO Marker ("H", "L", or space)
ONOP	Op equals No OPeration
ONP	Op equals Non Print
OP	Op equals "P" (Print Sign)

OPO	Op equals Punches Off
OPR	Op equals Print Restore
OQ	Op equals "Q" (Halt)
OR	Op equals "R" (Return)
ORR	Operation Register Reset ff
OSTP	Op equals SToP read
OTF	Op equals Tape Feed
OTO	Op equals Type Out
O0AD	Op equals 0 ADdress
O1AD	Op equals 1 ADdress
O1P	Op equals #1 Punch on
O1R	Op equals #1 Reader on
O2AD	Op equals 2 ADdress
O2P	Op equals #2 Punch on
O2R	Op equals #2 Reader on
PAMC	Pulse Address Modifier Counter
PDLG	Pulse Data LonG
PDTR	Pulse Data TRansfer
PECD	Permit Entry Count Digits
PERR	Parity ERRor
PLCR	Program Line-to-Common Register
PLDR	Parity Lamp DRiver
PLH	Program Line Home ff
PLIW	Program Line Intermediate Write
PLOR	Program Line-to-Operation Register
PLPL	Program Line-to-Program Line
PLPS	Program Line PreSet ff
PLRS	Program Line Read Signal
PLR	Program Line Read ff
PLSR	Program Line-to-Search Register
PLWS	Program Line Write Signal
PRNT	PriNT
PTCK	ProtecT CheckK ff
PTTF	Print Ten To Fifteen
RADP	Reset ADvance Prime flip-flop (test panel)
RBDP	Reset Both Decimal Point flip-flops
RCOF	Reset Carry OverFlow flip-flop
RCR	Reset Common Register
RDMF	Reset Divide, enter, Multiply & print Flip-flop
RECT	Reset Enable CounTer
RENT	Reader ENTry
RFS	Recall From Storage
RINS	Reset INput Signal flip-flop
ROED	Reset Only Enable Decimal point storage flip-flop
RR	Reader Run
RSR	Reset Search Register
RST	ReSeT ff
RTMA	ReTiMe Arithmetic ff
RTMP	ReTiMe Program ff

RTMS	ReTiMe Storage ff
RUBR	Reset Unconditional BRanch flip-flop
RWNG	Register 1 NeGative
R0T	Register 0 Time (arithmetic line timing chain)
R1DR	Reset 1 Decimal Register flip-flop
R1PH	Reset 1 Phase Hold flip-flop
R1T	Register 1 Time (arithmetic line timing chain)
R2DR	Reset 2 Decimal Register flip-flop
R2PH	Reset 2 Phase Hold flip-flop
R2T	Register 2 Time (arithmetic line timing chain)
R2X	Register 2 time eXtra
R3PH	Reset 3 Phase Hold flip-flop
R4DR	Reset 4 Decimal Register flip-flop
R4PH	Reset 4 Phase Hold flip-flop
R5CR	Reset 5 Common Register flip-flop
R6CR	Reset 6 Common Register flip-flop
R8DR	Reset 8 Decimal Register flip-flop
SADP	Set ADvance Prime flip-flop (test panel)
SASD	Storage & Arithmetic Shift Down
SASS	Storage Action Step Stop
SASX	Storage Action Stop eXtra
SCKP	Set ChecK Protection character
SCMF	Single Cycle Modifier Flip-flop (address modifier)
SCNT	Sign CoNTrol ff
SCOF	Set Carry OverFlow flip-flop
SDEC	Set DECimal (into common register)
SDGT	Significant DiGiT ff
SDMF	Set Divide, enter, Multiply, and print Flip-flop
SDTA	Store DaTA
SECB	Set Enable Carry/Borrow flip-flop
SEDP	Set Enable Decimal Point storage flip-flop
SEQ	SEQuence
SFF	Special Flexowriter Functions
SGN	SiGN ff
SH	Fractional bit time "SHift" (program line timing chain)
SLC	Storage Location Correct
SLDR	Sequence Lamp DRiver
SLH	Storage Line Home ff
SLHO	Storage Line Half Oscillator ff
SLHX	Storage Line Half oscillator eXtra
SLMD	Storage Line MiD oscillator ff
SLRS	Storage Line Read Signal
SLWS	Storage Line Write Signal
SNON	Set NON number (into common register)
SODP	Set Only Decimal Point storage flip-flop
SPA	SPAce (in common register)
SPAB	SPace or tAB (in common register)

SPC	Shift Phase Count ff
SPH	Shift Phase Hold ff
SRBR	Search Register BRanch marker
SRCR	Search Register-to-Common Register
SRFN	StoRage FiNished
SROD	Set Right Of Decimal
SROR	Search Register-to-Operation Register
SRPL	Search Register-to-Program Line
SRR	Search Register Reset ff
SRST	Set ReSeT flip-flop
SRTF	Search Register Tape Feed
SR0L	Shift Register 0 Left
SR1L	Shift Register 1 Left
SSDF	Set Significant Digit Flip-flop
SSTV	Set STep adVance flip-flop (test panel)
ST	STart
STAV	STep AdVance ff (test panel)
STOP	STOP ff (test panel)
STPX	SToP eXtra
STST	Set TeST flip-flop (test panel)
STXP	Another extra stop
SUB	SUBtract
SWLD	SWitch LoaD
SWRS	SWitch ReSet
SWTL	SWitch TransLate
SWTO	SWitch Type Only
SYNC	Arithmetic SYNC output (test panel)
S0D	Set 0 Decimal
S1CR	Set 1 Common Register flip-flop
S1DR	Set 1 Decimal Register flip-flop
S1M	Set 1 Mode
S1MD	Shift register 1 left in Multiply and Divide
S1P	Set 1 Phase
S1PH	Set 1 Phase Hold flip-flop
S2CR	Set 2 Common Register flip-flop
S2DR	Set 2 Decimal Register flip-flop
S2PH	Set 2 Phase Hold flip-flop
S3CR	Set 3 Common Register flip-flop
S3M	Set 3 Mode
S3PH	Set 3 Phase Hold flip-flop
S4CR	Set 4 Common Register flip-flop
S4DR	Set 4 Decimal Register flip-flop
S4PH	Set 4 Phase Hold flip-flop
S5CR	Set 5 Common Register flip-flop
S6CR	Set 6 Common Register flip-flop
S7CR	Set 7 Common Register flip-flop
S7P	Set 7 Phase
S8DR	Set 8 Decimal Register flip-flop
S13P	Set 13 Phase

TAB	TABulate
TADV	Test ADVance ff (test panel)
TAS0T	Test panel Action Step 0 Time
TAS1T	Test panel Action Step 1 Time
TAS2T	Test panel Action Step 2 Time
TAS3T	Test panel Action Step 3 Time
TAS4T	Test panel Action Step 4 Time
TAS5T	Test panel Action Step 5 Time
TCDR	Truncate Countdown Decimal Register
TCF	Terminate Constant Factor
TCP	Timing Chain Pulse
TEST	TEST ff (test panel)
TESTX	TEST eXtra
TESTY	Another extra test signal
TESX	Another extra test
THFF	Terminate Halt Flip-Flop
THLT	Terminate HaLT
TKEN	Ten Key ENtry ff
TMOD	Terminate Multiply Or Divide
TPRT	Terminate PRinT
TRD	TRanslate Decimal
TSRS	TRanslate Ring Shift
UBR	Unconditional BRanch ff
VRTA	InVert ReTime Arithmetic flip-flop
V1CR	InVert 1CR flip-flop
WRTD	WRiTe Direct
X	Math variable
XDIV	EXtra DIVide
XPLH	Extra Program Line Home
XPRN	EXtra PRiNt
XRCR	EXtra Reset Common Register
XRST	EXtra ReSeT ff
XS1L	EXtra Shift register 1 Left
XTST	EXtra TeST (test panel)
Y	Math variable
ZBCD	Zero Binary Coded Decimal
ZER	ZERo (in common register)
0D	Zero Decimal
1A	T1A time (program line timing chain)
1ALW	1 Arithmetic Line Write
1AMC	1 Address Modifier Counter ff
1AS	1 Action Step counter ff
1BC	1 Bit Counter ff (program line timing chain)
1CR	1 bit Common Register ff
1D	1 Decimal
1DR	1 Decimal Register ff
1DTR	1 bit Data Transfer Register ff
1LD	Arithmetic line home during LoaD

1MH	1 Mode Hold ff
1MR	1 Mode Register ff
1OR	1 bit Operation Register ff
1P	1 Phase
1 PH	1 Phase Hold ff
1PR	1 Phase Register ff
1RC	1 Register Counter ff (arithmetic line timing chain)
1RCR	1 mode Reset Common Register
1SR	1 bit Search Register ff
1TC	1 Tens Counter ff (storage line timing chain)
1UC	1 Units Counter ff (storage line timing chain)
1XM	1 Mode eXtra
2ALW	2 Arithmetic Line Write
2AMC	2 Address Modifier Counter ff
2AS	2 Action Step counter ff
2BC	2 Bit Counter ff (program line timing chain)
2CR	2 bit Common Register ff
2D	2 Decimal
2DR	2 Decimal Register ff
2DTR	2 bit Data Transfer Register ff
2M	2 Mode
2MH	2 Mode Hold ff
2MR	2 Mode Register ff
2MXJ	2 Mode, 2, 4, 8, 10 phase
2MXK	2 Mode, 2, 4 phase
2MXL	2 Mode, 2, 4, 10 phase
2MXM	2 Mode, 8, 10, 12 phase
2MXN	2 Mode, 4, 6 phase
2MXR	2 Mode, most even phases
2MXS	2 Mode, 14 phase & PLH
2MXY	2 Mode, eXtra
2MXZ	2 Mode, eXtra
2OR	2 bit Operation Register ff
2P	2 Phase
2PH	2 Phase Hold ff
2PR	2 Phase Register ff
2RC	2 Register Counter ff (arithmetic line timing chain)
2RCR	2 mode Reset Common Register
2RSR	2 mode Reset Search Register
2SR	2 bit Search Register ff
2TC	2 Tens Counter ff (storage line timing chain)
2UC	2 Units Counter ff (storage line timing chain)
2XM	2 Mode eXtra
3ALW	3 Arithmetic Line Write
3AS	3 Action Step counter ff
3BC	3 Bit Counter ff
3CR	3 bit Common Register ff
3CRX	3 bit Common Register eXtra

3D	3 Decimal
3DTR	3 bit Data Transfer Register ff
3M	3 Mode
3MH	3 Mode Hold ff
3MR	3 Mode Register ff
3MXL	3 Mode, 7-11 phase
3MXM	3 Mode, 10-12 phase
3MXN	3 Mode, 7, 9 phase
3MXR	3 Mode, 1, 3, 13, 15 phase
3M6A	3 Mode, 6A time
3OR	3 bit Operation Register ff
3P	3 Phase
3PH	3 Phase Hold ff
3PR	3 Phase Register ff
3RC	3 Register Counter ff
3RSR	3 mode Reset Search Register
3SR	3 bit Search Register ff
3XM	3 Mode eXtra
3XM6A	3 Mode 6A time eXtra
4A	T4A time (program line timing chain)
4AMC	4 Address Modifier Counter ff
4CR	4 bit Common Register ff
4D	4 Decimal
4DR	4 Decimal Register ff
4DTR	4 bit Data Transfer Register ff
4M	4 Mode
4MPLH	4 Mode Program Line Home
4MPLX	4 Mode Program Line home eXtra
4MXA	4 Mode 8 phase 6A time
4OR	4 bit Operation Register ff
4ORX	4 bit Operation Register eXtra
4P	4 Phase
4PH	4 Phase Hold ff
4PR	4 Phase Register ff
4RCR	4th Reset Common Register (1 mode)
4RSR	4 mode Reset Search Register
4SR	4 bit Search Register ff
4TC	4 Tens Counter ff (storage line timing chain)
4UC	4 Units Counter ff (storage line timing chain)
4XM	4 Mode eXtra
4XP	4 Phase eXtra
4YP	4 Phase, PLPS reset
5CR	5 bit Common Register ff
5D	5 Decimal
5F10	5 mode, op equals F10 (ten key)
5M	5 Mode
5MXK	5 Mode complete
5MXM	5 Mode complete, program line home
5P	5 Phase
5RCR	5 mode Reset Common Register
5XM	5 Mode eXtra

6A	T6A time (program line timing chain)
6B	T6B time (program line timing chain)
6C	T6C time (program line timing chain)
6CR	6 bit Common Register ff
6D	6 Decimal
6M	6 Mode
6OR	6 bit Operation Register ff
6P	6 Phase
6SH	T6SHift time (program line timing chain)
6SR	6 bit Search Register ff
6XA	T6A time eXtra
7CR	7 bit Common Register ff
7OR	7 bit Operation Register ff
7P	7 Phase
7SR	7 bit Search Register ff
7XP	7 Phase eXtra
8CR	8 bit Common Register ff
8DR	8 Decimal Register ff
8P	8 Phase
8UC	8 Units Counter ff (storage line timing chain)
9P	9 Phase
10P	10 Phase
10XP	10 Phase eXtra
11P	11 Phase
12P	12 Phase
13P	13 Phase
13XP	13 Phase eXtra
14P	14 Phase
14XP	14 Phase eXtra
14YP	14 Phase, PLPS reset
15P	15 Phase
15XP	15 Phase eXtra

TABLE OF CONTENTS

	Page
SECTION 2, PROGRAMMING	
THE OPERATING STACK	2-1
INSTRUCTION FORMAT	2-2
SWIFT PROGRAMMING LANGUAGE	2-3
SWIFT Mnemonics	2-3
Logical Instructions	2-5
Enter Instruction	2-8
Print Instructions	2-11
Arithmetic Instructions	2-13
Data Control Instructions	2-16
Storage Instructions	2-17
Flexowriter Instructions	2-19
Sample Program	2-26
Translation Format Commands	2-28
Error Sequence	2-28
ADVANCED PROGRAMMING	2-29
Operating in "Translate"	2-29
"Program Overlay"	2-29
Instructions not Included in SWIFT	2-29
"Macro" Instructions	2-30
Automatic Printing of Decimal During Entry	2-30
PREPARATION OF SOURCE PROGRAM TAPE	2-30
SOURCE PROGRAM TRANSLATION	2-33
REVERSE TRANSLATION	2-35
Features	2-35
Operating Instructions	2-35

SECTION 2, PROGRAMMING

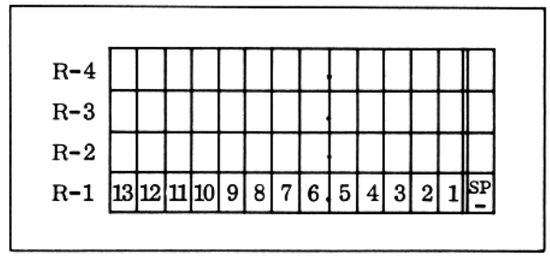
The basic concepts concerning programming in the 5610 were presented earlier in Section 1-B. This Section will delve into the subject in somewhat more detail, but will not attempt to cover it completely. Only those aspects of programming that are needed to understand the logic of the 5610 and to enable the technician to effectively communicate with the programmer are covered here. For more detailed information on programming the 5610, consult volumes DP11 and DP11.1 of the Branch Office Library, Data Processing Section.

A. THE OPERATING STACK

All programming operations are in one way or another directed at the "operating stack", or the arithmetic delay line. In fact, the entire operation of the 5610 revolves around this "stack". As stated in Section 1, the operating stack is made up of five registers, R0, R1, R2, R3, and R4. R0 is not accessible to the programmer: it is reserved for automatic use in the performance of arithmetic commands. All arithmetic instructions are performed on registers 1 and 2. R3 and R4 are "passive" registers, used for temporary storage.

The relationship of instructions such as Add, Subtract, Enter, Print, etc., to the arithmetic line is readily apparent. Other instructions (On-1, Non Print, etc.) can be termed "supportive" or secondary, in that they do not affect the operating stack directly, but instead control the input/output devices.

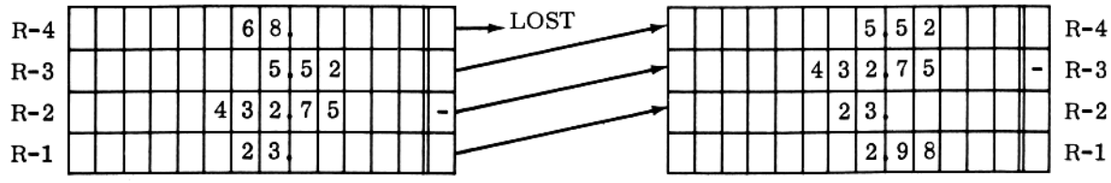
Each of the 5 operating registers has a capacity of 13 digits, and one position for the sign (see Figure 2-1). The sign may be a blank (for positive numbers) or a minus (for negative numbers). The decimal point is normally fixed between the fifth and sixth digit positions in each of these registers. Factory modification permits the decimal to be placed between the third and fourth positions or between the sixth and seventh positions. In this discussion, it will be assumed that the decimal point has been fixed in its normal position between the fifth and sixth digits.



FEATURES OF OPERATING REGISTERS

FIGURE 2-1

The operating registers are said to form a vertical "stack" because of their functional relationship to each other. For a number to be contained in one of these registers, it must initially be located in R1. An instruction which introduces a number into R1 will also cause the contents of the remaining registers to shift upwards (see Figure 2-2). An instruction which withdraws a number from R1 will cause the contents of the remaining registers to shift downwards (see Figure 2-3). The add, subtract, multiply, and divide functions use the numbers in R1 and R2. The result remains in R1, and the contents of the other registers shift downward. (See Figure 2-4).



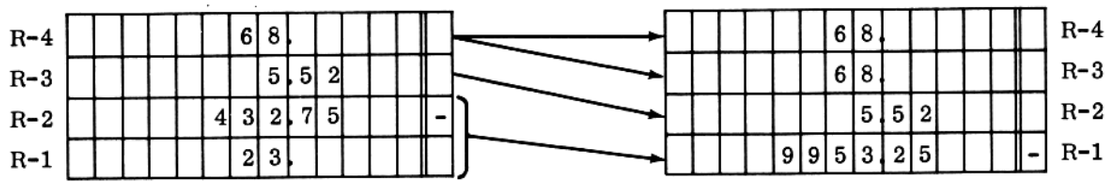
INTRODUCTION OF A NUMBER INTO R1

FIGURE 2-2



WITHDRAWAL OF A NUMBER FROM R1

FIGURE 2-3



EXECUTION OF ARITHMETIC INSTRUCTION (MULTIPLY)

FIGURE 2-4

This shifting action in the operating registers takes place automatically whenever data is transferred via R1, or when an arithmetic instruction is executed. For this reason, it is essential that the programmer record the shifting action of these registers on the programming sheet at every step in the program. It should be noted that the contents of R4 are lost whenever a number is entered in R1, but the contents of R4 are contained in both R3 and R4 when a number is withdrawn from R1, or when an arithmetic command is executed.

Those familiar with FRIDEN equipment will notice the similarity of the operating registers in the 5610 to the display on the 132 Calculator.

B. INSTRUCTION FORMAT

Each 5610 instruction consists of a command portion and an address portion. A command is that part of an instruction which specifies a particular 5610 function; an address is a set of characters following a command, specifying the nature or location of command execution. All 5610 commands consist of one writing machine character, but addresses may vary from 0 to 15 characters in length.

The command and address portions of an instruction specify the precise function to be performed. For example, in storage instructions, the command portion specifies the nature of the storing operation, and the address portion specifies the storage register. However, in some cases, the address portion of an instruction is implied by the nature of the command. For example, in arithmetic instructions, the address is implied to be R1 and R2; in format control instructions, the address is implied to be the writing machine. In such instructions, the address portion is left blank.

Program storage is completely independent of data storage. Up to 1119 characters can be stored on the program line. Instructions are stored consecutively, with each command requiring one character position, and addresses requiring from 0 to 15 character positions. These character positions may be occupied by any writing machine code using a combination of the first 7 channels, except tape feed. Each code requires one character position.

Commands are distinguished from addresses by the processor. For example, if an instruction requiring a 2-digit address is executed, the processor recognizes the 2 succeeding digits as the address for this command. On the other hand, if an instruction is executed and its address is implied by the nature of the command, the processor will execute the command without searching for an address.

Instructions are consecutively sorted in program storage in such a manner that each command is followed immediately by its address (if the command requires an address). The succeeding character in program storage is the command portion of the next instruction in the program.

C. SWIFT PROGRAMMING LANGUAGE

SWIFT (SoftWare Implemented Friden Translator) is the standard language used with the 5610. It consists of 44 instructions, each of which will be covered briefly in this section.

Because each 5610 program consists of a series of writing machine codes, programming directly in 5610 machine language can be very complex. The SWIFT translator has been developed to convert English-language statements into 5610 machine language. This translator is itself a 5610 program. When the translator is executed (in the translate mode), English-language instructions are converted into their corresponding machine language codes. Because of this capability, SWIFT considers only English-language instructions and the effect of executing them in the 5610.

The following instructions are typical of those used in SWIFT:

Command	Address
MULTIPLY	
DECIMAL PRINT	52
READER 1 ON	
NEGATIVE BRANCH	X
ENTER	42

For each of these instructions, the command is given in the English statement at the left, and the address is listed in a separate column to the right of the command. Those instructions in which the address is implied by the command are shown with a blank in the address column.

All 5610 instructions must be written with the command and the address in separate columns. This is essential for correct translation with the SWIFT translator. During translation, each command is converted into its corresponding 5610 object code, but the address is reproduced directly in the object program. For this reason, only those instructions in which the command and the address are listed in separate columns can be correctly translated with the SWIFT translator.

1. SWIFT MNEMONICS

Each command in SWIFT programming language consists of one, two, or three words. For example, Multiply is a one-word command, Negative Branch is a two-word command, and Non Decimal Print is a three-word command. Similarly, Reader 1 On is a three-word command in which the numeric "1" is counted as one word.

A 3-character "mnemonic" is associated with each command in SWIFT. A mnemonic is simply a set of three characters which represents the command. For example, the mnemonic MUL represents the Multiply command, the mnemonic NBR represents the Negative Branch command, and the mnemonic NDP represents the Non Decimal Print command. Each command in SWIFT has an associated mnemonic which uniquely identifies it.

The mnemonic for a command consists of three characters which are obtained from the command in one of the following ways:

(1) The first three characters of the first word.

(2) The first character of the first word and the first two characters of the second word.

(3) The first character of each of the first three words.

The table in Figure 2-5 presents the 44 commands in SWIFT, the mnemonic which represents each one, and the method by which each is obtained.

SWIFT COMMANDS AND ASSOCIATED MNEMONICS

Command	Mnemonic		
	Obtained by 1st Method	Obtained by 2nd Method	Obtained by 3rd Method
ACCUMULATE	ACC		
ADD	ADD		
BRANCH	BRA		
BACKSPACE	BAC		
CARRIAGE RETURN		CRE	
CHANGE SIGN		CSI	
CHECK PROTECT		CPR	
CONSTANT	CON		
DASH	DAS		
DECIMAL PRINT		DPR	
DELETE	DEL		
DIVIDE	DIV		
DUPLICATE	DUP		
ENTER	ENT		
HALT	HAL		
LOWER CASE		LCA	
MNEMONIC	MNE		
MULTIPLY	MUL		
NEGATIVE BRANCH		NBR	
NON DECIMAL PRINT			NDP
NON PRINT		NPR	
POSITIVE BRANCH		PBR	
PRINT RESTORE		PRE	
PRINT SIGN		PSI	
PUNCH 1 ON			P1O
PUNCH 2 ON			P2O
PUNCH OFF		POF	
READER 1 ON			R1O
READER 2 ON			R2O
READER OFF		ROF	
RECALL	REC		
RESUME	RES		
RETURN	RET		
SEQUENCE	SEQ		
SPACE	SPA		

Command	Mnemonic		
	Obtained by 1st Method	Obtained by 2nd Method	Obtained by 3rd Method
STORE	STO		
SUBTRACT	SUB		
TABULATE	TAB		
TRUNCATE	TRU		
TYPE	TYP		
UPPER CASE		UCA	
TRANSLATE	TRA		
END	END		
FINISH	FIN		

FIGURE 2-5

It is important to note that each mnemonic can be obtained from only one command, when the three methods are considered in turn. For example, the mnemonic DPR cannot be obtained from any command by the first and third methods, and is obtained only from the command Decimal Print by the second method. Similarly, the mnemonic NDP can be obtained only from the command Non Decimal Print, and only by the third method.

With a 3-character mnemonic to identify each command, the SWIFT translator is able to convert the commands from SWIFT programming language into 5610 machine language. For example, if the command Negative Branch is entered, the translator is able to identify this command by means of the mnemonic NBR. When the command has been identified in this way, the translator is able to convert it into the required 5610 object codes (codes meaningful to the 5610).

To obtain the 3-character mnemonic which is associated with each command, the processor scans the first three words of the command. A word is any set of writing machine characters, excluding space and tab. The end of each word is identified by a space.

If a combination of words is entered for which a corresponding mnemonic cannot be located on the program line, an error routine is executed. The search for a mnemonic is terminated as soon as the command is identified, or the error routine is executed.

Because only the first three words of the command are relevant for translation, additional words may be added without effect on the translation. For example, a suitable description of the function of each command may follow, providing that this description does not cross into the address column. Thus, the following instruction would be appropriate, provided that the column for the command portion of the instruction was sufficiently large to include the complete description.

STORE COST AMOUNT TOTAL S59

2. LOGICAL INSTRUCTIONS

The logical instructions (Sequence, Branch, Return, Positive Branch, Negative Branch, and Resume) are concerned with the initiation of program sequences, changes in the order in which instructions are to be executed, and repetitions of segments of a program.

The logical segments of a 5610 program are called "sequences". These sequences may vary in length and consist of those instructions which the programmer requires for execution of a specific portion of a program. Normally, each sequence is concerned with a particular routine which is required for one portion of the program. For example, an invoice program could consist of 3 sequences: one for the header, one for each invoice line, and one for the total line. However, the flexibility of 5610 programming permits a great deal of variation in this procedure; a program sequence can consist of any set of instructions which is required for the solution of a given problem.

a. SEQUENCE. The Sequence command is required to identify each program sequence. Although this command is translated into an object code by the SWIFT translator, it is not "executed" during program execution. Instead, the object code corresponding to the Sequence command is used only to separate the program sequences in a program. For this reason it is sometimes referred to as a "sequence separator" code.

The Sequence command requires a single-character address. This address may be any writing machine code other than tape feed that does not contain an 8-bit. The address for each Sequence command is called the "sequence code" for that command, because the address is used to identify the program sequences.

When the processor is not executing a sequence of instructions, the sequence light on the 5610 console will be on. This will be the case when the program is initially loaded into the processor, or when all the instructions in a specified program sequence have been executed. When the sequence light is on, the processor is awaiting the entry of a sequence code to identify a program sequence. When the sequence code is entered from the 2205 the processor will proceed to execute the instructions in that sequence.

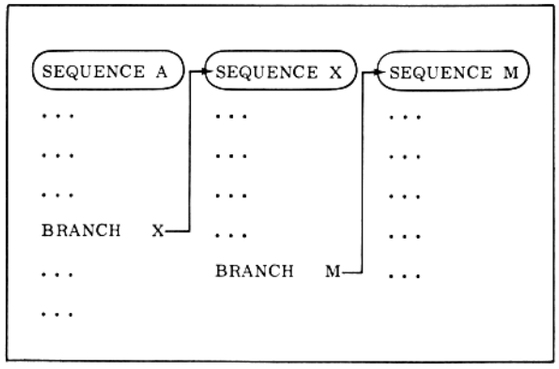
If an invalid sequence code is entered (i.e., a writing machine code which differs from the addresses of all Sequence commands in the program), the processor will immediately execute the instructions in the error sequence (to be explained shortly). The space code is reserved as the address of the error sequence. However, entry of either a space or a carriage return from the writing machine, when the processor is awaiting a sequence code, will be ignored by the processor.

The sequence code may be entered from either the keyboard or a reader. When a reader is used, an 8-bit may be added to prevent printing. Following execution of all instructions in the specified sequence, the sequence light will again be turned on, permitting the entry of a new sequence code. If a reader is in an "on" condition, this sequence code will come from the reader.

b. BRANCH. When the Branch instruction is executed, the processor interrupts the sequence of instructions being executed, proceeds to a Sequence command having an address identical to the address in the Branch instruction, and executes the instructions in this new sequence.

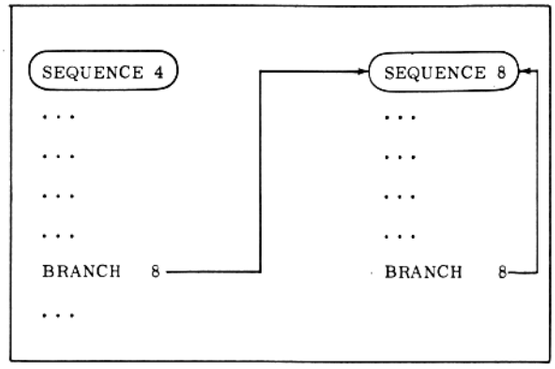
The address of the Branch command must be identical to the address of one of the Sequence commands in the program. Branches to the error sequence are also permitted. If the programmer develops a program having an invalid Branch instruction, the processor will automatically execute the instructions in the error sequence. This is of value during the debugging phase of programming.

Branches from any point in a program to any other point in the program are permitted, provided that a Branch instruction is used to interrupt the execution of instructions, and a Sequence command identifies the point at which execution is to resume. Examples of Branch instructions are given in Figures 2-6 and 2-7.



MULTIPLE BRANCHES

FIGURE 2-6



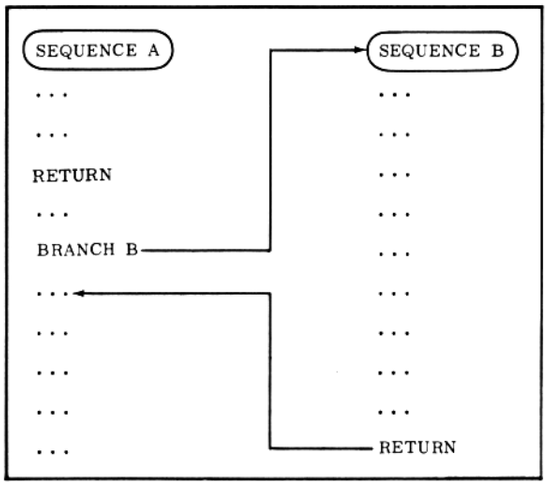
REPEATED BRANCHES

FIGURE 2-7

When a Branch instruction is executed, the processor leaves a marker at the point where the sequence is interrupted, providing no marker has been left previously. The marker can be erased by (1) execution of a Return instruction, (2) manual depression of the Reset switch on the 5610 console, or (3) execution of all instructions in a sequence, so that the processor is awaiting the entry of a sequence code specifying a new program sequence. Once the marker has been left, subsequent execution of Branch instructions has no effect on the marker. Only when the marker has been cleared, and a Branch instruction is executed, will a new marker be left.

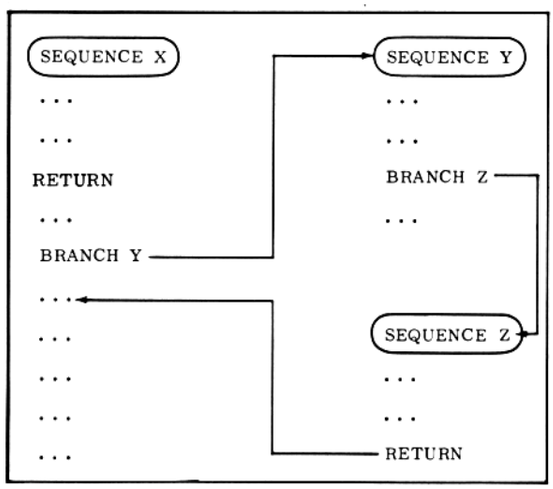
c. RETURN. Execution of the Return instruction causes the processor to interrupt the sequence being executed and return to the point where a marker was left by the execution of a previous Branch instruction. The marker is then cleared and the processor executes the instructions following the point where the marker had been left. A Return instruction has no effect unless a Branch instruction has been executed previously.

Examples of the use of the Return instruction are given in Figures 2-8 and 2-9.



RETURN

FIGURE 2-8



RETURN WITH INTERMEDIATE BRANCH

FIGURE 2-9

d. POSITIVE BRANCH. When the Positive Branch instruction is executed, the processor tests the sign of R1 and, if this sign is positive, interrupts the sequence being executed, proceeds to a Resume command having an identical address, and executes the instructions which follow this Resume instruction. If the sign of R1 is negative, a Positive Branch has no effect.

The Positive Branch command requires a single-digit address. The address may be any writing machine code except the carriage return code (8-bit) or tape feed.

e. NEGATIVE BRANCH. Execution of the Negative Branch instruction is identical to execution of the Positive branch instruction, except that the processor tests the sign of R1 for a negative condition.

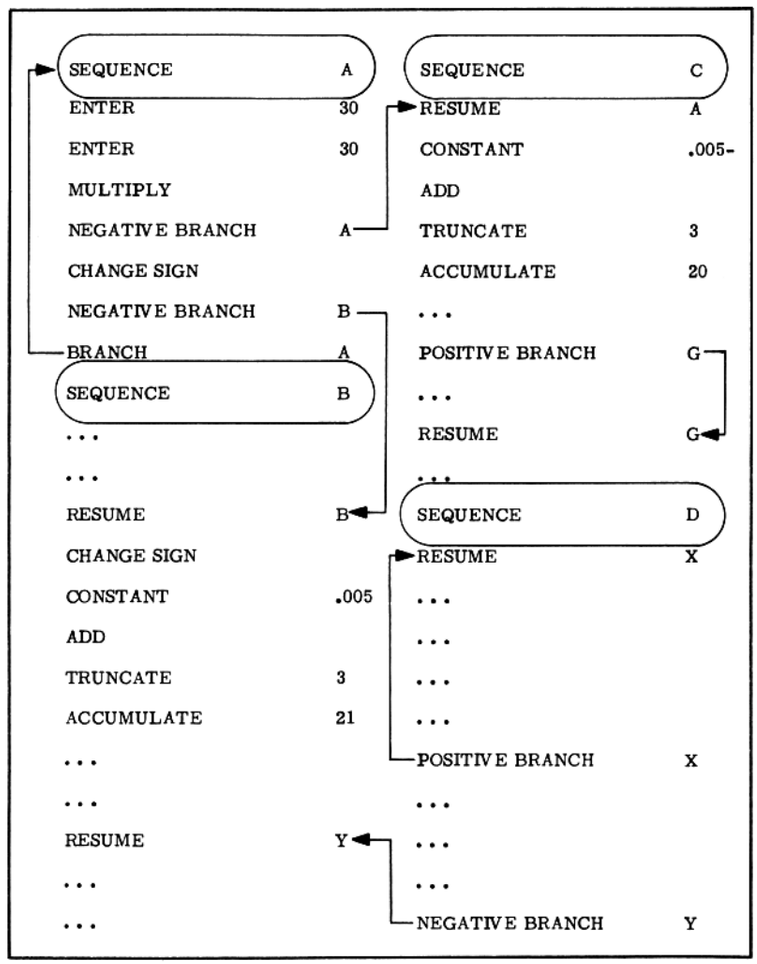
If R1 contains all zeros, the sign of R1 is always recognized as being positive.

f. RESUME. A Resume instruction terminates the search started by a previous Positive or Negative Branch, providing the addresses are the same. If the addresses are different, the search continues. If a Positive or Negative Branch has not been executed previously, sensing a Resume has no effect.

The address of a Resume command may be any single-digit code that is identical to the address of one of the Positive or Negative Branch commands in the program. Normally, duplication of resume codes must be avoided.

The assignment of all sequence and resume codes that are used in a given program can be recorded on the Sequence/Resume Assignment Sheet. (See Figure 2-54.) If there is in the program no Resume command whose address is identical to the address of a previously executed Positive Branch or Negative Branch command, the processor will continue to search indefinitely for the required command. It is essential for successful program execution that each Positive Branch and Negative Branch instruction have a corresponding Resume instruction.

An example of the use of Resume with Positive Branch and Negative Branch instructions is given in Figure 2-10.



RESUME INSTRUCTIONS

FIGURE 2-10

3. ENTER INSTRUCTION

Execution of the Enter instruction transfers data to R1 from the writing machine or its auxiliary input equipment. This instruction requires a 2-digit numeric address specifying the left- and right-of-decimal (LRD) requirements for the number. As a result, the number (and its sign, if any) is entered into R1 correctly aligned. If the entry is made from the 2205 keyboard,

printing takes place; if the entry is made from a reader, printing takes place - unless the 2205 is in a non print mode, or an 8-bit is added to each digit of the entry.

a. ADDRESS. The address of the Enter instruction indicates the size of the entry. The first digit of the address specifies the number of positions required for digits to the left of the decimal point. Since the decimal point is normally fixed between the fifth and sixth positions in the operating registers, the first digit of the address may be from 0 to 8, and the second digit from 0 to 6.

The appropriate address to be used with an Enter command depends upon the specific application. For example, if a 5-digit entry is to be made so the number is always to be entered without a decimal point, but the number may be either positive or negative, a suitable instruction is: Enter 60 (see Figure 2-11). This entry is satisfied when 5 digits, followed by a space or minus, are entered (or if 6 digits are entered). If an entry is to be made so the entry is always positive, and has 4 digits to the left of the decimal and 2 digits to the right, a suitable instruction is: Enter 42 (see Figure 2-12). When this instruction is executed, the first 4 digits are entered to the left of the decimal point and the succeeding 2 digits to the right of the decimal point.

SHIP'D	PRICE	DESC
200 45- 1000 XXXXXX		

ENTER 60 INSTRUCTION

FIGURE 2-11

NET AMOUNT	COST.	COST AMOUNT
	25.89 2.78 375.00 XXXX.XX	

ENTER 42 INSTRUCTION

FIGURE 2-12

Form alignment must be considered when either the decimal point or the sign is entered, since each of these requires one character position for printing.

Normally the decimal point must be entered by the operator during execution of an Enter instruction. However, it is possible to use a subroutine which causes automatic printing of the decimal point when the left-of-decimal requirements are fulfilled.

When the LRD requirements of the Enter instruction are satisfied, the processor executes the next instruction in the program.

b. RULES. The following rules apply to the Enter instruction:

(1) Prior to the entry of a significant digit, spacing decreases the left-of-decimal requirements by 1 digit for each space (see Figure 2-13).

(2) Prior to the entry of a significant digit, a tab or carriage return terminates the entry and causes R1 to contain zeros. The entry of any other non-numeric (except space or backspace) is ignored by the processor, except when the left-of-decimal requirement is 0 (see Figure 2-14).

NET AMOUNT	COST.	COST AMOUNT
	205.45 1200.00 XXXXX.XX	

ENTER 52 INSTRUCTION

FIGURE 2-13

SHIP'D	PRICE	
M200 C150 E 25 XXX		

ENTER 30 INSTRUCTION

FIGURE 2-14

(3) Prior to the entry of a significant digit, backspacing increases the left-of-decimal requirements by one digit for each backspace (see Figure 2-15).

(4) All entries are entered into R1 with the decimal point of the entry aligned with the fixed decimal position in R1. If no decimal is entered, digits are entered until the left-of-decimal requirements are fulfilled. Subsequent digits are entered right-of-decimal (see Figure 2-16).

NET AMOUNT	COST.	COST AMOUNT
	425.00 1205.95 XXX.XX	

ENTER 32 INSTRUCTION

FIGURE 2-15

NET AMOUNT	COST.	COST AMOUNT
	425.00 20000 XXX.XX	

ENTER 32 INSTRUCTION

FIGURE 2-16

(5) Entry of a decimal point satisfies the left-of-decimal requirements of the entry, both before and after the entry of a significant digit (see Figure 2-17).

(6) All entries are entered into R1 as positive numbers except when the entry is terminated with a minus, as shown in Figure 2-18. (Minus before the entry of a significant digit is ignored by the processor unless the left-of-decimal requirement is 0, in which case the entry is terminated.)

	SHIP'D	PRICE	
		2.55 120.00 XXX.XX	

ENTER 32 INSTRUCTION

FIGURE 2-17

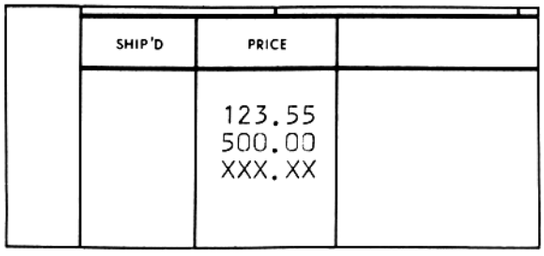
	NET AMOUNT	COST.	
	35.95- 125.00 - 38.75 XXXX.XXX		

ENTER 43 INSTRUCTION

FIGURE 2-18

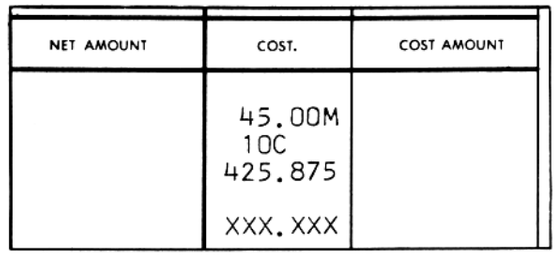
(7) The entry is terminated by satisfying the LRD requirements of the entry, as specified in the address portion of the instruction (see Figure 2-19).

(8) After the entry of a significant digit, the entry of a non-numeric character (space, tabulate, carriage return, upper case, lower case, alphabetic, or function code) terminates the entry, but backspace causes the keyboard of the writing machine to be locked and the RESET light to be turned on (see Figure 2-20).



ENTER 32 INSTRUCTION

FIGURE 2-19



ENTER 33 INSTRUCTION

FIGURE 2-20

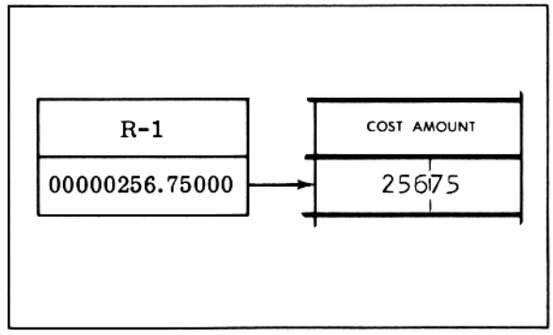
Entries may be made to the processor by the writing machine or any input device connected to the writing machine. This includes the 2205 keyboard or reader, and any auxiliary input unit.

4. PRINT INSTRUCTIONS

The Print instructions are Decimal Print, Non Decimal Print, Check Protect, Print Sign, and Type.

a. DECIMAL PRINT AND NON DECIMAL PRINT INSTRUCTIONS. Data is transferred from R1 to the writing machine and/or auxiliary output units, by means of a Decimal Print or Non Decimal Print instruction.

The Decimal Print instruction prints the contents of R1, including the decimal point, as specified by the address of the instruction. The Non Decimal Print instruction prints the contents of R1 without a decimal point, in accordance with the address of the instruction (see Figure 2-21). These two instructions differ only in the fact that a decimal point may or may not be printed. The programmer must be careful to allow one extra character position on the form when a Decimal Print instruction is to be executed, for printing of the decimal point.



NON DECIMAL PRINT 52 INSTRUCTION

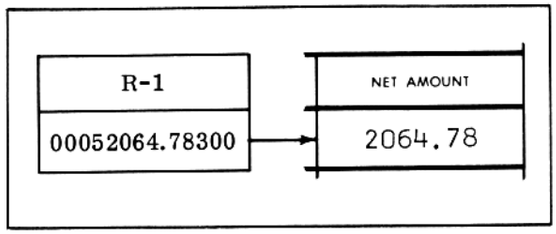
FIGURE 2-21

The Decimal Print and Non Decimal Print commands each require a 2-digit address specifying the LRD requirements of the data to be printed. The first digit of the address may be from 0 to 8; the second digit may be from 0 to 5. The address of each instruction restricts the portion of R1 which is to be printed. However, execution of either instruction does not affect the contents of R1.

The following rules apply to the Decimal Print and Non Decimal Print instructions:

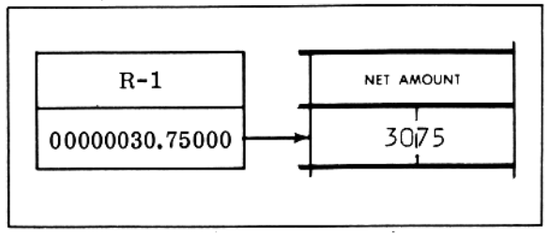
(1) The address of the instruction restricts printing to that portion of R1 which is specified by the LRD pattern in the address, as shown in Figure 2-22.

(2) Insignificant high-order zeros on the left of the decimal point in R1 are printed as spaces, as shown in Figure 2-23.



DECIMAL PRINT 42 INSTRUCTION

FIGURE 2-22

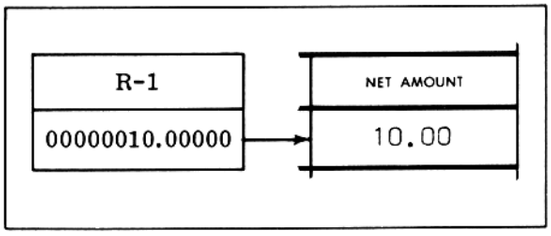


NON DECIMAL PRINT 52 INSTRUCTION

FIGURE 2-23

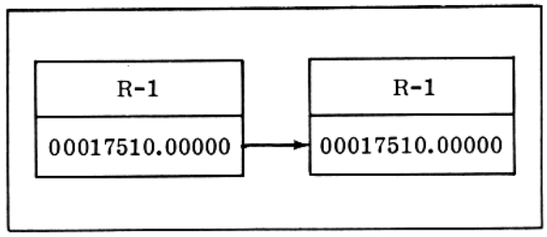
(3) All zeros to the right of the decimal point in R1 are printed, provided that they are within the specified LRD pattern of the address, as shown in Figure 2-24.

(4) Execution of either instruction has no effect on the data in R1, as shown in Figure 2-25.



DECIMAL PRINT 32 INSTRUCTION

FIGURE 2-24



NON DECIMAL PRINT 50 INSTRUCTION

FIGURE 2-25

The Decimal Print and Non Decimal Print instructions may also be used for output of data from R1 without printing. For example, if the 2205 is in a non print mode and the punch is on, the data is punched but not printed. The same rules apply for punching as for printing.

The Decimal Print and Non Decimal Print instructions are concerned with the contents of R1, and ignore the sign of R1.

b. CHECK PROTECT. The Check Protect instruction is used for automatic protection of the dollar amount of checks. Execution causes high order zeros in R1 to be replaced by the specified check protection symbol. When the Check Protect instruction, followed by a Decimal Print or Non Decimal Print instruction, is executed, the specified symbol will be printed instead of a space for all high order zeros.

Check protection must be initiated for each Decimal Print or Non Decimal Print instruction. Following execution of the Decimal Print or Non Decimal Print instruction, check protection will no longer be in effect and high order zeros will subsequently be printed as spaces. For this reason, the Check Protect instruction should immediately precede the Decimal Print or Non Decimal Print instruction for which check protection is required.

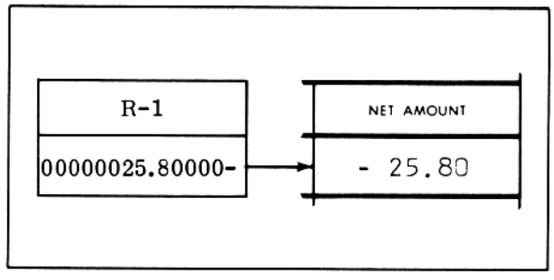
Any writing machine symbol may be used for check protection. If a zero is used as the check protection symbol, all high order zeros can be caused to print. By using a code that causes no carriage movement, left-justified printouts can be obtained. The writing machine must be set into the proper case prior to execution of the Decimal Print or Non Decimal Print instruction.

c. PRINT SIGN. When the Print Sign instruction is executed, the sign of R1 is printed. The sign is either a space (for positive numbers) or a minus (for negative numbers). The Print Sign instruction usually is used in conjunction with a Decimal Print or Non Decimal Print instruction. The sign in R1 can thus be printed either before or after the data. Execution of this instruction has no effect on the data in R1.

The following sequence of instructions illustrates the use of the Print Sign instruction:

```
PRINT SIGN
DECIMAL PRINT      42
```

When this sequence of instructions is executed, the sign of R1 is printed, followed by the contents of R1, as shown in Figure 2-26. After printing, the data in R1 remains unchanged. The Print Sign instruction may also be used for punching without printing. For example, if the writing machine is in a non print mode and the punch is on, the Print Sign instruction causes punching of the sign of R1 without printing. When the sign is to be printed, the programmer must allow one space on the form for printing of the sign.



PRINT SIGN, DECIMAL PRINT 42 INSTRUCTIONS

FIGURE 2-26

5. ARITHMETIC INSTRUCTIONS

The four arithmetic instructions, Add, Subtract, Multiply, and Divide, are performed on the contents of R1 and R2. The result appears in R1 and the contents of the other registers shift down. The address portion of these arithmetic instructions is always R1 and R2, so it is not stated.

During execution of an arithmetic instruction, both the quantity and the sign of the data are involved. Thus the result which is placed in R1 can be called the algebraic result of the arithmetic operation. A review of these algebraic rules follows:

a. ALGEBRAIC RULES. In the statement of each of the following rules (1) "a" and "b" are positive, (2) the "arithmetic result" is considered to be the result of an arithmetic operation on positive numbers, and (3) the "algebraic result" is considered to be the result of an arithmetic operation on numbers which may be positive or negative.

(1) ADDITION. (a) + (b). The algebraic sum of two positive numbers is always positive, and is equal to the arithmetic sum.

(a) + (-b) or (-a) + (b). The algebraic sum of a positive number and a negative number may be positive or negative. The sign of the result is equal to the sign of the larger number, and the quantity of the result is equal to the arithmetic difference between the two numbers.

(-a) + (-b). The algebraic sum of two negative numbers is always negative, with the quantity of the result being equal to the arithmetic sum of the two numbers.

(2) SUBTRACTION. (a) - (b). The algebraic difference between two positive numbers may be positive or negative. The sign of the result is positive if the first number is larger than the second, and negative if the second number is larger than the first. The quantity of the result is equal to the arithmetic difference between the two numbers.

(a) - (-b). The algebraic difference between a positive and a negative number is positive, and the quantity of the result is equal to the arithmetic sum of the two numbers.

(-a) - (b). The algebraic difference between a negative and a positive number is negative, and the quantity of the result is equal to the arithmetic sum of the two numbers.

(-a) - (-b). The algebraic difference between two negative numbers may be positive or negative. The sign of the result is positive if the first number is smaller than the second, and negative if the first number is larger than the second. The quantity of the result is equal to the arithmetic difference between the two numbers.

(3) MULTIPLICATION. (a) x (b). The algebraic product of two positive numbers is positive, and is identical to the arithmetic product.

(a) x (-b) or (-a) x (b). The algebraic product of a positive number and a negative number is negative, and the quantity of the result is equal to the arithmetic product.

(-a) x (-b). The algebraic product of two negative numbers is positive, and the quantity of the result is equal to the arithmetic product.

(4) DIVISION. (a) ÷ (b). The algebraic quotient of two positive numbers is positive, and is equal to the arithmetic quotient.

(a) ÷ (-b) or (-a) ÷ (b). The algebraic quotient of a positive number with a negative number is negative, and the quantity of the result is equal to the arithmetic quotient.

(-a) ÷ (-b). The algebraic quotient of two negative numbers is positive, and the quantity of the result is equal to the arithmetic quotient.

b. ADD. Execution of the Add instruction causes the contents of R1 to be algebraically added to the contents of R2, with the result of this operation being placed in R1. The contents of the other operating registers shift down. (See Figure 2-27).

	BEFORE EXECUTION	AFTER EXECUTION
R-4	2 4.3 6	2 4.3 6
R-3	3 9 4.7 8 3	2 4.3 6
R-2	5.9 5 -	3 9 4.7 8 3
R-1	0 0 0 0 0 0 1 4.7 5 0 0 0	0 0 0 0 0 0 0 8.8 0 0 0 0

ADD

FIGURE 2-27

c. SUBTRACT. When the Subtract instruction is executed, the contents of R1 are algebraically subtracted from the contents of R2, with the result of the operation being placed in R1. The contents of the other operating registers shift down. (See Figure 2-28.)

	BEFORE EXECUTION	AFTER EXECUTION
R-4	1 3 2.8 9 -	1 3 2.8 9 -
R-3	2 4.7 5	1 3 2.8 9 -
R-2	1 8.4 7 -	2 4.7 5
R-1	0 0 0 0 0 0 0 4.7 5 0 0 0	0 0 0 0 0 0 2 3.2 2 0 0 0 -

SUBTRACT

FIGURE 2-28

d. MULTIPLY. Execution of the Multiply instruction causes the contents of R1 to be algebraically multiplied by the contents of R2, with the result of this operation being placed in R1. The contents of the other operating registers shift down. (See Figure 2-29.)

	BEFORE EXECUTION	AFTER EXECUTION
R-4	1 4.8 9	1 4.8 9
R-3	0.0	1 4.8 9
R-2	3.7 5 -	0.0
R-1	0 0 0 0 0 0 1 2.0 0 0 0 0	0 0 0 0 0 0 4 5.0 0 0 0 0 -

MULTIPLY

FIGURE 2-29

e. DIVIDE. When the Divide instruction is executed, the contents of R1 are algebraically divided into the contents of R2, with the result of this operation being placed in R1. The contents of the other operating registers shift down. (See Figure 2-30.)

	BEFORE EXECUTION	AFTER EXECUTION
R-4	0.0	0.0
R-3	2 3.5 6 7	0.0
R-2	3 7.5 0	2 3.5 6 7
R-1	0 0 0 0 0 0 0 4.0 0 0 0 0 -	0 0 0 0 0 0 0 9.3 7 5 0 0 -

DIVIDE

FIGURE 2-30

Division by zero causes the red Error light to be turned on. This permits the operator to depress the Reset switch and initiate a new program sequence.

Overflow on the left or right in R1 cannot be detected by the processor when an arithmetic instruction is executed. The programmer must ensure that all data is within specified limits if overflow is not to occur.

6. DATA CONTROL INSTRUCTIONS

The data control instructions (Constant, Duplicate, Delete, Change Sign, and Truncate) permit data manipulation in the operating registers.

a. CONSTANT. The Constant instruction is used for repetitive constant factors required during program execution, without the use of data storages. Execution of this instruction causes the number constituting the address of the Constant command to be entered into R1. The contents of the operating registers shift up when this instruction is executed.

The address portion of a Constant instruction may include 0 to 8 digits to the left of the decimal, and 0 to 5 digits to the right of the decimal. The address must be terminated with a space (for positive constants) or a minus (for negative constants). The address may consist of 0 to 8 digits, without a decimal point, if the constant is to be entered left-of-decimal, but constants which contain more than 8 digits must include a decimal point.

The number of digits in the address of the Constant instruction will be counted correctly by the SWIFT translator during translation, provided that insignificant zeros have not been included and no decimal point is used for constants to be entered left-of-decimal only.

b. DUPLICATE. Execution of the Duplicate instruction causes the contents of R1 to be repeated in R2, with the contents of the other operating registers shifting up. (The previous contents of R4 are lost.)

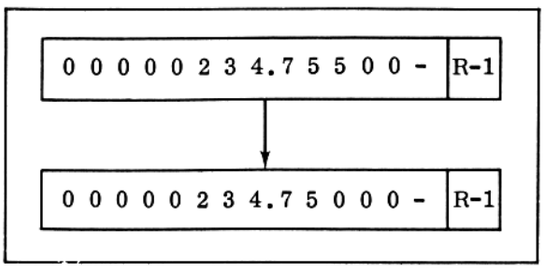
c. DELETE. When the Delete instruction is executed, the contents of R1 are destroyed and the contents of the other operating registers shift down. This instruction is used to shift the contents of the operating registers downward, thus making the contents of R2 available in R1. (The previous contents of R4 are contained in both R3 and R4 after command execution.)

d. CHANGE SIGN. Command execution will cause the sign of R1 to be changed from positive to negative, or vice versa. This instruction affects only the sign of R1, and has no effect on the contents of R1.

e. TRUNCATE. Execution of the Truncate instruction causes a number of digits in R1 to be replaced by zeros as specified by the address. From 1 to 13 digits may be replaced by zeros using the Truncate instruction. The address of this instruction indicates the number of digits (counting from the right) which are to be replaced. The single-character address may be any digit from 1 to 9, but additional symbols can also be used for truncation of 10 to 13 digits:

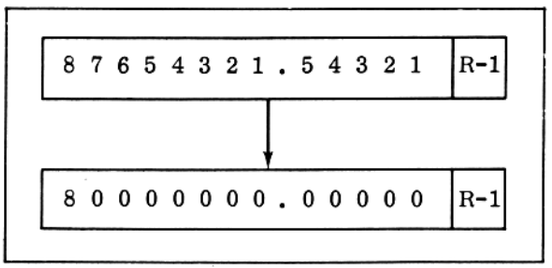
No. of Digits Replaced	Address
1 - 9	1 - 9
10	+
11	F1
12	F2
13	F8

The Truncate F8 instruction can be used to clear R1 without causing the contents of the operating registers to be shifted up or down. Examples of the effect of Truncate instructions are shown in Figures 2-31 and 2-32.



TRUNCATE 3 INTRUCTION

FIGURE 2-31



TRUNCATE F2 INSTRUCTION

FIGURE 2-32

The Truncate instruction does not affect the sign of R1. However, when a negative number is so truncated as to be completely replaced by zeros, its sign is sensed as being positive.

f. EXAMPLES OF USE OF DATA CONTROL INSTRUCTIONS. The Constant and Truncate instructions are used in rounding off data. For example, the following sequence is used to round off a positive amount to the nearest cent:

CONSTANT	.005 (space)
ADD	
TRUNCATE	3

Similarly, the same instructions can be used for rounding off a negative amount:

CONSTANT	.005-
ADD	
TRUNCATE	3

The programmer must be careful to determine whether the numbers to be rounded off are positive or negative.

A round-off can also be performed for data at any specified digit position. In the following example, positive data is rounded off to the nearest hundred:

CONSTANT	50 (space)
ADD	
TRUNCATE	7

The programmer will have no difficulty in modifying the round-off routine for a specific application.

7. STORAGE INSTRUCTIONS

There are 60 data storage registers in the 5610. Like the operating registers, each is capable of containing 13 digits plus sign, with a fixed decimal point. By the execution of an appropriate instruction, storage of data can be performed on either an accumulative or destructive basis. The storage registers are numbered from "00" to "59". Therefore, each storage instruction requires a 2-digit numeric address (from 00 to 59). There are three storage instructions; Store, Recall, and Accumulate.

a. STORE. Execution of the Store instruction causes the contents of R1 to be stored in the storage register specified by the 2-digit address following the Store command. The previous contents of the storage register are destroyed, and the contents of the operating registers shift down. (See Figure 2-33.)

	R-1	57
BEFORE EXECUTION	0 0 0 0 0 0 1 2.7 5 0 0 0	4 3.8 9
AFTER EXECUTION	---	1 2.7 5

STORE 57 INSTRUCTION

FIGURE 2-33

b. RECALL. The Recall instruction is used to retrieve data from storage. Execution of this instruction causes the contents of the storage register, specified in the 2-digit address following the Recall command, to be transferred to R1. The contents of the specified storage register are not affected by the execution of this instruction, but the contents of the operating registers shift up.

c. ACCUMULATE. The Accumulate instruction permits the addition of data in storage. When this instruction is executed, the contents of R1 are algebraically added to the previous contents of the storage register specified in the 2-digit address following the Accumulate command. The result of this addition remains in the storage register, and the contents of the operating registers shift down. Figure 2-34 illustrates the function of the Accumulate instruction.

Line No.	Command Statement	Address	R-1	R-2	R-3	R-4	05
001	ENTER	31	00000015.00000	---	---	---	432.85
002	ENTER	42	00000002.75000	15.0	---	---	432.85
003	MULTIPLY		00000041.25000	---	---	---	432.85
004	ACCUMULATE	05	---	---	---	---	474.10

ACCUMULATE INSTRUCTION

FIGURE 2-34

As illustrated, the Accumulate instruction permits the addition of data to storage. This is particularly valuable when totals are to be accumulated, because the operation of accumulation requires only one instruction.

d. ADDRESS MODIFIER. 5610's equipped with the address modifier feature have the capability of addressing a storage location specified by the arithmetic stack instead of the program line. In this case the program line contains a storage instruction having a "99" address. When such an instruction is retrieved, the address is selected from R1 (Recall) or R2 (Store and Accumulate). R1 or R2 can contain any data: the address will be that specified by the two digits immediately left-of-decimal.

The effect on the operating and storage registers of the execution of these instructions is also shown in Figure 2-35. Execution of each instruction in this sequence causes the operating registers to shift. Note that the contents of storages "08" and "09" are affected only by the Accumulate and Store instructions; the Recall instruction does not affect the contents of a storage register, so these contents are available for further use in the program.

Line No.	Command Statement	Address	R-1	R-2	R-3	R-4	08	09
001	ENTER	40	00000010.00000	---	---	---	147.50	42.80
002	ENTER	32	00000002.75000	10.	---	---	147.50	42.80
003	MULTIPLY		00000027.50000	---	---	---	147.50	42.80
004	STORE	08	---	---	---	---	27.50	42.80
005	RECALL	09	00000042.80000	---	---	---	27.50	42.80
006	ACCUMULATE	08	---	---	---	---	70.30	42.80
007	RECALL	08	00000070.30000	---	---	---	70.30	42.80
008	DECIMAL PRINT	52	00000070.30000	---	---	---	70.30	42.80

PROGRAM SEQUENCE ILLUSTRATING USE OF RECALL INSTRUCTION

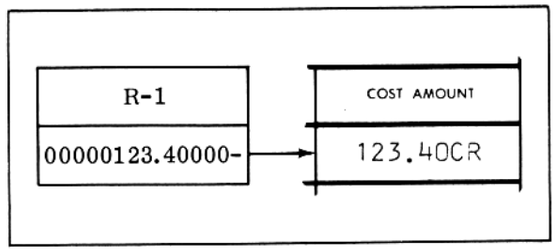
FIGURE 2-35

It is essential that a 2-digit numerical address from 00 - 59 be used in each of the storage instructions when specifying a storage register. This is particularly important for storage registers 00 - 09, since incorrect translation with the SWIFT translator otherwise results. In addition, the programmer is required to keep a record of the assignment of all storage registers, because translation cannot detect errors of assignment. A Storage Assignment Sheet (see Figure 2-56) has been provided for this purpose.

8. FLEXOWRITER INSTRUCTIONS

There are three distinct types of Flexowriter instructions. The first (print instructions) has already been covered. The second group is made up of the Type instruction and the format control commands. The last group is referred to as Special Flexowriter Functions. As its name implies, this group of commands performs only functions, and does not handle data or other information. In addition to these three groups, the Halt command can also be considered as a 2205-oriented instruction.

a. "TYPE". The Type instruction is used for printing symbols from the writing machine keyboard, and to execute writing machine functions. When the Type instruction is executed, its single-character address code is transmitted to the writing machine. The corresponding key on the writing machine keyboard is operated. The single-character address of the Type X instruction may be any writing machine code except the carriage return or tape feed code. For example, if the address is an alphabetic character, this character is printed; if the address is a function code, the writing machine function which is associated with that code by the wiring in the writing machine control panel is executed.



"TYPE" INSTRUCTION FOR PRINTING CREDITS

FIGURE 2-36

The following sequence and Figure 2-36 illustrate the use of the Type instruction:

TYPE	(F8)
DECIMAL PRINT	42
TYPE	C
TYPE	R
TYPE	(F9)

If the writing machine control panel is wired so that F8 causes the ribbon to shift to red, and F9 causes the ribbon to shift to black, this sequence of instructions can be used to print credit amounts in red.

b. FORMAT CONTROL INSTRUCTIONS. The format control instructions are Space, Backspace, Dash, Tabulate, Carriage Return, Upper Case, and Lower Case. They permit processor control of format for printing output. For example, these instructions are used to control the printing on the document and to ensure that the data in various columns is correctly aligned. Most of these instructions can be performed without execution of a Type command. However, two of them, Space and Dash, are not instructions in the normal sense, but "macro" instructions. A macro is a group of codes used as one instruction. "SPA" is actually a Type command followed by a space as the address. Likewise, "DAS" uses a Type command with a minus sign, or dash, as the address.

(1) SPACE. Execution of the Space instruction causes the carriage on the writing machine to advance one space.

(2) BACKSPACE. This instruction causes the carriage of the writing machine to move backward one space.

(3) DASH. When the Dash instruction is executed, the dash symbol is transmitted to the writing machine or its auxiliary equipment for printing or punching.

(4) TABULATE. Execution of the Tabulate instruction causes the carriage of the writing machine to advance to the next tab stop.

(5) CARRIAGE RETURN. When the Carriage Return instruction is executed, the carriage of the writing machine returns to the margin.

During execution of a Tabulate or Carriage Return instruction, the processor continues to execute the succeeding instructions in the program while the carriage is in motion, until an additional instruction specifying a writing machine function is to be executed. For example, Figure 2-37 illustrates appropriate positions of the Tabulate and Carriage Return instructions in a sequence. In this case, the Multiply command (line 005) is executed while the carriage is advancing to the next tab stop. The Duplicate, Accumulate and Store commands (lines 008 - 010) are executed while the carriage is returning to the margin.

If Reader 2 will go on and Reader 1 will be shut off (if on) when Reader 2 on is received.

Line No.	Command Statement	Address
001	ENTER	30
002	TABULATE	
003	ENTER	42
004	TABULATE	
005	MULTIPLY	
006	DECIMAL PRINT	72
007	CARRIAGE RETURN	
008	DUPLICATE	
009	ACCUMULATE	08
010	STORE	09

HIDING PROCESSOR TIME THROUGH USE OF WRITING MACHINE FUNCTIONS

FIGURE 2-37

(6) UPPER CASE. Execution of the Upper Case instruction causes the writing machine keyboard to shift to the upper case. It remains in this condition until a Lower Case instruction is executed or until the keyboard is manually shifted to the lower case. It is important that the keyboard be placed in the upper case for execution of the Decimal Print, Non Decimal Print, and Print Sign commands, since the numerics and other required symbols are in the upper case of the writing machine keyboard.

(7) LOWER CASE. Execution of the Lower Case instruction causes the writing machine keyboard to shift to the lower case. It remains in this condition until an Upper Case instruction is executed, or until the keyboard is manually shifted to the upper case.

c. SPECIAL FLEXOWRITER FUNCTIONS. The Special Flexowriter Functions (Reader 1 On, Reader 2 On, Reader Off, Punch 1 On, Punch 2 On, Punch Off, Non Print, and Print Restore) permit the control of reading, printing, and punching, by the processor during the execution of a program.

(1) READER 1 ON. When the Reader 1 On instruction is executed, the 2205 reader is placed in an "on" condition, but does not begin reading. If the reader is in an "on" condition, execution of an Enter instruction will initiate the reading action. The same rules apply to entries from the 2205 reader as to entries from the keyboard. As soon as the entry is terminated by the application of one of the rules applying to the Enter command, the reading action stops, but the reader remains in an "on" condition. Similarly, execution of a Halt command when the reader is in an "on" condition initiates the reading action; reading will continue until the "halt" condition is terminated. Depressing Start Read has the same effect as executing the Reader 1 On instruction.

(2) READER 2 ON. An auxiliary reader can be placed in an "on" condition by execution of the Reader 2 On instruction. The same rules apply to this instruction as to the Reader 1 On instruction, except that the auxiliary reader, rather than the 2205 reader, is conditioned. This "on" condition can be terminated by execution of the Reader Off instruction, or by manual depression of the Stop Read switch on the 2205 or the auxiliary reader.

Only one reader can be in an "on" condition at a time. However, the "on" condition can be transferred from one reader to another directly, without execution of an intervening Reader Off instruction. Execution of a Reader 1 On instruction, when the auxiliary reader is in an "on" condition, will transfer this "on" condition to the 2205 reader. Similarly, execution of a Reader 2 On instruction, when the 2205 reader is "on", will transfer this condition to the auxiliary reader. (If no auxiliary reader is attached, a Reader 2 On instruction has the same effect as a Reader 1 On instruction.) A Reader 1 On instruction, a Reader 2 On or a Reader Off instruction will have no effect if the required reader is in the specified condition before command execution.

The programmer must exercise a certain amount of care when a reader is used for entering data. For example, assume the following instructions are executed:

```
READER 1 ON
ENTER          32
```

If the entry in the reader contained 4 digits to the left of the decimal and 2 to the right, the data would be entered incorrectly. In this case, the first 3 digits would be entered left-of-decimal, and the fourth digit right-of-decimal. The entry of the decimal point would terminate the entry. The data entered would be incorrectly aligned in R1, and the 2 digits right-of-decimal would remain in the reader. The programmer must be certain that the Enter XX instruction has a sufficiently large LRD pattern for all entries to be made during program execution, or that a backspace is used for those entries which exceed the LRD pattern.

(3) READER OFF. Execution of the Reader Off instruction turns all readers off. When a reader entry is made, termination of the entry causes the reader to stop reading, but leaves the reader in an "on" condition; this condition remains until a Reader Off instruction is executed, or the operator depresses the Stop Read switch.

Successive entries may be made from a reader after it has been placed in an "on" condition. After completion of the last entry, a Reader Off instruction should be executed to prevent reading during execution of succeeding commands in the program. The following sequence illustrates the use of the reader control instructions.

```
READER 1 ON
ENTER          50
ENTER          42
READER OFF
```

Both entries are made from the 2205 reader in this case, because the reader has been conditioned by a Reader 1 On. After completion of both entries, execution of the Reader Off terminates the "on" condition and prevents reading action during subsequent program steps.

(4) PUNCH 1 ON. When the Punch 1 On instruction is executed, the 2205 punch will be turned on, providing that the Punch Control switch is in the "Select" position.

(5) PUNCH 2 ON. When the Punch 2 On instruction is executed, the auxiliary punch will be turned on, providing that the Auxiliary Punch Control switch is in the "Select" position.

(6) PUNCH OFF. When the Punch Off instruction is executed, both punches will be turned off, providing that the Punch Control switches are in the "Select" position.

The Punch Off instruction affects both punches, since both may be ON at any given time. Thus, if both punches are on, and one of them is to be turned off, the Punch Off instruction must be followed by a Punch On instruction designating the punch which is to be turned on.

(7) NON PRINT. Execution of the Non Print instruction places the 2205 in a non print mode of operation.

(8) PRINT RESTORE. When the Print Restore instruction is executed, the 2205 is restored to a normal print condition.

Figure 2-38 illustrates the use of the functional control instructions and the effect of each on the operating registers. This sequence involves the entry of data from both readers, and causes the result to be punched in the 2205 punch. Because the 2205 is in non print throughout the program, no printing occurs.

This program could be used, for example, to read a quantity from the 2205 reader, a price from the auxiliary reader, and cause the net amount to be punched in the 2205 punch. Because the punch is on during execution of the Carriage Return instruction, the carriage return code will be reproduced in the punch after the net amount. No other function codes will punch.

5610 MNEMONIC TRANSLATOR PROGRAMMING SHEET

COMMAND STATEMENT	ADDRESS	IN 1	IN 2	OUT 1	OUT 2	R1	R2	R3	R4
NON PRINT						---	---	---	
READER 1 ON		X				---	---	---	
ENTER	30	X				00000025.00000	---	---	
READER 2 ON			X			00000025.00000	---	---	
ENTER	42		X			00000002.75000	25.	---	
READER OFF						00000002.75000	25.	---	
MULTIPLY						00000068.75000	---	---	
PUNCH 1 ON				X		00000068.75000	---	---	
DECIMAL PRINT	52			X		00000068.75000	---	---	
CARRIAGE RETURN				X		00000068.75000	---	---	
PUNCH OFF						00000068.75000	---	---	
PRINT RESTORE						00000068.75000	---	---	

Prepared For _____ (Customer) Equipment _____

FRIDEN Office _____ Date Prepared _____ Page _____ Of _____ Pages

SP 9273

PRINTED IN U.S.A.

USE OF FUNCTIONAL CONTROL INSTRUCTIONS

FIGURE 2-38

Execution of a functional control instruction is independent of the wiring in the control panel of the 2205. These instructions are concerned only with the functions of the 2205 and cannot be used for the reproduction of their respective codes in the punch. Thus, for example, execution of the Print Restore instruction causes the 2205 to be restored to a normal print condition, but the print restore code is not reproduced in the punch, even if the punch is in an "all" condition. In order for the codes of these commands to be reproduced in the punch, the 2205 control panel must be specifically wired for this purpose, and a Type instruction must be executed.

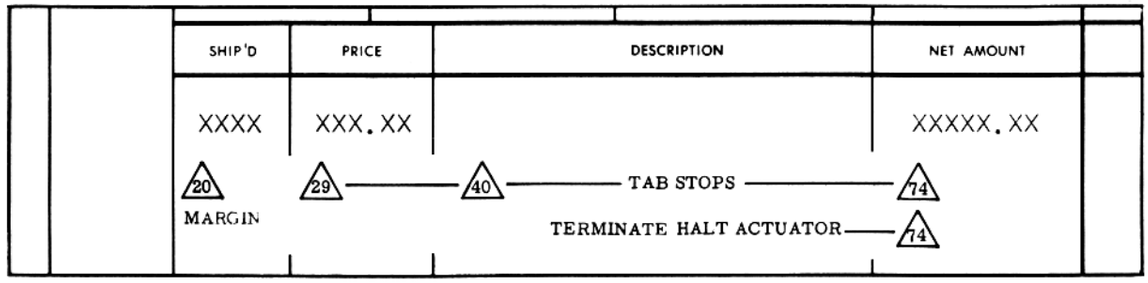
Each of the 13 function code keys on the writing machine keyboard may be wired in the control panel for any specific function required in a specific application, and a Type instruction can be used for the execution of this function in the program.

d. HALT. When a Halt instruction is executed, the processor interrupts the program and relinquishes control to the writing machine and its auxiliary equipment. During the "halt" condition, all writing machine operations are independent of the processor so that no input of data can occur.

The Halt instruction is used in a program when printing is required on the document, but no input is to be made to the central processor. For example, a "halt" condition is initiated when the description for a line of an invoice is printed. This description can come from the keyboard or a reader, and follows automatically from a reader if that reader is in an "on" condition prior to initiation of the Halt instruction.

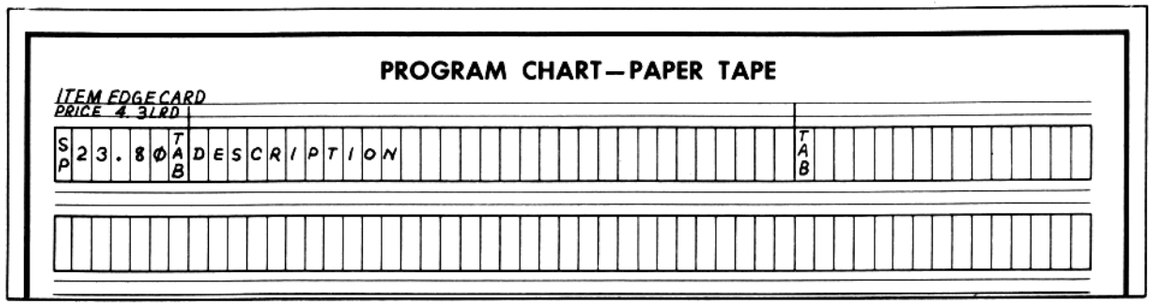
The "halt" condition is terminated by applying a pulse to the R16 hub in the 2205 program panel. Almost any means may be used to apply this pulse, but it is usually done via a field switch or a function code. The method to be used is determined by the actual wiring of the control panel, and is dependent upon program requirements.

The following program for a line of an invoice form illustrates the use of the format control and Halt instructions. The layout of the invoice form is shown in Figure 2-39, and the layout for the edge card to be used in preparing each line of the invoice is shown in Figure 2-40.



LAYOUT OF INVOICE FORM

FIGURE 2-39



LAYOUT OF EDGE CARD FOR INVOICE

FIGURE 2-40

In accordance with these customer requirements, the programmer develops the program shown in Figure 2-41.

Because no reader has been conditioned at the beginning of the program, the first entry can come from the keyboard. When this entry is terminated, the processor executes the Tabulate instruction to position the form for the entry of the price. The price is entered from the 2205 reader, since it has been conditioned. A "33" LRD pattern has been assigned to this entry to include the tab code in the edge card. The description follows from the reader since it remains in an "on" condition. The tab code following the description in the edge card causes the carriage to move to the halt actuator, thus restoring control of the program to the processor. The net amount remains in R1 and can be printed in the Net Amount column; a carriage return will return the carriage for the next line.

A description of the operating registers during execution of this program is also given in Figure 2-41. It may be noted that the format control commands have no effect on the operating registers. A typical output for this program is shown in Figure 2-42.

5610 MNEMONIC TRANSLATOR PROGRAMMING SHEET

COMMAND STATEMENT	ADDRESS	IN 1	IN 2	OUT 1	OUT 2	R1	R2	R3	R4
ENTER	40					00000012.00000	---	---	---
TABULATE						00000012.00000	---	---	---
READER 1 ON		X				00000012.00000	---	---	---
ENTER	33	X				00000023.80000	12.0	---	---
MULTIPLY		X				00000285.60000	---	---	---
HALT		X				00000285.60000	---	---	---
READER OFF						00000285.60000	---	---	---
DECIMAL PRINT	52					00000285.60000	---	---	---
CARRIAGE RETURN						00000285.60000	---	---	---

PROGRAM FOR INVOICE LINE

FIGURE 2-41

	SHIP'D	PRICE	DESCRIPTION	NET AMOUNT	
	12	23.80	PIN FEED PLATENS #628	285.60	

USE OF HALT INSTRUCTION

FIGURE 2-42

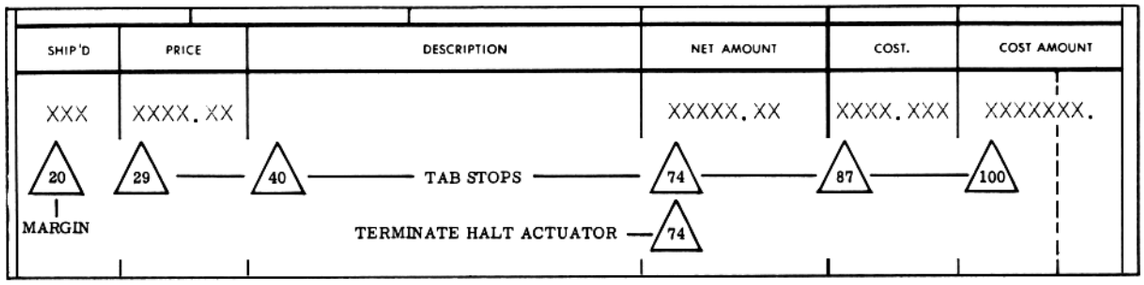
9. SAMPLE PROGRAM

The following program for one line of a sell-cost invoice illustrates use of many of the preceding instructions. This invoice is to be completed according to these customer specifications:

a. The quantity is positive, and is entered from the keyboard.

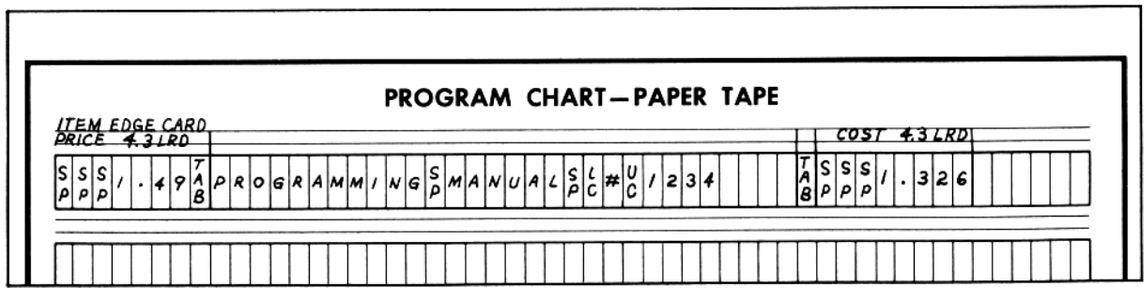
b. The price, description, and cost are read from edge cards.

The layout for the invoice form is given in Figure 2-43, and the layout for the edge cards in Figure 2-44. The instructions used and the condition of the operating registers during execution are shown in Figure 2-46. The output for the program is presented in Figure 2-45.



LAYOUT OF INVOICE FORM

FIGURE 2-43



LAYOUT OF EDGE CARD FOR INVOICE

FIGURE 2-44

SHIP'D	PRICE	DESCRIPTION	NET AMOUNT	COST.	COST AMOUNT
100	1.49	PROGRAMMING MANUAL #1234	149.00	1.326	132.60

INVOICE LINE PRODUCED FROM SAMPLE PROGRAM

FIGURE 2-45

5610 MNEMONIC TRANSLATOR PROGRAMMING SHEET

COMMAND STATEMENT	ADDRESS	IN 1	IN 2	OUT 1	OUT 2	R1	R2	R3	R4
ENTER	30					00000100.00000	---	---	---
DUPLICATE						00000100.00000	100.	---	---
TABULATE						00000100.00000	100.	---	---
READER 1 ON		X				00000100.00000	100.	---	---
ENTER	43	X				00000001.49000	100.	100.	---
HALT		X				00000001.49000	100.	100.	---
MULTIPLY		X				00000149.00000	100.	---	---
DECIMAL PRINT	52	X				00000149.00000	100.	---	---
TABULATE		X				00000149.00000	100.	---	---
ACCUMULATE	42	X				00000100.00000	---	---	---
ENTER	43	X				00000001.32600	100.	---	---
READER OFF						00000001.32600	100.	---	---
TABULATE						00000001.32600	100.	---	---
MULTIPLY						00000132.60000	---	---	---
CONSTANT	.005(space)					00000000.00500	132.60	---	---
ADD						00000132.60500	---	---	---
TRUNCATE	3					00000132.60000	---	---	---
NON DECIMAL PRINT	52					00000132.60000	---	---	---
CARRIAGE RETURN						00000132.60000	---	---	---
ACCUMULATE	44					---	---	---	---

Prepared For _____ Customer Equipment _____

FRIDEN Office _____ Date Prepared _____ Page _____ Of _____ Pages

SP 9273

PRINTED IN U.S.A.

FIGURE 2-46 PROGRAM FOR COST-SELL INVOICE

10. TRANSLATION FORMAT COMMANDS

The translation format commands (Translate, End, and Finish) are required in all programs which are to be translated by the SWIFT translator. These commands do not translate into any 5610 object codes; their function is restricted to control of translation by the SWIFT translator.

a. TRANSLATE. The Translate command is used to initiate translation. Execution of this command clears the storages required for Total Object Count, Line Count, and Sequence Object Count. In addition, execution of the Translate command permits typing of a description of the program.

b. END. The End command is used to indicate the end of a program sequence. When this instruction is executed, the processor identifies the end of the program sequence, provides the Sequence Object Count and the Total Object Count, and continues to translate the instructions in the next sequence.

c. FINISH. The Finish command terminates translation. When executed, this instruction identifies the end of the program, provides the Total Object Count, and turns off the reader.

d. MNEMONIC. The Mnemonic instruction is used to identify program sequences which are to be executed only in the translate mode. This instruction is translated by the SWIFT translator into an object code which separates the program sequences in the translate mode.

The Mnemonic command requires a 3-character address. These characters may be any writing machine codes except the carriage return, tab, and tape feed codes. The space code may be used only as the third character of the address, or as the second and third characters. However, a 3-character address consisting of 3 spaces, or having a space to the left of another character, is unacceptable.

In the translate mode, the Mnemonic instruction performs the same function as did the sequence instruction in the operate mode. When the processor is not executing a program sequence, the Sequence light will be on, and a 3-character address may be entered. The address may include other characters for description, since only the mnemonics obtained by the three methods of search are relevant.

If an invalid mnemonic is obtained by all 3 methods of search, the processor will search for the error routine and execute the instructions in this sequence. For this reason, an error sequence must be included in every program.

Both the Sequence and Mnemonic instructions can be used in a program to be executed in the translate mode. However, only those program sequences which are identified by a Mnemonic can be initiated by the entry of codes from the writing machine when the processor is awaiting codes to identify a program sequence. The Sequence instruction is required for those program sequences which are to be executed after execution of a Branch instruction.

11. ERROR SEQUENCE

An error sequence must be included in every 5610 program, whether executed in the operate or translate mode. The instructions in the error sequence will be executed if: invalid sequence codes are used to identify a program sequence, a Branch instruction specifying the error sequence is executed, or, an invalid Branch instruction is executed.

If a program does not contain an error sequence, entry of an invalid sequence code or execution of an invalid Branch instruction will cause the processor to search indefinitely for the error sequence. In this case, it is necessary to reload the program. If the error sequence is included in the program, detection of an error condition by the processor is followed by execution of the error sequence.

The error sequence must have a space as the address of the Sequence command. However, the error sequence can not be executed by the entry of a space from the writing machine, when the processor is awaiting a code to identify a program sequence.

The instructions to be executed in the error sequence are determined by the specific application. A typical error routine might be as follows:

SEQUENCE	(space)
READER OFF	
CARRIAGE RETURN	
CARRIAGE RETURN	
TYPE	E
TYPE	R
TYPE	R
TYPE	O
TYPE	R
CARRIAGE RETURN	
CARRIAGE RETURN	
PUNCH OFF	
END	

The instructions in this sequence can be modified to meet the requirements of an application. Therefore, the programmer can develop an error routine to meet these requirements. Additional error routines can be included in a program, but only the error sequence can be executed automatically by the processor when an invalid sequence code is entered, or an invalid Branch instruction is executed.

D. ADVANCED PROGRAMMING

There are certain advanced techniques used by programmers with which the service technician should be acquainted. Space does not permit a complete listing of these techniques here: instead a few representative samples are presented to provide the techinican with an idea of what might be expected in this area.

1. OPERATING IN "TRANSLATE"

The 5610 can operate in "Translate" as well as in its normal operating condition: this is the case when using the SWIFT translator. During Translate there are major differences in the way sequences are selected, and minor differences in the way some commands (e.g., Branch, Enter) are executed. These differences are covered in detail in Section 7.

2. "PROGRAM OVERLAY"

"Program overlay" is a term used to describe the process of loading a second program into the machine without disturbing the totals developed as a result of operating the first program. An example of this might be a billing program which uses approximately 1000 program characters. After the billing is accomplished, a second program could be loaded and run, using the totals developed by the first program, to compute commissions, taxes, etc. The second program is necessary when the total number of characters in both programs exceeds 1119.

3. INSTRUCTIONS NOT INCLUDED IN SWIFT

There are a few instructions that the 5610 can carry out that are not included in the SWIFT translator. They are 2205 functions that need not be preceded by a Type command. Some such instructions (Tab, Backspace, Carriage Return, Upper Case, etc.) are included in SWIFT, but there are six others that are not: F7, F8, F9, F10, F12, and Comma. These instructions may be used alone or as the address of a Type command: in either case they will cause the operation of their respective keylevers.

4. "MACRO" INSTRUCTIONS

Some instructions that appear in the SWIFT translator are not instructions in the normal sense, but "macro" instructions. A macro is actually a group of codes used as one instruction. Two macros used in the SWIFT translator are Space and Dash: SPA is actually a Type command followed by a space code as the address. Likewise, DAS uses a Type command with a minus sign, or dash, as the address. Other macros used in SWIFT are TRA, END, and FIN, which do not translate into object codes. Still other macros can be used, but only those mentioned are a part of SWIFT.

5. AUTOMATIC PRINTING OF DECIMAL DURING ENTRY

A decimal cannot be made to print during any single entry, but by providing a short subroutine the effect of the automatic decimal can be accomplished. Consider the following sequence:

```
ENTER          50
TYPE           .
ENTER          03
ADD
```

These four instructions allow a 5/3 entry and print the decimal automatically. The Add is necessary because the second Enter command shifts the stack up, causing the first entry to move into R2.

Note, however, that the preceding sequence will not work with negative numbers. The following sequence of instructions will print the decimal and function equally well with either positive or negative numbers. If the second entry is terminated with a minus sign, the entire entry becomes negative.

```
CONSTANT       .00001
ENTER          50
TYPE           .
MULTIPLY
Z ENTER        04
```

In this case the code for the letter "Z" is used for the second Enter command, instead of the "/". The "Z" causes the same operation as the "/", except the shift up of the operating stack is eliminated. The constant .00001 can be retrieved from the program line any time prior to the Multiply, and is necessary to "shift" the first entry to the LSD (Least Significant Digit) side of R1. As the second entry is completed, proper decimal alignment is achieved.

E. PREPARATION OF SOURCE PROGRAM TAPE

Preparation of the source program in tape form eliminates the chance of errors during translation and provides a tape documented program that lends itself to simpler program modification. Preparation of the source program can be done off line or on the 2205 with the processor in the type only mode.

A layout of a 2205 control panel is shown in Figure 2-47, which includes all patching necessary for program tape preparation, translation, reproduction of tapes and program loading.

The following list of instructions includes all steps necessary to prepare source program tapes for use with the SWIFT translator.

1. Place the 5610 processor in the Type Only mode. This provides for off line use of the 2205.
2. Place the Punch Select switch in the "All" position.
3. Depress Tape Feed to provide a leader on the source program tape.
4. Restore all program modification switches on the 2205.
5. Punch the source program tape in the following manner.

a. Type "TRANSLATE" as the first command of the program. A description of the program to be translated can follow this command. End the description with a tab to the address column and then carriage return.
b. Type "SEQUENCE" to identify the beginning of a program sequence. A description of the sequence may be typed prior to tabbing to the address column. End the description and tab to the address column.
c. Type the single character address of the sequence. Then carriage return.
d. Repeat the following for each command in the sequence.
   (1) Type the command and the description prior to tabbing to the address column.
   (2) Tab to the address column and type the appropriate address.
   (3) Carriage return to end the instruction.
e. After all commands and addresses in a sequence have been typed, type "END". Then tab and carriage return to identify the end of a program sequence.
f. Repeat steps b, c, d and e, for all sequences in the program.
g. At the end of the entire program, type "FINISH", tab and carriage return.

A1 - D1
A2 - A7
A15 - G12

B1 - D9
B7 - C7 (May be removed for select punching)

C1 - G1
C2 - D3
C2 - E2
C15 - G16

D2 - E1
D3 - G13

E1 - G11

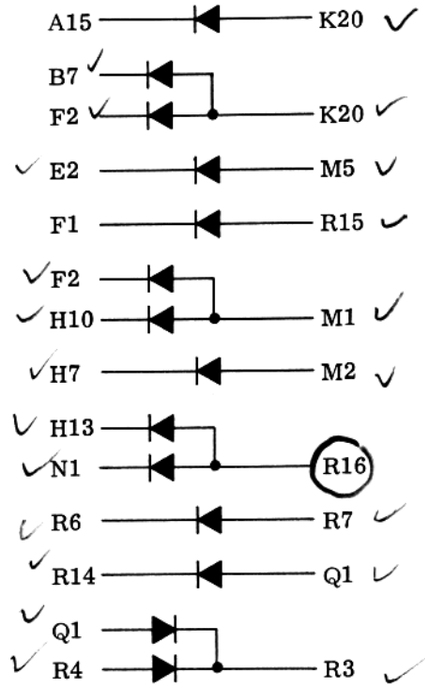
F1 - G10
F3 - K10
F9 - K9

G7 - G8
G8 - G13
G10 - G12

H1 - K1
H8 - M6
H11 - K8
H12 - M5
H16 - P1

M2 - R20

R5 - R6
R7 - R17



Prog Mod Switch 1 - Load and Translate

Prog Mod Switch 2 - Reverse Translate

Prog Mod Switch 3 - Add 8 Bit

Tape Skip Switch initiates "Medium Speed Read" during load. (Does not stop on F1 codes).

Auxiliary Selector 1 - Halt Control

Patch Hub R14 - Provides a pulse from all codes that are patched to reproduce. In the case of translation, all codes reproduce. Thus, all codes emit a pulse at R14.

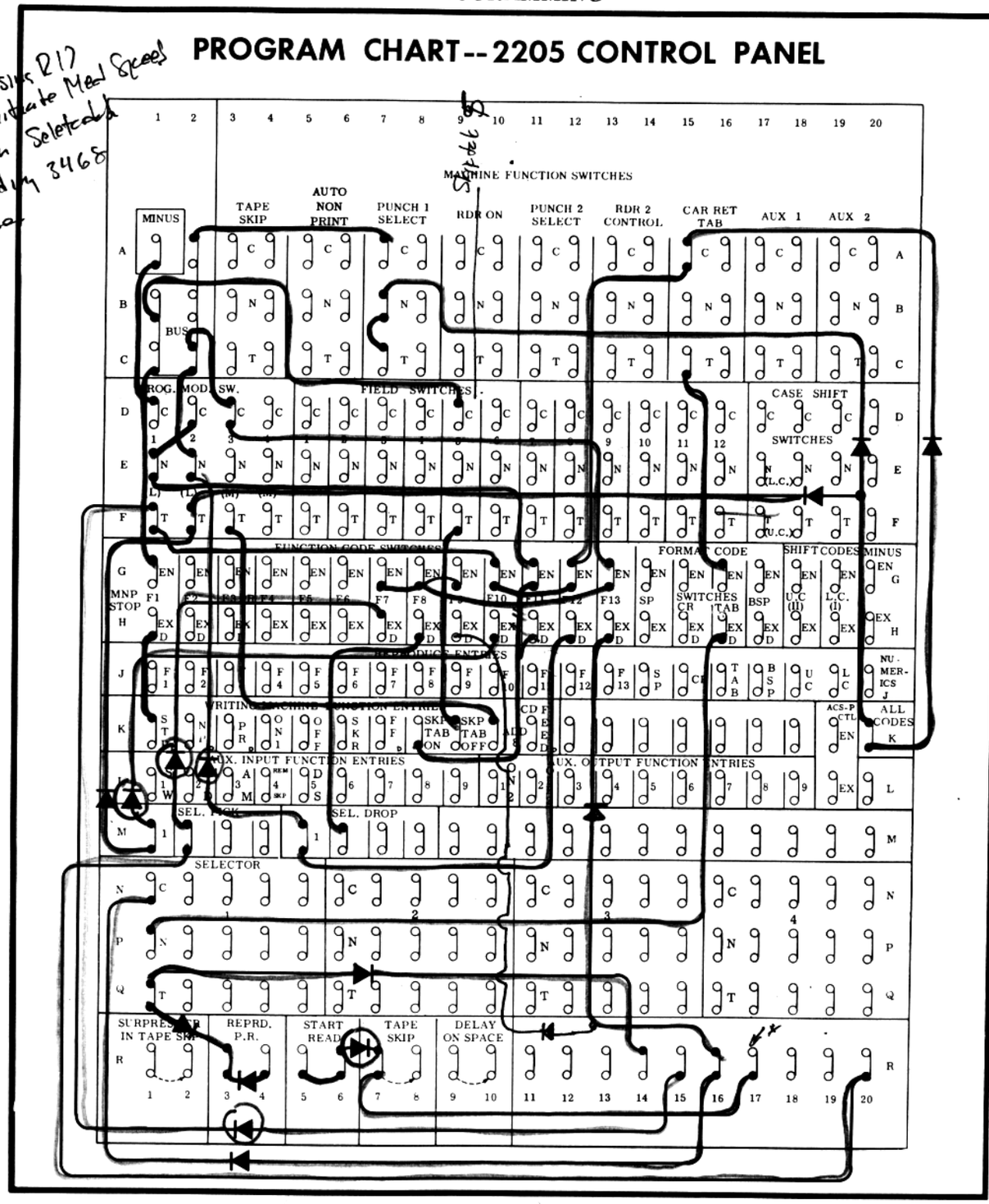
Note: This layout provides control of Halt during translation without the use of field switches.

SWIFT 2205 PATCH LIST

FIGURE 2-47 (a)

TP-161SL

PROGRAM CHART--2205 CONTROL PANEL



Prepared For _____ Customer Equipment _____

FRIDEN Office _____ Date Prepared _____ Page _____ Of _____ Pages

SP 9275

STANDARD SWIFT 2205 PROGRAM PANEL LAYOUT

PRINTED IN U.S.A.

FIGURE 2-47 (b)

```
TRANSLATE

SEQUENCE                          A
HALT
READER OFF
RECALL                            00
NON DECIMAL PRINT                 20
TYPE                              /
NON DECIMAL PRINT                 02
TYPE                              /
CONSTANT                          100
MULTIPLY
NON DECIMAL PRINT                 02
TABULATE
RECALL                            01
NON DECIMAL PRINT                 50
TABULATE
CONSTANT                          1
ACCUMULATE                        01
READER 1 ON
ENTER                             50
TABULATE
READER OFF
ENTER                             60
TABULATE
ENTER                             40
READER 1 ON
HALT
READER OFF
END

SEQUENCE
ENTER                             24
STORE                             00
ENTER                             50
CARRIAGE RETURN
END

FINISH
```

SAMPLE SOURCE PROGRAM

FIGURE 2-48

F. SOURCE PROGRAM TRANSLATION

The purpose of translation in the 5610 is to convert source programs, written with English statements, into machine (object) language. Translation in the 5610 is a function of both hardware and software, requiring that the SWIFT translator be loaded on the program line of the 5610 before translation can occur.

Instructions for translation:

1. Place 5610 processor in Load mode.
2. Place Punch Select switch in "Select" position.
3. Insert SWIFT translator program tape in reader.
4. Depress Program Modification switch #1 on 2205.

5. Depress Non Print.
6. After SWIFT tape has been loaded, place 5610 in Translate mode.
7. Depress Tape Feed to provide leader on object tape.
8. Insert source program in reader on tape feed codes.
9. Depress Start Read.
10. Translation of source program to object program will be automatic.

```
TRANSLATE

SEQUENCE                HA          001
HALT                    Q           002
READER OFF              F1          003
RECALL                  V00         004
NON DECIMAL PRINT       T20         005
TYPE                    ½/          006
NON DECIMAL PRINT       T02         007
TYPE                    ½/          008
CONSTANT                J100        009
MULTIPLY                0           010
NON DECIMAL PRINT       T02         011
TABULATE                            012
RECALL                  V01         013
NON DECIMAL PRINT       T50         014
TABULATE                            015
CONSTANT                J1          016
ACCUMULATE              W01         017
READER 1 ON             F11         018
ENTER                   /50         019
TABULATE                            020
READER OFF              F1          021
ENTER                   /60         022
TABULATE                            023
ENTER                   /40         024
READER 1 ON             F11         025
HALT                    Q           026
READER OFF              F1          027
END

     56 S
     56 T

SEQUENCE F1             H           028
ENTER                   /24         029
STORE                   S00         030
ENTER                   /50         031
CARRIAGE RETURN         +           032
END

     12 S
     68 T

FINISH

     68 T
```

SAMPLE SWIFT TRANSLATION

FIGURE 2-49

G. REVERSE TRANSLATION

The reverse translate program can be a valuable troubleshooting tool. With the reverse translator, the technician can convert an object program tape to a pure source program tape in the SWIFT language. At the same time, a hard copy may be produced (Figure 2-50) which is helpful in becoming familiar with the program.

1. FEATURES

Other features of the reverse translate program are as follows:

a. All instructions punched in the source program tape are followed by a space and a stop code (F1). This allows off-line editing of the pure source program tape, for insertion of descriptive information prior to SWIFT translation.

b. The function codes listed below, that may appear in the object code tape as commands, are converted to addresses of Type commands for use with the SWIFT translator. During reverse translation an asterisk will type after any such conversions for ease of detection.

F7
F8
F9
F10
F11
Comma

c. All invalid codes in the object tape are detected and will stop the program without affecting the source program output tape.

2. OPERATING INSTRUCTIONS

a. Set 2205 punch switch to "Select".

b. Depress program modification switch #2.

c. Load program tape.

d. Depress Translate switch.

e. Type "T", then tab.

f. Restore Translate switch.

g. Insert object tape in reader; depress Start Read.*

h. When translation is completed, depress Translate switch.

i. Type "END", then tab.

* Reverse translation can be performed in manual non print to create a source program tape without printing. This will reduce the time required to create the source program tape.

```
T
      TRANSLATE

H     SEQUENCE                          A
      READER OFF
V     RECALL                            00
T     NON DECIMAL PRINT                 20
½     TYPE                              /
T     NON DECIMAL PRINT                 02
½     TYPE                              /
J     CONSTANT                          100
O     MULTIPLY
T     NON DECIMAL PRINT                 02
      TABULATE

V     RECALL                            01
T     NON DECIMAL PRINT                 50
      TABULATE

J     CONSTANT                          1
W     ACCUMULATE                        01
      READER 1 ON
/     ENTER                             50
      TABULATE

      READER OFF
/     ENTER                             60
      TABULATE

/     ENTER                             40
      READER 1 ON
Q     HALT
      READER OFF

H     END

      SEQUENCE
/     ENTER                             24
S     STORE                             00
/     ENTER                             50
+     CARRIAGE RETURN
      END

FINISH
```

SAMPLE REVERSE TRANSLATION OBJECT TO "SWIFT"

FIGURE 2-50

A second reverse translator is also available which converts the object tape to a mnemonic tape, listing only the mnemonics for each command instead of the complete English description. A sample hard copy output is shown in Figure 2-51.

```
T

TRANSLATE OBJECT TO MNEMONICS

H

      S
      T

    SEQ   A
Q   HAL
    ROF
V   REC   00
T   NDP   20
½   TYP   /
T   NDP   02
½   TYP   /
J   CON   100
O   MUL
T   NDP   02
    TAB

V   REC   01
T   NDP   50
    TAB

J   CON   1
W   ACC   01
    R1O
/   ENT   50
    TAB

    ROF
/   ENT   60
    TAB

/   ENT   40
    R1O
Q   HAL
    ROF

H

    56S
    56T

    SEQ
/   ENT   24
S   STO   00
/   ENT   50
+   CRE
    END

    12S
    68T
```

SAMPLE REVERSE TRANSLATION OBJECT TO MNEMONICS

FIGURE 2-51

Command	Object	Line No.
MNEMONIC FOR MNEMONIC COMMAND	LMNE	001
HALT FOR DESCRIPTION	Q	002
PUNCH 1 ON FOR OBJECT CODE	F4	003
TYPE OBJECT CODE	½L	004
ENTER FIRST CHARACTER OF ADDRESS	/↑0	005
ENTER SECOND CHARACTER OF ADDRESS	/↑0	006
ENTER THIRD CHARACTER OF ADDRESS	/↑0	007
PUNCH OFF FOR PRINTING ONLY	F6	008
CONSTANT FOR OBJECT COUNT	J4	009
TABULATE TO LINE COUNT COLUMN		010
BRANCH FOR LINE AND OBJECT COUNT	EW	011
END OF SEQUENCE		
24 S		
24 T		
MNEMONIC FOR SEQUENCE COMMAND	LSEQ	012
HALT FOR DESCRIPTION	Q	013
TYPE F10 FOR HALT CONTROL	½	014
PUNCH 1 ON FOR OBJECT CODE	F4	015
TYPE OBJECT CODE	½H	016
PRINT RESTORE FOR DELAY	F3	017
PRINT RESTORE FOR DELAY	F3	018
HALT FOR SINGLE-CHARACTER ADDRESS	Q	019
PUNCH OFF FOR PRINTING ONLY	F6	020
UPPER CASE FOR PRINTING CONTROL		021
BRANCH FOR LINE AND OBJECT COUNT	EY	022
END OF SEQUENCE		
17 S		
41 T		
MNEMONIC FOR ADD COMMAND	LADD	023
HALT FOR DESCRIPTION	Q	024
PUNCH 1 ON FOR OBJECT CODE	F4	025
TYPE OBJECT CODE	½F	026
BRANCH FOR LINE AND OBJECT COUNT	EX	027
END OF SEQUENCE		
10 S		
51 T		
MNEMONIC FOR SUBTRACT COMMAND	LSUB	028
HALT FOR DESCRIPTION	Q	029
PUNCH 1 ON FOR OBJECT CODE	F4	030
TYPE OBJECT CODE	½B	031
BRANCH FOR LINE AND OBJECT COUNT	EX	032
END OF SEQUENCE		
10 S		
61 T		

TP-161SL TRANSLATE SWIFT TRANSLATOR

FIGURE 2-52

Command	Object	Line No.
MNEMONIC FOR MULTIPLY COMMAND	LMUL	033
HALT FOR DESCRIPTION	Q	034
PUNCH 1 ON FOR OBJECT CODE	F4	035
TYPE OBJECT CODE	½0	036
BRANCH FOR LINE AND OBJECT COUNT	EX	037
END OF SEQUENCE		
10 S		
71 T		
MNEMONIC FOR DIVIDE COMMAND	LDIV	038
HALT FOR DESCRIPTION	Q	039
PUNCH 1 ON FOR OBJECT CODE	F4	040
TYPE OBJECT CODE	½K	041
BRANCH FOR LINE AND OBJECT COUNT	EX	042
END OF SEQUENCE		
10 S		
81 T		
MNEMONIC FOR CHANGE SIGN COMMAND	LCSI	043
HALT FOR DESCRIPTION	Q	044
PUNCH 1 ON FOR OBJECT CODE	F4	045
TYPE OBJECT CODE	½-	046
BRANCH FOR LINE AND OBJECT COUNT	EX	047
END OF SEQUENCE		
10 S		
91 T		
MNEMONIC FOR DUPLICATE COMMAND	LDUP	048
HALT FOR DESCRIPTION	Q	049
PUNCH 1 ON FOR OBJECT CODE	F4	050
TYPE OBJECT CODE	½N	051
BRANCH FOR LINE AND OBJECT COUNT	EX	052
END OF SEQUENCE		
10 S		
101 T		
MNEMONIC FOR TRUNCATE COMMAND	LTRU	053
HALT FOR DESCRIPTION	Q	054
PUNCH 1 ON FOR OBJECT CODE	F4	055
TYPE OBJECT CODE	½I	056
ENTER SINGLE-DIGIT ADDRESS	/10	057
BRANCH FOR LINE AND OBJECT COUNT	EY	058
END OF SEQUENCE		
13 S		
114 T		

Command	Object	Line No.
MNEMONIC FOR DELETE COMMAND	LDEL	059
HALT FOR DESCRIPTION	Q	060
PUNCH 1 ON FOR OBJECT CODE	F4	061
TYPE OBJECT CODE	½D	062
BRANCH FOR LINE AND OBJECT COUNT	EX	063
END OF SEQUENCE		
10 S		
124 T		
MNEMONIC FOR STORE COMMAND	LSTO	064
HALT FOR DESCRIPTION	Q	065
PUNCH 1 ON FOR OBJECT CODE	F4	066
TYPE OBJECT CODE	½S	067
BRANCH FOR ADDRESS	ES	068
END OF SEQUENCE		
10 S		
134 T		
MNEMONIC FOR ACCUMULATE COMMAND	LACC	069
HALT FOR DESCRIPTION	Q	070
PUNCH 1 ON FOR OBJECT CODE	F4	071
TYPE OBJECT CODE	½W	072
BRANCH FOR ADDRESS	ES	073
END OF SEQUENCE		
10 S		
144 T		
MNEMONIC FOR RECALL COMMAND	LREC	074
HALT FOR DESCRIPTION	Q	075
PUNCH 1 ON FOR OBJECT CODE	F4	076
TYPE OBJECT CODE	½V	077
BRANCH FOR ADDRESS	ES	078
END OF SEQUENCE		
10 S		
154 T		
MNEMONIC FOR CONSTANT COMMAND	LCON	079
HALT FOR DESCRIPTION	Q	080
PUNCH 1 ON FOR OBJECT CODE	F4	081
TYPE OBJECT CODE	½J	082
ENTER CONSTANT FACTOR	/86	083
PUNCH OFF FOR PRINTING ONLY	F6	084
BRANCH FOR CONSTANT COUNT	EJ	085
END OF SEQUENCE		
14 S		
168 T		

Command	Object	Line No.
MNEMONIC FOR ENTER COMMAND	LENT	086
HALT FOR DESCRIPTION	Q	087
PUNCH 1 ON FOR OBJECT CODE	F4	088
TYPE OBJECT CODE	½/	089
ENTER 2-DIGIT ADDRESS	/20	090
BRANCH FOR LINE AND OBJECT COUNT	EZ	091
END OF SEQUENCE		
13 S		
181 T		
MNEMONIC FOR DECIMAL PRINT COMMAND	LDPR	092
HALT FOR DESCRIPTION	Q	093
PUNCH 1 ON FOR OBJECT CODE	F4	094
TYPE OBJECT CODE	½X	095
ENTER 2-DIGIT ADDRESS	/20	096
BRANCH FOR LINE AND OBJECT COUNT	EZ	097
END OF SEQUENCE		
13 S		
194 T		
MNEMONIC FOR NON DECIMAL PRINT COMMAND	LNDP	098
HALT FOR DESCRIPTION	Q	099
PUNCH 1 ON FOR OBJECT CODE	F4	100
TYPE OBJECT CODE	½T	101
ENTER 2-DIGIT ADDRESS	/20	102
BRANCH FOR LINE AND OBJECT COUNT	EZ	103
END OF SEQUENCE		
13 S		
207 T		
MNEMONIC FOR PRINT SIGN COMMAND	LPSI	104
HALT FOR DESCRIPTION	Q	105
PUNCH 1 ON FOR OBJECT CODE	F4	106
TYPE OBJECT CODE	½P	107
BRANCH FOR LINE AND OBJECT COUNT	EX	108
END OF SEQUENCE		
10 S		
217 T		
MNEMONIC FOR BRANCH COMMAND	LBRA	109
HALT FOR DESCRIPTION	Q	110
TYPE F10 FOR HALT CONTROL	½	111
PUNCH 1 ON FOR OBJECT CODE	F4	112
TYPE OBJECT CODE	½E	113
PRINT RESTORE FOR DELAY	F3	114
PRINT RESTORE FOR DELAY	F3	115
HALT FOR SINGLE-CHARACTER ADDRESS	Q	116
PUNCH OFF FOR PRINTING ONLY	F6	117

Command	Object	Line No.
UPPER CASE FOR PRINTING CONTROL		118
BRANCH FOR LINE AND OBJECT COUNT	EY	119
END OF SEQUENCE		
17 S		
234 T		
MNEMONIC FOR POSITIVE BRANCH COMMAND	LPBR	120
HALT FOR DESCRIPTION	Q	121
TYPE F10 FOR HALT CONTROL	½	122
PUNCH 1 ON FOR OBJECT CODE	F4	123
TYPE OBJECT CODE	½C	124
PRINT RESTORE FOR DELAY	F3	125
PRINT RESTORE FOR DELAY	F3	126
HALT FOR SINGLE-CHARACTER ADDRESS	Q	127
PUNCH OFF FOR PRINTING ONLY	F6	128
SPACE FOR MARKER	½	129
SPACE FOR MARKER	½	130
UPPER CASE FOR PRINTING CONTROL		131
TYPE + FOR POSITIVE BRANCH MARKER	½+	132
BRANCH FOR LINE AND OBJECT COUNT	EY	133
END OF SEQUENCE		
23 S		
257 T		
MNEMONIC FOR NEGATIVE BRANCH COMMAND	LNBR	134
HALT FOR DESCRIPTION	Q	135
TYPE F10 FOR HALT CONTROL	½	136
PUNCH 1 ON FOR OBJECT CODE	F4	137
TYPE OBJECT CODE	½G	138
PRINT RESTORE FOR DELAY	F3	139
PRINT RESTORE FOR DELAY	F3	140
HALT FOR SINGLE-CHARACTER ADDRESS	Q	141
PUNCH OFF FOR PRINTING ONLY	F6	142
SPACE FOR MARKER	½	143
SPACE FOR MARKER	½	144
UPPER CASE FOR PRINTING CONTROL		145
DASH FOR NEGATIVE BRANCH MARKER	½-	146
BRANCH FOR LINE AND OBJECT COUNT	EY	147
END OF SEQUENCE		
23 S		
280 T		
MNEMONIC FOR RESUME COMMAND	LRES	148
HALT FOR DESCRIPTION	Q	149
TYPE F10 FOR HALT CONTROL	½	150
PUNCH 1 ON FOR OBJECT CODE	F4	151
TYPE OBJECT CODE	½A	152
PRINT RESTORE FOR DELAY	F3	153
PRINT RESTORE FOR DELAY	F3	154

Command	Object	Line No.
HALT FOR SINGLE-CHARACTER ADDRESS	Q	155
PUNCH OFF FOR PRINTING ONLY	F6	156
SPACE FOR MARKER	½	157
SPACE FOR MARKER	½	158
LOWER CASE FOR * SYMBOL		159
TYPE ASTERISK FOR RESUME MARKER	½8	160
UPPER CASE FOR PRINTING CONTROL		161
BRANCH FOR LINE AND OBJECT COUNT	EY	162
END OF SEQUENCE		
24 S		
304 T		
MNEMONIC FOR RETURN COMMMAND	LRET	163
HALT FOR DESCRIPTION	Q	164
PUNCH 1 ON FOR OBJECT CODE	F4	165
TYPE OBJECT CODE	½R	166
BRANCH FOR LINE AND OBJECT COUNT	EX	167
END OF SEQUENCE		
10 S		
314 T		
MNEMONIC FOR READER 1 ON COMMAND	LR10	168
HALT FOR DESCRIPTION	Q	169
PUNCH 1 ON FOR OBJECT CODE	F4	170
TYPE OBJECT CODE (F11)	½	171
PUNCH OFF FOR PRINTING ONLY	F6	172
TYPE F FOR CODE IDENTIFICATION	½F	173
TYPE 1 FOR CODE IDENTIFICATION	½1	174
TYPE 1 FOR CODE IDENTIFICATION	½1	175
BRANCH FOR LINE AND OBJECT COUNT	EX	176
END OF SEQUENCE		
17 S		
331 T		
MNEMONIC FOR READER 2 ON COMMAND	LR20	177
HALT FOR DESCRIPTION	Q	178
PUNCH 1 ON FOR OBJECT CODE	F4	179
TYPE OBJECT CODE (F13)	½	180
PUNCH OFF FOR PRINTING ONLY	F6	181
TYPE F FOR CODE IDENTIFICATION	½F	182
TYPE 1 FOR CODE IDENTIFICATION	½1	183
TYPE 3 FOR CODE IDENTIFICATION	½3	184
BRANCH FOR LINE AND OBJECT COUNT	EX	185
END OF SEQUENCE		
17 S		
348 T		

Command	Object	Line No.
MNEMONIC FOR READER OFF COMMAND	LROF	186
HALT FOR DESCRIPTION	Q	187
PUNCH 1 ON FOR OBJECT CODE	F4	188
TYPE OBJECT CODE (F1)	½	189
PUNCH OFF FOR PRINTING ONLY	F6	190
TYPE F FOR CODE IDENTIFICATION	½F	191
TYPE 1 FOR CODE IDENTIFICATION	½1	192
BRANCH FOR LINE AND OBJECT COUNT	EX	193
END OF SEQUENCE		
15 S 363 T		
MNEMONIC FOR PUNCH 1 ON COMMAND	LP10	194
HALT FOR DESCRIPTION	Q	195
PUNCH 1 ON FOR OBJECT CODE	F4	196
TYPE OBJECT CODE (F4)	½	197
PUNCH OFF FOR PRINTING ONLY	F6	198
TYPE F FOR CODE IDENTIFICATION	½F	199
TYPE 4 FOR CODE IDENTIFICATION	½4	200
BRANCH FOR LINE AND OBJECT COUNT	EX	201
END OF SEQUENCE		
15 S 378 T		
MNEMONIC FOR PUNCH 2 ON COMMAND	LP20	202
HALT FOR DESCRIPTION	Q	203
PUNCH 1 ON FOR OBJECT CODE	F4	204
TYPE OBJECT CODE (F5)	½	205
PUNCH OFF FOR PRINTING ONLY	F6	206
TYPE F FOR CODE IDENTIFICATION	½F	207
TYPE 5 FOR CODE IDENTIFICATION	½5	208
BRANCH FOR LINE AND OBJECT COUNT	EX	209
END OF SEQUENCE		
15 S 393 T		
MNEMONIC FOR PUNCH OFF COMMAND	LPOF	210
HALT FOR DESCRIPTION	Q	211
PUNCH 1 ON FOR OBJECT CODE	F4	212
TYPE OBJECT CODE (F6)	½	213
PUNCH OFF FOR PRINTING ONLY	F6	214
TYPE F FOR CODE IDENTIFICATION	½F	215
TYPE 6 FOR CODE IDENTIFICATION	½6	216
BRANCH FOR LINE AND OBJECT COUNT	EX	217
END OF SEQUENCE		
15 S 408 T		

Command	Object	Line No.
MNEMONIC FOR NON PRINT COMMAND	LNPR	218
HALT FOR DESCRIPTION	Q	219
PUNCH 1 ON FOR OBJECT CODE	F4	220
TYPE OBJECT CODE (F2)	½	221
PUNCH OFF FOR PRINTING ONLY	F6	222
TYPE F FOR CODE IDENTIFICATION	½F	223
TYPE 2 FOR CODE IDENTIFICATION	½2	224
BRANCH FOR LINE AND OBJECT COUNT	EX	225
END OF SEQUENCE		
15 S		
423 T		
MNEMONIC FOR PRINT RESTORE COMMAND	LPRE	226
HALT FOR DESCRIPTION	Q	227
PUNCH 1 ON FOR OBJECT CODE	F4	228
TYPE OBJECT CODE (F3)	½	229
PUNCH OFF FOR PRINTING ONLY	F6	230
TYPE F FOR CODE IDENTIFICATION	½F	231
TYPE 3 FOR CODE IDENTIFICATION	½3	232
BRANCH FOR LINE AND OBJECT COUNT	EX	233
END OF SEQUENCE		
15 S		
438 T		
MNEMONIC FOR HALT COMMAND	LHAL	234
HALT FOR DESCRIPTION	Q	235
PUNCH 1 ON FOR OBJECT CODE	F4	236
TYPE OBJECT CODE	½Q	237
BRANCH FOR LINE AND OBJECT COUNT	EX	238
END OF SEQUENCE		
10 S		
448 T		
MNEMONIC FOR TYPE COMMAND	LTYP	239
HALT FOR DESCRIPTION	Q	240
TYPE F10 FOR HALT CONTROL	½	241
PUNCH 1 ON FOR OBJECT CODE	F4	242
TYPE OBJECT CODE	½½	243
PRINT RESTORE FOR DELAY	F3	244
PRINT RESTORE FOR DELAY	F3	245
HALT FOR SINGLE-CHARACTER ADDRESS	Q	246
PUNCH OFF FOR PRINTING ONLY	F6	247
UPPER CASE FOR PRINTING CONTROL		248
BRANCH FOR LINE AND OBJECT COUNT	EY	249
END OF SEQUENCE		
17 S		
465 T		

Command	Object	Line No.
MNEMONIC FOR SPACE COMMAND	LSPA	250
HALT FOR DESCRIPTION	Q	251
PUNCH 1 ON FOR OBJECT CODE	F4	252
TYPE OBJECT CODE	$\frac{1}{2}\frac{1}{2}$	253
SPACE FOR ADDRESS	$\frac{1}{2}$	254
BRANCH FOR LINE AND OBJECT COUNT	EY	255
END OF SEQUENCE		
12 S 477 T		
MNEMONIC FOR DASH COMMAND	LDAS	256
HALT FOR DESCRIPTION	Q	257
PUNCH 1 ON FOR OBJECT CODE	F4	258
TYPE OBJECT CODE	$\frac{1}{2}\frac{1}{2}$	259
DASH FOR ADDRESS	$\frac{1}{2}$-	260
BRANCH FOR LINE AND OBJECT COUNT	EY	261
END OF SEQUENCE		
12 S 489 T		
MNEMONIC FOR TABULATE COMMAND	LTAB	262
HALT FOR DESCRIPTION	Q	263
PUNCH 1 ON FOR OBJECT CODE	F4	264
TABULATE TO LINE COUNT COLUMN		265
PUNCH OFF FOR PRINTING ONLY	F6	266
TRUNCATE 13 (F8) FOR CLEARING 60 COUNTER	I	267
STORE IN 60 COUNTER	S04	268
CONSTANT FOR OBJECT COUNT	J1	269
BRANCH FOR LINE AND OBJECT COUNT	EW	270
END OF SEQUENCE		
18 S 507 T		
MNEMONIC FOR CARRIAGE RETURN COMMAND	LCRE	271
HALT FOR DESCRIPTION	Q	272
PUNCH 1 ON FOR OBJECT CODE	F4	273
TYPE OBJECT CODE	$\frac{1}{2}$+	274
BRANCH FOR LINE AND OBJECT COUNT	EX	275
END OF SEQUENCE		
10 S 517 T		
MNEMONIC FOR UPPER CASE COMMAND	LUCA	276
HALT FOR DESCRIPTION	Q	277
PUNCH 1 ON FOR OBJECT CODE	F4	278
UPPER CASE FOR OBJECT CODE		279
BRANCH FOR LINE AND OBJECT COUNT	EX	280
END OF SEQUENCE		
9 S 526 T		

Command	Object	Line No.
MNEMONIC FOR LOWER CASE COMMAND	LLCA	281
HALT FOR DESCRIPTION	Q	282
PUNCH 1 ON FOR OBJECT CODE	F4	283
LOWER CASE FOR OBJECT CODE		284
PUNCH OFF FOR PRINTING ONLY	F6	285
UPPER CASE FOR PRINTING CONTROL		286
BRANCH FOR LINE AND OBJECT COUNT	EX	287
END OF SEQUENCE		
11 S		
537 T		
MNEMONIC FOR BACKSPACE COMMAND	LBAC	288
HALT FOR DESCRIPTION	Q	289
PUNCH 1 ON FOR OBJECT CODE	F4	290
BACKSPACE FOR OBJECT CODE		291
BRANCH FOR LINE AND OBJECT COUNT	EX	292
END OF SEQUENCE		
9 S		
546 T		
MNEMONIC FOR CHECK PROTECT COMMAND	LCPR	293
HALT FOR DESCRIPTION	Q	294
PUNCH 1 ON FOR OBJECT CODE	F4	295
TYPE OBJECT CODE	½&	296
BRANCH FOR LINE AND OBJECT COUNT	EX	297
END OF SEQUENCE		
10 S		
556 T		
MNEMONIC FOR TRANSLATE COMMAND	LTRA	298
HALT FOR DESCRIPTION	Q	299
TYPE F12 TO RESET HALT CONTROL	½	300
TRUNCATE 13 (F8) FOR CLEARING COUNTERS	I	301
DUPLICATE TO R2	N	302
DUPLICATE TO R3	N	303
DUPLICATE TO R4	N	304
STORE IN LINE COUNTER	S01	305
STORE IN OBJECT COUNTER FOR SEQUENCE	S02	306
STORE IN TOTAL OBJECT COUNTER	S03	307
STORE IN PAGE COUNTER	S06	308
CONSTANT FOR 60 COUNTER	J60-	309
DUPLICATE TO R2	N	310
STORE IN 60 COUNTER	S04	311
STORE IN SWITCH FOR CARRIAGE CONTROL	S08	312
BRANCH FOR PRINTING PAGE NUMBER	EQ	313
END OF SEQUENCE		
37 S		
593 T		

Command	Object	Line No.
MNEMONIC FOR END COMMAND	LEND	314
HALT FOR DESCRIPTION	Q	315
PUNCH 1 ON FOR CARRIAGE RETURNS	F4	316
CARRIAGE RETURN FOR FORM CONTROL	+	317
CARRIAGE RETURN FOR FORM CONTROL	+	318
PUNCH OFF FOR PRINTING ONLY	F6	319
RECALL OBJECT COUNTER FOR SEQUENCE	V02	320
CONSTANT FOR CLEARING SEQUENCE COUNTER	J0	321
STORE IN SEQUENCE COUNTER	S02	322
NON DECIMAL PRINT SEQUENCE COUNT	T40	323
SPACE FOR FORM CONTROL	½	324
TYPE S FOR DESCRIPTION	½S	325
CARRIAGE RETURN FOR FORM CONTROL	+	326
CONSTANT FOR 60 COUNTER	J60-	327
STORE IN 60 COUNTER	S04	328
CONSTANT FOR UPDATING PAGE LINE COUNTER	J5	329
ACCUMULATE IN PAGE LINE COUNTER	W07	330
BRANCH FOR PRINTING TOTAL OBJECT COUNT	EV	331
RECALL PAGE LINE COUNTER	V07	332
CONSTANT FOR CHECKING COUNTER	J3	333
ADD TO PAGE LINE COUNT	F	334
CONSTANT FOR CARRIAGE CONTROL SWITCH	J1 -	335
STORE IN SWITCH	S08	336
BRANCH FOR CHECKING PAGE LINE COUNT	EP	337
END OF SEQUENCE		
56 S 649 T		
MNEMONIC FOR FINISH COMMAND	LFIN	338
HALT FOR DESCRIPTION	Q	339
CARRIAGE RETURN FOR FORM CONTROL	+	340
CARRIAGE RETURN FOR FORM CONTROL	+	341
BRANCH FOR TOTAL OBJECT COUNT	EV	342
CARRIAGE RETURN FOR FORM CONTROL	+	343
CARRIAGE RETURN FOR FORM CONTROL	+	344
READER OFF FOR END OF TRANSLATION	F1	345
END OF SEQUENCE		
12 S 661 T		
SEQUENCE FOR PRINTING TOTAL OBJECT COUNT	HV	346
RECALL TOTAL OBJECT COUNTER	V03	347
NON DECIMAL PRINT TOTAL OBJECT COUNT	T40	348
SPACE FOR FORM CONTROL	½	349
TYPE T FOR DESCRIPTION	½T	350
RETURN TO MARKER FOR BRANCH	R	351
END OF SEQUENCE		
13 S 674 T		

Command	Object	Line No.
SEQUENCE FOR OBJECT COUNT OF 1	HX	352
PUNCH OFF FOR PRINTING ONLY	F6	353
TABULATE TO LINE COUNT COLUMN		354
CONSTANT FOR OBJECT COUNT	J1	355
BRANCH FOR PRINTING LINE COUNT	EW	356
END OF SEQUENCE		
9 S 683 T		
SEQUENCE FOR OBJECT COUNT OF 2	HY	357
PUNCH OFF FOR PRINTING ONLY	F6	358
TYPE F12 FOR HALT CONTROL	½	359
TABULATE TO LINE COUNT COLUMN		360
CONSTANT FOR OBJECT COUNT	J2	361
BRANCH FOR PRINTING LINE COUNT	EW	362
END OF SEQUENCE		
11 S 694 T		
SEQUENCE FOR OBJECT COUNT OF 3	HZ	363
RESUME FOR CHECKING STORAGE ADDRESSES	AZ *	364
PUNCH OFF FOR PRINTING ONLY	F6	365
TABULATE TO LINE COUNT COLUMN		366
CONSTANT FOR OBJECT COUNT	J3	367
BRANCH FOR PRINTING LINE COUNT	EW	368
END OF SEQUENCE		
11 S 705 T		
SEQUENCE FOR PRINTING LINE COUNT	HW	369
CONSTANT FOR UPDATING LINE COUNT	J.001	370
RECALL LINE COUNTER	V01	371
ADD TO PREVIOUS LINE COUNT	F	372
NON DECIMAL PRINT LINE COUNT	T03	373
STORE NEW LINE COUNT	S01	374
ENTER CARRIAGE RETURN	/10	375
CHANGE SIGN FOR CHECKING CARRIAGE RETURN	-	376
NEGATIVE BRANCH IF ENTRY NON-ZERO	GB -	377
DELETE CARRIAGE RETURN ENTRY	D	378
DUPLICATE OBJECT COUNT TO R2	N	379
DUPLICATE OBJECT COUNT TO R3	N	380
ACCUMULATE OBJECT COUNT IN SEQUENCE COUNT	W02	381
ACCUMULATE OBJECT COUNT IN TOTAL COUNTER	W03	382
ACCUMULATE OBJECT COUNT IN 60 COUNTER	W04	383
RECALL 60 COUNTER	V04	384
NEGATIVE BRANCH IF LESS THAN 60	GA -	385
CONSTANT FOR SETTING COUNTER	J60-	386
STORE IN 60 COUNTER	S04	387
NON PRINT FOR CARRIAGE RETURN	F2	388

Command	Object	Line No.
PUNCH 1 ON FOR PUNCHING CARRIAGE RETURN	F4	389
CARRIAGE RETURN FOR OBJECT TAPE	+	390
PUNCH OFF FOR PRINTING ONLY	F6	391
PRINT RESTORE FOR NORMAL PRINT MODE	F3	392
RESUME FOR CHECKING PAGE LINE COUNTER	AA *	393
CONSTANT FOR INCREASING COUNTER	J1	394
DUPLICATE TO R2	N	395
ACCULULATE IN PAGE LINE COUNTER	W07	396
STORE IN CARRIAGE CONTROL SWITCH	S08	397
RECALL PAGE LINE COUNTER	V07	398
NEGATIVE BRANCH FOR TERMINATION	GC -	399
CONSTANT FOR CONTROL TO NEW PAGE	J11-	400
BRANCH FOR NEW PAGE	ER	401
END OF SEQUENCE		
76 S		
781 T		
SEQUENCE CONSTANT COUNT	HJ	402
TABULATE TO LINE COUNT COLUMN		403
NEGATIVE BRANCH FOR COUNTING	G1 -	404
CHANGE SIGN FOR ZERO CHECK	-	405
NEGATIVE BRANCH FOR COUNTING	G1 -	406
CONSTANT FOR COUNTING CONSTANT ZERO	J3	407
BRANCH FOR PRINTING LINE COUNT	EW	408
RESUME COUNTING FOR NON-ZERO CONSTANTS	A1 *	409
CONSTANT FOR MINIMUM OBJECT COUNT	J2	410
STORE IN OBJECT COUNTER FOR CONSTANT	S05	411
DUPLICATE CONSTANT TO R2	N	412
TRUNCATE FOR LEFT-OF-DECIMAL CHECK	I5	413
DUPLICATE LEFT-OF-DECIMAL	N	414
POSITIVE BRANCH FOR NO LEFT-OF-DECIMAL	C2 +	415
RESUME FOR LEFT-OF-DECIMAL COUNT	A3 *	416
CONSTANT FOR INCREASING COUNT	J1	417
ACCUMULATE IN CONSTANT COUNTER	W05	418
CONSTANT FOR SHIFTING	J10	419
DIVIDE TO SHIFT RIGHT 1 POSITION	K	420
TRUNCATE TO CHECK LEFT-OF-DECIMAL	I5	421
NEGATIVE BRANCH FOR ADDITIONAL DIGITS	G3 -	422
RESUME FOR RIGHT-OF-DECIMAL COUNT	A2 *	423
DELETE LEFT-OF-DECIMAL DIGITS	D	424
SUBTRACT TO ISOLATE RIGHT-OF-DECIMAL	B	425
POSITIVE BRANCH FOR NO RIGHT-OF-DECIMAL	C4 +	426
RESUME FOR RIGHT-OF-DECIMAL COUNT	A5 *	427
CONSTANT TO INCREASE COUNT	J1	428
ACCUMULATE IN CONSTANT COUNTER	W05	429
DUPLICATE FOR RIGHT-OF-DECIMAL	N	430
TRUNCATE TO ISOLATE LEFT-OF-DECIMAL	I5	431
SUBTRACT TO ISOLATE RIGHT-OF-DECIMAL	B	432
CONSTANT TO SHIFT LEFT	J10	433
MULTIPLY TO SHIFT LEFT 1 POSITION	O	434
NEGATIVE BRANCH FOR ADDITIONAL DIGITS	G5 -	435

Command	Object	Line No.
RESUME FOR END OF COUNT	A4 *	436
RECALL CONSTANT COUNT	V05	437
BRANCH FOR PRINTING LINE COUNT	EW	438
END OF SEQUENCE		
76 S 857 T		
SEQUENCE FOR ERROR ROUTINES	H	439
HALT FOR DESCRIPTION	Q	440
CARRIAGE RETURN FOR FORM CONTROL	+	441
CARRIAGE RETURN FOR FORM CONTROL	+	442
TYPE C FOR ERROR IDENTIFICATION	½C	443
TYPE O FOR ERROR IDENTIFICATION	½O	444
TYPE M FOR ERROR IDENTIFICATION	½M	445
TYPE M FOR ERROR IDENTIFICATION	½M	446
TYPE A FOR ERROR IDENTIFICATION	½A	447
TYPE N FOR ERROR IDENTIFICATION	½N	448
TYPE D FOR ERROR IDENTIFICATION	½D	449
RECALL LINE COUNTER	V01	450
CONSTANT TO INCREASE LINE COUNT	J.001	451
ADD TO PREVIOUS LINE COUNT	F	452
POSITIVE BRANCH FOR PRINTING ERROR	CF +	453
RESUME FOR NON-CARRIAGE RETURN ENTRY	AB *	454
CARRIAGE RETURN FOR FORM CONTROL	+	455
CARRIAGE RETURN FOR FORM CONTROL	+	456
TYPE A FOR ERROR IDENTIFICATION	½A	457
TYPE D FOR ERROR IDENTIFICATION	½D	458
TYPE D FOR ERROR IDENTIFICATION	½D	459
TYPE R FOR ERROR IDENTIFICATION	½R	460
TYPE E FOR ERROR IDENTIFICATION	½E	461
TYPE S FOR ERROR IDENTIFICATION	½S	462
TYPE S FOR ERROR IDENTIFICATION	½S	463
RECALL LINE COUNTER	V01	464
RESUME FOR PRINTING ERROR	AF *	465
SPACE FOR FORM ALIGNMENT	½	466
TYPE E FOR PRINTING ERROR	½E	467
TYPE R FOR PRINTING ERROR	½R	468
TYPE R FOR PRINTING ERROR	½R	469
TYPE O FOR PRINTING ERROR	½O	470] Drop form suffix
TYPE R FOR PRINTING ERROR	½R	471]
SPACE FOR FORM ALIGNMENT	½	472
TYPE I FOR IDENTIFICATION	½I	473
TYPE N FOR IDENTIFICATION	½N	474
NON DECIMAL PRINT LINE COUNT	T13	475
CONSTANT FOR CORRECTING LINE COUNT	J.001-	476
ADD TO PREVIOUS COUNT	F	477
STORE CORRECTED LINE COUNT	S01	478
CARRIAGE RETURN FOR FORM ALIGNMENT	+	479
CARRIAGE RETURN FOR FORM ALIGNMENT	+	480
READER OFF FOR MANUAL ERROR CORRECTION	F1	481
END OF SEQUENCE		
88 S 945 T		

Command	Object	Line No.
SEQUENCE FOR CHECKING STORAGE ADDRESSES	HS	482
ENTER 2 DIGIT ADDRESS	/20	483
CONSTANT FOR CHECKING DIRECT ADDRESS	J60	484
SUBTRACT FROM ADDRESS	B	485
NEGATIVE BRANCH FOR CORRECT ADDRESS	GZ -	486
CONSTANT FOR CHECKING INDIRECT ADDRESS	J39	487
SUBTRACT FROM ADDRESS	B	488
POSITIVE BRANCH FOR CORRECT ADDRESS	CZ +	489
PUNCH OFF FOR PRINTING ONLY	F6	490
CONSTANT FOR UPDATING LINE COUNT	J.001	491
ACCUMULATE IN LINE COUNT	W01	492
NEGATIVE BRANCH FOR ADDRESS ERROR	GB -	493
END		
31 S 976 T		
SEQUENCE FOR CHECKING PAGE LINE COUNT	HP	494
NEGATIVE BRANCH FOR COUNTING ON PAGE	GC -	495
CONSTANT FOR CONTROL TO NEW PAGE	J16-	496
ADD TO PREVIOUS PAGE LINE COUNT	F	497
BRANCH FOR NEW PAGE	ER	498
RESUME FOR TERMINATION	AC *	499
END OF SEQUENCE		
13 S 989 T		
SEQUENCE FOR NEW PAGE	HR	500
RESUME FOR ADDITIONAL CARRIAGE RETURN	AD *	501
CARRIAGE RETURN FOR NEW PAGE	+	502
CONSTANT TO INCREASE CONTROL COUNTER	J1	503
ADD TO CONTROL COUNTER	F	504
NEGATIVE BRANCH FOR ADD. CARRIAGE RETURN	GD -	505
READER OFF FOR INSERTION OF NEW PAGE	F1	506
HALT FOR INSERTING PAGE	Q	507
READER 1 ON TO CONTINUE TRANSLATION	F11	508
BRANCH FOR PRINTING PAGE AND NUMBER	EQ	509
END OF SEQUENCE		
16 S 1005 T		
SEQUENCE FOR PRINTING PAGE AND NUMBER	HQ	510
TABULATE TO PAGE NUMBER COLUMN		511
TYPE P FOR DESCRIPTION	½P	512
TYPE A FOR DESCRIPTION	½A	513
TYPE G FOR DESCRIPTION	½G	514
TYPE E FOR DESCRIPTION	½E	515
CONSTANT TO INCREASE PAGE NUMBER COUNT	J1	516
ACCUMULATE IN PAGE COUNTER	W06	517
RECALL PAGE COUNTER	V06	518

add sequence for checking storage address of 5

EXTERNAL INTERFACING 561C

Uses F7 F8 F9 F10 F11

Program is SWIFTER

SWIFT plus routines for Four additional Mnemonics.

EXTERNAL MEMORY FEED	EMF	F7
EYTERNAL MEMORY READ	EMR	F8
EYTERNAL TERMINATE	EXT	~~F9~~ F10
EYTERNAL COMMAND ECO	~~EXC~~	~~F10~~. F9

~~LE~~

MNEMONIC FOR EYTERNAL MEMORY FEED	LEMF	528
HALT FOR DESCRIPTION	Q	529
PUNCH 1 ON FOR OBJECT CODE	F4	530
TYPE OBJECT CODE (F7)	~~B~~ 1/2 (F7)	531
PUNCH OFF FOR PRINTING ONLY	F6	532
18S TYPE F FOR CODE IDENTIFICATION	1/2 F	533
TYPE 7 FOR CODE IDENTIFICATION	1/2 7	534
BRANCH FOR LINE AND OBJECT COUNT	EX	535
END OF SEQUENCE		
17		
MNEMONIC FOR EYTERNAL MEMORY READ	LEMR	536
HALT FOR DESCRIPTION	Q	537
PUNCH 1 ON FOR OBJECT CODE	F4	538
15, TYPE OBJECT CODE (F8)	1/2 (F8)	539
PUNCH OFF FOR PRINTING ONLY	F6	540
TYPE F FOR CODE IDENTIFICATION	1/2 F	541
TYPE 8 FOR CODE IDENTIFICATION	1/2 8	542
BRANCH FOR LINE AND OBJECT COUNT	EX	543.
17		

(2) SWIFTER

	MNEMONIC FOR EXTERNAL COMMAND	LECO	544.
	HALT FOR DESCRIPTION	Q	545
	PUNCH 1ON FOR OBJECT CODE	F4	546.
155	TYPE OBJECT CODE (F9)	½(F9)	547
	PUNCH OFF FOR PRINTING ONLY	F6	548
✓	TYPE F FOR CODE IDENTIFICATION	½ F	549.
	TYPE 9 FOR CODE IDENTIFICATION	½ 9	550
	BRANCH FOR LINE AND OBJECT COUNT	EY	551
	END OF SEQUENCE #15		

	MNEMONIC FOR EXTERNAL STOP	LEST	552
	HALT FOR DESCRIPTION	Q	553
155	PUNCH 1 ON FOR OBJECT CODE	F4	554
	TYPE OBJECT CODE	½ M	555
	~~PUNCH OFF FOR PRINTING ONLY~~	~~F6~~	~~556~~
	~~TYPE OBJECT~~ BRANCH FOR LINE & OBJ COUNT	EX	556
	END OF SEQUENCE 155		

	MNEMONIC FOR EXTERNAL TYPE	LETY	556.7
	HALT FOR DESCRIPTION	Q	558
105	PUNCH 1 ON FOR OBJECT CODE	F4	558.9
	TYPE OBJECT CODE	½ Z	560.
	BRANCH FOR LINE AND OBJECT COUNT	EY	561
	END OF SEQUENCE 105		

	MNEMONIC FOR EXTERNAL VIEW	LEVI	562
	HALT FOR DESCRIPTION	Q	563
	PUNCH 1 ON FOR OBJECT CODE	F4	564
105	TYPE OBJECT CODE	½ U	565
	BRANCH FOR LINE AND OBJECT COUNT	EY	566
	END OF SEQUENCE		

	MNEMONIC FOR EXTERNAL ~~CLEAR~~ WIPE SCREEN	LESC	567
	HALT FOR DESCRIPTION	Q	568
	PUNCH 1 ON FOR OBJECT CODE	F4	569
105	TYPE OBJECT CODE	½ Y	570
	BRANCH FOR LINE AND OBJECT CODE	EX	571
	END OF SEQUENCE		

Command	Object	Line No.
NON DECIMAL PRINT PAGE NUMBER	T30	519
RECALL CARRIAGE RETURN CONTROL SWITCH	V08	520
NEGATIVE BRANCH FOR 1 CARRIAGE RETURN	GE -	521
CARRIAGE RETURN FOR FORM ALIGNMENT	+	522
CARRIAGE RETURN FOR FORM ALIGNMENT	+	523
RESUME FOR SINGLE CARRIAGE RETURN	AE *	524
CARRIAGE RETURN FOR FORM ALIGNMENT	+	525
CONSTANT FOR SETTING PAGE LINE COUNTER	J52-	526
STORE IN PAGE LINE COUNTER	S07	527
END OF SEQUENCE		
40 S		
1045 T		
FINISH TRANSLATION OF TRANSLATOR		
1045 T		

In swidler

2 deletions 470 & 471

Changes ERROR to ERR

T. Count is now 1043

+75 for New Inst

T = 1118

1119 is limit

Command	Object	Line No.
MNEMONIC TO INITIALIZE PROGRAM	LT	001
UPPER CASE		002
CARRIAGE RETURN	+	003
TABULATE		004
PUNCH 1 ON	F4	005
TYPE	½T	006
TYPE	½R	007
TYPE	½A	008
TYPE	½N	009
TYPE	½S	010
TYPE	½L·	011
TYPE type « TRANS. -6CL.	½A·	012
TYPE :-6. ~~-8 b Cha.~~	½T·	013
TYPE	½E·	014
PRINT SIGN FOR SPACE	P	015
TYPE F1	½	016
TABULATE		017
CONSTANT TO SET END CONTROL SWITCH	J1 -	018
STORE END CONTROL SWITCH	S01	019
CARRIAGE RETURN	+	020
CARRIAGE RETURN	+	021
PUNCH OFF	F6	022
TRUNCATE 13 TO CLEAR R1	I	023
END		
41 S -P		
41 T		
SEQUENCE FOR ACCUMULATE COMMAND	HW	024
TABULATE		025
PUNCH 1 ON	F4	026
TYPE	½A	027
TYPE	½C	028
TYPE	½C	029
TYPE	½U	030
TYPE type ACCUM.	½M	031
TYPE	½U	032
TYPE -10	½L	033
TYPE	½A	034
TYPE	½T	035
TYPE	½E	036
RESUME	AV *	037
PRINT SIGN FOR SPACE	P	038
TYPE F1	½	039
TABULATE		040
ENTER ADDRESS	/02	041
CARRIAGE RETURN	+	042
PUNCH OFF	F6	043
END		
35 S		
76 T		

TP-161SL TRANSLATE REVERSE TRANSLATOR

FIGURE 2-53

Command	Object	Line No.
SEQUENCE FOR ADD COMMAND	HF	044
TABULATE		045
PUNCH 1 ON	F4	046
TYPE	½A	047
TYPE	½D	048
TYPE	½D	049
RESUME	AU *	050
PRINT SIGN FOR SPACE	P	051
TYPE F1	½	052
TABULATE		053
CARRIAGE RETURN	+	054
PUNCH OFF	F6	055
END		
18 S		
94 T		
SEQUENCE FOR BRANCH COMMAND	HE	056
TABULATE		057
PUNCH 1 ON	F4	058
RESUME	A6 *	059
TYPE	½B	060
TYPE	½R	061
TYPE	½A	062
TYPE	½N	063
TYPE	½C	064
TYPE	½H	065
POSITIVE BRANCH TO 434	CW +	066
END		
20 S		
114 T		
SEQUENCE FOR BACKSPACE COMMAND	H	067
TABULATE		068
PUNCH 1 ON	F4	069
TYPE	½B	070
TYPE	½A	071
TYPE	½C	072
TYPE TYPE BKSPACE	½K	073
TYPE :-4	½S	074
TYPE	½P	075
TYPE	½A	076
TYPE	½C	077
TYPE	½E	078
RESUME	AT *	079
PRINT SIGN FOR SPACE	P	080
TYPE F1	½	081
TABULATE		082
CARRIAGE RETURN	+	083
PUNCH OFF	F6	084
END		
30 S		
144 T		

Command	Object	Line No.
SEQUENCE FOR CARRIAGE RETURN COMMAND	H+	085
TABULATE		086
PUNCH 1 ON	F4	087
TYPE	½C	088
TYPE	½A	089
TYPE CARR.	½R	090
TYPE	½R	091
TYPE -6.	½I	092
TYPE	½A	093
TYPE	½G	094
TYPE	½E	095
PRINT SIGN FOR SPACE	P	096
POSITIVE BRANCH TO 394	C4 +	097
END		
23 S		
167 T		
SEQUENCE FOR CHANGE SIGN COMMAND	H-	098
TABULATE		099
PUNCH 1 ON	F4	100
TYPE	½C	101
TYPE CHNG.	½H	102
TYPE	½A	103
TYPE -2	½N	104
TYPE	½G	105
TYPE	½E	106
POSITIVE BRANCH TO 314	C2 +	107
END		
18 S		
185 T		
SEQUENCE FOR CONSTANT COMMAND	HJ	108
TABULATE		109
PUNCH 1 ON	F4	110
TYPE	½C	111
TYPE CONST.	½O	112
TYPE	½N	113
TYPE -4	½S	114
TYPE	½T	115
TYPE	½A	116
TYPE	½N	117
TYPE	½T	118
PRINT SIGN FOR SPACE	P	119
TYPE F1	½	120
TABULATE		121
ENTER ADDRESS (VALUE OF CONSTANT)	/86	122
CARRIAGE RETURN	+	123
PUNCH OFF	F6	124
TRUNCATE 13 TO CLEAR R1	I	125
END		
31 S		
216 T		

Command	Object	Line No.
SEQUENCE FOR DECIMAL PRINT COMMAND	HX	126
TABULATE		127
RESUME	A5 *	128
PUNCH 1 ON	F4	129
TYPE	½D	130
TYPE	½E	131
TYPE DEC	½C	132
TYPE	½I	133
TYPE -6	½M	134
TYPE	½A	135
TYPE	½L	136
PRINT SIGN FOR SPACE	P	137
TYPE	½P	138
TYPE	½R	139
TYPE PNT.	½I	140
TYPE -2	½N	141
TYPE	½T	142
POSITIVE BRANCH TO 037	CV +	143
END		
33 S 249 T		
SEQUENCE FOR DELETE COMMAND	HD	144
TABULATE		145
PUNCH 1 ON	F4	146
TYPE	½D	147
TYPE	½E	148
TYPE	½L	149
TYPE	½E	150
TYPE	½T	151
TYPE	½E	152
POSITIVE BRANCH TO 050	CU +	153
END		
18 S 267 T		
SEQUENCE FOR DIVIDE COMMAND	HK	154
TABULATE		155
PUNCH 1 ON	F4	156
TYPE	½D	157
TYPE DIV.	½I	158
TYPE	½V	159
TYPE -4	½I	160
TYPE	½D	161
TYPE	½E	162
POSITIVE BRANCH TO 050	CU +	163
END		
18 S 285 T		

Command	Object	Line No.
SEQUENCE FOR DUPLICATE COMMAND	HN	164
TABULATE		165
PUNCH 1 ON	F4	166
TYPE	½D	167
TYPE	½U	168
TYPE DUP.	½P	169
TYPE	½L	170
TYPE	½I	171
TYPE	½C	172
TYPE -10	½A	173
TYPE	½T	174
TYPE	½E	175
POSITIVE BRANCH TO 050	CU +	176
END		
24 S		
309 T		
SEQUENCE FOR ENTER COMMAND	H/	177
TABULATE		178
PUNCH 1 ON	F4	179
TYPE	½E	180
TYPE	½N	181
TYPE	½T	182
TYPE	½E	183
TYPE	½R	184
POSITIVE BRANCH TO 037	CV +	185
END		
16 S		
325 T		
SEQUENCE FOR HALT COMMAND	HQ	186
TABULATE		187
PUNCH 1 ON	F4	188
TYPE	½H	189
TYPE	½A	190
TYPE	½L	191
TYPE	½T	192
POSITIVE BRANCH TO 050	CU +	193
END		
14 S		
339 T		
SEQUENCE FOR LOWER CASE COMMAND	H	194
TABULATE		195
UPPER CASE		196
PUNCH 1 ON	F4	197
TYPE	½L	198
TYPE	½O	199
TYPE	½W	200

Command	Object	Line No.
TYPE	½E	201
TYPE	½R	202
RESUME	A3 *	203
PRINT SIGN FOR SPACE	P	204
TYPE	½C	205
TYPE	½A	206
TYPE	½S	207
TYPE	½E	208
POSITIVE BRANCH TO 079	CT +	209
END		
28 S		
367 T		
SEQUENCE FOR MNEMONIC COMMAND	HL	210
RECALL END CONTROL SWITCH	V01	211
NEGATIVE BRANCH TO 214	G7 -	212
POSITIVE BRANCH TO 407	C9 +	213
RESUME	A7 *	214
TRUNCATE 13 TO CLEAR R1	I	215
DUPLICATE TO CLEAR R2	N	216
STORE TO RESET END CONTROL SWITCH	S01	217
TABULATE		218
PUNCH 1 ON	F4	219
TYPE	½M	220
TYPE	½N	221
TYPE	½E	222
TYPE	½M	223
TYPE	½O	224
TYPE	½N	225
TYPE	½I	226
TYPE	½C	227
PRINT SIGN FOR SPACE	P	228
TYPE F1	½	229
TABULATE		230
ENTER FIRST CHARACTER OF ADDRESS	/01	231
ENTER SECOND CHARACTER OF ADDRESS	/01	232
ENTER THIRD CHARACTER OF ADDRESS	/01	233
CARRIAGE RETURN	+	234
PUNCH OFF	F6	235
UPPER CASE		236
END		
51 S		
418 T		
SEQUENCE FOR MULTIPLY COMMAND	HO	237
TABULATE		238
PUNCH 1 ON	F4	239
TYPE	½M	240
TYPE	½U	241
TYPE	½L	242

Command	Object	Line No.
TYPE	½T	243
TYPE	½I	244
TYPE	½P	245
TYPE	½L	246
TYPE	½Y	247
POSITIVE BRANCH TO 050	CU +	248
END		
22 S		
440 T		
SEQUENCE FOR NON DECIMAL PRINT COMMAND	HT	249
TABULATE		250
PUNCH 1 ON	F4	251
TYPE	½N	252
TYPE	½O	253
TYPE	½N	254
PRINT SIGN FOR SPACE	P	255
POSITIVE BRANCH TO 128	C5 +	256
END		
13 S		
453 T		
SEQUENCE FOR NEGATIVE BRANCH COMMAND	HG	257
TABULATE		258
PUNCH 1 ON	F4	259
TYPE	½N	260
TYPE	½E	261
TYPE NEG.	½G	262
TYPE	½A	263
TYPE	½T	264
TYPE -8.	½I	265
TYPE	½V	266
TYPE	½E	267
PRINT SIGN FOR SPACE	P	268
POSITIVE BRANCH TO 059	C6 +	269
END		
23 S		
476 T		
SEQUENCE (F2) NON PRINT COMMAND	H	270
TABULATE		271
PUNCH 1 ON	F4	272
TYPE	½N	273
TYPE	½O	274
TYPE	½N	275
PRINT SIGN FOR SPACE	P	276
TYPE	½P	277
TYPE	½R	278
TYPE	½I	279

Command	Object	Line No.
TYPE	½N	280
TYPE	½T	281
POSITIVE BRANCH TO 079	CT +	282
END		
23 S		
499 T		
SEQUENCE FOR POSITIVE BRANCH COMMAND	HC	283
TABULATE		284
PUNCH 1 ON	F4	285
TYPE	½P	286
TYPE	½O	287
TYPE POS.	½S	288
TYPE	½I	289
TYPE	½T	290
TYPE -8	½I	291
TYPE	½V	292
TYPE	½E	293
PRINT SIGN FOR SPACE	P	294
POSITIVE BRANCH TO 059	C6 +	295
END		
23 S		
522 T		
SEQUENCE (F3) PRINT RESTORE COMMAND	H	296
BRANCH TO 306	EP	297
TYPE	½R	298
TYPE	½E	299
TYPE	½S	300
TYPE REST.	½T	301
TYPE	½O	302
TYPE -4	½R	303
TYPE	½E	304
POSITIVE BRANCH TO 079	CT +	305
END		
20 S		
542 T		
SEQUENCE FOR PRINT SIGN COMMAND	HP	306
TABULATE		307
PUNCH 1 ON	F4	308
TYPE	½P	309
TYPE	½R	310
TYPE	½I	311
TYPE	½N	312
TYPE	½T	313
RESUME	A2 *	314
PRINT SIGN FOR SPACE	P	315
RETURN	R	316

Command	Object	Line No.
TYPE	½S	317
TYPE	½I	318
TYPE	½G	319
TYPE	½N	320
POSITIVE BRANCH TO 079	CT +	321
END		
28 S		
570 T		
SEQUENCE (F4) PUNCH 1 ON COMMAND	H	322
TABULATE		323
PUNCH 1 ON	F4	324
TYPE	½P	325
TYPE	½U	326
TYPE	½N	327
TYPE	½C	328
TYPE	½H	329
RETURN	R	330
POSITIVE BRANCH TO 351	CX +	331
END		
17 S		
587 T		
SEQUENCE (F5) PUNCH 2 ON COMMAND	H	332
BRANCH TO 322	E	333
RESUME	AY *	334
PRINT SIGN FOR SPACE	P	335
TYPE	½2	336
PRINT SIGN FOR SPACE	P	337
TYPE	½O	338
TYPE	½N	339
POSITIVE BRANCH TO 079	CT +	340
END		
16 S		
603 T		
SEQUENCE (F6) PUNCH OFF COMMAND	H	341
BRANCH TO 322	E	342
RESUME	AZ *	343
PRINT SIGN FOR SPACE	P	344
TYPE	½O	345
TYPE	½F	346
TYPE	½F	347
POSITIVE BRANCH TO 079	CT +	348
END		
15 S		
618 T		

Command	Object	Line No.
SEQUENCE (F11) READER 1 ON COMMAND	H	349
BRANCH TO 361	E	350
RESUME	AX *	351
PRINT SIGN FOR SPACE	P	352
TYPE	½1	353
PRINT SIGN FOR SPACE	P	354
TYPE	½0	355
TYPE	½N	356
POSITIVE BRANCH TO 079	CT +	357
END		
16 S		
634 T		
SEQUENCE (F13) READER 2 ON COMMAND	H	358
BRANCH TO 361	E	359
POSITIVE BRANCH TO 334	CY +	360
END		
6 S		
640 T		
SEQUENCE (F1) READER OFF COMMAND	H	361
TABULATE		362
PUNCH 1 ON	F4	363
TYPE	½R	364
TYPE	½E	365
TYPE	½A	366
TYPE	½D	367
TYPE	½E	368
TYPE	½R	369
RETURN	R	370
POSITIVE BRANCH TO 343	CZ +	371
END		
19 S		
659 T		
SEQUENCE FOR RECALL COMMAND	HV	372
TABULATE		373
PUNCH 1 ON	F4	374
TYPE	½R	375
TYPE	½E	376
TYPE	½C	377
TYPE	½A	378
TYPE	½L	379
TYPE	½L	380
POSITIVE BRANCH TO 037	CV +	381
END		
18 S		
677 T		

Command	Object	Line No.
SEQUENCE FOR RESUME COMMAND	HA	382
TABULATE		383
PUNCH 1 ON	F4	384
TYPE	½R	385
TYPE	½E	386
TYPE	½S	387
TYPE	½U	388
TYPE	½M	389
TYPE	½E	390
POSITIVE BRANCH TO 434	CW +	391
END		
18 S		
695 T		
SEQUENCE FOR RETURN COMMAND	HR	392
TABULATE		393
RESUME	A4 *	394
PUNCH 1 ON	F4	395
TYPE	½R	396
TYPE RET.	½E	397
TYPE	½T	398
TYPE -6	½U	399
TYPE	½R	400
TYPE	½N	401
POSITIVE BRANCH TO 079	CT +	402
END		
20 S		
715 T		
SEQUENCE FOR SEQUENCE COMMAND	HH	403
RECALL END CONTROL SWITCH	V01	404
NEGATIVE BRANCH TO 420	G8 -	405
CONSTANT TO CONTROL CONDITIONAL BRANCHING	J1 -	406
RESUME	A9 *	407
TABULATE		408
PUNCH 1 ON	F4	409
TYPE	½E	410
TYPE	½N	411
TYPE	½D	412
TYPE	½	413
TYPE F1	½	414
TABULATE		415
CARRIAGE RETURN	+	416
CARRIAGE RETURN	+	417
PUNCH OFF	F6	418
POSITIVE BRANCH TO 214	C7 +	419
RESUME	A8 *	420
TRUNCATE 13 TO CLEAR R1	I	421
DUPLICATE TO CLEAR R2	N	422
STORE ZERO TO RESET END CONTROL SWITCH	S01	423

Command	Object	Line No.
TABULATE		424
PUNCH 1 ON	F4	425
TYPE	½S	426
TYPE	½E	427
TYPE	½Q	428
TYPE SEQU.	½U	429
TYPE	½E	430
TYPE	½N	431
TYPE -6	½C	432
TYPE	½E	433
RESUME	AW *	434
PRINT SIGN FOR SPACE	P	435
TYPE F1	½	436
TABULATE		437
PRINT RESTORE FOR DELAY	F3	438
PRINT RESTORE FOR DELAY	F3	439
HALT FOR ADDRESS	Q	440
PRINT RESTORE FOR DELAY	F3	441
CARRIAGE RETURN	+	442
PUNCH OFF	F6	443
END		
68 S		
783 T		
SEQUENCE FOR STORE COMMAND	HS	444
TABULATE		445
PUNCH 1 ON	F4	446
TYPE	½S	447
TYPE	½T	448
TYPE	½O	449
TYPE	½R	450
TYPE	½E	451
POSITIVE BRANCH TO 037	CV +	452
END		
16 S		
799 T		
SEQUENCE FOR SUBTRACT COMMAND	HB	453
TABULATE		454
PUNCH 1 ON	F4	455
TYPE	½S	456
TYPE	½U	457
TYPE SUBT.	½B	458
TYPE	½T	459
TYPE -6	½R	460
TYPE	½A	461
TYPE	½C	462
TYPE	½T	463
POSITIVE BRANCH TO 050	CU +	464
END		
22 S		
821 T		

Command	Object	Line No.
SEQUENCE FOR TABULATE COMMAND	H	465
PUNCH 1 ON	F4	466
TYPE	½T	467
TYPE	½A	468
TYPE	½B	469
TYPE	½U	470
TYPE	½L	471
TYPE	½A	472
TYPE	½T	473
TYPE	½E	474
POSITIVE BRANCH TO 079	CT +	475
END		
21 S		
842 T		
	. .	
SEQUENCE FOR TRUNCATE COMMAND	HI	476
TABULATE	. .	477
PUNCH 1 ON	F4	478
TYPE	½T	479
TYPE	½R	480
TYPE	½U	481
TYPE	½N	482
TYPE	½C	483
TYPE	½A	484
TYPE	½T	485
TYPE	½E	486
PRINT SIGN FOR SPACE	P	487
TYPE F1	½	488
TABULATE	. .	489
ENTER ADDRESS	/01	490
CARRIAGE RETURN	+	491
PUNCH OFF	F6	492
END		
29 S		
871 T		
SEQUENCE FOR TYPE COMMAND	H½	493
TABULATE		494
PUNCH 1 ON	F4	495
TYPE	½T	496
TYPE	½Y	497
TYPE	½P	498
TYPE	½E	499
RETURN	R	500
POSITIVE BRANCH 434	CW +	501
END		
15 S		
886 T		

Command	Object	Line No.
SEQUENCE FOR UPPER CASE COMMAND	H	502
TABULATE		503
PUNCH 1 ON	F4	504
TYPE	½U	505
TYPE	½P	506
TYPE	½P	507
TYPE	½E	508
TYPE	½R	509
POSITIVE BRANCH TO 203	C3 +	510
END		
16 S		
902 T		
SEQUENCE FOR COMMA	H,	511
BRANCH TO 493	E½	512
TABULATE		513
TYPE	½,	514
RESUME	A1 *	515
NON PRINT	F2	516
CARRIAGE RETURN	+	517
PRINT RESTORE	F3	518
PUNCH OFF	F6	519
PRINT SIGN FOR SPACE	P	520
PRINT SIGN FOR SPACE	P	521
PRINT SIGN FOR SPACE	P	522
LOWER CASE		523
TYPE * TO INDICATE COMMAND CONVERSION	½8	524
UPPER CASE		525
CARRIAGE RETURN	+	526
END		
21 S		
923 T		
SEQUENCE (F7) FUNCTION 7	H	527
BRANCH TO 493	E½	528
PRINT SIGN FOR SPACE	P	529
TYPE	½F	530
TYPE	½7	531
PRINT SIGN FOR SPACE	P	532
TYPE F1	½	533
TABULATE		534
TYPE F7 (ADDRESS)	½	535
POSITIVE BRANCH TO 515	C1 +	536
END		
17 S		
940 T		

TP-161SL

delete for Swedes.

Command	Object	Line No.
SEQUENCE (F8) FUNCTION 8	H	537
BRANCH TO 493	E½	538
PRINT SIGN FOR SPACE	P	539
TYPE	½F	540
TYPE	½8	541
PRINT SIGN FOR SPACE	P	542
TYPE F1	½	543
TABULATE		544
TYPE F8 (ADDRESS)	½	545
POSITIVE BRANCH TO 515	C1 +	546
END		
17 S		
957 T		
SEQUENCE (F9) FUNCTION 9	H	547
BRANCH TO 493	E½	548
PRINT SIGN FOR SPACE	P	549
TYPE	½F	550
TYPE	½9	551
PRINT SIGN FOR SPACE	P	552
TYPE F1	½	553
TABULATE		554
TYPE F9 (ADDRESS)	½	555
POSITIVE BRANCH TO 515	C1 +	556
END		
17 S		
974 T		
SEQUENCE (F10) FUNCTION 10	H	557
BRANCH TO 493	E½	558
PRINT SIGN FOR SPACE	P	559
TYPE	½F	560
TYPE	½1	561
TYPE	½0	562
PRINT SIGN FOR SPACE	P	563
TYPE F1	½	564
TABULATE		565
TYPE F10 (ADDRESS)	½	566
POSITIVE BRANCH TO 515	C1 +	567
END		
19 S		
993 T		
SEQUENCE (F12) FUNCTION 12	H	568
BRANCH TO 493	E½	569
PRINT SIGN FOR SPACE	P	570
TYPE	½F	571
TYPE	½1	572
TYPE	½2	573

Command	Object	Line No.
PRINT SIGN FOR SPACE	P	574
TYPE F1	½	575
TABULATE		576
TYPE F12 (ADDRESS)	½	577
POSITIVE BRANCH TO 515	C1 +	578
END		
19 S		
1012 T		
SEQUENCE FOR CHECK PROTECT COMMAND	H&	579
TABULATE		580
PUNCH 1 ON	F4	581
TYPE	½C	582
TYPE CK.	½H	583
TYPE	½E	584
TYPE	½C	585
TYPE	½K	586
PRINT SIGN FOR SPACE	P	587
TYPE	½P	588
TYPE	½R	589
TYPE PCT.	½O	590
TYPE	½T	591
TYPE	½E	592
TYPE -4	½C	593
TYPE	½T	594
POSITIVE BRANCH TO 050	CU +	595
END		
31 S		
1043 T		
SEQUENCE FOR ERROR (NO-OP CODES)	H	596
READER OFF (Will restore manual non print)	F1	597
PUNCH OFF	F6	598
TABULATE		599
TYPE	½E	600
TYPE	½R	601
TYPE	½R	602
TYPE	½O	603
TYPE	½R	604
CARRIAGE RETURN	+	605
END		
16 S		
1059 T		
MNEMONIC FOR PROGRAM FINISH	LEND	606
HALT	Q	607
PUNCH 1 ON	F4	608
NON PRINT	F2	609
TYPE	½E	610

Command	Object	Line No.
TYPE	½N	611
TYPE	½D	612
PRINT SIGN FOR SPACE	P	613
TYPE F1	½	614
TABULATE		615
PRINT RESTORE	F3	616
CARRIAGE RETURN	+	617
CARRIAGE RETURN	+	618
TYPE	½F	619
TYPE	½I	620
TYPE	½N	621
TYPE	½I	622
TYPE	½S	623
TYPE	½H	624
PRINT SIGN FOR SPACE	P	625
TYPE F1	½	626
TABULATE		627
CARRIAGE RETURN	+	628
CARRIAGE RETURN	+	629
PUNCH OFF	F6	630
END REVERSE TRANSLATE PROGRAM		
39 S		
1098 T		
FINISH TRANSLATION		
1098 T		

SWIFTER REVERSE TRANSLATCK

SEQUENICE (F8) EXTERNAL MEMORY FEED		H	631
TABULATE			632
PUNCH 1 ON		F4	633
TYPE		½ E	634
TYPE		½ X	635
TYPE		½ T	636
TYPE EXT.	½ .	~~½ .~~	637
TYPE		~~½ R~~	638
TYPE		~~½ N~~	639
TYPE		~~½ A~~	640
TYPE		~~½ L~~	641
30s PRINT SIGN FOR SPACE	P	~~P~~ P	642
TYPE		½ M	643
TYPE		½ E	644
TYPE		½ M	645
TYPE	½ .	~~½ O .~~	646
TYPE		~~½ R~~	647
TYPE		~~½ X~~	648
PRINT SIGN FOR SPACE	P		649
TYPE		½ F	650
TYPE		½ E	651
TYPE		½ E	652
TYPE		½ D	653
POSITIVE BRANCH TO 050 END		CU+	654
SEQUENCE (F8) EXTERNAL MEMORY READ		H	655
TABULATE			656
PUNCH 1 ON		F4	657
TYPE		½ E	656
TYPE		½ X	659
TYPE		½ T	660
TYPE	½ .	~~½ E~~	661
TYPE		½ R	662
TYPE		½ N	663
TYPE		½ A	664
TYPE		½ L	665
TYPE			

SWIFTER REL TRANSLATE

PRINT SIGN FOR SPACE		P	666
TYPE		½ M	667
TYPE		½ E	668
TYPE		½ M	669
TYPE	½•	~~½ O~~	670
TYPE		~~½ R~~	671
TYPE		~~½ Y~~	672
~~TYPE~~ PRINT SIGN FOR SPACE		P	673
TYPE		½ R	674
TYPE		½ E	675
TYPE		½ A	676
TYPE		½. D	677.
POSITIVE BRANCH TO 050		CU +	678
END.			

SEQUENCE (F 9) FOR EXTERNAL COMMAND		H	679
TABULATE — PUNCH 1 ON		F4 681	680
TYPE		½ E	682
TYPE		½ X	683
TYPE		½ T	684
TYPE	½•	~~½ E~~	685
~~TYPE~~		~~½ R~~	686
~~TYPE~~		~~½ N~~	687
~~TYPE~~		~~½ A~~	688
~~TYPE~~		~~½ L~~	689
PRINT SIGN FOR SPACE		P	690
TYPE		½ C	691
TYPE		½ O	692
TYPE		½ M	693
TYPE		½ M	694
TYPE	½•	~~½ A~~	695
TYPE		~~½ N~~	696
TYPE		~~½ D~~	697.
POSITIVE BRANCH TO C50		CU +	698

③

SEQUENCE FOR EXTERNAL STOP		HM	699
TABULATE			700
PUNCH 1 ON		F4	701
TYPE		½ E	702
TYPE		½ X	703
TYPE	½ T	~~½ E~~	704
TYPE	½ •	~~½ R~~	705
TYPE		~~½ N~~	706
25 S TYPE		~~½ A~~	707
TYPE		~~½ L~~	708
PRINT SIGN FOR SPACE		P	709
TYPE		½ S	710
TYPE		½ T	711
TYPE		½ O	712
TYPE		½ P	713.
POSITIKE BRANCH TO 050		CU +	714
END.			

SEQUENCE FOR EXTERNAL TYPE		H2	713
TABULATE			714
PUNCH 1 ON		F4	715
TYPE		½ E	716
TYPE		½ X	717
TYPE		½ T	718
TYPE	½ •	~~½ E~~	719
23 S TYPE		~~½ R~~	720
TYPE		~~½ N~~	721
TYPE		~~½ A~~	722
PRINT SIGN FOR SPACE		~~½ L~~	723
~~POSITIVE BRANCH TO 0~~		P	724
TYPE		½ T	725
TYPE		½ Y	726
TYPE		½ P	727
TYPE		½ E	728
POSITIVE BRANCH TO 050		CU+	729
END			

SEQUENCE FOR EXTERNAL VIEW		HU	730
TABULATE			731
PUNCH 1 ON		F4	732
TYPE		½E	733
TYPE		½X	734
TYPE	EXT	½T	735
TYPE	½.	~~½E~~	736
TYPE		~~½R~~	737
TYPE		~~½N~~	738
TYPE		~~½A~~	739
TYPE		~~½L~~	740
PRINT SIGN FOR SPACE		P	741
TYPE		½V	742
TYPE		½I	743
TYPE		½E	744
TYPE		½W	745
POSITIVE BRANCH TO 050		CU+	746
END			

SEQUENCE FOR EXTERNAL SCREEN CLEAR		HY	747
TABULATE			745
PUNCH 1 ON		F4	746.
TYPE		½E	747
TYPE		½X	748
TYPE		½T	749
TYPE	.	~~½E~~	750
TYPE		~~½R~~	751
TYPE		~~½N~~	752
TYPE		~~½A~~	753
TYPE		~~½L~~	754
PRINT SIGN FOR SPACE		P	755
TYPE		½S	756
TYPE		½C	757
TYPE	½.	~~½R~~	758
TYPE		~~½E~~	759
TYPE		~~½E~~	760
TYPE		~~½N~~	761
PRINT SIGN FOR SPACE		P.	762

(5)

TYPE	1/2 W	763
TYPE	1/2 T	764
TYPE	1/2 P	765
TYPE	1/2 E	766
POSITIVE BRANCH TO 050	CU +	767.
END.		

5610 SEQUENCE – RESUME CODE ASSIGNMENT SHEET

SEQUENCE ASSIGNMENTS		RESUME ASSIGNMENTS	
SEQ.	DESCRIPTION	RES.	DESCRIPTION

Prepared For _____ Equipment _____
Customer

FRIDEN Office _____ Date Prepared _____ Page ____ Of ____ Pages

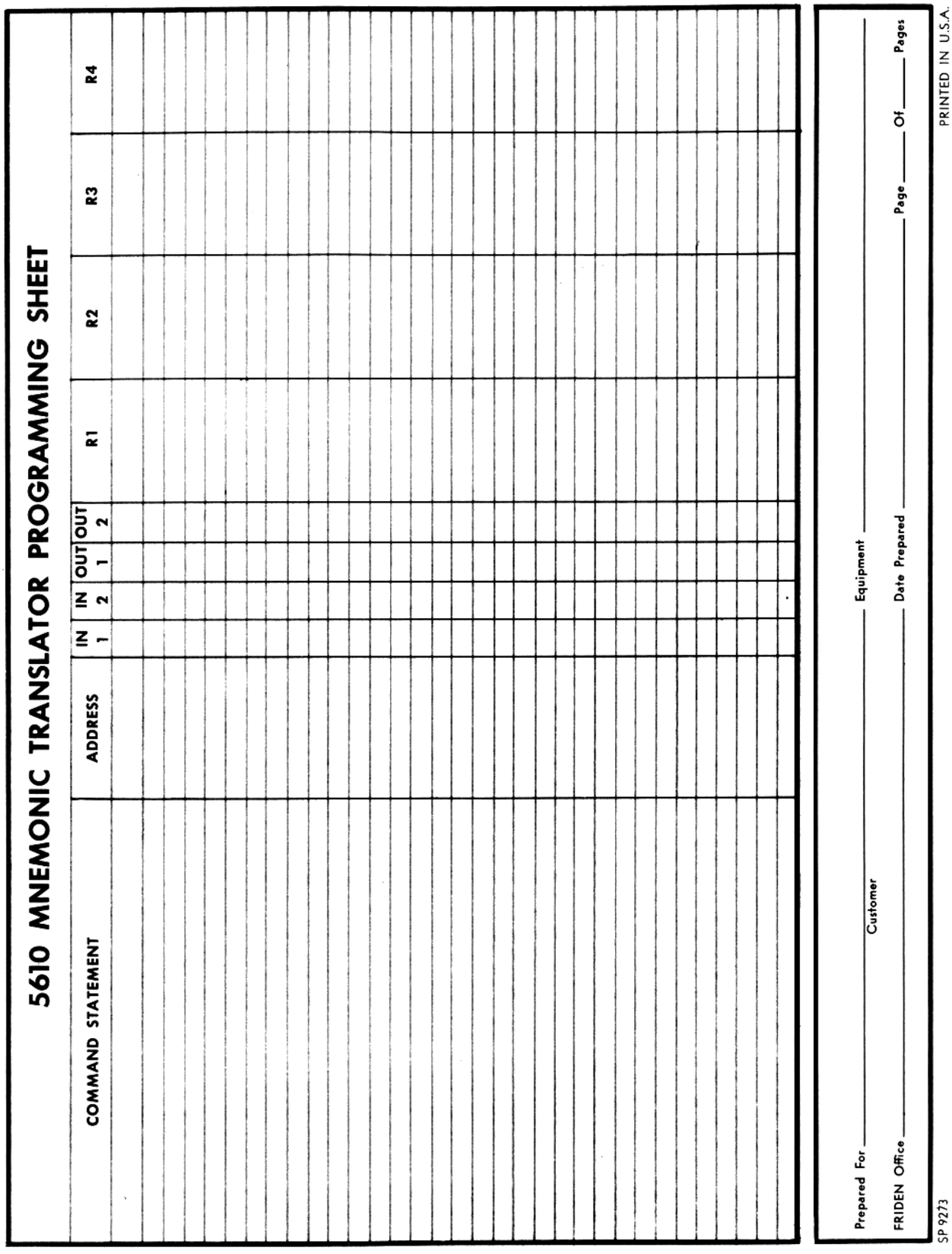
SP 9269

SEQUENCE - RESUME CODE ASSIGNMENT SHEET

PRINTED IN U.S.A.

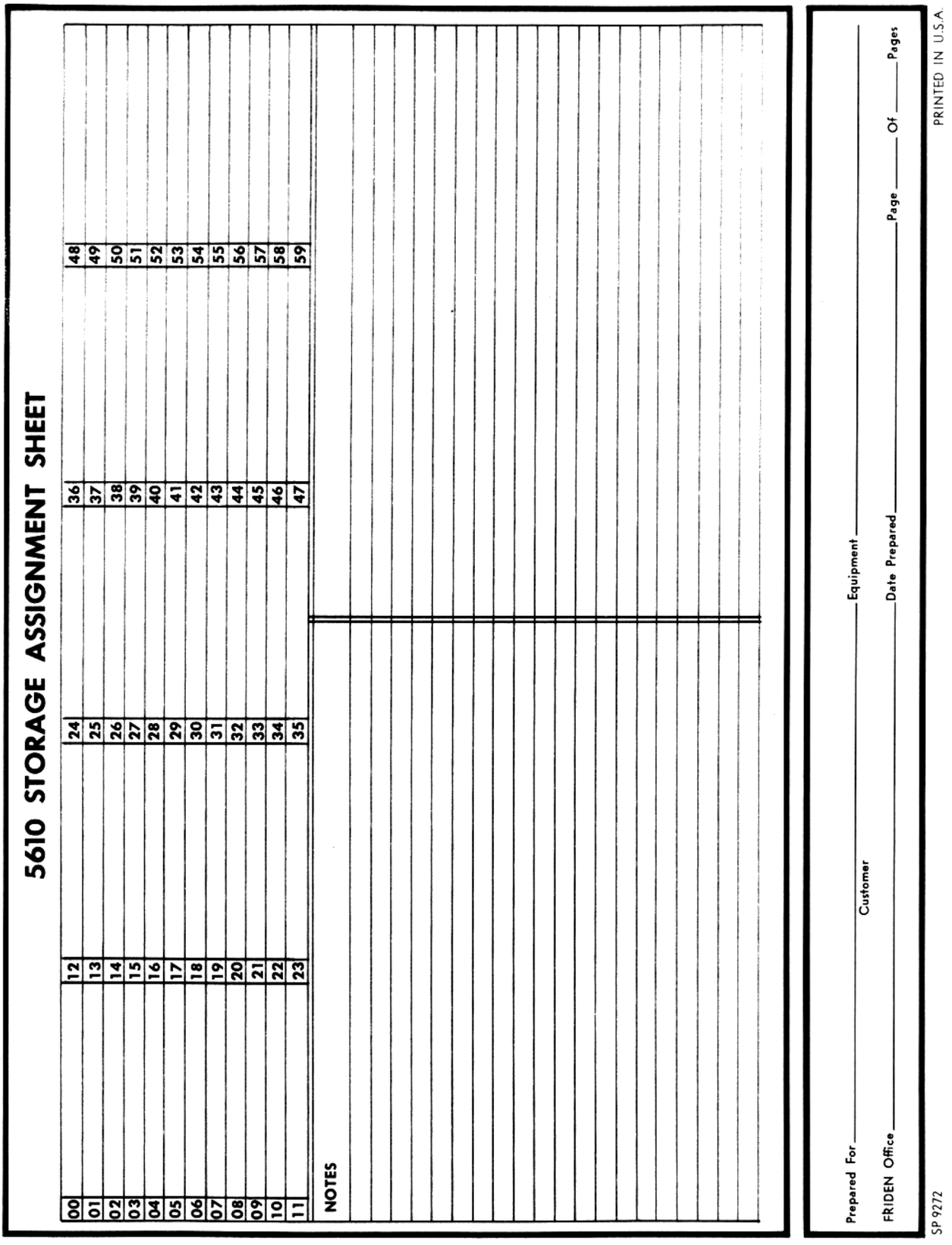
FIGURE 2-54

TP-161SL



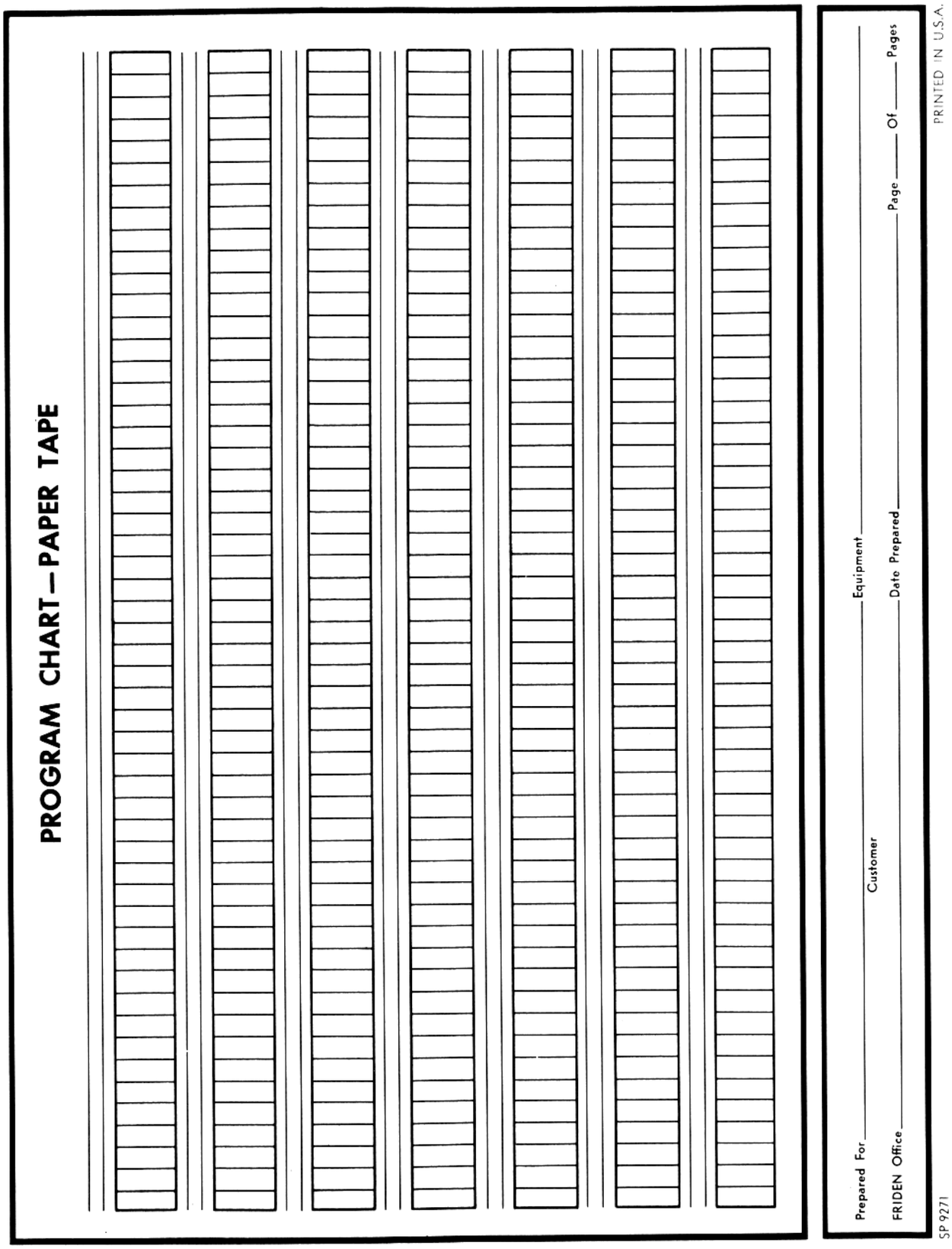
MNEMONIC TRANSLATOR PROGRAMMING SHEET

FIGURE 2-55



STORAGE ASSIGNMENT SHEET

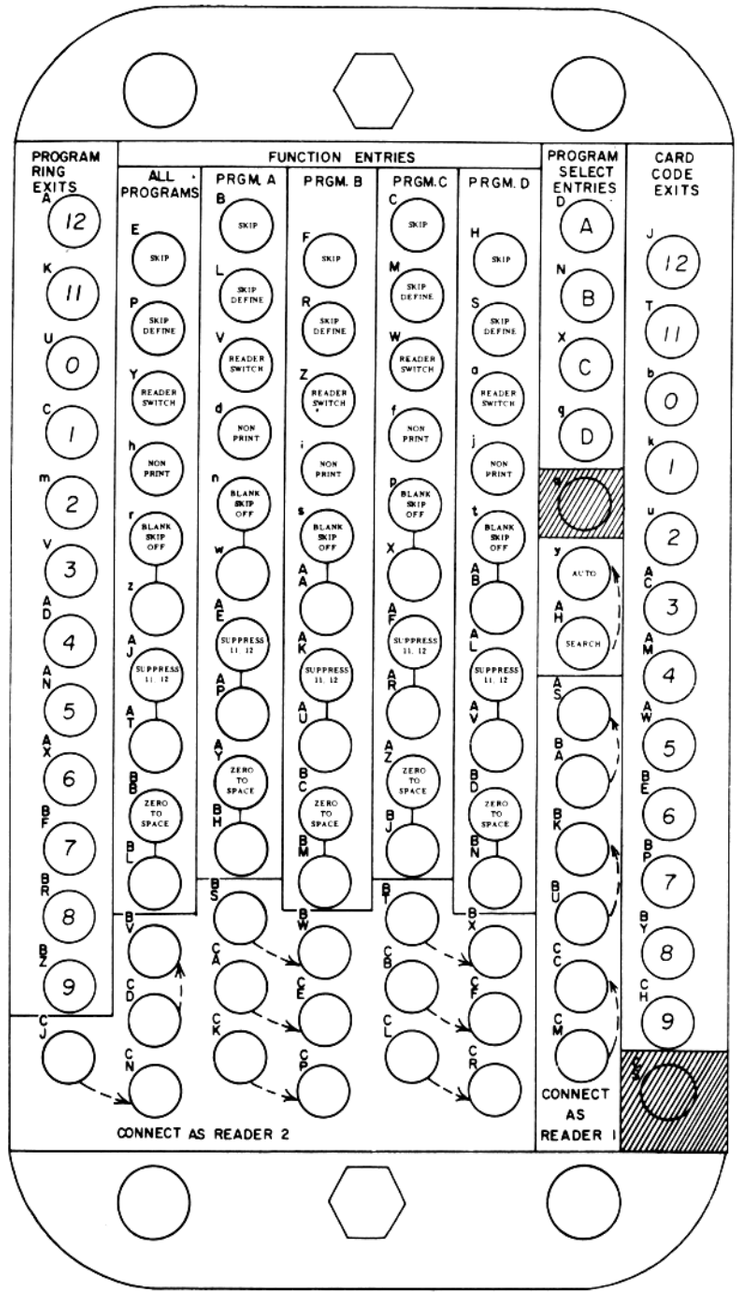
FIGURE 2-56



PROGRAM CHART - PAPER TAPE

FIGURE 2-57

PROGRAM CHART-- 2313 TERMINAL BLOCK



Prepared For _____ Equipment _____
Customer

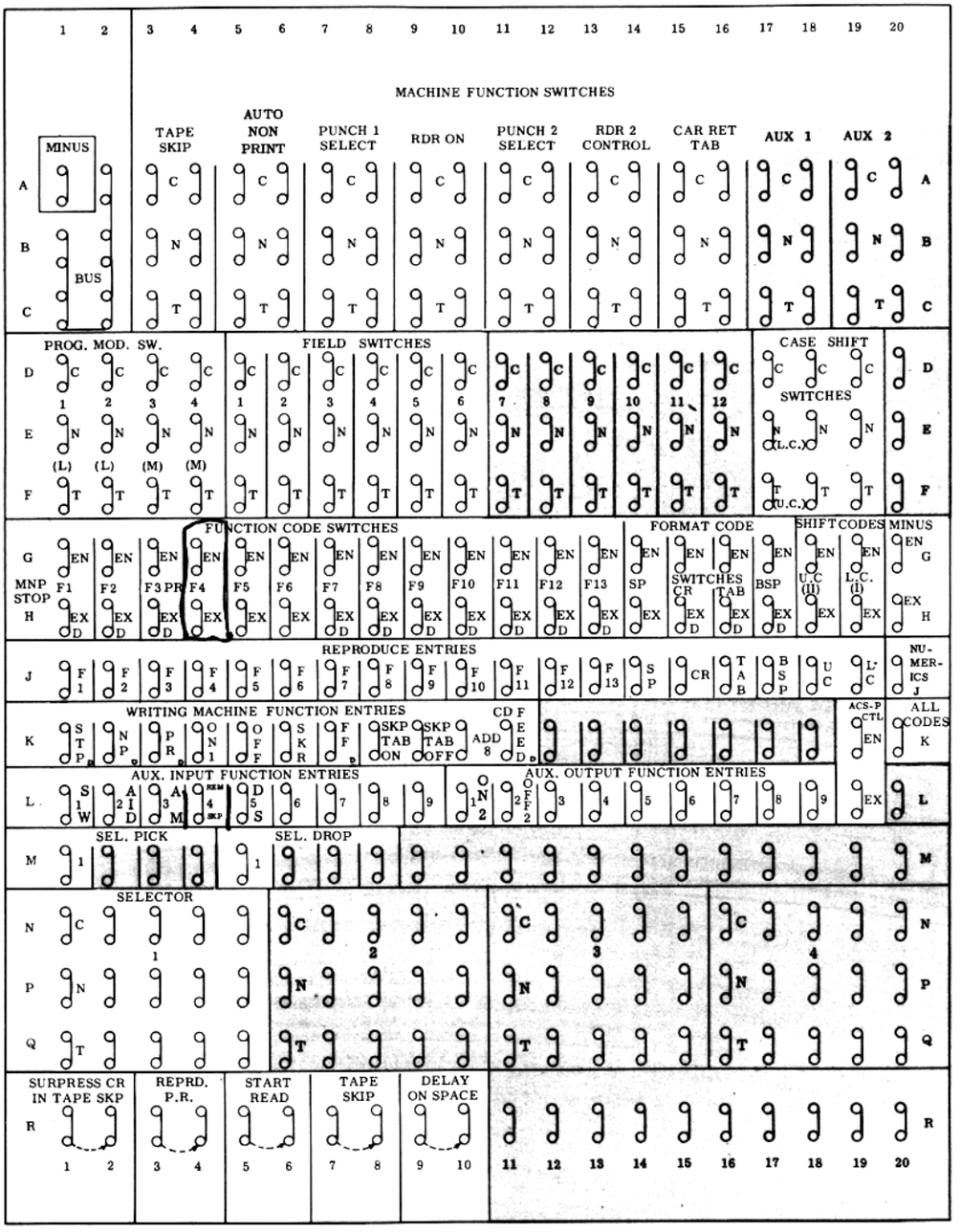
FRIDEN Office _____ Date Prepared _____ Page ____ Of ____ Pages

PROGRAM CHART - 2313 TERMINAL BLOCK

FIGURE 2-58

TP-161SL

PROGRAM CHART--2201 CONTROL PANEL



Prepared For _____ Equipment _____
Customer
FRIDEN Office _____ Date Prepared _____ Page ____ Of ____ Pages

SP 9274

PRINTED IN U.S.A.

PROGRAM CHART - 2201 CONTROL PANEL

FIGURE 2-59

TP-161SL

TABLE OF CONTENTS

	Page
SECTION 3, PROGRAM LINE LOGIC	
PROGRAM DELAY LINE	3-1
Loading the Program	3-2
General Operation	3-2
Line Routings	3-4
Data Organization	3-5
Basic Timing	3-6
Sync Pulse	3-6
Program Line Home	3-6
End of Program Detector	3-6
Other Important Flip-flops	3-7
PROGRAM LINE TIMING CHAIN	3-7
Oscillator	3-7
Fractional Bit Counter	3-7
Bit Counter	3-8
PROGRAM LINE OPERATION	3-9
Basic Read/Write Logic	3-9
Operation at Program Line Home	3-11
Launch Program Line Sync Pulse	3-13
START SIGNALS	3-15
Start Pushbutton (S9)	3-15
Logic	3-15
BASIC PROGRAM LINE LOGIC	3-16
Mode Register	3-16
Phase Counter	3-18
Control Registers	3-25
Reset Switch	3-40
1 MODE, PROGRAM LOAD	3-43
General Operation	3-43
Line Routings	3-47
Enabling the Phase Counter	3-47
Preliminary Phases	3-49
Loading the Line	3-52
Parity Error	3-54
Load Overflow	3-54
Terminate Load	3-54
Subsequent Loads	3-56

	Page
3 MODE, SEARCH	3-65
General Operation	3-65
Phases	3-66
Decimal Settings	3-67
Basic Search Patterns	3-67
Normal Sequence Search	3-73
Marker Search	3-86
Other Types of Search	3-91
4 MODE, INSTRUCTION RETRIEVAL	3-93
General Operation	3-93
Line Routings	3-101
Enabling the Phase Counter	3-101
Standard Operation	3-102
Special Operation	3-104
Summary	3-111
UNCONDITIONAL BRANCH	3-115
General Operation	3-115
Line Routings	3-115
Retrieval of the Branch Instruction	3-119
Unconditional Branch Sequence Search	3-121
"RETURN" COMMAND	3-123
CONDITIONAL BRANCH	3-124
General Operation	3-125
Line Routings	3-129
Enabling the Counter	3-129
Retrieving the Conditional Branch Instruction	3-129
Sequence Search	3-130
Count Down Decimal Register	3-131
"Resume" Search	3-132
Effects of UBR	3-134
"Resume" Command	3-135
Flow Chart	3-135

SECTION 3, PROGRAM LINE LOGIC

The program line logic can be called the most important part of the 5610 because it tells the rest of the system WHAT to do. Basically, the program line logic performs four functions. First, it allows loading of a program onto the program delay line. Second, it performs a search of the delay line for certain specified codes. Third, it retrieves instructions (commands and addresses) from the program line so that they may be executed. Last, it executes the three branching instructions after they are retrieved. This last function is mainly a matter of modifying the search function.

Certain parts of the program line logic are basic to the operation of the entire machine. These are the program delay line, the mode register, the phase counter, and the control registers.

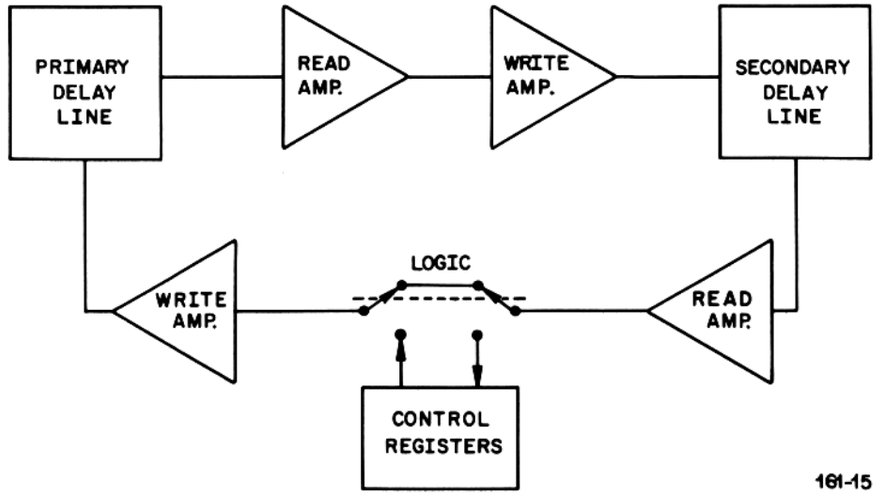
A. PROGRAM DELAY LINE

The 5610 is a stored program device, utilizing the delay line method for program storage. Delay line storage is dynamic. This means that the program is continually recirculating on the delay line. If the 5610 is turned off, the program is lost. This is the opposite of "static" storage, such as is found in the patchboard wiring of the CTP, 5010, 6010, and the 2205.

The program delay line, commonly referred to as the "program line", is actually composed of two 6 ms. delay lines connected in series. A twist from the output of the first, or primary, delay line is sensed by a read amplifier, which converts the mechanical energy into electrical energy. This read amplifier immediately sends an electronic signal to a write amplifier, which converts the electrical energy back into mechanical power sufficient to "write" a twist into the second (secondary) delay line.

When this twist emerges from the secondary delay line, it is sensed by an identical read amplifier which again converts the mechanical delay line energy into an electrical pulse. This pulse is fed into a logic network from which it can be sent directly to another write amplifier to be rewritten back onto the primary delay line, or it can be diverted to one of the control registers to initiate some logic function.

Figure 3-1 illustrates a simple block diagram of program line operation. In order to maintain uniformity in the packaging of all delay lines, two shorter delay lines are used instead of one long one.



BLOCK DIAGRAM
PROGRAM LINE OPERATION

FIGURE 3-1

1. LOADING THE PROGRAM

In operating the 5610, the first step is to "load" the program. Then the program must be recirculated on the delay line until it is needed for operation. This recirculation is a process of continual reading and writing. The sequence of events is as follows:

(1) The program is initially written onto the delay line. This is called "Program Load". Each program character is written onto the delay line immediately behind the previous character.

(2) After each character is written, it travels down the length of the delay line and emerges from the other end about 12 ms. later. When it emerges, it is immediately rewritten onto the beginning of the line. In this way, each character maintains its position in respect to the other characters.

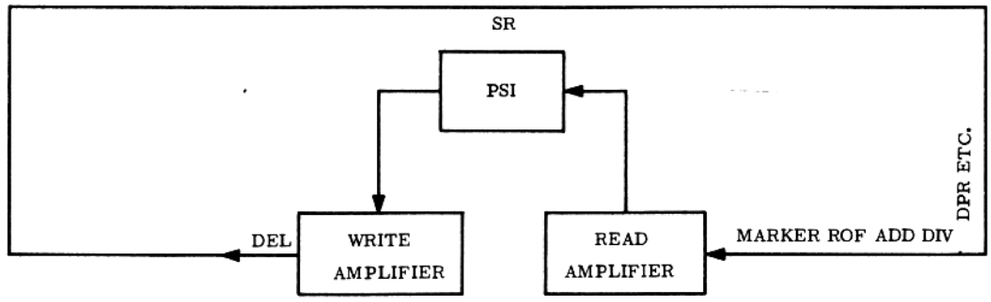
(3) When another character is to be loaded (entered via the 2205), it is held in a buffer register until the last character previously entered has been rewritten. It is then written onto the line right behind the previous character. This character is then recirculated along with the others.

(4) When the entire program has been loaded, all characters keep recirculating until the operator begins to "run" the program.

2. GENERAL OPERATION

Before a series of instructions is performed, a "marker" is automatically placed on the delay line before the first command. The marker is the tape feed code. (A tape feed cannot be loaded onto the program line from the 2205.) As each instruction is executed, the marker moves along the delay line to the next instruction, so that it is always in front of the next instruction to be performed.

When one instruction is completed, a search for the marker is initiated. During this search, the program is no longer recirculated directly, but diverted through a control register instead. This control register is a shift register, called the Search Register (SR). See Figure 3-2 (a).



BASIC PROGRAM LINE OPERATION

FIGURE 3-2 (a)

Each character emerging from the delay line is shifted into SR. As the next character is sensed, it is shifted into SR and the previous character is shifted out of SR back onto the program line, as shown in Figure 3-2 (b).

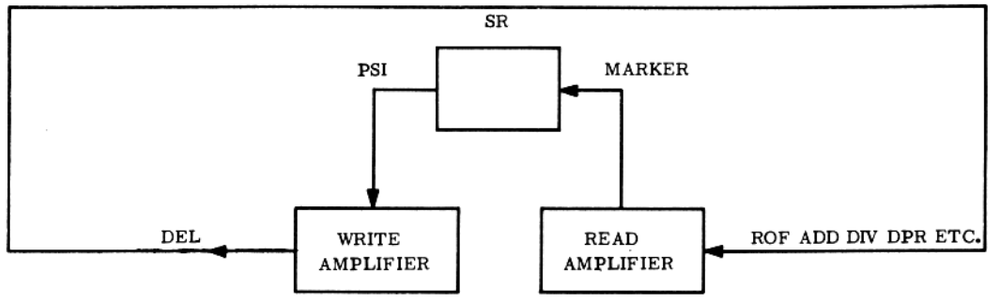


FIGURE 3-2 (b)

This process continues until the marker is shifted into SR. As soon as the marker is detected in SR, the program line output is diverted to another control register, called the Operation Register (OR). In this way, the command immediately following the marker is shifted into OR. This command is also sent directly to the write amplifier to be rewritten onto the delay line at the same time it is shifted into OR. See Figure 3-2 (c).

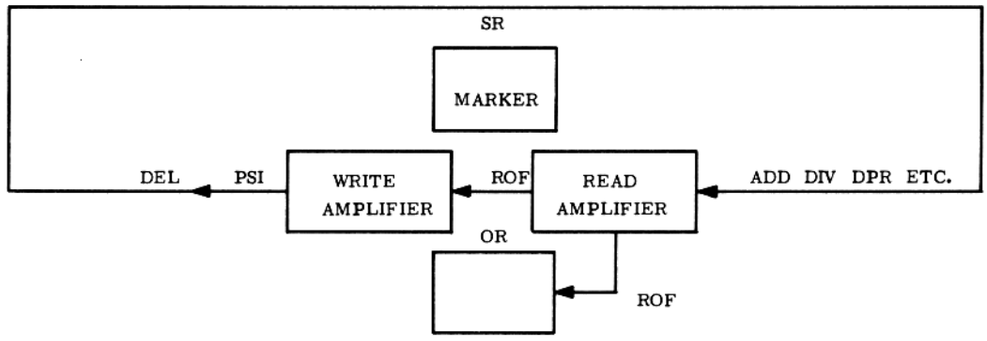


FIGURE 3-2 (c)

As soon as the command is completely shifted into OR, the line output is transferred to SR, and the output of SR is again sent to the program line, as shown in Figure 3-2 (d).

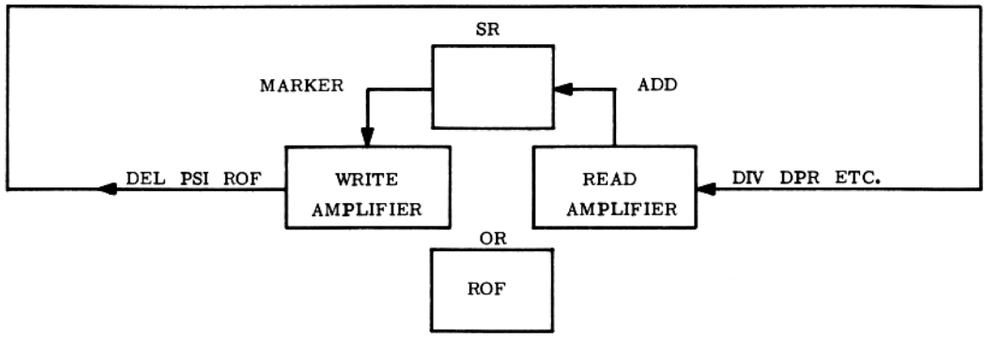


FIGURE 3-2 (d)

Since the marker was the last code in SR, it is written onto the line immediately after the command being performed. In this way, the marker is moved one character. See Figure 3-2 (e).

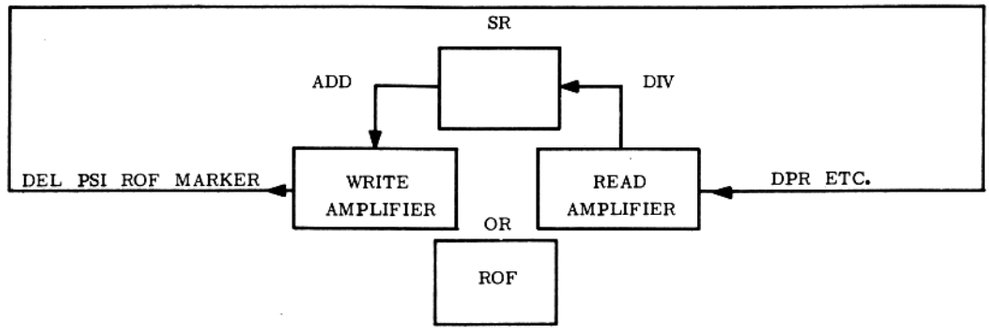


FIGURE 3-2 (e)

The operation remains the same until the end of the program appears, at which time the program resumes direct recirculation as shown in Figure 3-2 (f).

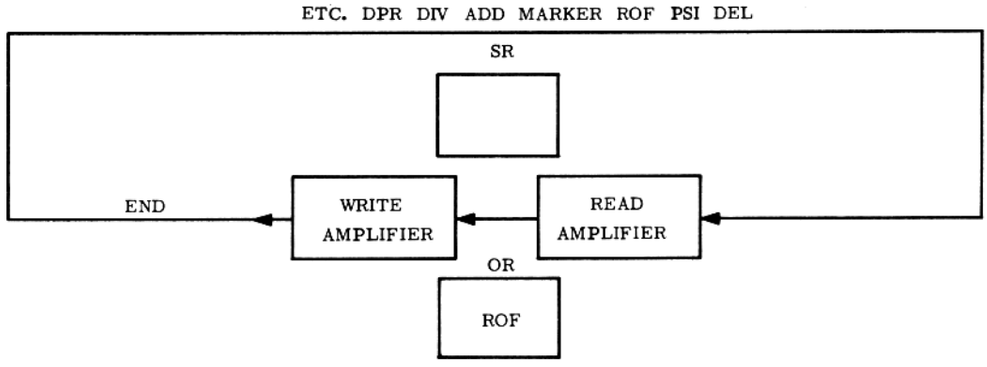


FIGURE 3-2 (f)

At this time the logic functions necessary to execute the command in OR are initiated. While the command in OR is being executed, data continues recirculating as shown in Figure 3-2 (f). When the command execution is complete, another search for the marker is initiated and the preceding operation is repeated.

3. LINE ROUTINGS

When data is recirculating on the program line, as described previously, the operation is termed "program line-to-program line", and is abbreviated PLPL. When the program is passing through the Search Register, the operation is termed Program Line-to-Search Register (PLSR) and Search Register-to-Program Line (SRPL). The program can also flow through the Common Register (CR) in which case the operation is termed PLCR, CRPL. When the program line data is routed into the Operation Register, the function is called PLOR. Note, however, that data cannot flow from OR back onto the program line. Therefore, in order to preserve the program on the delay line, PLOR must be accompanied by either PLPL, SRPL, or CRPL. It is possible for the output of the program line to go to several registers at one time, but only ONE source of data can be used to write back onto the line at any one time.

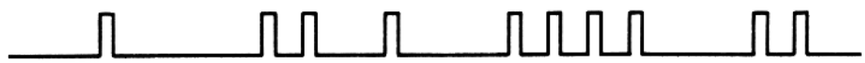
4. DATA ORGANIZATION

Each character loaded onto the program line is made up of a series of pulses representing a combination of bits. The maximum number of bits in any one character is six. These six bits, numbered 1-2-3-4-5-6, correspond to the 1-2-3-4-6-7 bits in the 2205 tape. The 5-bit and the 8-bit are not loaded onto the program line in order to conserve space, thus making it possible to load a larger total number of characters (1119). The 5-bit is not necessary because parity check is accomplished electronically. The carriage return code, which is a lone 8-bit in the 2205 tape, is converted to a different code when stored on the program line, and then it is re-converted back to an 8-bit when the carriage return is called out.

The chart below compares a few sample tape codes to their corresponding program line codes.

TAPE CODE	FLEXOWRITER CHARACTER OR FUNCTION	PROGRAM LINE CODE
1	1	1
5	Space	(blank)
1-6-7	A	1-5-6
1-2-5	3	1-2
2-3-5-6-7	F	2-3-5-6
8	Carriage Return	2-4
2-3-6	W	2-3-5
1-2-3-4-5-6-7	Tape Feed	1-2-3-4-5-6

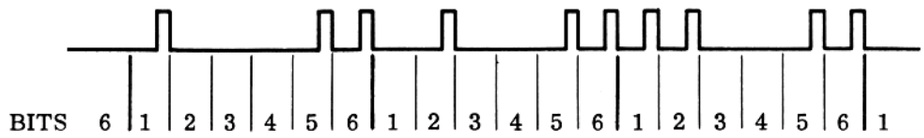
a. CHARACTERS. Figure 3-3 represents the read amplifier output waveform as the letters A, B, and C are read coming off the program line. This data alone is useless: it is impossible to tell where one character ends and where the next one begins, and it is impossible to tell what "bit" each pulse represents. It is apparent that some type of timing signal is also necessary to aid in decoding this waveform.



PROGRAM LINE DATA

FIGURE 3-3

Figure 3-4 shows the same waveform, only this time the individual bits are identified. Now the pulses can be translated into meaningful codes: 1-5-6, 2-5-6, 1-2-5-6, which represent the 2205 codes 1-6-7, 2-6-7, and 1-2-(5)-6-7 (A, B, and C).



DATA IN RELATION TO TIME

FIGURE 3-4

b. BITS. The 5610 also requires bit timing to which the read amplifier output can be compared. This timing is provided by the bit counter, a three-stage series shift counter. The six different states of the bit counter are decoded into equal "T" times, T1 through T6. Each T time is 1.6 μsec. long, so each 6-bit character time equals 9.6 μsec.

c. FRACTIONAL BITS. Each bit time is further divided into four equal fractional bit times; SH (SHift), A, B, and C. Each fractional bit time is 400 nsec. (nanoseconds) long. The fractional bit times are produced by the fractional bit counter, a two-stage series shift counter. The fractional bits are necessary to guide the routing of data between bits as they are sensed.

The fractional bit counter and the bit counter are referred to together as the Program Line Timing Chain, and this term will be used when referring to them in subsequent material.

5. BASIC TIMING

In order to sense data at the correct bit time, it must previously have been written at the proper time. After data is written onto the line, it holds the same time relationship to other data that was written, and maintains this same relationship to the other data as it comes off the delay line. However, due to different total delay times caused by minute differences in the length of delay lines, temperature changes, etc., a pulse written at T1 time would not necessarily come off of the line at T1 time. For this reason, the timing chain is stopped after the last program character is written onto the line, and then restarted as the first data again comes off of the line. In this way, the timing chain is synchronized with the data coming off the line. Since the data being read is in the correct time relationship to other data, the 5610 merely syncs the timing chain to the first pulse, and this causes all subsequent data to be sensed at the correct time. Since the timing chain is running in sync with the data being read, it is also used to control the rewriting of data back onto the delay line. In this way the data is recirculated in the correct time relationship.

6. SYNC PULSE

Different programs can begin with different characters, so the first character written onto the program line need not always be the same character. Consequently, the first data bit on the line can be any bit from 1 to 6. In order to start the timing chain correctly in relation to the data, a special pulse, called the "sync pulse", is always written ahead of the data. The sync pulse is always written in the same T time, so the timing chain always starts up in the same place, regardless of the position of the first data bit. It will be seen later that other variable data also precedes the first program character.

7. PROGRAM LINE HOME

Once the last bit of data is written onto the line the timing chain is no longer needed, so it stops running until the first data (sync pulse) again comes off the line. The longest program will use up about 10.75 ms. (1119 characters x 9.6 μsec.), and since the total delay time is approximately 12 ms., there will be about 1.25 ms. of dead time when all data is on the program line and no data is being read or written. This time is called "program line home". The length of program line home will vary depending upon the length of the program. If a very short program is loaded, program line home can be as long as 10 ms. or more.

8. END OF PROGRAM DETECTOR

As stated before, the timing chain stops running after the last bit of data is read. However, the only way the 5610 can detect the last bit of data is to detect an absence of data following the last bit. This is accomplished by an electronic circuit called the End of Program Detector (EPD). While data pulses are being sensed, the EPD allows the timing chain to run. When the last bit is read the EPD continues to look for more pulses. If no pulses are sensed in the next 40 μsec., the EPD stops the timing chain. In normal operation there will never be an absence of pulses within the program longer than about 30 μsec., so a 40 μsec. lapse in pulses will reliably indicate the end of the program. The operation of the End of Program Detector is explained in Section 8, Electronic Circuits.

9. OTHER IMPORTANT FLIP-FLOPS

Certain other flip-flops are necessary to the operation of the program line, and they will be mentioned briefly here. Their operation will be covered in detail shortly.

a. PROGRAM LINE READ. The PLR flip-flop is set by every pulse coming off the program line. The read amplifier output, called Program Line Read Signal (PLRS4) is normally true and goes false each time a pulse is sensed. This signal is inverted and PLRS3 sets PLR. Fractional bit time "SH" (SH3) resets PLR at the beginning of each T time, T1 through T6.

b. RE-TIME PROGRAM. The Re-TiMe Program flip-flop (RTMP) is set each time a pulse is to be written onto the line. If the program data is being recirculated, RTMP is enabled by PLR. During many machine operations the data flow can also be from PLR to one of the control registers, and then back to RTMP. In this case RTMP is enabled by some other logic signal. RTMP is set only at fractional bit "SH" time, the pulse is written out at "A" time, and RTMP is reset at "C" time.

c. PROGRAM LINE PRESET. The PLPS flip-flop is set at T4A time when the EPD senses the end of the program. It is reset by the sync pulse. The function of the PLPS flip-flop will be more easily explained later.

d. PROGRAM LINE HOME. The PLH flip-flop sets at the beginning of program line home and is reset by the sync pulse at the end of program line home.

B. PROGRAM LINE TIMING CHAIN

As mentioned before, the timing chain provides the signals which control the reading and writing of pulses as data is recirculated on the program line. As will be seen later, it also provides many of the signals necessary to the sequential performance of other program line logic operations.

Actual electronic operation of the special circuits in the timing chain will be covered in Section 8, Electronic Circuits; this section will discuss the timing chain operation primarily from the logic point of view.

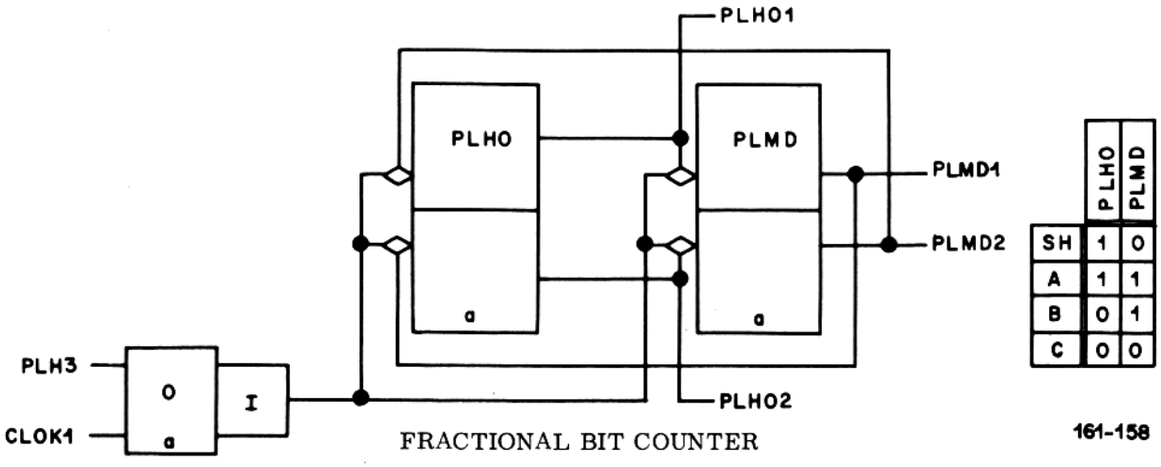
1. OSCILLATOR

The primary timing device (clock) in the timing chain is a 2.5 megahertz (megacycles-per-second) oscillator. This oscillator runs continuously whenever system power is "on". When the PLH flip-flop is reset, the oscillator drives the fractional bit counter.

2. FRACTIONAL BIT COUNTER

The fractional bit counter is a two-stage series shift counter made up of the Program Line Half Oscillator (PLHO) and Program Line MiD oscillator (PLMD) flip-flops. The outputs of PLHO gate the setting and resetting of PLMD, and the opposite outputs of PLMD are fed back to gate the setting and resetting of PLHO

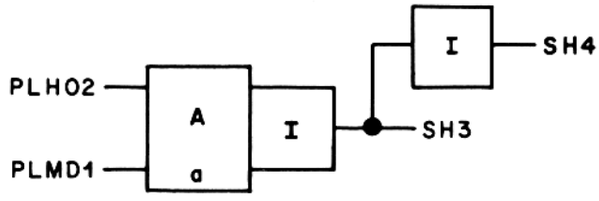
Figure 3-5 illustrates the fractional bit counter and shows how it is driven by the oscillator when the PLH flip-flop is reset. The truth table shows the settings of both flip-flops for each of the fractional bit times, SH, A, B, and C.



FRACTIONAL BIT COUNTER

FIGURE 3-5

Figure 3-6 illustrates the method in which the fractional bit counter flip-flops are used to develop the shift signals, SH3 and SH4. Similar gates are used for the other fractional bit signals.

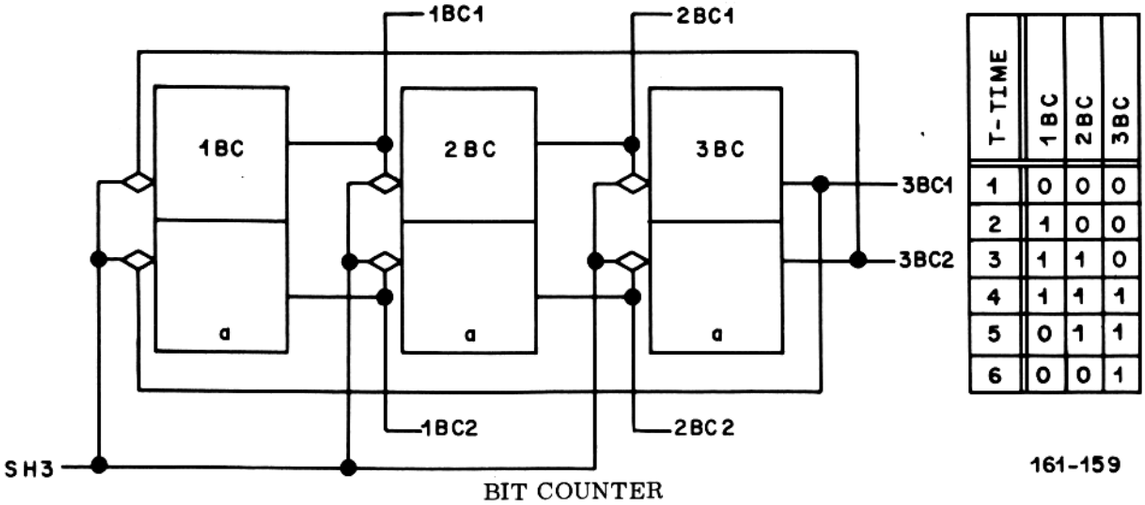


DECODING FRACTIONAL BITS

FIGURE 3-6

3. BIT COUNTER

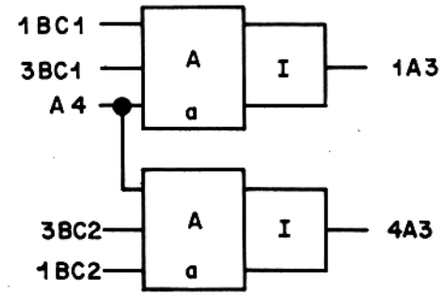
The bit counter is a three-stage series shift counter made up of the 1BC, 2BC, and 3BC flip-flops. It is essentially the same counter as the fractional bit counter with one more flip-flop added. Figure 3-7 illustrates the bit counter, and shows that it is driven by the SH3 signal, which goes true each time PLHO becomes set. The truth table shows the condition of the bit counter flip-flops during the various T times.



BIT COUNTER

FIGURE 3-7

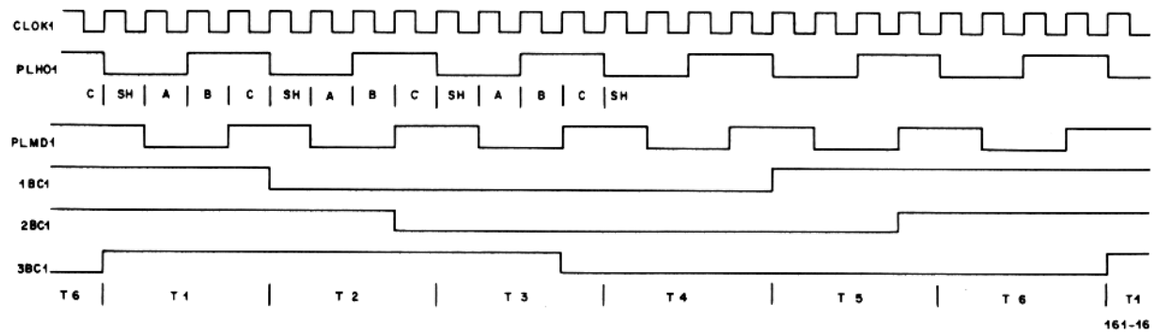
No 5610 logic operations require the timing signals T1, T2, T3, etc., but many operations do require specific fractional bit signals such as T6A, T6B, T6C, etc. The signals T1A and T4A are very important to the operation of the program line around program line home. The utilization of the bit counter flip-flops in the development of these signals is shown in Figure 3-8. Other necessary timing signals are decoded similarly.



DECODING SPECIFIC FRACTIONAL BITS

FIGURE 3-8

Figure 3-9 shows the timing relationships between all program line timing chain flip-flops.



TIMING CHART
PROGRAM LINE TIMING CHAIN

FIGURE 3-9

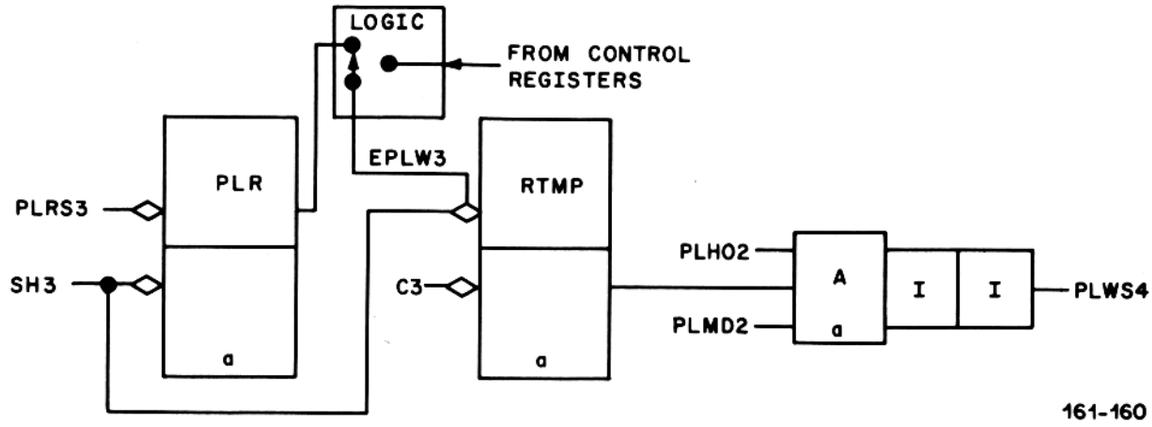
C. PROGRAM LINE OPERATION

Program line operation can be divided into three basic sections: the basic read/write logic, operation at program line home, and different line routings. The latter has already been mentioned briefly in this section, and it will be expanded upon later as some of the individual instructions and the control registers are studied. For the following explanation, assume that the line routing is PLPL.

1. BASIC READ/WRITE LOGIC

The basic read/write logic is illustrated in Figure 3-10. The read amplifier output is inverted and applied to the PLR flip-flop as PLRS3. Every pulse from the program line sets PLR. The PLR output enables the set gate of RTMP by causing Enable Program Line Write (EPLW3) to become true. The next SH3 signal going true both sets RTMP and resets PLR. When SH3 then

goes false, fractional bit "A" time becomes true. During "A" time PLHO2 and PLMD2 are both false, and with RTMP set, all three legs of the AND gate are false, so the Program Line Write Signal (PLWS4) goes false. PLWS4 goes to the write amplifier and causes a pulse to be written onto the line.

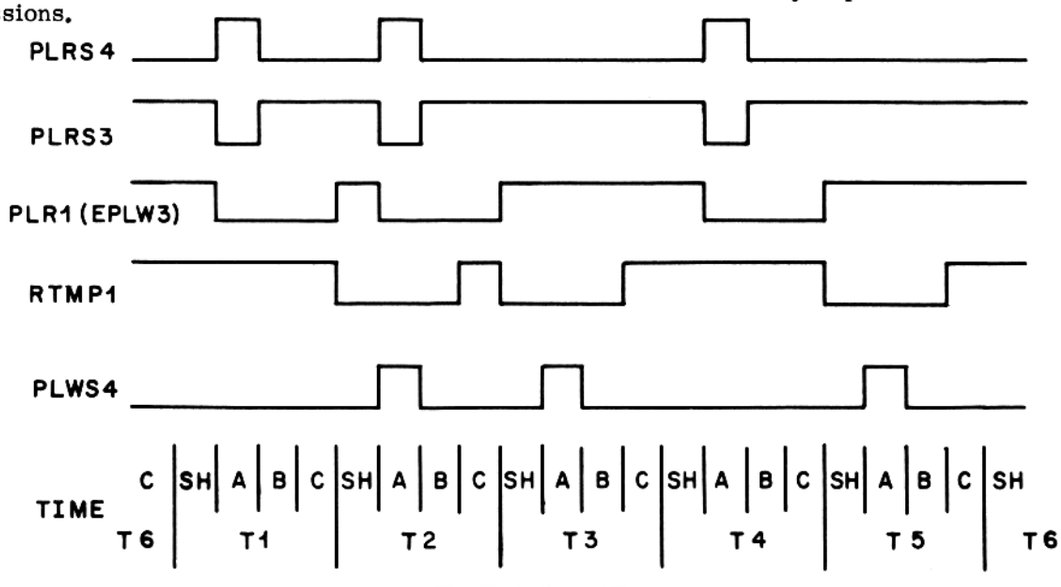


BASIC READ/WRITE LOGIC

FIGURE 3-10

When "A" time goes false, PLHO2 becomes true, and this drives PLWS4 true again (its normal state). At the next "C" time, C3 resets RTMP. If another pulse had been sensed by the read amplifier, PLR would be set again by this time, so as the timing chain progressed through "C" time and back to "SH" time, RTMP would set again, and another pulse would be written.

Figure 3-11 illustrates the timing relationships of the important signals involved in the previous discussion. Note that each pulse is given a value depending upon its relation to the timing chain as it is WRITTEN, and not when it is read. This fact will become very important in future discussions.



TIMING CHART
BASIC READ/WRITE
Example: Letter "W", 2-3-5 (2-3-6 tape code)

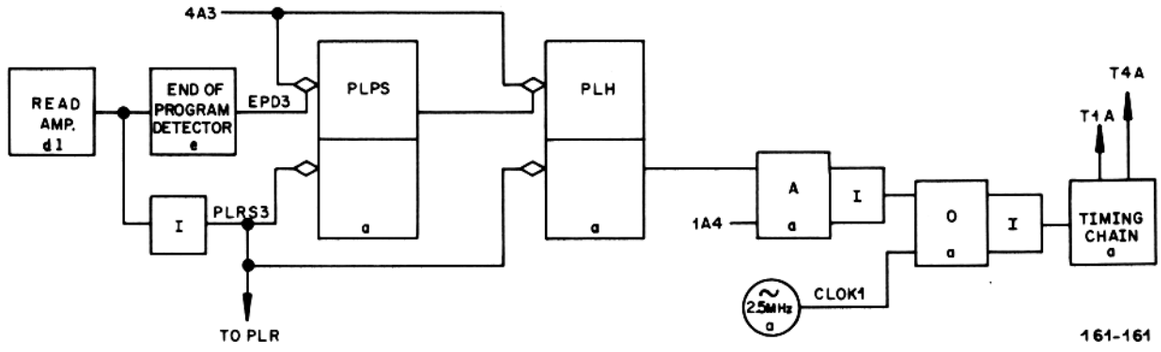
FIGURE 3-11

Note also in Figure 3-11 that when a pulse is read it is not perfectly synchronized with the timing chain. This is because the timing chain is driven by the oscillator, which never stops, and it is impossible to perfectly synchronize the entire delay time to the oscillator. In other words, each time a pulse is written, it is written in sync with the timing chain. By the time the pulse has traveled the entire length of the delay line, it is no longer synchronized with the timing. However, it is re-synchronized when it is written again.

2. OPERATION AT PROGRAM LINE HOME

Program line operation at program line home is primarily concerned with two processes: (1) stopping the timing chain after the last pulse is written and restarting it when the sync pulse is sensed, and (2) rewriting the sync pulse after the timing chain restarts.

a. TIMING CHAIN OPERATION. Figure 3-12 shows the basic elements that come into play at program line home. As long as the PLH flip-flop is reset, the upper input to the OR gate is false, and the OR gate output is the inverse of the oscillator signal, alternating at a 2.5 MHz (megahertz) rate and driving the timing chain. In order to stop the timing chain both inputs to the AND gate must become false.



OPERATION AT PROGRAM LINE HOME

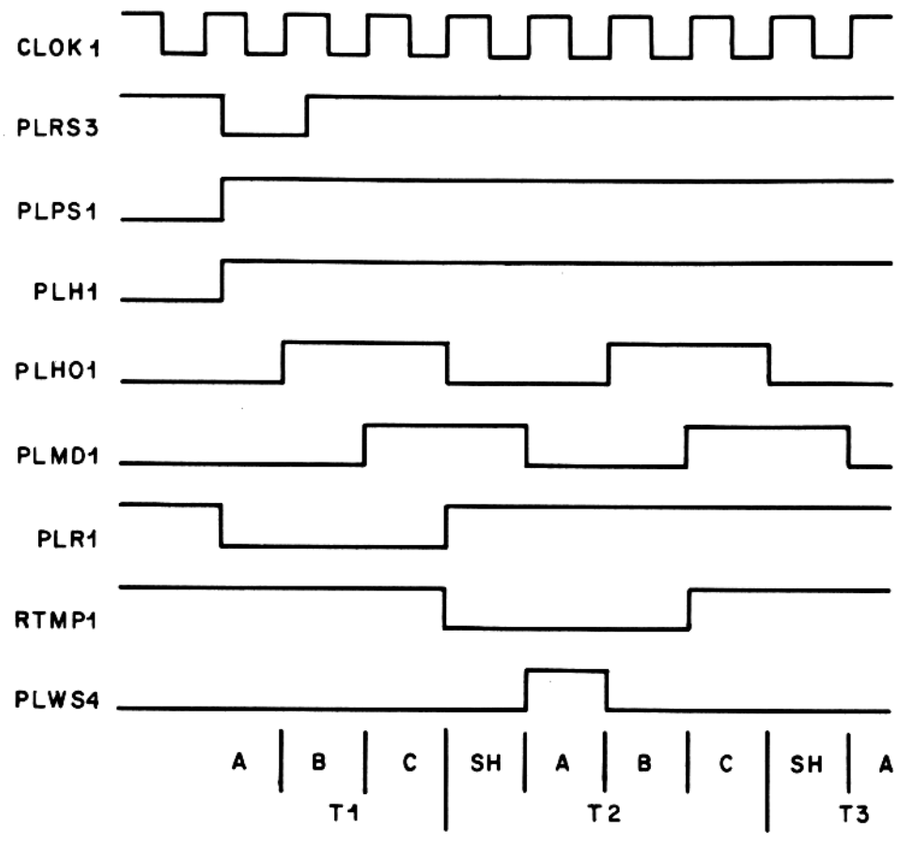
FIGURE 3-12

Whenever a pulse is sensed, PLRS3 resets PLPS and PLH, and the timing chain continues to run. After the last bit of data on the line is sensed the EPD looks for more pulses, and if none are sensed for 40 μsec., EPD3 goes true and enables the set gate of PLPS. Because the timing chain is still running, 4A3 will go true within 9.6 μsec. and set PLPS. PLPS1 enables the set gate of PLH, and 9.6 μsec. later another 4A3 going true sets PLH.

When PLH sets, the upper input to the AND gate goes false. The timing chain continues to run until T1A time when 1A4 goes false and causes the upper input to the OR gate to become true. With this input true the timing chain drive signal is clamped false and the timing chain is locked at T1A time.

PLPS and PLH stay in the set condition and the timing chain stays at T1A until the sync pulse is sensed by the read amplifier. When the sync pulse arrives, PLRS3 resets both PLPS and PLH. With PLH reset, the upper input to the OR gate goes false and the oscillator again drives the timing chain.

b. REWRITING THE SYNC PULSE. (Figure 3-13.) When the sync pulse is sensed and it unlocks the timing chain, it also sets PLR. The basic read/write logic then waits for the next SH3 signal which shifts the sync pulse from PLR to RTMP. At the next "A" time the sync pulse is written onto the line, as shown by the PLWS4 waveform. The timing chain then continues to run until the end of the program is sensed.



TIMING CHART
REWRITE SYNC PULSE

FIGURE 3-13

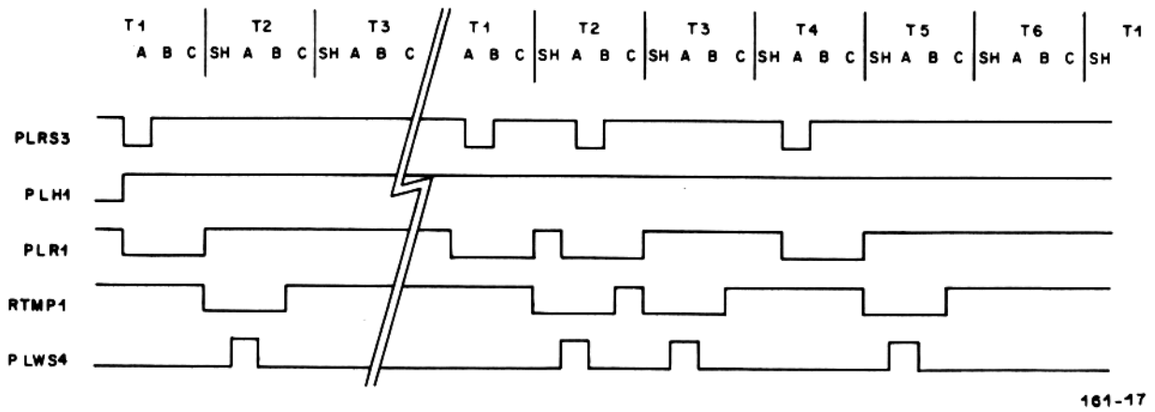
Also note in Figure 3-13 that the sync pulse is sensed during T1A time. This is because the timing chain is locked in the T1A state. The sync pulse is then written onto the line in T2A time, thus providing a delay of approximately one T time. This delay is also present for all subsequent data, and it is very important to the operation of the 5610, as will be seen later.

A third point to notice in Figure 3-13, and one that has been mentioned previously, is that the sync pulse is sensed at some random time in relation to the oscillator, but when it is written back onto the delay line it is back in sync with the oscillator and the timing chain.

Figure 3-14 illustrates the rewriting of the sync pulse and one additional character. Note that even though each pulse is written slightly after it is sensed, (approximately one T time), the amount of delay in all cases is equal, and when the pulses are rewritten they have the same time relationship to each other as they had when they were read.

Again note (in Figure 3-14) that the sync pulse is read in T1 time and written in T2 time. The first pulse in the next character is also sensed at T1 and written at T2. It actually represents a 2-bit. (A 1-bit would be sensed during the previous T6 time.) On the next line pass (the next time the data comes off of the delay line), the sync pulse will again unlock the timing chain at T1A and be written at T2 time. Since the timing chain always starts "earlier" in respect to when the data was last written, the data will be sensed "early" and be rewritten at the same T time that it was previously written.

Again, this ability to sense the data "early" in respect to the timing chain will become very important later on.



TIMING CHART
RECIRCULATION OF DATA

FIGURE 3-14

3. LAUNCH PROGRAM LINE SYNC PULSE

Once the basic operation of the program line and its delay line is understood, the last item to be covered before entering into the program load operation is that of launching the program line sync pulse.

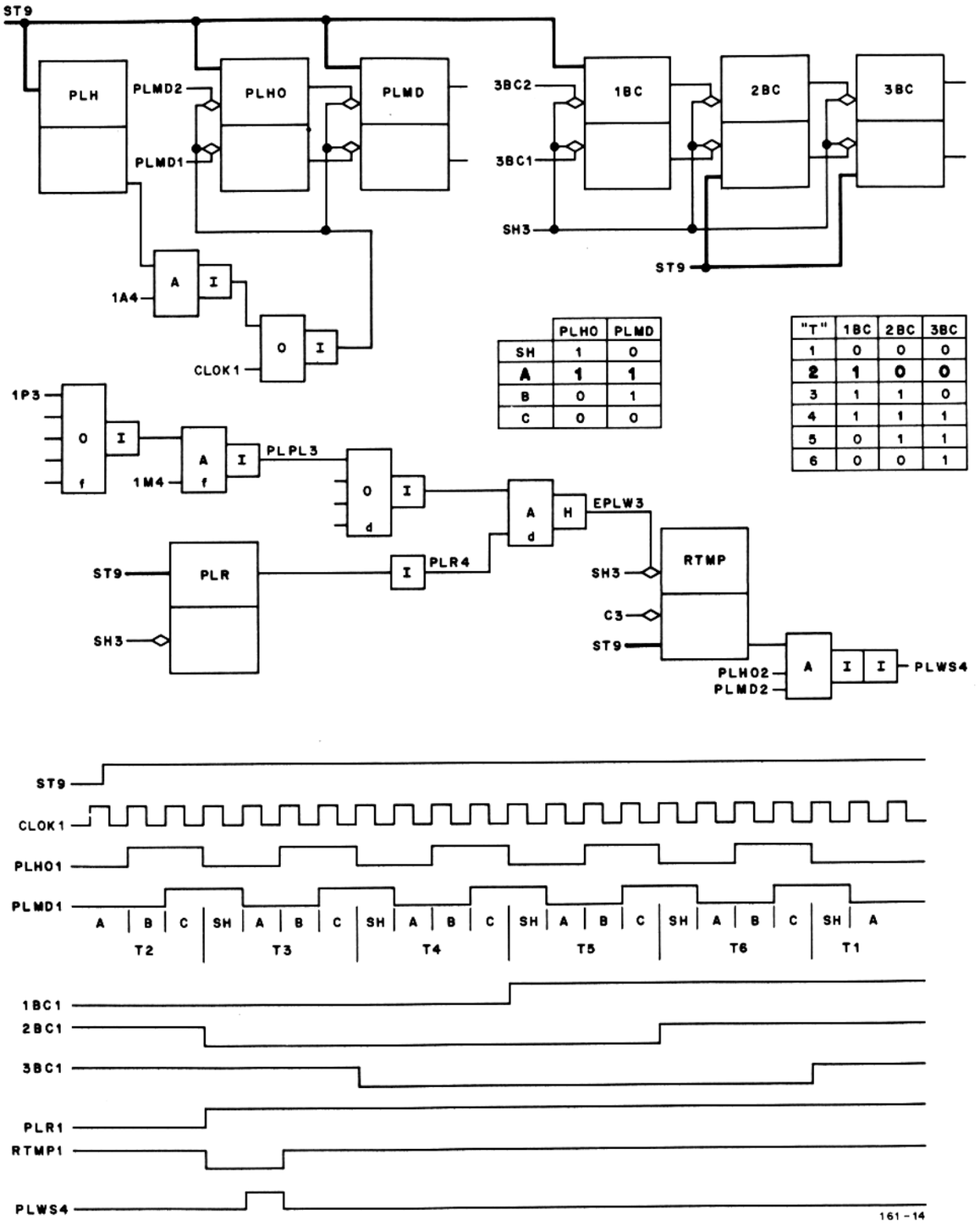
When power is first applied to the system, the STart signal, ST9, sets PLH, PLHO, PLMD, 1BC, and PLR. It resets 2BC, 3BC, and RTMP. The effect on the program line timing chain, then, is to automatically set it in the T2A condition. Since T1A is false at this time, the oscillator is trying to drive the fractional bit counter, but it cannot because PLHO and PLMD are clamped by ST9. This is illustrated in Figure 3-15.

ST9 also sets the mode register to 1 mode and the phase counter to 1 phase. Still referring to Figure 3-15, note that the combination of 1M4 being false and 1P3 being true makes the Program Line-to-Program Line routing (PLPL) true. Since PLR is now set, PLR4 is false. These two signals combine to drive Enable Program Line Write, EPLW3, true, thus enabling the set gate of ReTiMe Program, RTMP. All of the preceding conditions are maintained while ST9 is true.

When ST9 goes false it removes the clamp from the counters and the timing chain starts to count. Nothing happens during T2B and T2C, but at T3SH time RTMP becomes set and PLR resets. When PLR resets, EPLW3 becomes false. At T3A, PLWS4 goes false and the sync pulse is written onto the program line.

Next, at T3B time, PLWS4 again goes true, and at T3C time RTMP resets. Since PLR is now reset and EPLW3 is false, RTMP stays reset. The timing chain continues to count until T1A time when the input to PLHO and PLMD is clamped false. No further action occurs until 12 ms. later when the sync pulse is sensed by the read amplifier and PLH is reset, allowing the timing chain to unlock and rewrite the sync pulse.

It will be noted shortly that this same procedure also occurs whenever the Start pushbutton, S9, is operated.



	PLHO	PLMD
SH	1	0
A	**1**	**1**
B	0	1
C	0	0

"T"	1BC	2BC	3BC
1	0	0	0
2	**1**	**0**	**0**
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1

LAUNCH PROGRAM LINE SYNC PULSE
(All on "A" card unless noted)

FIGURE 3-15

D. START SIGNALS

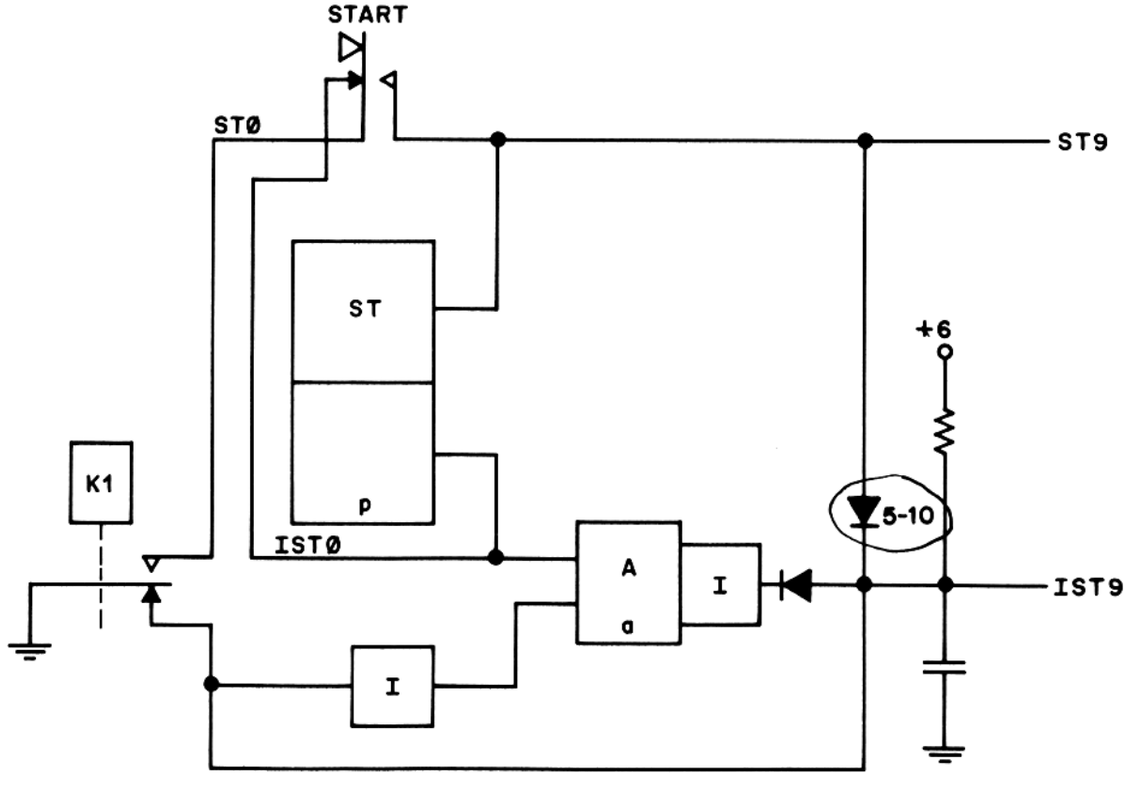
There are two "start" signals, STart, ST9, and Initial STart, IST9. IST9 becomes true only during the first second after the 5610 power is turned on. ST9 is also true at this time, but in addition, it becomes true whenever the Start pushbutton, S9, is operated. ST9 resets all of the flip-flops needed to establish the start-up conditions. IST9 resets the storage line timing chain and launches the storage line sync pulse. (See Section 6, Storage Line Logic.)

1. START PUSHBUTTON (S9)

The Start pushbutton, S9, is not normally used by the 5610 operator. It is, however, a useful tool to both the technician and the programmer. When operated, it causes the reset of all of the flip-flops required to "restart" the 5610 without disturbing the data on the storage line. It is normally used to restart the 5610 after it falls into an endless search for non-existent codes (non-existent Resume titles, etc.), caused by software errors or hardware malfunctions.

2. LOGIC (See Figure 3-16)

During the first second after the machine is turned on, Initial STart, IST9, is true. The circuit used to energize K1 after 1 second is covered in Section 8, Electronic Circuits. This resets the storage line timing chain and, through diode 5-10, drives ST9 true. ST9 sets and resets the other flip-flops throughout the machine necessary to establish the proper start-up conditions. It also sets the ST flip-flop. This drives IST0 false, enabling one leg of the AND gate. IST9 inverted enables the other leg, causing the output of the gate to be true. This, then, holds IST9 true.



START LOGIC
FIGURE 3-16

TP-161SL

As the K1 relay begins to pick there is an instant of time when neither stationary contact is touching the common contact. During this time IST9 remains true because both inputs to the gate are false. Also during this time, IST9 holds ST9 true.

As soon as the K1 common contact touches the normally-open contact, ground is applied to one input of the gate, driving its output and IST9 false. At the same time, the ST flip-flop resets, so even if the relay contacts bounce, the input to the gate remains true. This drives the gate output false, and +6 volts through the 1.8K resistor holds IST9 false. This false state is inverted by the other inverter, holding the input to the gate true. Thus the gate output remains false until after the machine is turned off and turned back on again.

If the Start pushbutton is now operated, ground is applied to ST9 and the ST flip-flop sets. IST9 remains false because diode 5-10 is reverse biased. As the pushbutton is depressed, there is an instant of time when no two contacts are touching. During this time ST9 remains false because the ST flip-flop is reset. As soon as the switch transfers, ST sets and ST9 becomes true. If the switch contacts should bounce, the output of ST9 is held smooth by the ST flip-flop. A similar but opposite action occurs during the restore of the switch.

Note that even though operation of the pushbutton applies a false signal to one input of the AND gate, its output remains false because of the true signal on its other input.

E. BASIC PROGRAM LINE LOGIC

There are three basic divisions of the program line logic which have not yet been covered. These are the Mode Register, the Phase Counter, and the Control Registers. The latter is actually a group of registers and counters, each of which will be given individual attention. The general operation of the Reset switch is also covered here.

1. MODE REGISTER

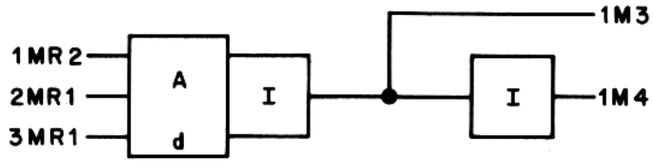
The mode register keeps record of the mode in which the 5610 is operating. The six modes of operation are as follows:

(1) Program Load
(2) Translate
(3) Search
(4) Command Retrieval
(5) Execute
(6) Special Flexowriter Functions

The operator can put the 5610 into 1 mode by depressing the Load switch, into 2 mode by depressing the Translate switch, or into 3 mode by re-depressing the Load switch when the machine is already in the Load mode. The operation through the other modes is automatic, with signals being developed at the proper times to cause the jump from one particular mode to another.

It should be noted that the mode register is NOT a counter: the present setting of the register does not necessarily dictate the next setting. Rather, each time the operations in a given mode are finished, part of the program line logic will set or reset those flip-flops necessary to change modes.

The mode register is made up of six flip-flops. Three of them, 1 MR, 2MR, and 3MR, have their outputs decoded to let the system know which of the six modes is occurring. Figure 3-17 shows the decoding of the mode register flip-flop outputs to develop the 1M3 and 1M4 signals. The other mode signals are decoded similarly. The truth table shows that the decoding of the entire register is purely binary.



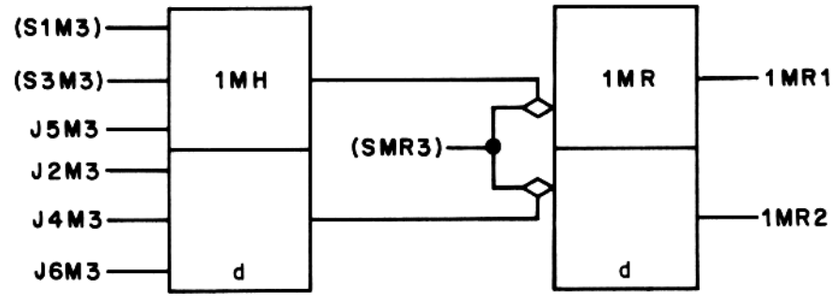
MODE	1MR	2MR	3MR
1	1	0	0
2	0	1	0
3	1	1	0
4	0	0	1
5	1	0	1
6	0	1	1

MODE REGISTER DECODING

FIGURE 3-17

The other three flip-flops, 1MH, 2MH, and 3MH, are the Mode Hold flip-flops. They contain the setting of the next mode to occur. Just before all operations in a given mode are completed, the code for the next mode to occur is set into the mode hold flip-flops. Then, just as the given mode finishes, the settings of the mode hold flip-flops are shifted into the mode register flip-flops.

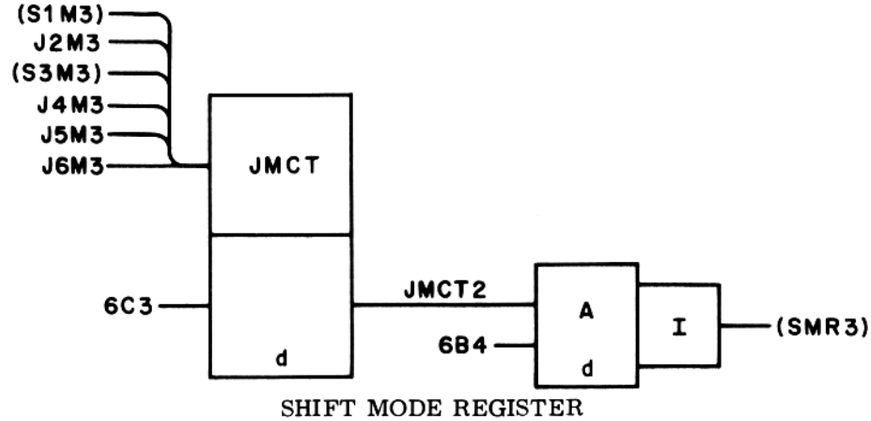
Figure 3-18 illustrates the 1MH and 1MR flip-flops. Various signals can set or reset 1MH, and when the shift pulse (SMR3) occurs, the contents of 1MH are shifted into 1MR. The flip-flops representing the binary 2 and binary 4 are arranged similarly.



MODE HOLD ACTION ON MODE REGISTER

FIGURE 3-18

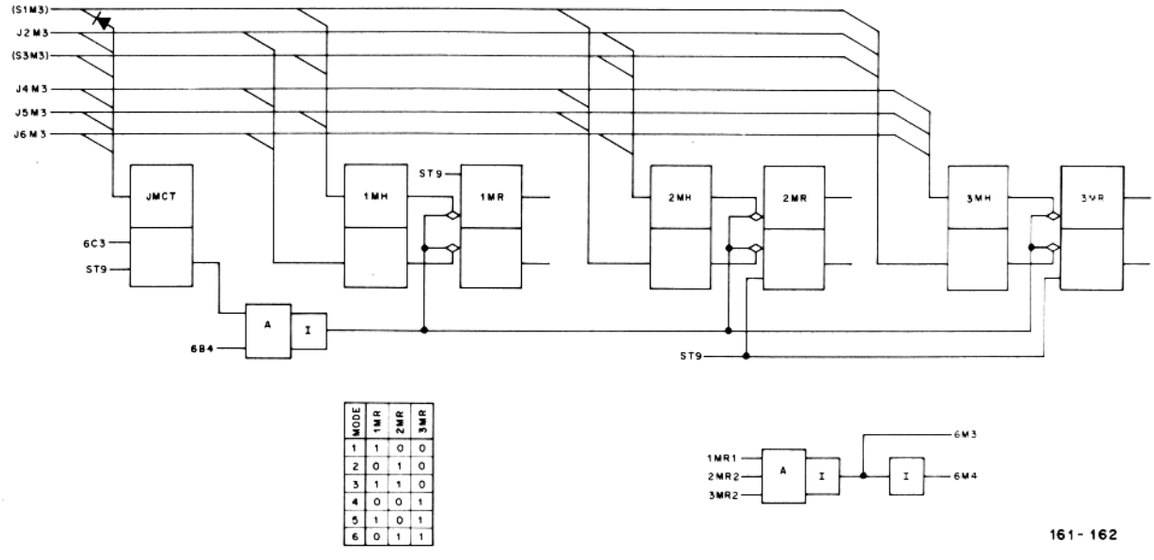
The Shift Mode Register signal (SMR3) is developed as shown in Figure 3-19. All signals that either set or reset any of the mode hold flip-flops also set the Jump Mode CounT flip-flop, JMCT.



SHIFT MODE REGISTER

FIGURE 3-19

At the following T6B, 6B4 goes false, driving SMR3 true and shifting all mode hold flip-flop settings into the mode register flip-flops. 400 nsec. later 6C3 becomes true and resets JMCT, preventing any further changes in the mode register until JMCT again sets.



MODE REGISTER
(All on "D" card)

FIGURE 3-20

The development of all of the signals that set JMCT and the mode hold flip-flops will be covered later. For now it is sufficient to note that ST9 sets the mode register to 1 mode when system power is first applied. It also resets JMCT. Refer to Figure 3-20 and note that when the machine is first turned on the mode hold flip-flops will be set in a random configuration. However, when the program load is finished, the logic will cause a jump to a specific mode (3 mode) by setting JMCT and the required MH flip-flops.

Figure 3-20 also shown another example of decoding the MR flip-flop outputs to produce the mode logic signals.

2. PHASE COUNTER

The purpose of the phase counter is to enable groups of operations to be performed in the correct order, or sequence. For example, when performing a Negative Branch instruction, the phase counter first allows the decision to be made as to whether or not the branch will be made (is the sign of R1 negative?). After this determination is made, the phase counter allows the branch to occur, or, if R1 is positive, causes the next instruction to be retrieved, thus ignoring the Negative Branch. In other words, the phase counter allows for the ORDERLY performance of each step in the execution of an instruction.

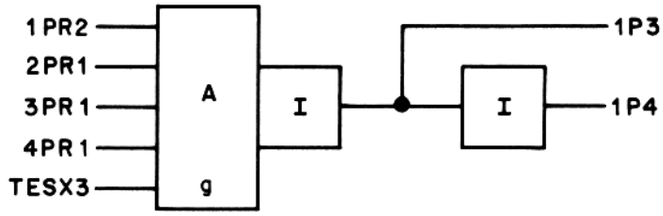
The phase counter accomplishes the foregoing by controlling all line routings, all sets and resets to registers and individual flip-flops, the operation of the mode counter, and even its own operation. In other words, the proper operation of the phase counter is imperative to proper operation of the rest of the system: any other part of the 5610 can malfunction if the phase counter is operating incorrectly.

The phase counter is a modified four-stage series shift counter. Its internal gating is such that it has fifteen possible states. In addition to counting through the various states, the phase counter has the ability to jump over unwanted states (it can jump either "forward" or "backward"). In actual operation, it rarely counts through more than four or five consecutive states before jumping ahead to another state or back to some prior state.

The phase counter is triggered by the program line timing chain so that all phases (with one exception, that will be covered shortly) are changed near the end of a character time (at T6B). The decoded phase counter outputs are combined with the mode register outputs and other logic signals to provide the various line routings and other logic functions.

a. COUNTER MAKE-UP. There are eight flip-flops in the basic counter, four Phase Register (PR) flip-flops and four Phase Hold (PH) flip-flops. The PH flip-flops operate somewhat like the mode hold flip-flops in that their contents are shifted into the PR flip-flops when it is time to change phases. The major difference is that the outputs of each PR flip-flop control the setting and resetting of the next higher PH flip-flop, thus making a counter out of the system.

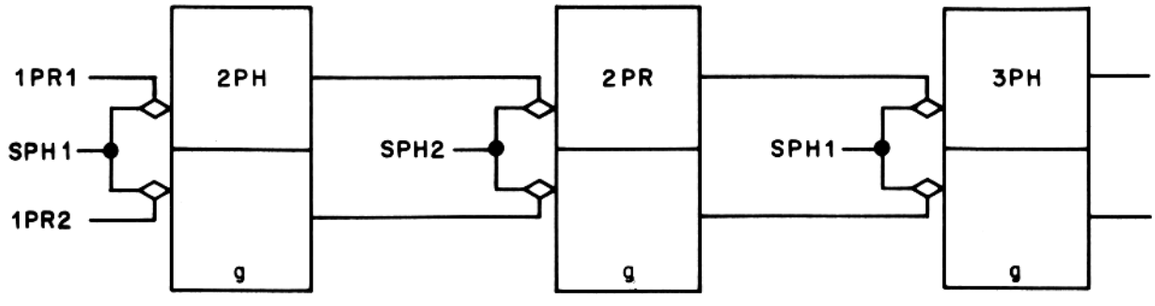
b. DECODING. As in the mode register, the PR flip-flop outputs are decoded to provide the various logic signals necessary for the different line routings and other program line logic operations. An example of decoding a phase signal is shown in Figure 3-21. TESX3 is a test signal which enables the technician to stop the machine operation when a certain phase has been reached. Other phases are decoded in a similar manner.



PHASE COUNTER DECODING

FIGURE 3-21

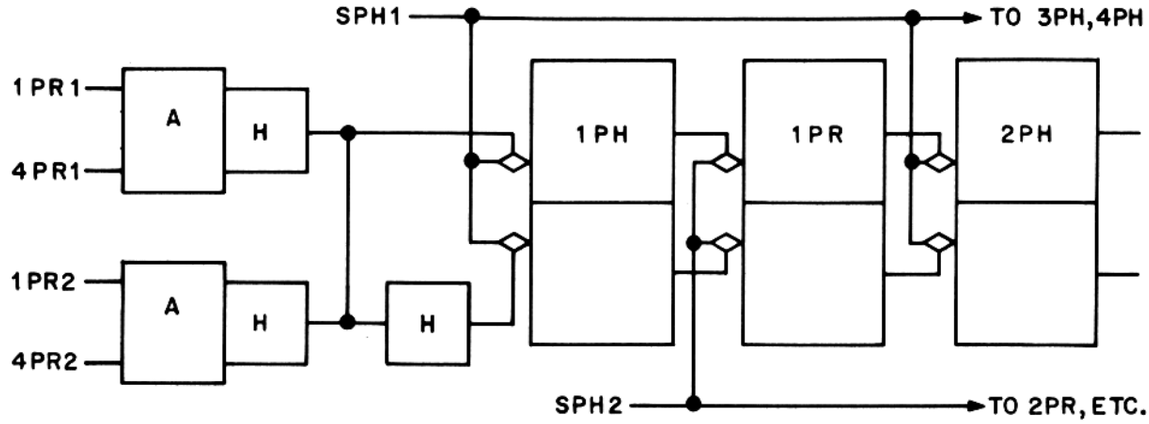
c. STEPPING OPERATION. Figure 3-22 illustrates three of the flip-flops in the counter. Note that as SPH1 goes true the contents of 1PR and 2PR are shifted into 2PH and 3PH, respectively. (The operation of the SPH flip-flop will be covered shortly.) In this way, the phase register setting is set into the next higher phase hold setting. When SPH2 then goes true, the contents of 2PH and 3PH are shifted into 2PR and 3PR, respectively. Since it is the PR flip-flop outputs that are decoded into the phase signals, the phase actually changes when SPH2 becomes true and the PR flip-flops change.



PHASE COUNTER SHIFTING ACTION

FIGURE 3-22

Since the overall counter operation is that of a shift register, the input to 1PH determines whether a true or a false (1 or 0) will be shifted through all of the flip-flops. Figure 3-23 illustrates the input to 1PH. If either 1PR or 4PR is set, a 0 will be shifted into 1PH. If 1PR and 4PR are in the same condition (either set or reset), a 1 will be shifted in.



PHASE COUNTER ACTION DURING COUNTING
(All on "G" card)

FIGURE 3-23

Assume that all flip-flops are reset (15 phase). 1PR1 and 4PR1 are both false, so the set gate to 1PH is enabled, and 1PH will set when SPH1 goes true. When SPH2 becomes true the contents of 1PH shift into 1PR. This disables the set gate to 1PH and enables the reset gate, so when the SPH flip-flop again sets and SPH1 goes true, 1PH resets. (At the same time, the set condition of 1PR shifts into 2PH.) When SPH resets and SPH2 goes true, 1PR will reset and 2PR will set. By following this procedure from beginning to end, the truth table shown in Figure 3-24 can be developed.

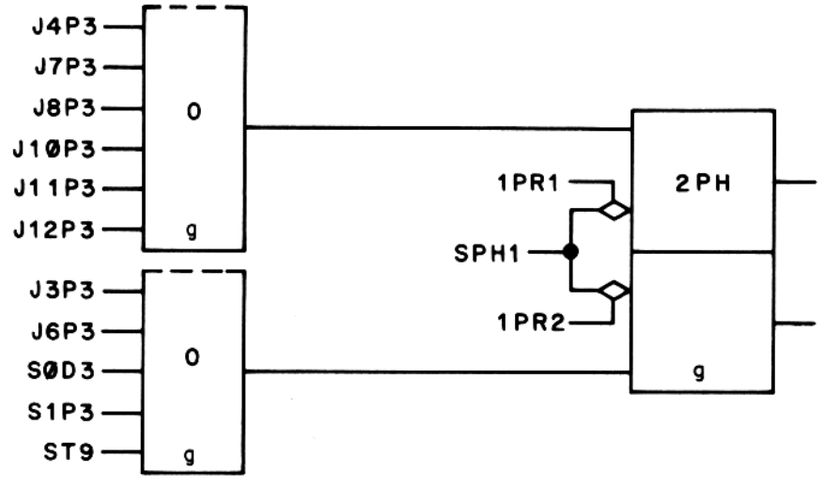
PHASE	1PR	2PR	3PR	4PR
1	1	0	0	0
2	0	1	0	0
3	1	0	1	0
4	0	1	0	1
5	0	0	1	0
6	1	0	0	1
7	1	1	0	0
8	0	1	1	0
9	1	0	1	1
10	1	1	0	1
11	1	1	1	0
12	0	1	1	1
13	0	0	1	1
14	0	0	0	1
15	0	0	0	0

TRUTH TABLE
PHASE COUNTER

FIGURE 3-24

Another way of defining the input to 1PH is as follows: if 1PR is set, 1PH will next assume the state now held by 4PR; if 1PR is reset, 1PH will next assume the state opposite to that now held by 4PR. Still another way of saying it is that if 4PR is set, 1PR does not change: if 4PR is reset, 1PR will toggle.

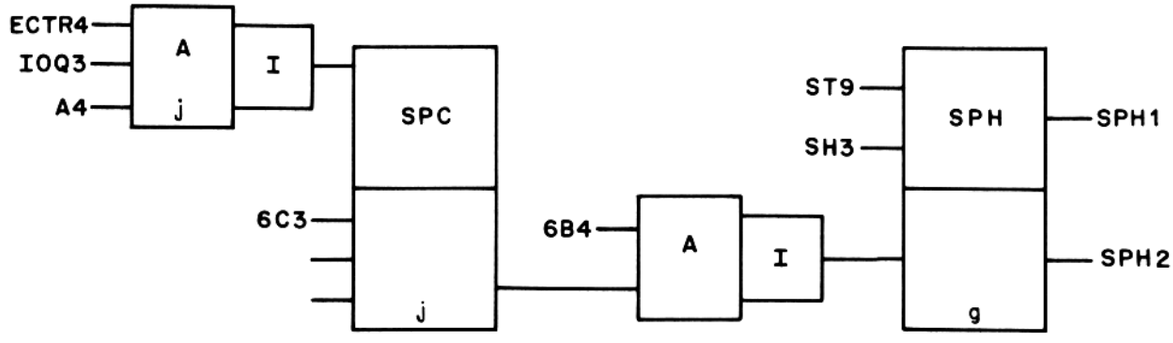
d. JUMPING OPERATION. In addition to the normal stepping action just described, the phase counter also has the ability to jump. This is accomplished in a fashion similar to that employed by the mode register. Different signals set and reset the various PH flip-flops into the desired configuration, and when SPH2 goes true the PR flip-flops assume the new phase setting, regardless of what the old setting was. Note, however, that the new phase setting must be set into the PH flip-flops AFTER the SPH flip-flop sets, for whenever SPH sets, the PH flip-flops each assume the state of the previous PR flip-flop. Figure 3-25 shows some of the signals that set and reset 2PH in order to perform jumps. Each of the PH flip-flops has a total of 18 inputs to the two OR gates. These inputs provide the jumps to the proper phase settings as different logic operations are performed.



INPUTS TO PH FLIP-FLOPS

FIGURE 3-25

e. TRIGGERING. The basic triggering element of the phase counter is the Shift Phase Hold (SPH) flip-flop. This flip-flop is set at every program line fractional bit "SH" time, as shown in Figure 3-26.



PHASE COUNTER TRIGGERING

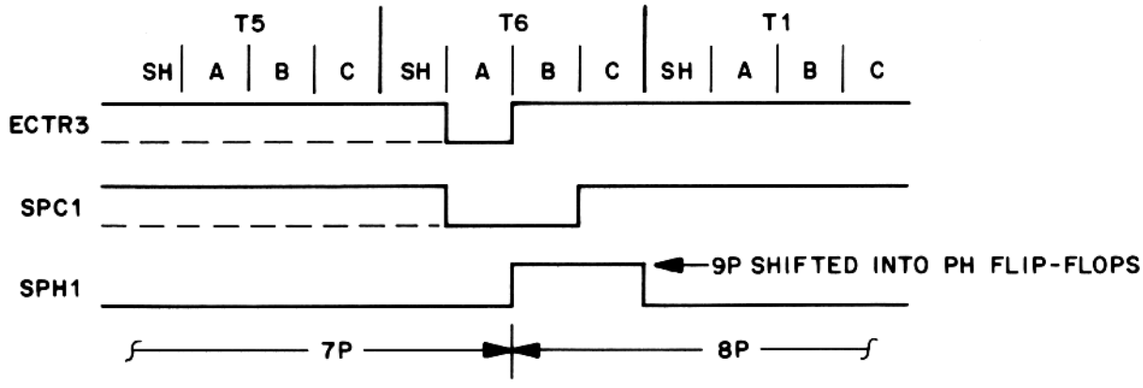
FIGURE 3-26

It will reset at T6B time after the Shift Phase Count (SPC) flip-flop becomes set. SPC sets at fractional bit "A" time when Enable CounTeR (ECTR) is true, and resets at the next T6C time. (IOQ3 becomes true only during execution of the Halt command: this is covered in Section 6, Flexowriter Commands.) ECTR can be generated in many different ways: its development will be covered when program line operation is analyzed more closely. The phase actually changes when SPH resets and SPH2 becomes true. Thus, the sequence of events is as follows:

(1) Set SPH
(2) Develop ECTR3
(3) Set SPC
(4) Reset SPH, which changes the phase
(5) Reset SPC
(6) Set SPH at the next SH3. This sets the PH flip-flops to the next higher phase configuration. Wait for the next ECTR3.

Note that SPH sets at "SH" time. If it resets at T6B, it will again set at T1SH. Therefore, the next phase is set into the PH flip-flops only 800 nsec. after a given phase is reached.

Figure 3-27 illustrates the timing relationships involved in a typical step from 7 phase to 8 phase. ECTR3 is often gated by T6A time as shown by the solid lines. It is also possible for ECTR3 to remain true from one phase to another, and this condition is represented by the dotted lines in Figure 3-27. Every gate that produces ECTR3 has a phase signal for one of the inputs, so as the phase is changed, ECTR3 often becomes false. The important point to note is that regardless of when ECTR3 becomes true, the phase setting does not change until T6B time. (The one and only exception to this follows.)

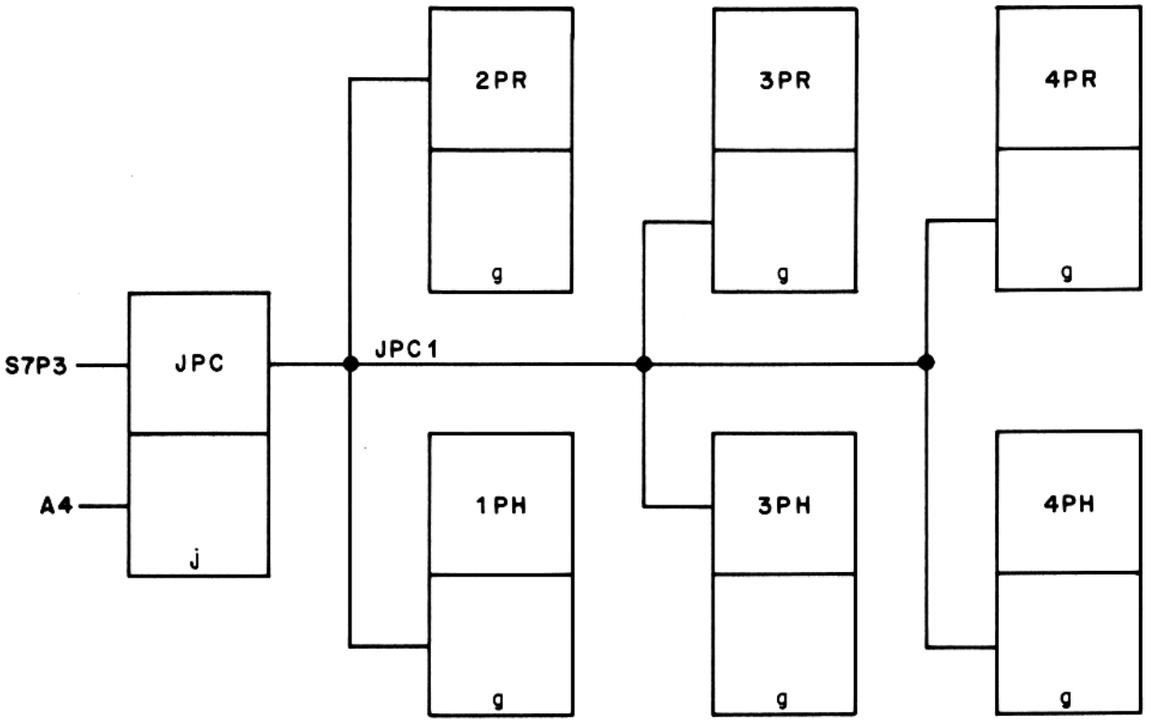


TIMING CHART
PHASE COUNTER STEP

FIGURE 3-27

f. SET 7 PHASE. In one case only, a jump is made at other than T6B time. This is the jump from 9 phase to 7 phase. This jump occurs during 3 mode while the various searches (marker search, sequence separator search, etc.) are being conducted. The reason for having this jump occur immediately will be covered later in this section under G, Mode 3, Search.

The signal Set 7 Phase, S7P3, can occur at any fractional bit "C" time, T1 through T6, during 9 phase. It immediately sets the Jump Phase Count (JPC) flip-flop (see Figure 3-28). As soon as JPC sets, it immediately sets 2PR and resets 3PR and 4PR. By referring to the truth table, note that this condition of 2PR set with 3PR and 4PR reset yields either 2 phase or 7 phase. As will be seen later, S7P3 can be developed only during 9 phase, at which time 1PR is set. Therefore 1PR and 2PR are set, and 3PR and 4PR are reset, yielding 7 phase.



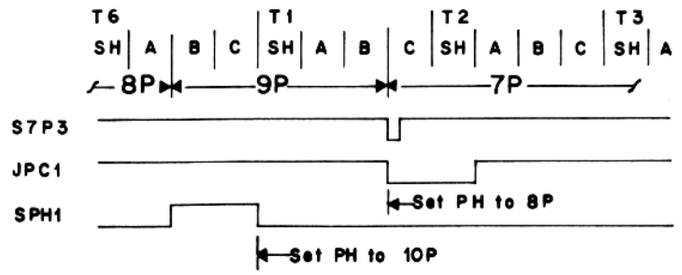
SET 7 PHASE

FIGURE 3-28

The method of setting the PH flip-flops is slightly different, depending upon when S7P3 occurs.

(1) S7P3 During T1 Through T5. If S7P3 occurs during T1 through T5 time, the SH3 signal has already been true in 9 phase, and the PH flip-flops contain the 10 phase setting (1PH, 2PH, and 4PH set, 3PH reset). When JPC sets, JPC1 sets 3PH and resets 1PH and 4PH. This leaves 2PH and 3PH set, and 1PH and 4PH reset, or the 8 phase configuration. Thus JPC sets the PR flip-flops to 7 phase and the PH flip-flops to 8 phase. Note that ECTR3 is not necessary for this jump to occur.

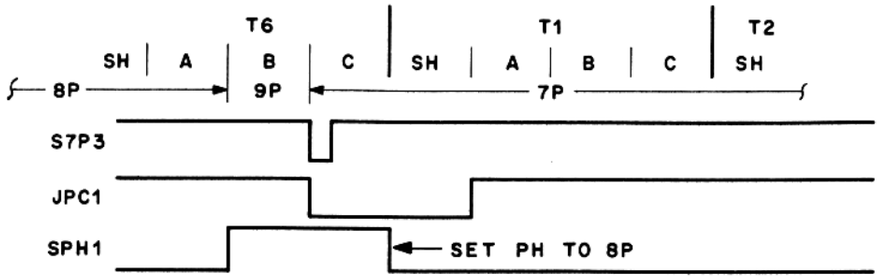
Figure 3-29 (a) illustrates the timing relationships involved in this 9 phase to 7 phase jump. Since 9 phase is one of the qualifications necessary to produce S7P3, the jump from 9 phase to 7 phase drives S7P3 false. S7P3 is actually about 100 nsec. long due to the delay in switching times of the many gates involved.



TIMING CHART
S7P3 DURING T1-T5

FIGURE 3-29 (a)

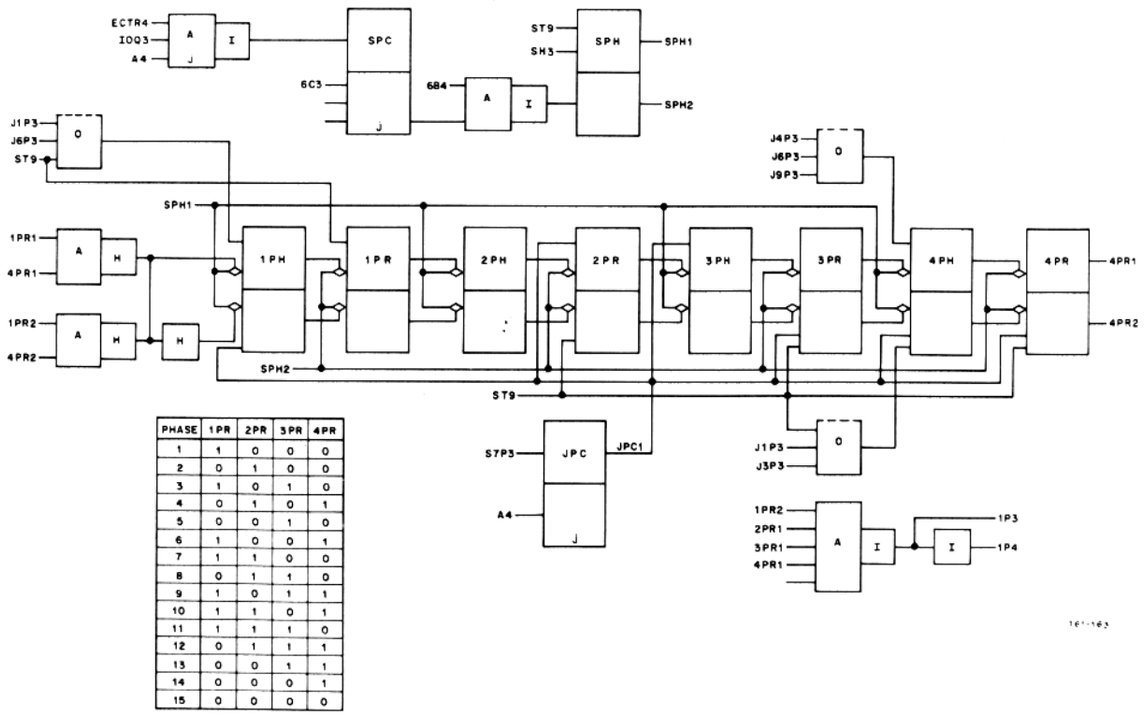
(2) S7P3 at T6C. If S7P3 occurs immediately after entering 9 phase, at T6C time, the PH flip-flops still contain the 9 phase setting, 1-0-1-1. However, as soon as S7P3 occurs, the PR flip-flops assume the 7 phase setting. Then, when SPH sets at T1SH time, the PH flip-flops assume the 8 phase setting because they are gated by the PR flip-flops. This is shown in Figure 3-29 (b).



TIMING CHART
S7P3 DURING T6

FIGURE 3-29 (b)

g. START CONDITIONS. Figure 3-30 brings together all of the flip-flops associated with the phase counter. Note that when power is first applied, ST9 sets 1PH and 1PR, and resets 2PR, 3PR, 4PH, and 4PR. Although not shown in the drawing, ST9 also resets 2PH and 3PH, through OR gates similar to those shown for 1PH and 4PH. In other words, ST9 sets the phase counter to 1 phase. ST9 also sets SPH. It was shown earlier that this has significance in launching the program line sync pulse.



PHASE	1PR	2PR	3PR	4PR
1	1	0	0	0
2	0	1	0	0
3	1	0	1	0
4	0	1	0	1
5	0	0	1	0
6	1	0	0	1
7	1	1	0	0
8	0	1	1	0
9	1	0	1	1
10	1	1	0	1
11	1	1	1	0
12	0	1	1	1
13	0	0	1	1
14	0	0	0	1
15	0	0	0	0

PHASE COUNTER OPERATION
(All on "G" card unless noted)

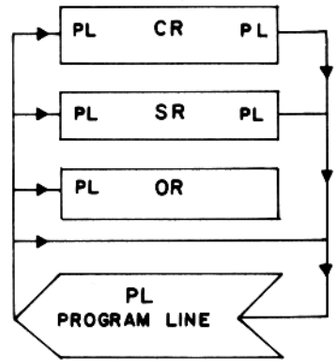
FIGURE 3-30

3. CONTROL REGISTERS

The four units that make up the group called the control registers are as follows:

(1) Operation Register (OR)
(2) Search Register (SR)
(3) Common Register (CR)
(4) Decimal Register (DR)

The first three registers in this list can operate in conjunction with the program line. As stated earlier, data can flow from the program line to the operation register, search register, or common register, and from either the search register or common register back onto the program line. Data cannot flow from the operation register to the program line. See Figure 3-31.



BLOCK DIAGRAM
PROGRAM LINE DATA FLOW

FIGURE 3-31

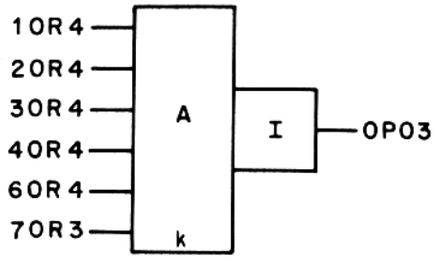
The decimal register is named for its use during input and output instructions, but it is also used as a type of phase counter by the program line logic during the various searches.

One additional register, the data transfer register, might also be considered a control register. However, it is used only during those instructions dealing with the storage delay line, and it will be covered when the storage line logic is discussed.

a. OPERATION REGISTER. The Operation Register (OR) is a six-stage series shift register composed of the 1OR, 2OR, 3OR, 4OR, 6OR, and 7OR flip-flops. Its main function is to receive commands from the program line and to decode these commands into logic signals suitable to initiate the operation.

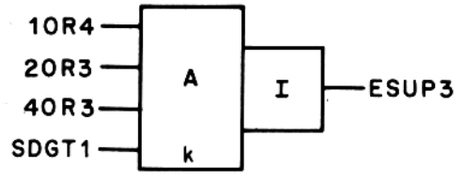
(1) Decoding. The outputs of the OR flip-flops are decoded in a group of gates, called the OR Decoder, or Op Decoder. The outputs of these gates are routed throughout the machine to control the various operations necessary to execute each instruction.

(a) Commands. When a given command is in OR, the OR Decoder will usually produce a signal having the name of the command. For instance, if the Punch Off function is being performed, 7OR will be reset and all other OR flip-flops will be set. The decoder will produce the Operation Punch Off signal, OPO3, as shown in Figure 3-32.



OP DECODER OUTPUT - COMMAND

FIGURE 3-32



OP DECODER OUTPUT - FUNCTION

FIGURE 3-33

(b) Other Signals. In addition to the command signals, other logic signals are developed by the OR Decoder. Each command in OR is decoded as to whether it is a zero, one, or two address command. Other logic signals, such as Enable Shift UP (ESUP3), Enable Shift DowN (ESDN3), etc., are produced by the OR Decoder. An example is shown in Figure 3-33. If 1OR is set and 2OR and 4OR are reset, and if the Significant DiGiT flip-flop is reset, ESUP3 will be true.

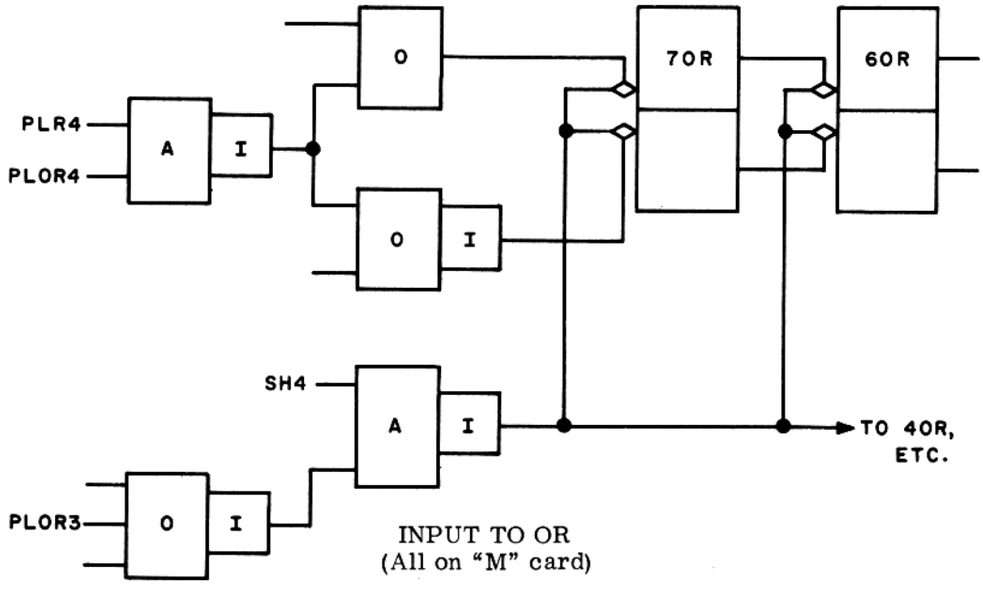
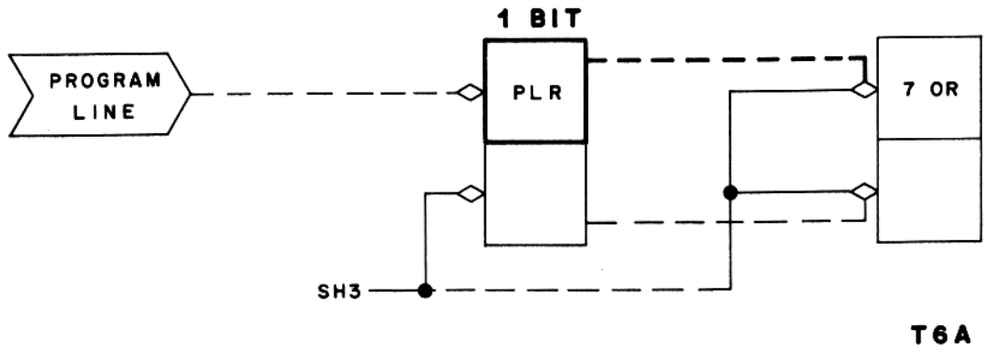


FIGURE 3-34

(2) Basic Operation. Figure 3-34 illustrates the method used to shift a code into the operation register from the program line. (The unmarked inputs are used when shifting a code in from the search register. This occurs during Translate.) Assume Program Line-to-Operation Register (PLOR3) is true. At every fractional bit "SH" time the contents of 7OR will shift into 6OR, the contents of 6OR will shift into 4OR, and so on down. 7OR will assume the state dictated by PLR. The importance of the fact that data is read early in relation to the timing chain now becomes apparent. Assume that the Reader Off code, 1-2-4, is to be transferred from the program line to the operation register. The 1-bit is actually sensed (sets PLR) during T6 time of the previous character. This enables the set gate of 7OR, as shown in Figures 3-34 and 3-35 (a). (The dotted lines in Figure 3-35 represent logical connections made possible by PLOR3, and not wired connections.)



BASIC SHIFTING SEQUENCE

FIGURE 3-35 (a)

At the following "SH" time (T1SH), 7OR becomes set and PLR becomes reset. (See Figure 3-35 (b).

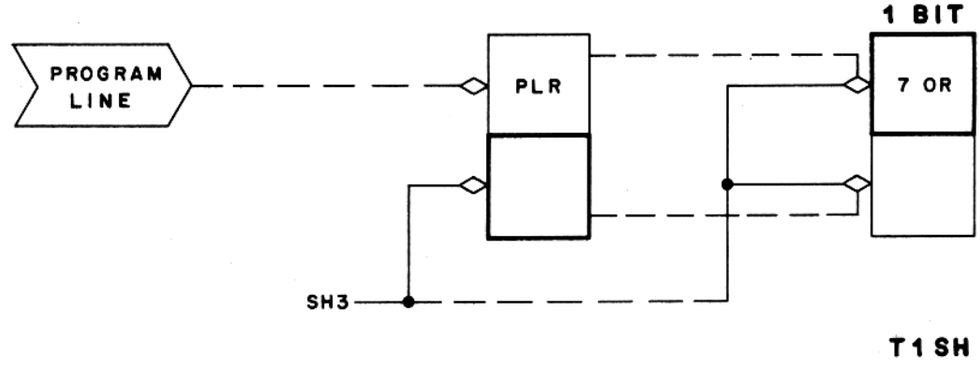


FIGURE 3-35 (b)

During T1 time PLR again becomes set by the 2-bit from the program line, thus enabling the set gate of 7OR again.

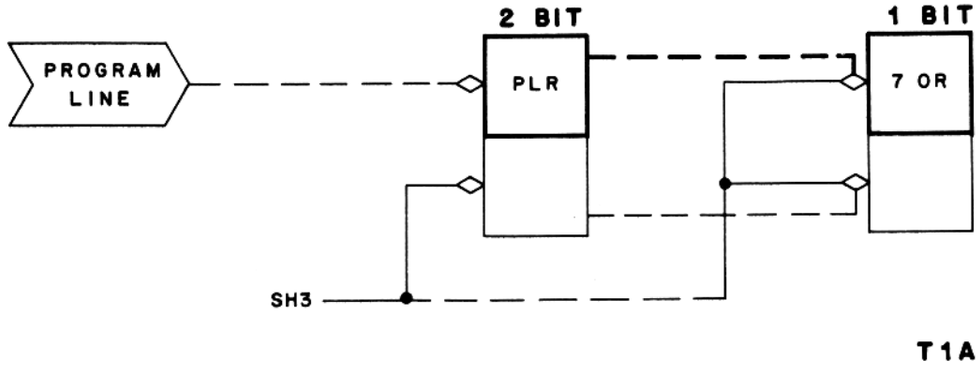


FIGURE 3-35 (c)

At T2SH the set condition of 7OR is shifted into 6OR. 7OR remains set because its set gate was enabled. PLR resets.

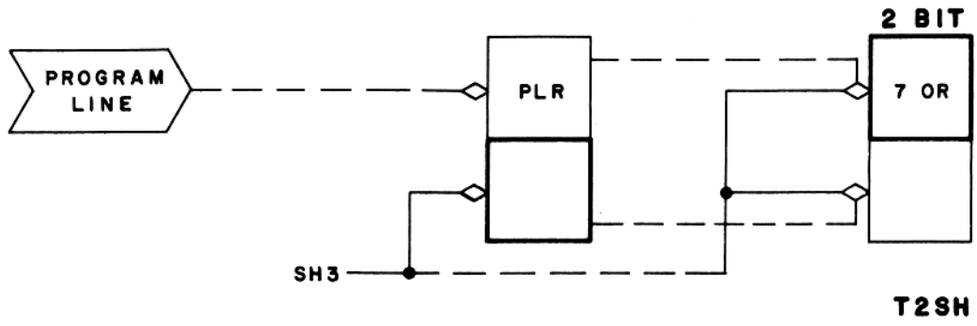


FIGURE 3-35 (d)

During the balance of T2 time no bit is sensed coming off the program line, so PLR remains reset. The set gate of 7OR is disabled and the reset gate is enabled.

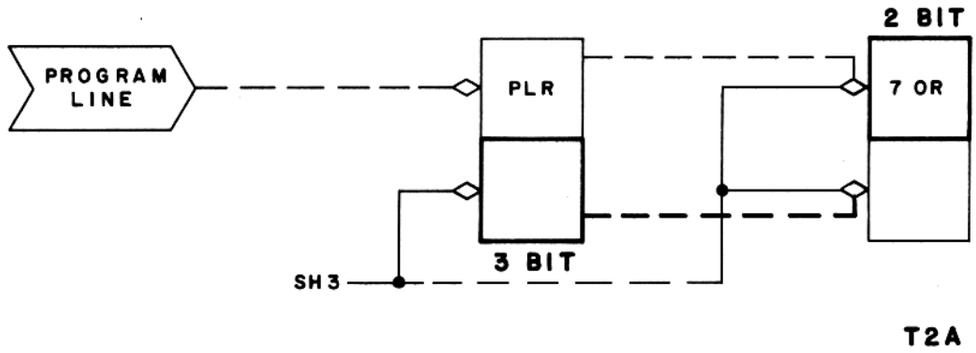


FIGURE 3-35 (e)

At T3SH time 7OR becomes reset, but its previously set condition is shifted into 6OR and the previously set condition of 6OR is shifted into 4OR. PLR remains reset.

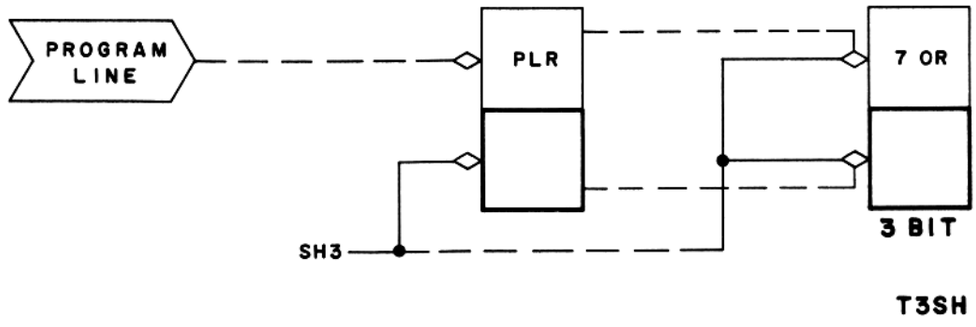


FIGURE 3-35 (f)

During T3 time PLR is again set by the 4-bit coming off the program line, and the set gate of 7OR is again enabled.

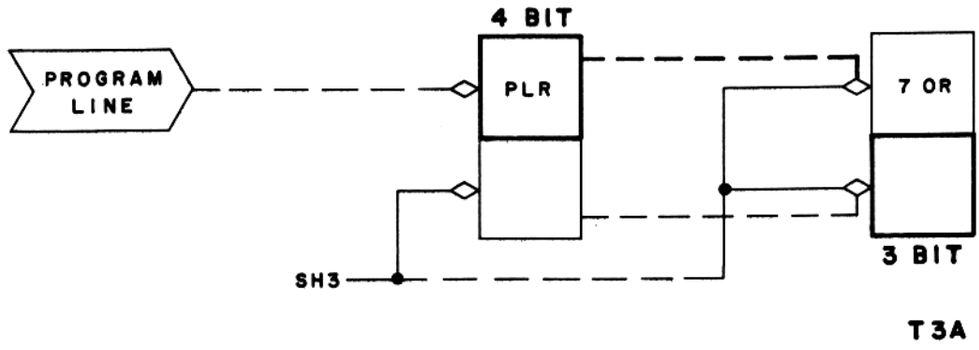


FIGURE 3-35 (g)

At T4SH the set conditions of 6OR and 4OR are shifted into 4OR and 3OR, respectively; the reset condition of 7OR is shifted into 6OR; and 7OR becomes set. PLR again resets.

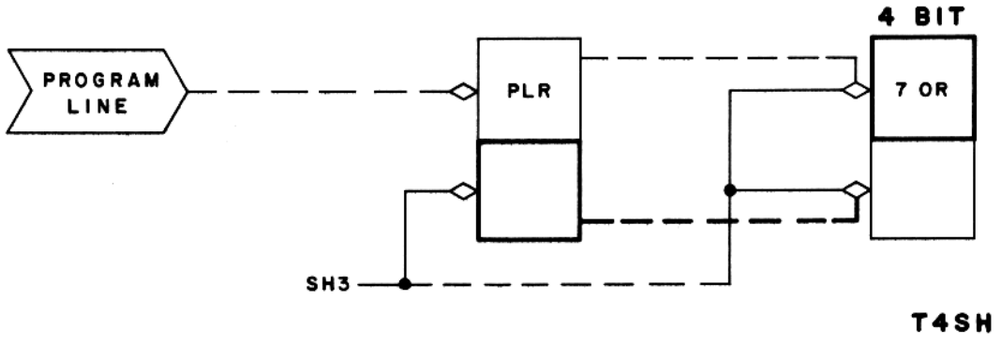


FIGURE 3-35 (h)

During T4 and T5 time no more bits are sensed coming off the program line, so PLR remains reset and 7OR remains with its reset gate enabled. However, three more "SH" signals occur (T4SH, T5SH, and T6SH), which shift the contents of the OR flip-flops each T time.

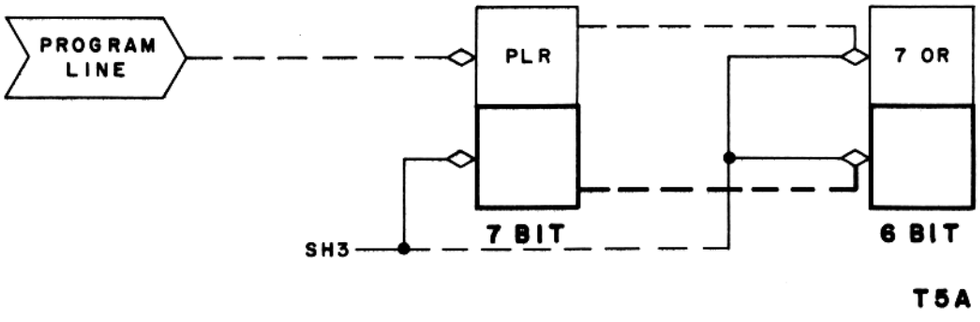
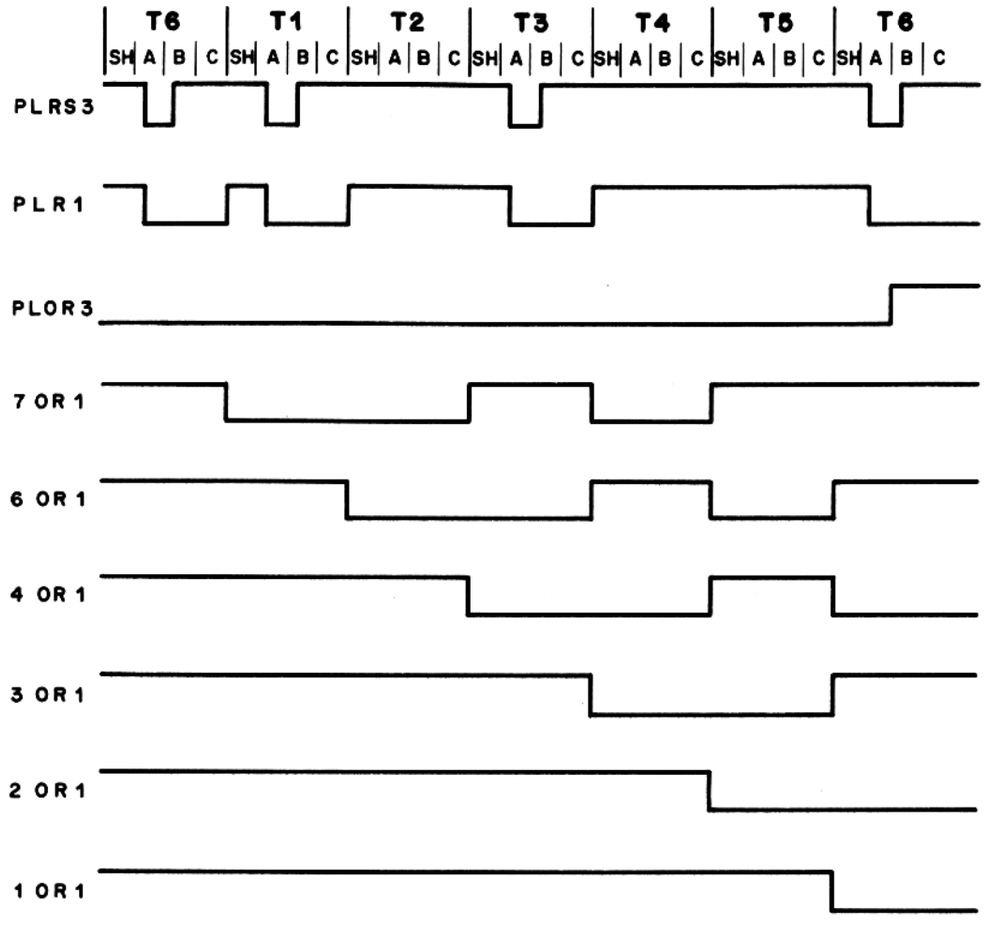


FIGURE 3-35 (i)

After T6SH occurs, the 1OR, 2OR, and 4OR flip-flops are set and 3OR, 6OR, and 7OR are reset.



TIMING CHART
SHIFTING 1-2-4 INTO OR

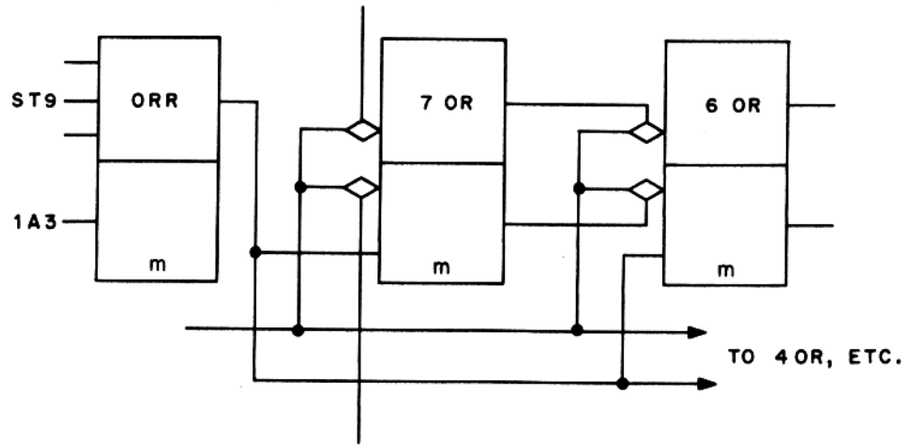
FIGURE 3-36

Figure 3-36 illustrates the representative waveforms. Note that each bit coming off the program line appears in PLR early in respect to the timing chain. By the time the entire character has been shifted into the operation register (at T6SH), 1OR, 2OR, and 4OR are set. As each shift occurs, the data previously in 1OR is shifted out and lost. The last pulse shown on PLRS3 is a 1-bit in the next character.

Note also in Figure 3-36 that PLOR3 becomes false at the T6B time after the ROF code is completely shifted into the operation register. If PLOR3 did not become false at this time, the next code would be shifted into OR and the ROF code would be shifted out and lost. Similarly, any code previously in the operation register is shifted out and lost as the ROF code is shifted in. The way in which PLOR3 is controlled by the phase counter settings will be covered later.

(3) RESET CONDITIONS. When system power is first applied, ST9 sets the Operation Register Reset flip-flop, ORR. This flip-flop resets the six OR flip-flops, assuring that the operation register is in the reset condition when 5610 operation begins. See Figure 3-37.

Figure 3-37 also shows that ORR can be set by two other signals during the course of 5610 operation. After the set signal to ORR is removed, it resets at the next T1A time.



RESETTING OR

FIGURE 3-37

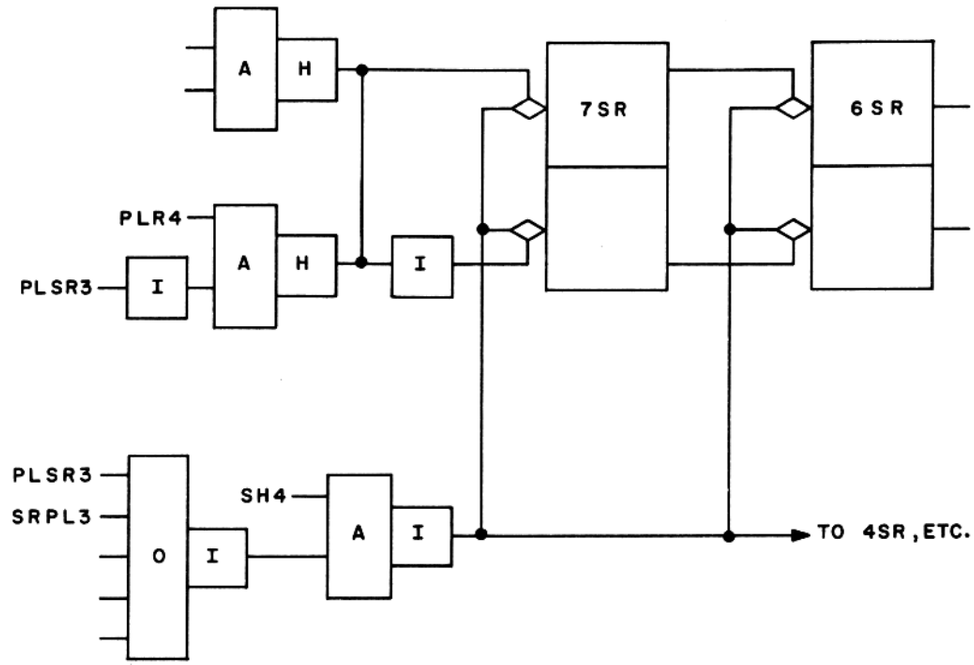
b. SEARCH REGISTER. The search register is very similar to the operation register, being a six-stage series shift register made up of the 1SR, 2SR, 3SR, 4SR, 6SR, and 7SR flip-flops. The main purpose of the search register is to monitor the program line output and let the program line logic know when the marker has been sensed.

(1) Decoding. The search register has very few decoded outputs. The two main ones are the Search Register Tape Feed, SRTF, and the Search Register BRanch marker, SRBR. SRTF is the marker in normal operation and SRBR is the marker during unconditional branch.

(2) Operation. The basic operation of the search register is very similar to that of the operation register. Two major differences are the fact that the search register output can be routed back onto the program line, and the two marker codes can be set into the search register from an external source, without having been sensed by PLR.

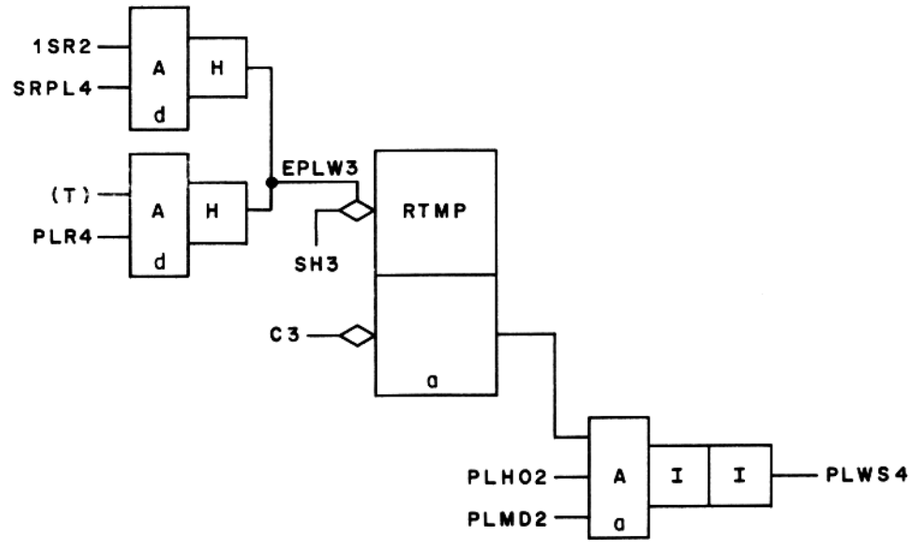
(a) Input. Figure 3-38 illustrates the input to 7SR. The unmarked inputs are used only during translation, to route data from the common register into the search register (CRSR). When the program line logic makes PLSR3 true, the data flowing through PLR is shifted into 7SR and through the other SR flip-flops. This operation is identical to that of the operation register during PLOR3.

(b) Output. Most of the time that PLSR3 is true, SRPL3 is also true. In this way, the output of the search register (1SR) is written onto the program line as shown in Figure 3-39. While SRPL3 is true, EPLW3 will become true whenever 1SR is set. (Note in Figure 3-38 that either PLSR3 or SRPL3 will enable the contents of the search register to shift at "SH" time. Thus, these two operations can occur separately, although they usually occur together.) With EPLW3 true, SH3 sets RTMP and at fractional bit time "A" the pulse is written onto the program line. Figure 3-40 shows the waveforms involved in shifting the Reader Off code (1-2-4) from the search register to the program line.



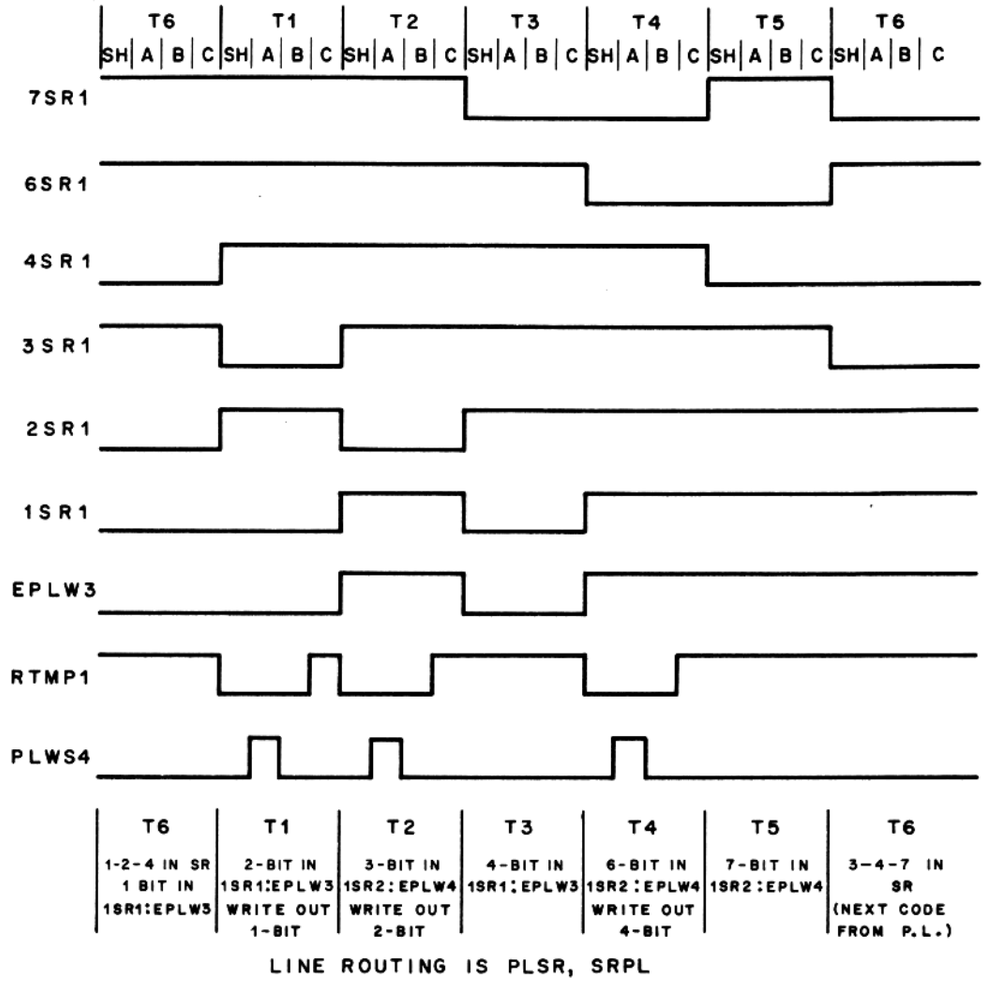
INPUT TO SR
(All on "M" card)

FIGURE 3-38



EFFECT OF SRPL ON PROGRAM LINE WRITE

FIGURE 3-39



TIMING CHART
SHIFTING 1-2-4 FROM SR TO PL

FIGURE 3-40

(c) Setting Markers Into The Search Register. When a marker is to be written onto the program line initially, it is first set into the search register. Then at the proper time, SRPL3 goes true and the marker is written out. If the normal marker is desired, the Normal MarKer Signal, NMKS3, is developed. This signal immediately sets each of the six SR flip-flops through a level set input. If the unconditional branch condition is true, the Unconditional MarKer Signal, UMKS3, is developed instead. UMKS3 sets 2SR, 4SR, and 7SR, and resets 1SR, 3SR, and 6SR, thus setting the branch marker (2-4-7) into the search register.

(3) Reset Conditions. ST9 sets the Search Register Reset flip-flop (SRR), which in turn resets all SR flip-flops. This ensures a reset condition of the search register upon machine turn-on. SRR, like ORR, is reset by 1A3. There are six other signals that can set SRR. These will be covered later.

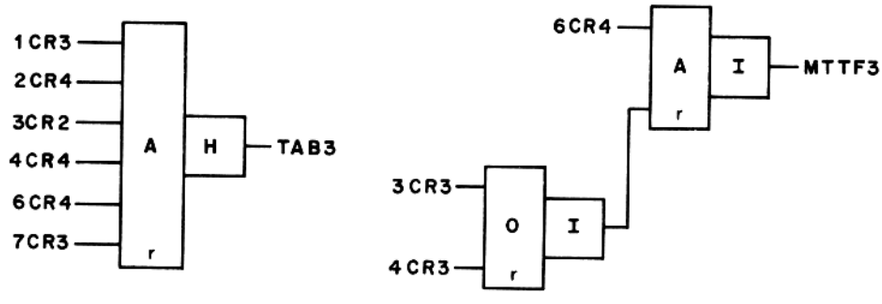
c. COMMON REGISTER. The common register is also a six flip-flop series shift register, but it has two additional flip-flops to correspond to the 2205 5-bit and 8-bit. All CR flip-flops can be set or reset by external signals.

The name of common register is very appropriate, for this register is truly common to all parts of the 5610. All data loaded onto the program line from the 2205 passes through the common register. All data entered into the processor or printed out by the 2205 passes through it. All data that enters or leaves the arithmetic and storage delay lines passes through the common register.

Certain other functions are also performed by the common register. It checks all codes entered for parity. When a character is to be printed, the 5-bit is added, if necessary, in the common register.

All of these functions will not be covered in detail in this section. Instead, the basic operation of the common register and its relation to the different line routings will be explained.

(1) Decoding. The common register has more decoded outputs than the search register, but fewer than the operation register. Figure 3-41 (a) illustrates a decoded output which is used by the program line logic. Figure 3-41 (b) shows another output used only by the arithmetic logic (MTTF represents Math Ten To Fifteen).



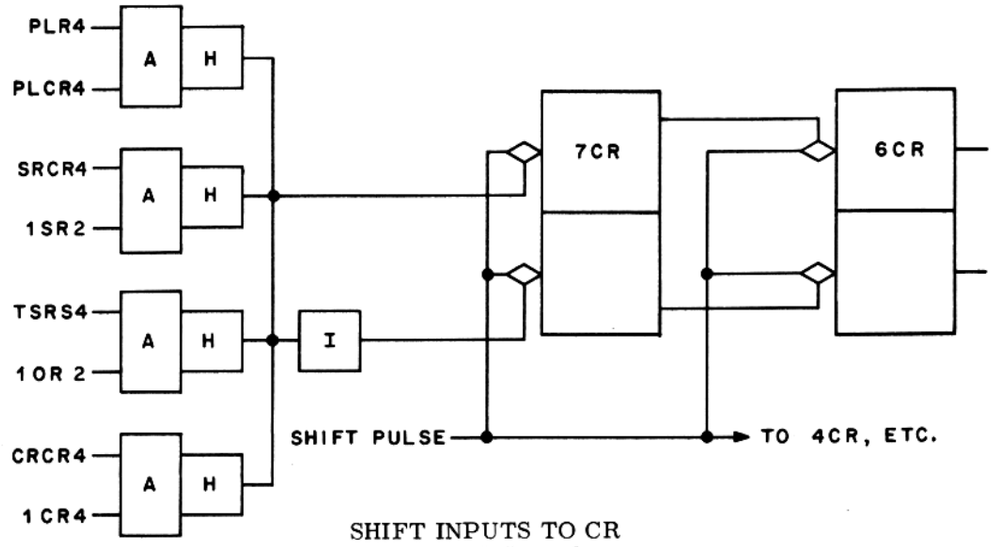
(a) (b)

CR DECODER OUTPUTS

FIGURE 3-41

(2) Basic Operation. There are basically three different ways in which a code can be set into the common register. It can be set into 7CR and shifted across, the same as is done with the operation and search registers. It can also be set into 6CR and shifted across. This method is used when binary coded decimal characters (using only the first four bits) are shifted from the arithmetic delay line into the common register. Characters can also be set into the common register in parallel from the 2205. In this case, all CR flip-flops are set into the desired code at the same time, and no shifting occurs.

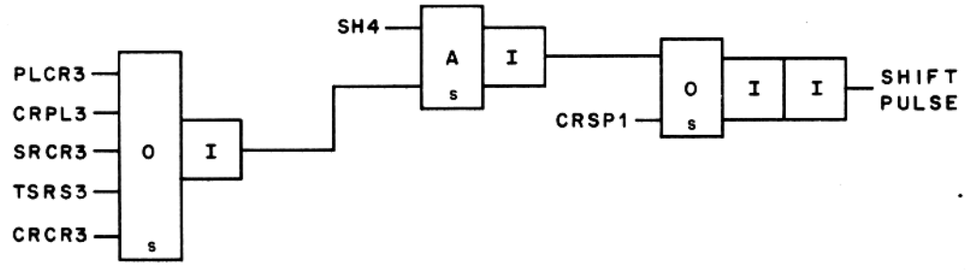
(a) Line Routing Inputs. There are four different line routings (see Figure 3-42) which can cause a code to be shifted into the common register via 7CR. Program Line-to-Common Register (PLCR) and Search Register-to-Common Register (SRCR) need no further explanation at this point. TSRS stands for TranSlate Ring Shift, and shifts the contents of the operation register into the common register. CRCR, otherwise called "ring shift", is used during parity check and output instructions. In each of these cases, the code is shifted into the six CR flip-flops just as it is into the operation and search registers. 5CR and 8CR do not enter the picture yet.



SHIFT INPUTS TO CR
(All on "S" card)

FIGURE 3-42

Figure 3-43 illustrates the development of the pulse which shifts the contents of the CR flip-flops. Note that either PLCR3 or CRPL3 can enable the shift pulse. Both of these signals must be on the gate so that one of these functions can be performed without the other. In other words, these two line routings can occur independently of one another.



DEVELOPMENT OF THE CR SHIFT PULSE

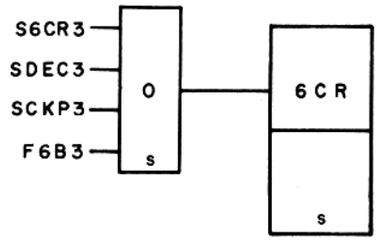
FIGURE 3-43

All five of these line routings are ANDed with fractional bit "SH" time to have the actual shift occur in sync with the program line timing chain.

When the input to the common register is from the arithmetic delay line, the Common Register Shift Pulse, CRSP3, is developed in sync with the arithmetic timing chain to provide the correct shift pulse timing.

(b) Arithmetic and Storage Logic Inputs. When the input to the common register is from the arithmetic or storage delay line, all of the line routing signals pertaining to the common register will be false, so whenever a shift pulse occurs, 7CR will become reset and enable the reset gate of 6CR. If a bit is to be shifted into the common register, the Set 6CR signal, S6CR3 can only be developed BETWEEN shift pulses, so that the following shift pulse transfers the condition of 6CR into 4CR. In this case, 6CR holds the code to be shifted in, the same way that 1OR, PLR, and 1SR hold the code during TSRS, PLCR, and SRCR, respectively. When the code is completely shifted in, it is held in 1CR, 2CR, 3CR, and 4CR. 6CR and 7CR are reset.

(c) 2205 and Similar Inputs. All level set inputs to the CR flip-flops (except 5CR and 8CR) are applied through OR gates. The number of inputs to each OR gate varies. The inputs shown in the example in Figure 3-44 are listed here to give a basic idea of the reason for each.



CR LEVEL SET INPUTS

FIGURE 3-44

(1) S6CR3. As explained earlier, this signal is used to set 6CR when shifting a number into the common register from the arithmetic or storage line.

(2) SDEC3. This signal sets the 1-2-4-6-7 CR flip-flops when the decimal is to be printed.

(3) SCKP3. This input may or may not be present. It is wired per customer requirements to set the desired check protection character into the common register before it is printed.

(4) F6B3. This is the 6-bit signal from the 2205. Whenever a character is entered from the 2205, the code is first set into the common register. It may then go from the common register to the program line (Load) or to the arithmetic line (Entry). 5CR and 8CR are also set by the F5B3 and F8B3 signals, but in these two cases the input is direct, and not through an OR gate.

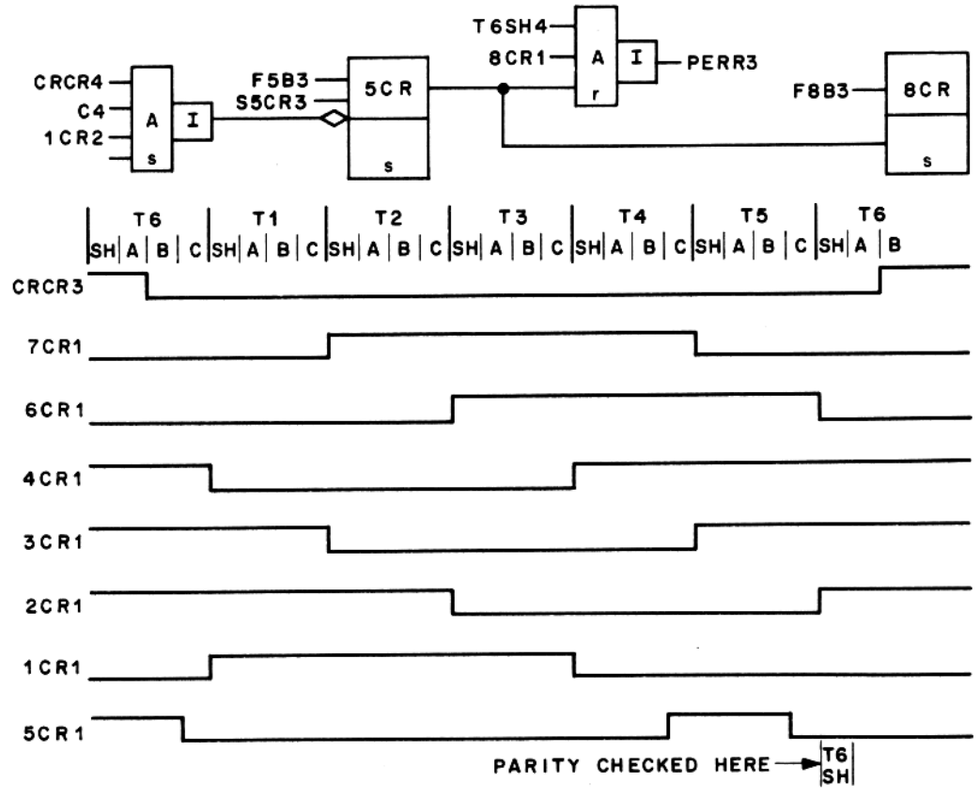
(3) Special Operations. Three additional operations occur in the common register. They have to do with carriage return output, parity check on the 2205 input, and adding the 5-bit during output to the writing machine.

(a) Carriage Return. When any 2205 function is to be performed, the program line code for that function is automatically shifted into the operation and search registers, and the common register output is then routed to the 2205. In the case of the carriage return, the program line code, 2-4 must first be converted to a lone 8-bit. To do this, the operation register develops the Op Carriage Return signal, OCR3, which resets 2CR and 4CR and sets 8CR through level inputs (not through OR gates). The common register output (8-bit) is then fed to the writing machine to perform the carriage return.

(b) Parity Check. All codes entered via the 2205 are checked to make sure they are odd-parity codes. If an even-parity code is detected, the Parity ERRor signal, PERR3, causes the program line logic to alter its course and stop machine operation. An exception to this is found with the 8-bit. Any code (except carriage return) containing an 8-bit is checked for odd-parity minus the 8-bit. In other words, the 8-bit is ignored and the other bits are checked. Related logic prevents a parity error when an 8-bit alone is entered.

The basic process in parity check is the ring shift (CRCR3). During ring shift, the contents of 1CR are shifted into 7CR as the contents of the other CR flip-flops are shifted through the common register. The net result is that the contents of the six shifting CR flip-flops are back to their original configuration after six shifts occur. However, as the shifts take place, 5CR is toggled whenever 1CR is set: each time a bit is shifted into 1CR, 5CR will be toggled. If no bit is sensed by 1CR, 5CR will not be affected. If 5CR is reset at the end of six shifts, it denotes a parity error.

Take the case of the letter "A" entered via the writing machine. The 2205 sets 1CR, 6CR, and 7CR. CRCR3 becomes true at T6B time, and at T6C time 5CR becomes set (see Figure 3-45). At T1SH the contents of the common register shift, and 1CR resets. 5CR stays set. After two more shifts nothing has changed. However, at T4SH the 6-bit is shifted into 1CR, and at T4C time, 5CR is again toggled and becomes reset. At T5SH the 7 bit appears in 1CR, and at T5C time 5CR is again toggled and sets. At T6SH time the common register again contains the "A" code and at T6B time CRCR3 goes false to prevent any further shifting. 5CR is now set, but does not affect the entry: if the "A" is to be written onto the program line next (CRPL), it is shifted out of the six shifting CR flip-flops. No data shifts out of 5CR.



PARITY CHECK

FIGURE 3-45

In the case of a 5-bit entered via the writing machine, such as the digit "3" (1-2-5), 5CR starts out set by the 2205, is toggled an even number of times during the ring shift, and ends up in the set condition. If an even parity code is entered, such as a 1-5 code, 5CR starts out set and is toggled an odd number of times, so it ends up reset, allowing PERR3 to become true.

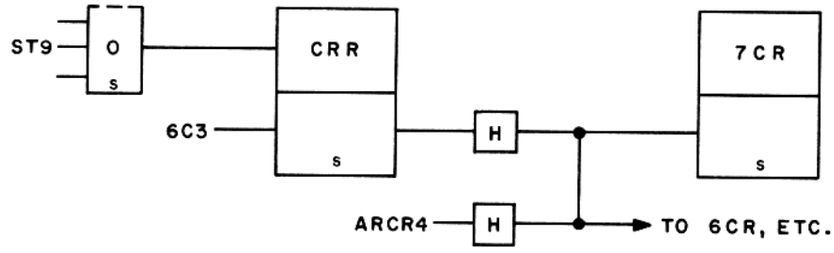
The last item to consider under parity check is that of the 8-bit, which, when entered, sets 8CR. Whenever 5CR sets, 5CR1 applies a true signal to the level reset input of 8CR (refer to Figure 3-45). Therefore, if an 8-bit is entered along with any other bits (to prevent printing), 8CR becomes reset during the ring shift. With no other signals available to set 8CR, it stays reset throughout the ring shift. In this way the 8-bit is ignored whenever it is accompanied by any other bit.

When a lone 8-bit is entered, there are no other bits available to set 5CR during the ring shift, so 8CR remains set. In this case, 5CR is still reset at the end of the ring shift, but PERR3 is held false by virtue of 8CR1 being true (refer again to Figure 3-45).

(c) Adding 5-bit On Output. Since the 5-bit is not recorded on the program line, any code which normally contains the 5-bit must have it added before the code is sent to the writing machine. Actually, the common register adds the 5-bit to all even-parity codes being sent to the 2205. If the code is odd without the 5-bit, the 5 is not added.

The mechanism employed to add the 5-bit is, again, the ring shift. After PLCR shifts the code into the common register, S5CR3 is developed, which immediately sets 5CR. Next, the ring shift takes place, and toggles 5CR once for each bit. If the code from the program line contains an even number of bits, 5CR remains set at the end of the ring shift; if the code has an odd number of bits, 5CR is reset. In this way, all codes sent to the 2205 contain an odd number of bits. A special logic circuit prevents S5CR3 from becoming true if the output is a carriage return (OCR3).

(4) Reset Conditions. During regular 5610 operations, 8 signals, including ST9, can set the Common Register Reset flip-flop, CRR, which immediately resets all eight CR flip-flops through level reset gates. Since the arithmetic logic also operates in conjunction with the common register, the Arithmetic Reset Common Register signal, ARCR, is developed during some instructions dealing with the arithmetic delay line. CRR is reset at T6C time, as shown in Figure 3-46.



RESET CR

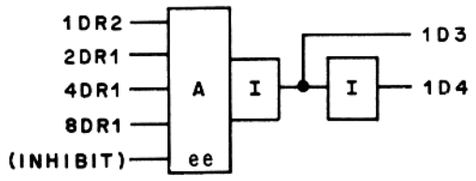
FIGURE 3-46

d. DECIMAL REGISTER. The Decimal Register (DR) is different from the other control registers in that it is a binary counter, rather than a series shift register. Its purpose is also somewhat different: whereas the other control registers operate primarily with the program line, the decimal register cannot accept codes directly from the program line, and its data handling functions are rather limited (it is used mainly as a counter).

The decimal register is made up of the 1DR, 2DR, 4DR, and 8DR flip-flops. It is named for its use during entry and printout, when it counts digits. However, it has many other functions: it is used by the program line logic to dictate exactly what type of search is to be conducted in 3 mode. It is used by the storage line logic as part of a compare circuit to locate the proper storage location. The arithmetic line logic uses it during most functions.

(1) Decoding. Many of the DR flip-flop outputs feed special circuits to produce signals such as Decimal CoMPare (DCMP3), used in entry and printout, and Storage Location Correct (SLC3), used in Store, Recall, and Accumulate. These circuits are quite involved and are covered in Section 8, Electronic Circuits.

The decoded outputs used by the program line logic are 1D3, 2D3, 3D3, etc., and are produced as shown in Figure 3-47. These signals are referred to as the "decimal settings". The purposes of these signals will be covered later when the program line logic is studied in greater detail. Although the decimal register has sixteen states, numbered 0 through 15, only the first seven states, 0D3 through 6D3, are decoded for use by the program line logic. The truth table in Figure 3-47 indicates that the decimal register decoding is binary.



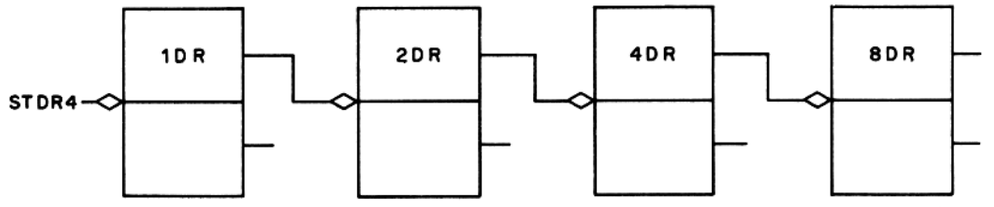
	1DR	2DR	4DR	8DR
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

DR DECODING

FIGURE 3-47

(2) Basic Operation, The decimal register is a "down counter". That is, from 0 it first steps to 15, then to 14, 13, 12, etc. Like the phase counter, it can also jump over some configurations to assume a decimal setting entirely different from the previous one. In addition, the decimal register can be set into specific states corresponding to the addresses of commands such as Store, Truncate, Enter, etc.

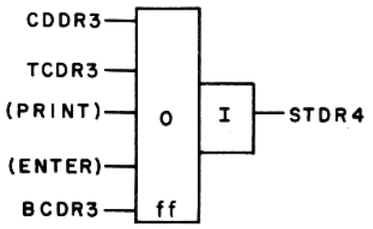
(a) Counting Operation. As a starting point, assume that all DR flip-flops are reset. Referring to the truth table in Figure 3-47, note that this is the 0 decimal setting (0D3 is true). Now see Figure 3-48. If STep DR (STDR4) goes true, 1DR is toggled and it sets. This causes 2DR to become set, which sets 4DR, etc. After one pulse, all DR flip-flops are set, yielding the 15 decimal setting. A second STDR4 resets 1DR, yielding 14 decimal. Each pulse counts the decimal register down by one setting, so after 16 pulses it is back to the 0D3 condition. Note that the counting pulse, STDR4, is a normally true signal, and the actual count occurs when STDR4 becomes true after first going false.



COUNTING DOWN DR
(All on "FF" card)

FIGURE 3-48

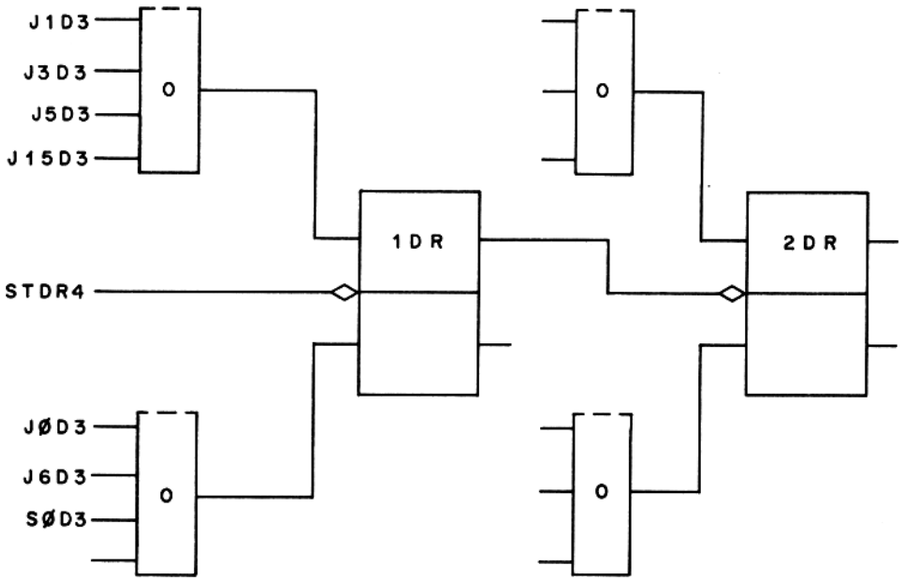
The various signals which produce STDR4 are illustrated in Figure 3-49. The signal most often used to step the decimal register is Count Down Decimal Register, CDDR3. TCDR3 is used during Truncate. The next two signals are produced at the proper times during the two print instructions and during Enter to keep track of the number of digits. Backspace Count Decimal Register (BCDR3) can also be referred to as "Back Count DR". 15 pulses are produced when the backspace key is operated during Entry. This steps the decimal register fifteen times, giving the net effect of counting it UP one time. Note again that the decimal register actually changes its setting when one of the five inputs to the OR gate in Figure 3-49 returns to false.



DEVELOPMENT OF STDR4

FIGURE 3-49

(b) Jumping Operation. Each signal causing a decimal jump is applied to a combination of OR gates in order to set or reset the proper DR flip-flops immediately. Each flip-flop has an OR gate on both the set and reset side. An example is shown in Figure 3-50.



TYPICAL DR INPUTS
(All on "FF" card)

FIGURE 3-50

All of the signals used to set or reset 1DR shown in Figure 3-50 are functions of the program line logic. Jumps can also be initiated by the arithmetic line logic, and a jump also occurs when the contents of the common register are set into the decimal register for Truncate. These additional jumps will be covered later.

(3) Reset Conditions. ST9 sets 8DR upon machine turn on. This assures that the decoded decimal outputs, 0D3 through 6D3, will all be false when machine operation begins. The states of 1DR, 2DR, and 4DR are of no consequence at this time.

4. RESET SWITCH

Operation of the Reset switch results in destruction of the marker, thus terminating the sequence. In order to simplify the gating and still eliminate any possibility of error due to development of conflicting signals, two flip-flops are used in conjunction with the Reset switch to allow the Reset function to occur only at specific times.

Figure 3-51 shows that the XRST flip-flop is set whenever the Reset switch is operated. It remains set until RST becomes set. This happens as a result of one of the following four conditions:

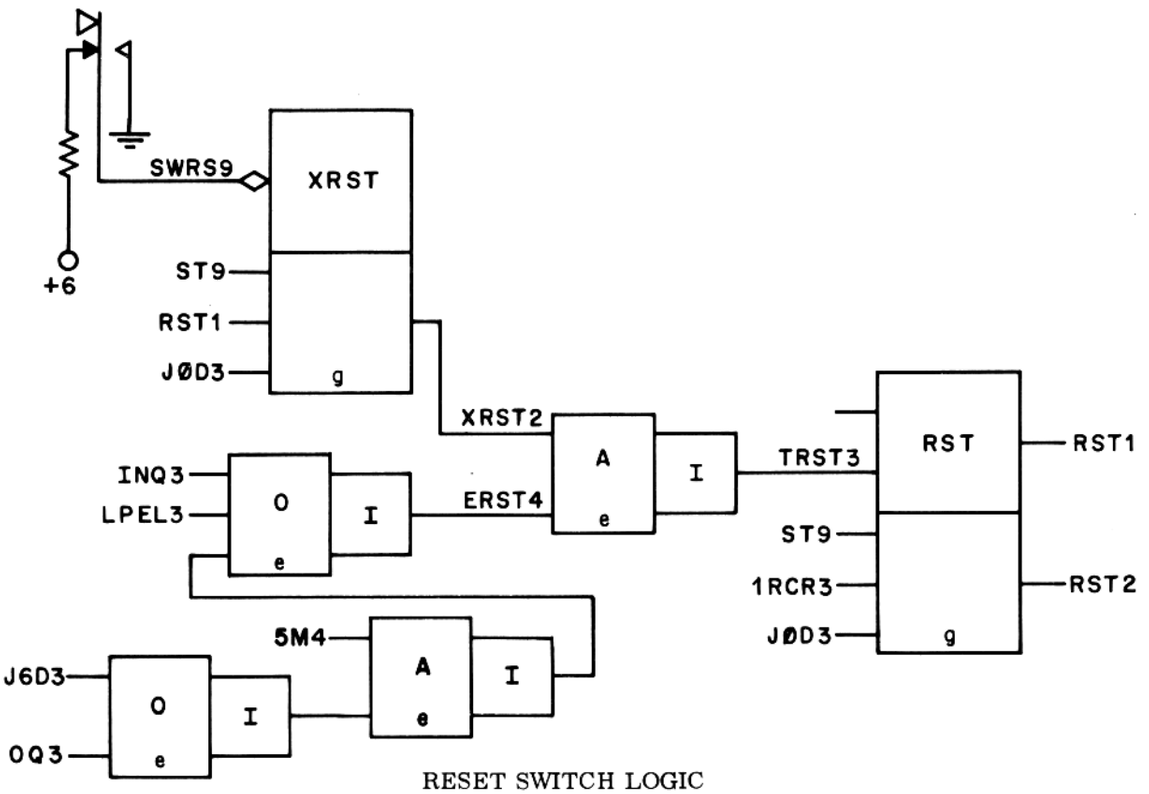
(1) A Halt command is retrieved and the logic attains 5 mode.

(2) An arithmetic or printout instruction is completed.

(3) An input to the 5610 is enabled (Enter instruction, Load, or entry of new sequence).

(4) An error condition occurs.

When one of these four conditions becomes true, ERST4 goes false and drives Trigger ReSeT, TRST3, true. This sets RST, resetting XRST and initiating a routine (to be covered later in this section) which searches out the marker and destroys it, thus leaving the system out of program. Another sequence title must then be entered in order to resume operation.



RESET SWITCH LOGIC

FIGURE 3-51

Still referring to Figure 3-51, note that when a Halt is retrieved, OQ3 becomes true, and when the logic reaches 5 mode, ERST4 is driven false. Whenever an arithmetic or printout instruction is completed, the logic jumps from 5 mode to 3M6,4 to begin a new marker search: the J6D3 occurring at the end of 5 mode also provides ERST4. Whenever an input is enabled, INput Qualification, INQ3, goes true. An error condition develops Light Parity Error Lamp, LPEL3. Either of these two signals develops ERST4 directly.

Note that there are two ways for this reset operation to occur. In the case of an error (parity error, incorrect backspace entry during an Enter instruction, etc.), the error will occur first, driving ERST4 false. The logic then remains in this state, waiting for the Reset switch to be depressed, after which the marker is destroyed. On the other hand, the Reset switch can be operated first, setting XRST. In this case, the logic performs normally until the next Halt, input, or error occurs, or until an arithmetic or output instruction is completed.

F. 1 MODE, PROGRAM LOAD

The 5610 cannot operate without a program first being loaded onto the program line. For this reason, the processor is automatically set into the Load mode (1 mode) when power is first applied.

Certain electronic circuits, mainly in the 2205 interface, that are necessary to the load operation, are discussed separately in Section 8, Electronic Circuits. Only the basic operation and timing of the associated signals are discussed here.

Subsequent load operations, which can occur after the system has been operating, are covered later in this section.

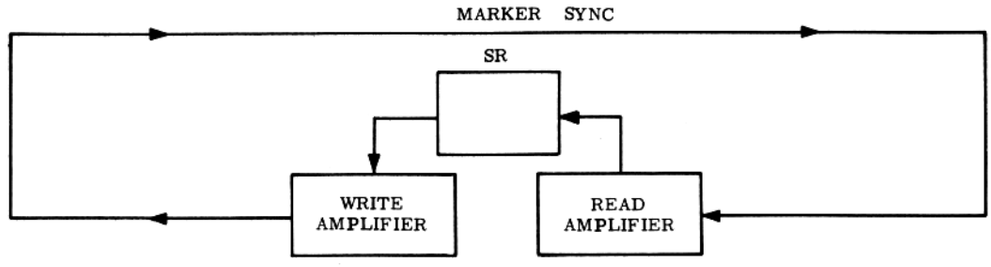
1. GENERAL OPERATION

Certain machine functions occur automatically as the result of the initial application of power to the 5610. Among these are the launching of the program line sync pulse (already covered), the writing out of the marker onto the program line, and the development of the signals necessary to allow the entry of characters from the writing machine. Once the initial operations of launching the sync pulse and marker are completed, program characters may be loaded onto the delay line from the 2205. This latter function is repetitive in nature, the same operations being performed once for each character entered.

The sync pulse and marker are launched during the first six phases. Phases 7 through 11 are used for loading the characters, and phases 12 and 13 are used when a parity error occurs.

a. PRELIMINARY PHASES. The early portion of the program line can be thought of as being made up of four characters: the first character is the sync pulse, the second and third characters are "blanks", and the fourth character is the marker. The blanks will be used later during conditional branch, and the marker will be replaced by the first program character from the 2205.

b. LOADING THE LINE. After the sync pulse and the marker are launched, they simply recirculate on the program line, waiting for the first character to be loaded. This is shown in Figure 3-52 (a). Each pulse is sensed by the read amplifier, sent to the search register, and then written back onto the program line. The line routing is PLSR, SRPL.



BLOCK DIAGRAM
LOADING FIRST CODE

FIGURE 3-52 (a)

As the first character is entered on the 2205, it is stored in the common register. Very shortly after the entry, a parity check occurs (CRCR), as shown in Figure 3-52 (b). During the parity check, the line routing remains PLSR, SRPL.

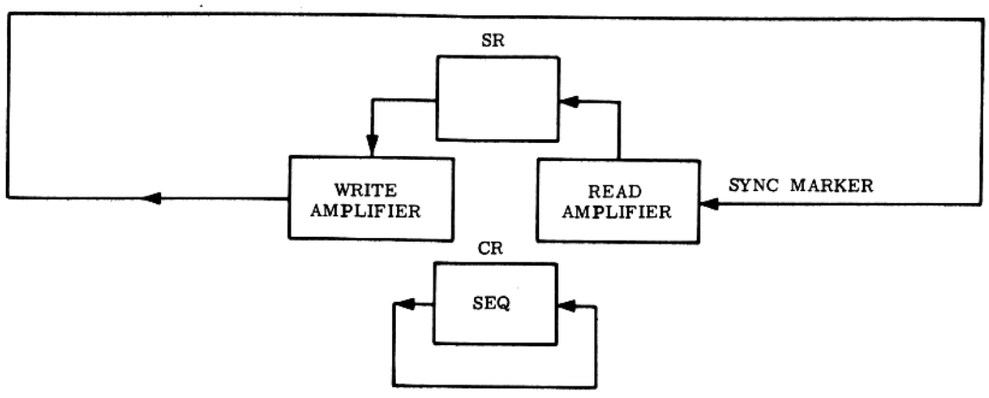


FIGURE 3-52 (b)

This first code waits in the common register until the marker is sensed by the read amplifier and shifted into the search register. Then the line routing is quickly changed to CRPL (only) and the character in the common register is loaded onto the program line in place of the marker. See Figure 3-52 (c).

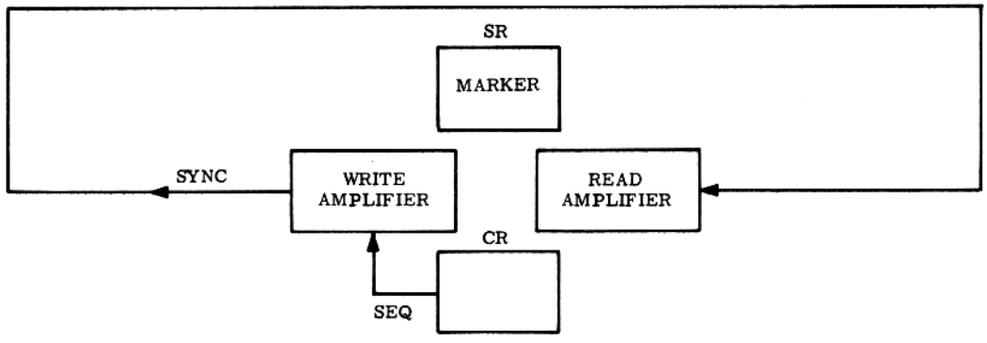


FIGURE 3-52 (c)

As soon as this first program character is completely written onto the program line, the line routing returns to its former state of PLSR, SRPL, and the marker is written out, as shown in Figure 3-52 (d).

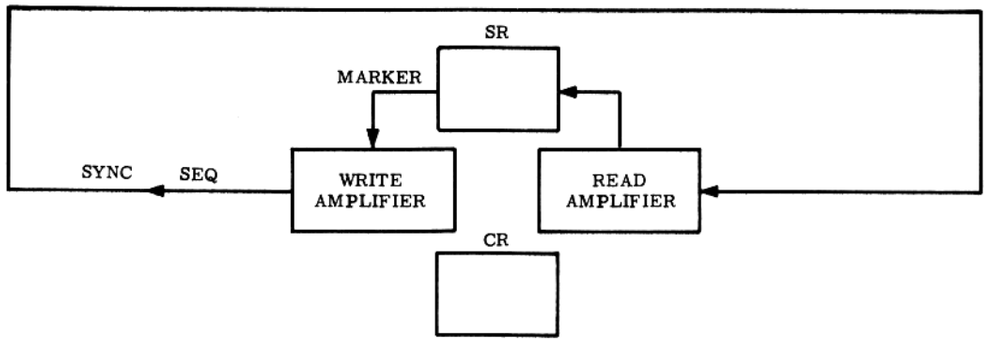


FIGURE 3-52 (d)

Note in Figure 3-52 (e) that the marker now FOLLOWS the newly-loaded program character.

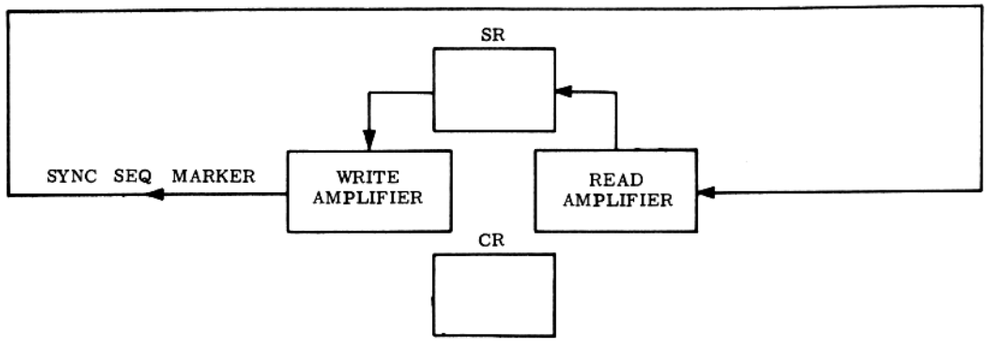
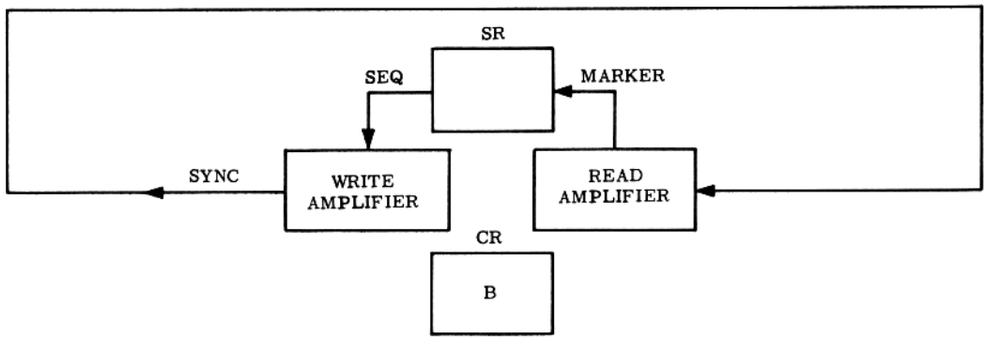


FIGURE 3-52 (e)

When the next character is entered on the 2205, it again waits (after the parity check) in the common register until the marker is detected in the search register (Figure 3-53 (a)).



BLOCK DIAGRAM
LOADING SECOND CODE

FIGURE 3-53 (a)

Then the line routing changes to CRPL and the second program character is loaded onto the program line, immediately behind the first character (Figure 3-53 (b)).

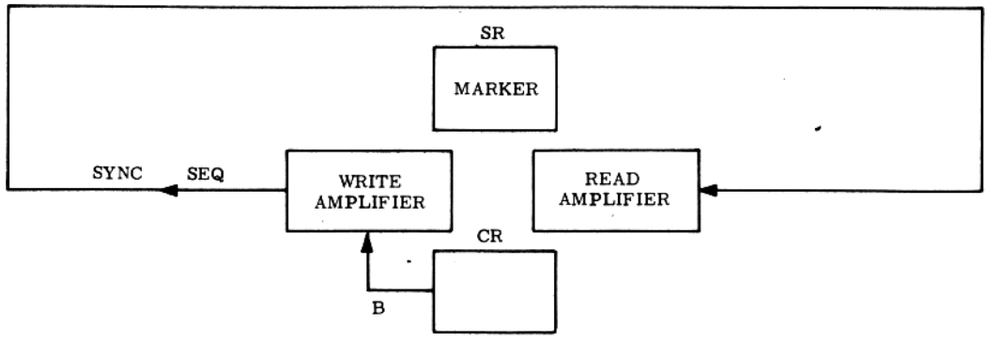


FIGURE 3-53 (b)

When this operation is complete, the line routing returns to PLSR, SRPL, and the marker is written out, again AFTER the character just loaded (Figure 3-53 (c)).

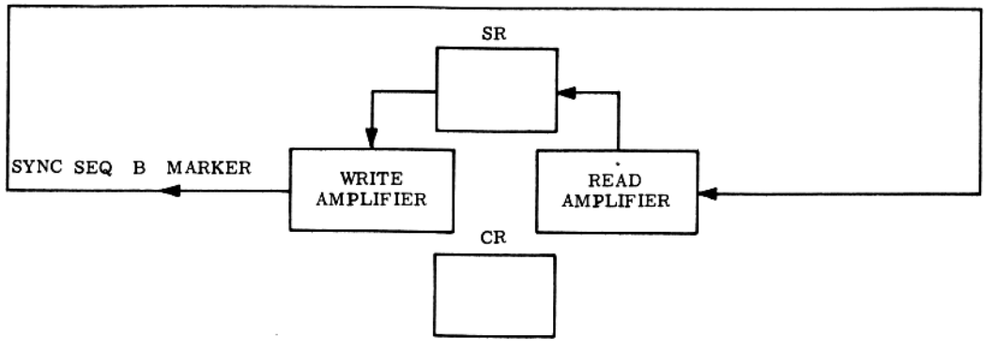


FIGURE 3-53 (c)

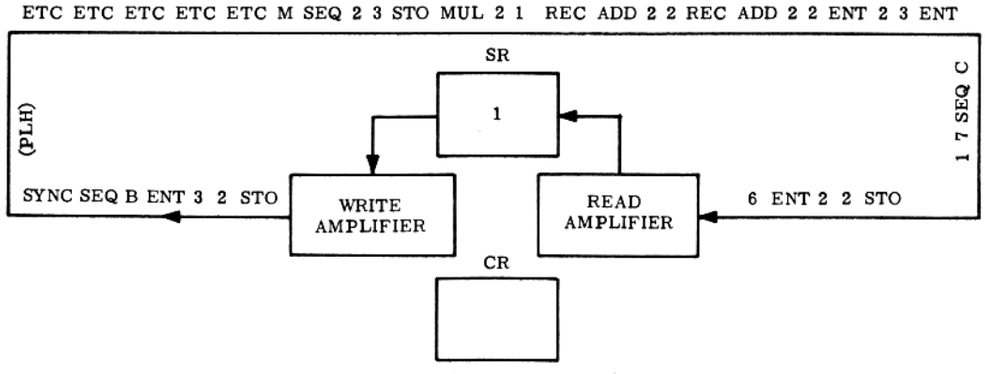
Note that even though the marker may appear in the search register many times (once every 12 milliseconds) between characters entered on the 2205, it is only recognized ONCE after each character; the act of entering a character is what actually triggers the search for the marker.

The same action occurs as subsequent characters are entered on the 2205: each character is written onto the program line immediately behind the preceding character, and the marker is moved to a position behind the character just loaded.

Should a parity error occur, the 5610 stops until the operator takes corrective action.

Each code entered increments a counter in the arithmetic logic, and this counter stops the entry of further characters once a total of 1119 has been reached.

When the entire program has been loaded (Figure 3-54), and the operator depresses the Load switch (or the Reset switch) to take the system out of the Load mode, the marker is searched out and destroyed. This last operation will not be covered at this time, since it will be much easier to understand after normal marker search has been covered.



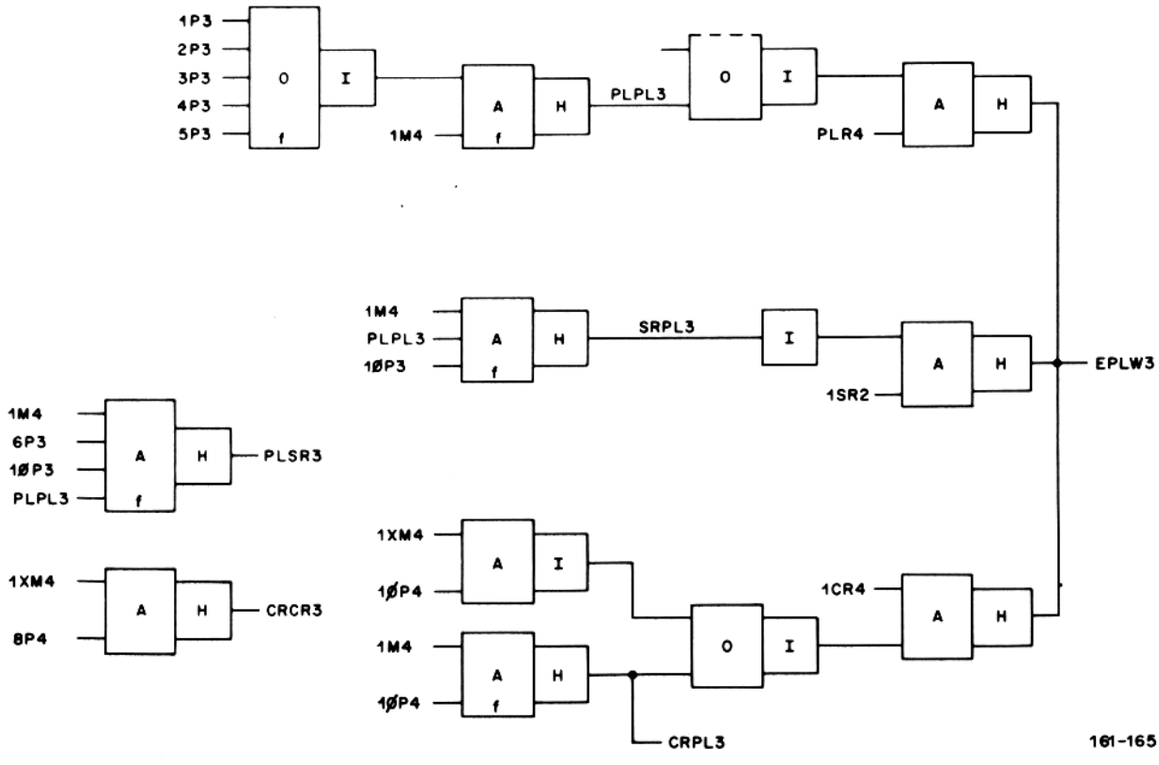
BLOCK DIAGRAM
ENTIRE PROGRAM LOADED

FIGURE 3-54

2. LINE ROUTINGS

The easiest way to cover all of the various events taking place during program load is to study them phase-by-phase: take each phase and cover all of the action that occurs during that phase. However, in order to simplify this discussion of the various phases, the line routings and the development of ECTR3 in each phase are discussed separately.

The logic used to develop the various line routings is pictured in Figure 3-55. The PLPL line routing in phases 1 through 5 is needed to launch and recirculate the sync pulse. In 6 phase, SRPL allows the marker to be written out of the search register onto the program line. The PLSR, SRPL routing used in all subsequent phases except 10 enables the search for the marker and allows for "opening up" the program line when a new program code is inserted between the previous code entered and the marker. CRCR during 8 phase allows for the ring shift (parity check), and CRPL in 10 phase enables the new code to be written out onto the line after the marker has been located.

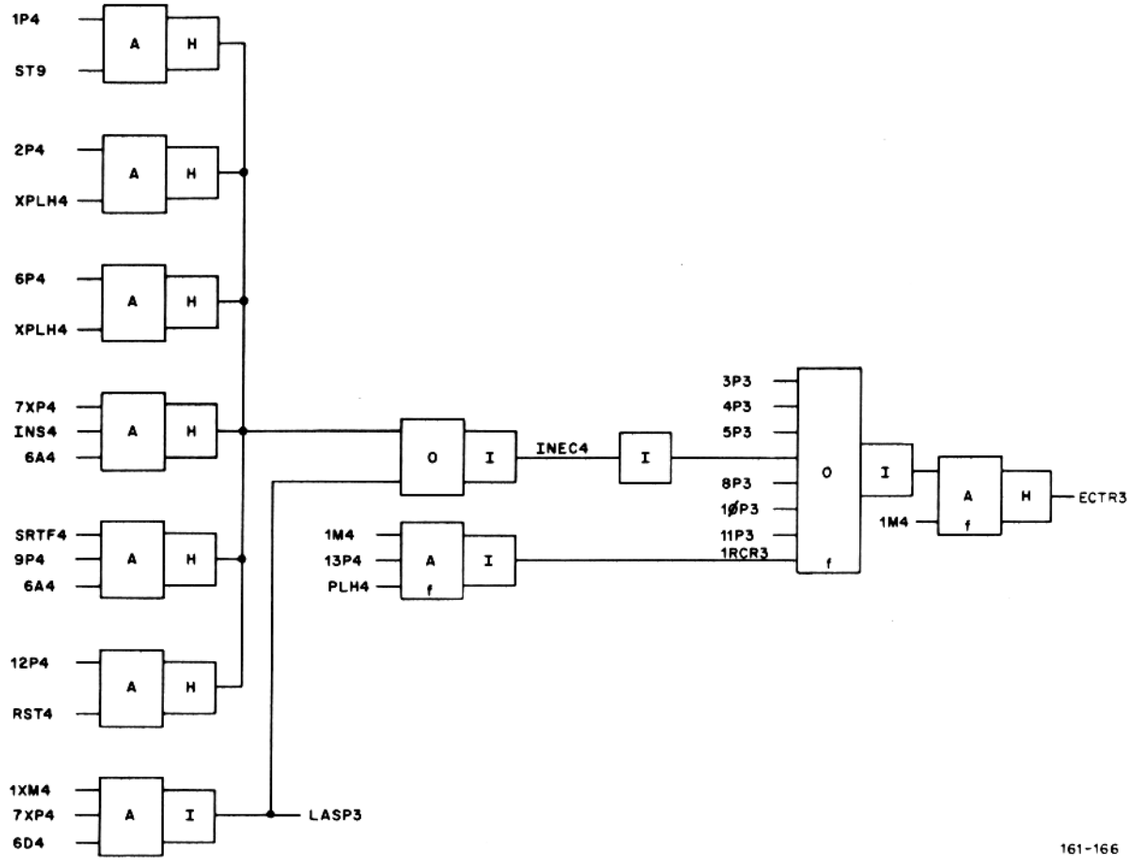


LINE ROUTINGS IN PROGRAM LOAD
(All on "D" card unless otherwise noted)

FIGURE 3-55

3. ENABLING THE PHASE COUNTER

As explained previously, the phase counter cannot step or jump until the Enable CounTeR signal, ECTR3, is developed. The development of ECTR3 is of prime importance throughout the operation of the 5610, and in most cases, the qualifications needed to develop ECTR3 during a given phase give some indication as to the function of that particular phase. Figure 3-56 illustrates the development of ECTR3 in all phases of 1 mode.



DEVELOPMENT OF ECTR3 DURING LOAD
(All on "D" card unless otherwise noted)

FIGURE 3-56

In 1 phase ECTR3 is developed as soon as ST9 goes false: INput Enable Counter, INEC4, is driven false when ST9 goes away. Whenever INEC4 goes false, ECTR3 goes true and the phase counter changes at the next T6B. In 2 phase, INEC4 goes false when the Program Line Home flip-flop, PLH, sets. This provides a step to 3 phase at the T6B occurring after PLH sets.

In 3 phase, ECTR3 is "automatic", meaning no qualifications other than 1 mode and 3 phase are required to produce ECTR3. Note, however, that 3 phase is reached at the T6B occurring during program line home, and even though ECTR3 is developed immediately, the phase counter cannot step until after the timing chain restarts (after the sync pulse is sensed) and a T6B occurs. Thus the step to 4 phase occurs at the first T6B following the sensing of the sync pulse.

ECTR3 is "automatic" in phases 4 and 5 too, so the phase counter simply steps through the next two phases at each T6B. This provides the two "blank" characters on the program line between the sync pulse and the marker. In 4 phase the marker is set into the search register, and after the phase counter steps through 5 phase, the marker is written out onto the line in 6 phase. After the marker is written out, the phase counter waits for PLH to set, and ECTR3 is developed for the step to 7 phase.

In 7 phase, INEC4 is not driven false until INput Signal, INS, is developed. This occurs after a character has been entered via the 2205. Thus, after each character is entered, ECTR3 is developed and the phase counter steps to 8 phase. ECTR3 is automatic in 8 phase, where the parity check occurs, so the phase counter steps to 9 phase at the very next T6B.

In 9 phase a search for the marker occurs: the character is entered into the common register via the 2205 (in 7 phase) at some random time in relation to the program line, so in 9 phase the code waits in the common register until the marker is sensed. When it is, the phase counter steps to 10 phase, where the CRPL line routing allows the code last entered to be written onto the delay line in the position last occupied by the marker. Since ECTR3 is automatic in 10 phase, the phase counter then steps to 11, where the marker is written out behind the character. Next, a jump occurs back to 7 phase (ECTR3 is again automatic), where the counter waits for the next character to be entered.

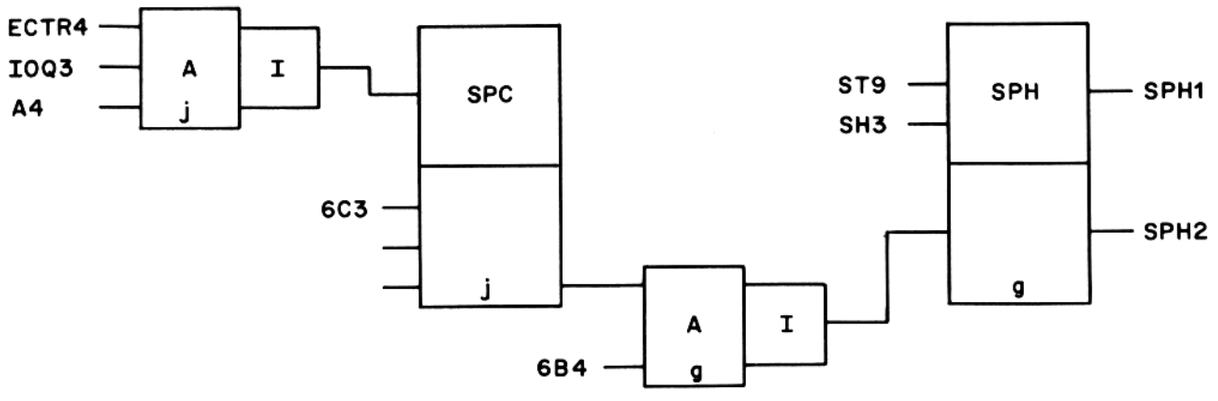
Should a parity error occur in 8 phase, a jump to 12 phase occurs instead of a step to 9 phase. Once in 12 phase, ECTR3 can be developed only after the Reset switch is operated. This allows a step to 13 phase, where the "bad" code is reset out of the common register and a jump back to 7 phase occurs at program line home.

When the load is finally completed, the 5610 is still waiting in 7 phase for more codes to be entered. If the Load switch is operated at this time, it allows the decimal register to be set to 6, driving 6D4 false. This produces Launch Arithmetic Sync Pulse, LASP3, which performs a variety of functions, including the development of ECTR3 for the jump out of 1 mode.

4. PRELIMINARY PHASES

The preliminary phases, 1 through 6, are concerned with launching the sync pulse and the marker. These two operations occur automatically each time system power is first applied and each time the Start pushbutton is operated: in other words, each time ST9 is developed.

a. 1 PHASE. The phase counter operation during the first program line "pass" after ST9 returns to false is somewhat unusual. It was shown earlier that when system power is first applied, the phase counter is set to 1 phase, and that the phase hold flip-flops also contain the 1 phase setting. The phase always shifts at T6B time when the SPH flip-flop resets. However, SPH cannot reset until SPC is set, and SPC cannot set until Enable CounTeR (ECTR) is made true. This is illustrated in Figure 3-57.



TRIGGERING THE PHASE COUNTER

FIGURE 3-57

The problem, then, is to develop ECTR3. In 1 phase this is done as soon as ST9 goes false, as shown earlier in Figure 3-56.

ST9 also sets the program line timing chain to T2A time, so with A4 and ECTR4 both false SPC sets. At the next T6B time SPH resets and causes the phase register flip-flops to assume the state of the phase hold flip-flops. Since ST9 has set 1 phase into both the phase register and phase hold flip-flops, a jump is actually made from 1 phase to 1 phase. At T6C time SPC resets, and at T1SH time, SH3 sets SPH and shifts the 2 phase condition into the phase hold flip-flops.

1 phase is still true, and INEC4 remains false and ECTR3 remains true (refer to Figure 3-56). The following A4 (T1A) again sets SPC.

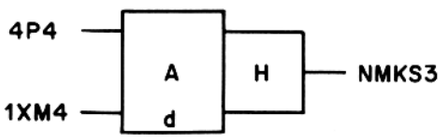
At this point the timing chain stops (because PLH was set by ST9) and nothing further occurs for about 12 ms. Then the sync pulse is sensed and PLH resets.

When the sync pulse is sensed by the read amplifier, PLRS3 resets PLPS and PLH, and sets PLR. PLR4 and PLPL combine to yield EPLW3, and the sync pulse is rewritten (at T2A time). SPC and SPH are set, and at the following T6B time, SPH resets and shifts the phase counter to 2 phase. Note that this occurs at the first T6B time after the sync pulse.

b. 2 PHASE. The timing chain continues to run, but since the sync pulse is the only data on the line, the End of Program Detector output, EPD3, goes true after 40 μsec. and the next T4A sets PLPS. 9.6 μsec. later PLH sets at T4A time and immediately drives INEC4 false. This drives ECTR3 true, allowing the phase counter to step to 3 at the next T6B, which occurs during program line home. Finally, at T1A time the timing chain locks up and waits for the sync pulse to be sensed.

c. 3 PHASE. INEC4 is not necessary to produce ECTR3 during 3 phase. Instead, ECTR3 becomes true at T6B time, as soon as 3 phase is reached. The following "A" time, which is T1A, sets SPC. SPH remains set, however, because the timing chain locks up at T1A and waits for the sync pulse. Nothing further occurs in 3 phase until the sync pulse arrives on the next line pass. When this occurs, the timing chain restarts and the sync pulse is rewritten. The next T6B resets SPH and the phase counter steps to 4 phase.

d. 4 PHASE. In 4 phase the Normal MarKer Signal, NMKS3, is developed (Figure 3-58), which sets the marker into the search register by setting all six SR flip-flops. Then, because ECTR3 is automatic in 4 phase, the next T6B causes a step to 5 phase.



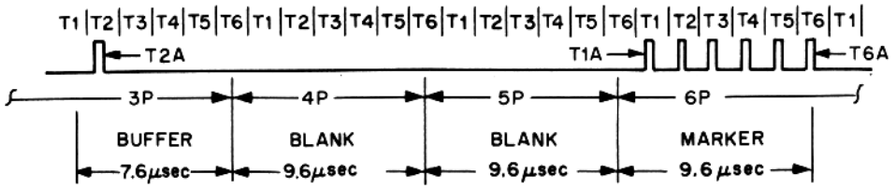
DEVELOPMENT OF NMKS3 IN 1M4P

FIGURE 3-58

e. 5 PHASE. This phase is identical to 4 phase except that nothing happens to the marker: it stays in the search register. Phases 4 and 5 together pass the time required to provide the two "blanks" on the program line.

f. 6 PHASE. In 6 phase the line routing changes from PLPL to SRPL. Since the marker was set into the search register in 4 phase, it is now written onto the program line. Note that the sync pulse was last sensed in 3 phase, and the step was made to 4 phase, 5 phase, and 6 phase at consecutive T6B times. This leaves the time from T2A, where the sync pulse was written,

until the third following T1A, where the marker begins, with nothing written on the line. This is pictured in Figure 3-59. The time between the sync pulse and the next T1SH is referred to as a "buffer", and the next two character times are now blank. These "blanks" will be used later as the program is executed, but it is sufficient to merely note their existence at this time.



LAYOUT - HEAD OF PROGRAM LINE

FIGURE 3-59

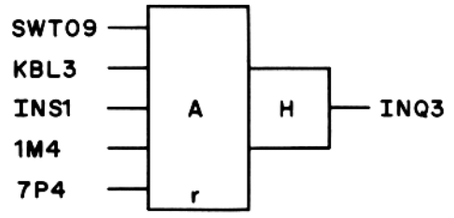
Note that there is no input to the search register at this time, so as the marker is shifted out onto the line, the search register is merely cleared (reset). Since the line routing remains SRPL throughout 6 phase, nothing is written onto the line after the marker.

By the time the marker is completely written out, approximately 30 μsec. have passed since the sync pulse was sensed. Shortly thereafter EPD3 goes true, and PLPS sets at the next T4A. 9.6 μsec. later T4A3 sets PLH. 6P4 and XPLH4 produce ECTR3, and the next T6B provides the step to 7 phase.

g. 7 PHASE. As shown earlier in Figure 3-55, the line routing remains SRPL in 7 phase. Note, however, that PLSR3 is also produced starting in 7 phase. It can now be stated that the line routing will be PLSR, SRPL, until 10 phase is reached.

In order to produce INEC4 to get out of 7 phase, INput Signal, INS, is needed. In other words, the phase counter stays in 7 phase until a character is received from the 2205.

One of the signals needed to produce INS is INput Qualification, INQ3. This is produced in 1 mode 7 phase, as shown in Figure 3-60, as long as the Type Only switch is not operated and the KeyBoard Lock condition is false. (KBL3 becomes true when 1119 characters have been loaded. By disabling INQ3, KBL3 prevents the entry of additional characters.)



DEVELOPMENT OF INQ3 IN 1M7P

FIGURE 3-60

This is the condition in which the 5610 is idling when it is first turned on, before any entry is made. Figure 3-71, page 3-56, illustrates the timing relationships of the signals involved in launching the sync pulse and stepping to this point.

5. LOADING THE LINE

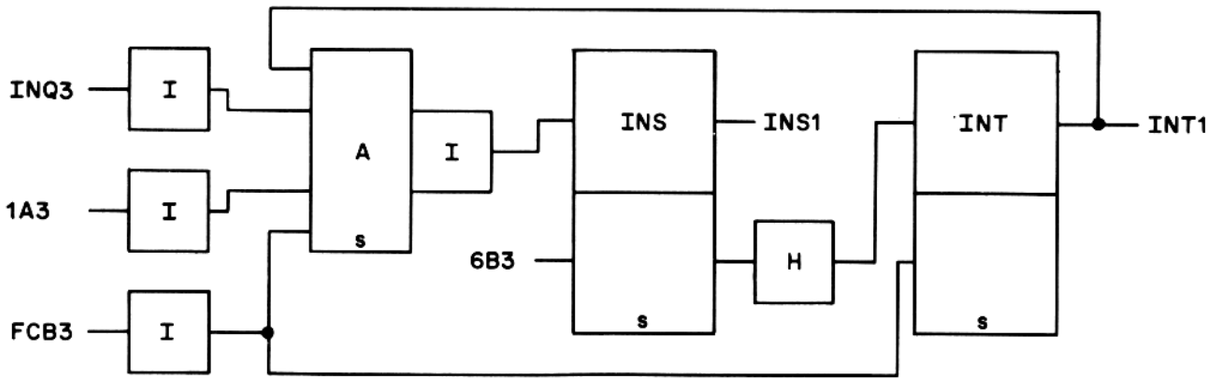
Phases 7 through 11 are used for loading each code onto the delay line. The character is entered into the common register in 7 phase, and is checked for parity in 8 phase. In 9 phase the marker is located, in 10 phase the new code is written out in place of the marker, and in 11 phase the marker is written out behind the new code. The line routing remains PLSR, SRPL in all phases but 10, where CRPL is needed to write out the new code.

There are two types of signals produced by the writing machine during entry. These are classified as "Data signals" and "Control signals". The data signals load the code, representing the character being entered, into the common register. The control signals allow the program line logic to move on from 7 phase and complete the transfer of data from the common register to the program line.

There are eight data signals, named for the eight tape channels. F1B3 through F8B3 are produced by the 2205 reader or selector code contacts. The primary control signal is Flexowriter Common Bit, FCB3. It is produced by the 2205 reader or selector common contacts. The actual circuits involved in producing FCB3 and the data signals are covered in Section 8, Electronic Circuits. Note, however, that FCB3 is needed to produce INS3, and INS3 is needed to produce F1B3, F2B3, etc.

a. 7 PHASE. The primary control signal, FCB3, is produced by the 2205 at some random time in relation to the program line timing chain. The length of FCB3 is determined by the length of time the reader or selector common contact circuits are complete. The first T1A after FCB3 goes true produces INS1 and INT1, as shown in Figure 3-61. When INS1 goes true, the data signals are produced; this sets the code corresponding to the character being loaded into the common register. The INS flip-flop resets at the next T6B time, but INT remains set until FCB3 again goes false. INT prevents INS from setting again, so the character entered can be loaded only once. It acts as an "anti-repeat" flip-flop.

The T6B that resets INS also causes the step to 8 phase.



INS/INT LOGIC

FIGURE 3-61

b. 8 PHASE. ECTR3 is produced immediately in 8 phase and the step is made to 9 phase at the next T6B time, or 9.6 μsec. later. However, during 8 phase, CRCR3 is also developed, which performs the common register ring shift to check parity. If an even parity code is detected, a jump to 12 phase will occur instead of a step to 9 phase. This possibility will be covered later in this section.

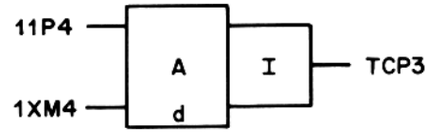
c. 9 PHASE. The logic remains in 9 phase until the marker is in the search register: SRTF4 must be false in order to produce INEC4. The line routing in 9 phase is still PLSR, SRPL, so each character on the program line is passing through the search register before being rewritten. When the marker is shifted into the search register, all six SR flip-flops are set and SRTF4 becomes false. This occurs as the last shift takes place, at T6SH time. 400 nsec. later, at T6A time, INEC4 becomes false, and at T6B time the phase counter shifts to 10 phase.

d. 10 PHASE. The phase counter steps to 10 phase at T6B time when the marker is in the search register. 10 phase immediately drives SRPL3 and PLSR3 both false (refer to Figure 3-55). With both of these signals false, there is no pulse available to shift the search register, so the marker remains there. At the same time, CRPL3 becomes true and the code in the common register is shifted out and written onto the program line. Because no other line routing signals are true at this time, there is no input to the common register, and it is cleared as the character is shifted out. ECTR3 is developed immediately in 10 phase, so at the next T6B the phase counter steps to 11 phase.

e. 11 PHASE. Basically, three things happen in 11 phase: the marker is written out, the character just loaded is counted, and a jump back to 7 phase is initiated.

Since 10P3 no longer inhibits PLSR3 and SRPL3 (Figure 3-55), this line routing is resumed and the marker is shifted out of the search register and written back onto the program line, immediately behind the character just loaded.

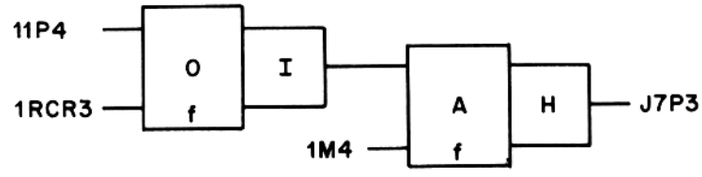
The Timing Chain Pulse, TCP3, is developed, as shown in Figure 3-62, and is routed to the arithmetic timing chain to increment the 1119 counter one time. This count actually occurs at the end of 11 phase, when TCP3 returns to false.



DEVELOPMENT OF TCP3

FIGURE 3-62

J7P3 is developed as shown in Figure 3-63. This signal immediately sets the phase hold flip-flops to 7 phase, and since ECTR3 is true, a jump to 7 phase occurs at the next T6B.



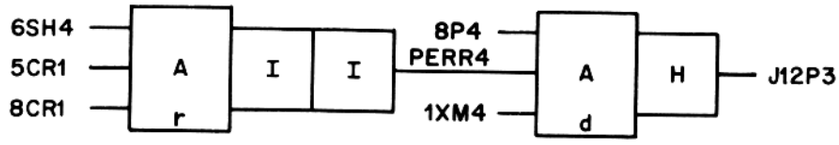
DEVELOPMENT OF J7P3 IN 1M11P

FIGURE 3-63

Once back in 7 phase, INQ3 again becomes true and the processor waits for the next 2205 entry (Figure 3-60). Note, however, that FCB3 may still be true from the character just loaded. If this is the case, INT1 will still be true, and the set signal to the INS flip-flop will be blocked (refer to Figure 3-61), thus preventing the same character from being set into the common register. Before the system can leave 7 phase, FCB3 (and INT1) must first go false. Then another entry must be made in order to develop INS1, to ultimately yield ECTR3.

6. PARITY ERROR

Figure 3-64 illustrates the development of the jump to 12 phase when a parity error is detected during 8 phase. The ECTR3 necessary for this jump is automatic, the same as for the step from 8 phase to 9 phase, and was illustrated in Figure 3-56.

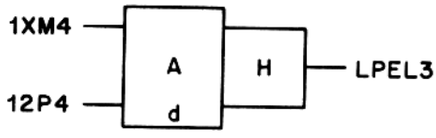


DEVELOPMENT OF J12P3 BY PARITY ERROR

FIGURE 3-64

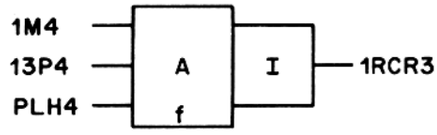
When 12 phase is realized, Light Parity Error Lamp, LPEL3, is developed as shown in Figure 3-65. This signal is applied to a lamp driver (see Section 8, Electronic Circuits) to light the Error lamp on the front of the processor.

The system remains in this state until the operator depresses the Reset switch. This sets the RST flip-flop, producing INEC4 and ECTR3, which allows a step to 13 phase at the next T6B time. 13 phase remains until program line home, when 1RCR3 is produced, as shown in Figure 3-66.



DEVELOPMENT OF LPEL3 IN 1M12P

FIGURE 3-65



DEVELOPMENT OF 1RCR3

FIGURE 3-66

1RCR3 then does the following:

(1) Sets the CRR flip-flop to reset the "bad" code out of the common register.
(2) Resets the RST flip-flop.
(3) Produces J7P3 to set the phase hold flip-flops to 7 phase (see Figure 3-63).
(4) Develops ECTR3 so that the first T6B to occur when the timing chain restarts will jump the phase counter to 7 phase, where the processor will wait for another entry.

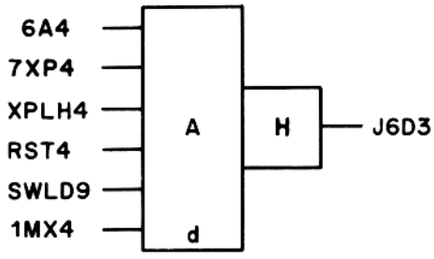
7. LOAD OVERFLOW

It was noted earlier that TCP3 increments the 1119 counter in the arithmetic section during 11 phase each time a program character is loaded. The method by which the arithmetic timing chain does this is explained in Section 4, Arithmetic Line Logic. That section also shows how KBL3 is developed when 1119 characters have been loaded. KBL3 opens the LKL circuit in the 2205, thus locking the keyboard and stopping the reader.

8. TERMINATE LOAD

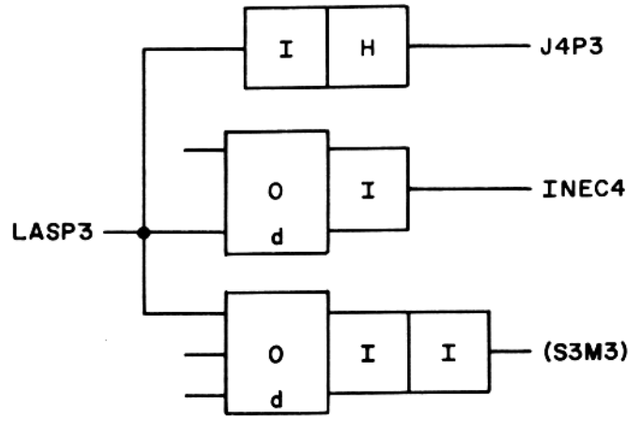
The load condition can be terminated by depressing either the Load switch or the Reset switch. Either of these switches will set the RST flip-flop. With RST set, the Jump to 6 Decimal signal, J6D3, is produced at the T6A time occurring after PLH sets. See Figure 3-67. Note that if RST was set by depressing the Load switch, J6D3 does not occur until after the switch is released.

J6D3 immediately sets the decimal register to 6, and this produces Launch Arithmetic Sync Pulse, LASP3, as shown earlier in Figure 3-56. In addition to launching the sync pulse onto the arithmetic delay line (to be covered later), LASP3 also develops J4P3, INEC4, and (S3M3), as shown in Figure 3-68.



DEVELOPMENT OF J6D3 IN 1M7P

FIGURE 3-67

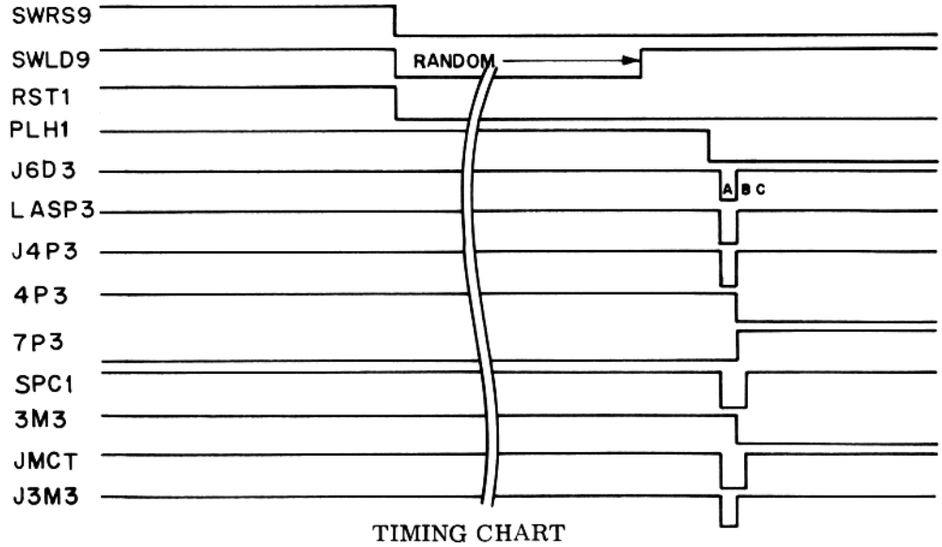


JUMPING TO 3M4P AT END OF LOAD

FIGURE 3-68

Timing becomes very important at this point. The only timing signal used so far in terminating the load condition has been the T6A time occurring during program line home. At this time, J6D3, LASP3, J4P3, INEC4, and (S3M3) have all been developed. INEC4 produces ECTR3, and this, together with fractional bit time "A" (T6A), sets SPC so that the next T6B can reset SPH and jump the phase counter to 4 phase. (S3M3) has set JMCT, 1MH, and 2MH, so the same T6B will set the mode register to 3 mode. Thus, at T6B time the program line logic is in 3 mode, 6 decimal, and 4 phase.

The waveforms associated with terminating Load are illustrated in Figure 3-69. Note that 3 mode is finally realized during program line home, and no more phase changes occur until the timing chain restarts after the program line sync pulse has again been sensed on the next line pass.



TIMING CHART
TERMINATING LOAD

FIGURE 3-69

This is acutally the end of the load mode. However, before the Load operation can be called complete, the marker must be searched out and erased. This is the function of the 3M6,4 setting with RST set, which will be explained shortly when normal marker search is covered. Remember the 3M6,4 setting at program line home, as it will be mentioned often during other operations.

9. SUBSEQUENT LOADS

Sometimes it is necessary to load a second program after the 5610 has been operating. One way to do this would be to simply turn the machine off and back on: the processor would automatically be in the load condition when it is turned back on.

This is not always practical, however. If the data accumulated by the execution of the first program must remain in storage, to be used in performing the second program, the system cannot be turned off. Instead, the operator has only to depress the Load switch again. This conditions the 5610 to accept another program (thus destroying the original) without disturbing the stored data. Loading a second program in this manner is termed "program overlay". Since the arithmetic timing chain is used to count the number of program characters loaded, data on the arithmetic delay line is lost.

When the processor is idling, after being in operation, it is in 3 mode, 0 decimal, 1 phase (3M0,1), waiting for the operator to enter the next sequence title. Depression of the Load switch at this time causes a jump to 1 mode (and 1 phase). Refer to Figure 3-70. SWLD9 sets the LD flip-flop. At this time, the only signal preventing (S1M3) from becoming true is SWLD9. As soon as the Load switch is released, (S1M3) becomes true and sets 1 mode into the mode hold flip-flops, and also sets JMCT. (At the same time J0D3 insures the reset of the RST flip-flop. RST is set by SWLD9 to bring the processor to the 3M0,1 setting, should the Load switch be depressed while the 5610 is executing a program sequence). At the next T6B time the 1 mode setting is shifted into the mode register flip-flops, and 1M3 immediately resets LD.

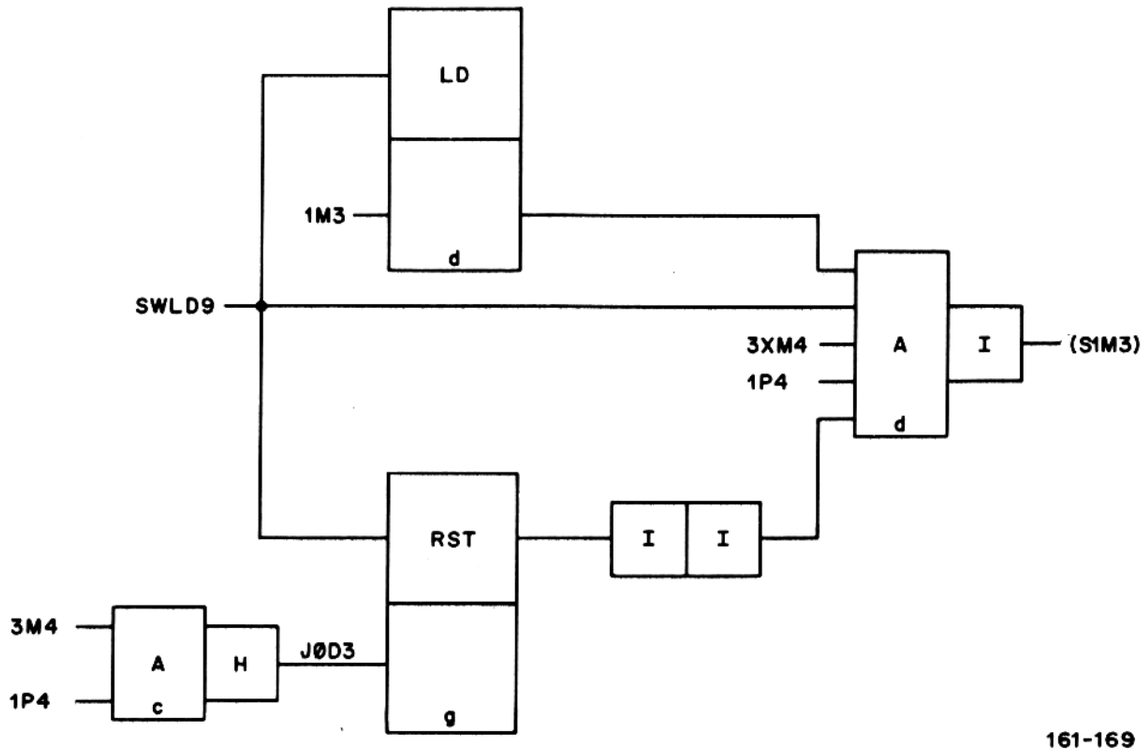
At this point the processor is in 1 mode and 1 phase. Compare this condition to the 1M1P setting when the 5610 is first turned on. ST9 does not enter the picture this time, so no sync pulse is launched (instead, the existing sync pulse will be used). Also, whereas the phase counter made a 1 phase-to-1 phase jump after the initial turn-on, the phase hold flip-flops now contain the 2 phase setting, and the next T6B causes a step to 2 phase. The processor then waits for program line home before developing ECTR3 for the step to 3 phase. This and subsequent operation in 1 mode is identical to that described earlier.

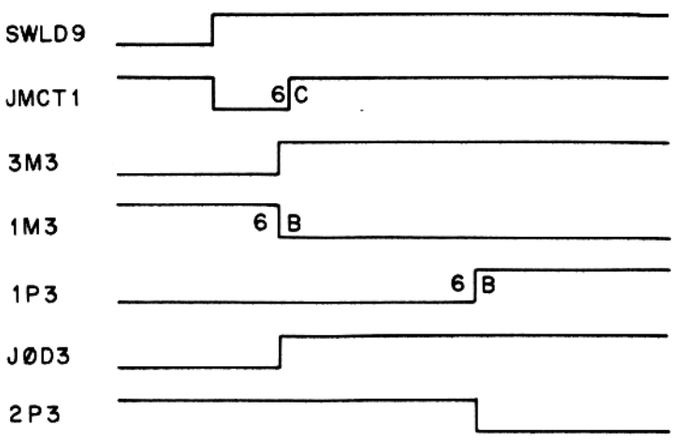
Up until this point, and until 6 phase, when the marker is written out, nothing has yet affected the data ("old" program) on the program line. However, it was pointed out earlier that in 6 phase the line routing is SRPL ONLY. Therefore, the marker is written out (after the second "blank"), but no data is written onto the line after it. 6 phase remains until program line home, when all previous data has passed through PLR and then been lost. In this way the old program is destroyed.

All other aspects of "program overlay" are the same as for the first load.

Figures 3-71 and 3-72 illustrate the timing relationships of the more important signals involved in 1 mode.

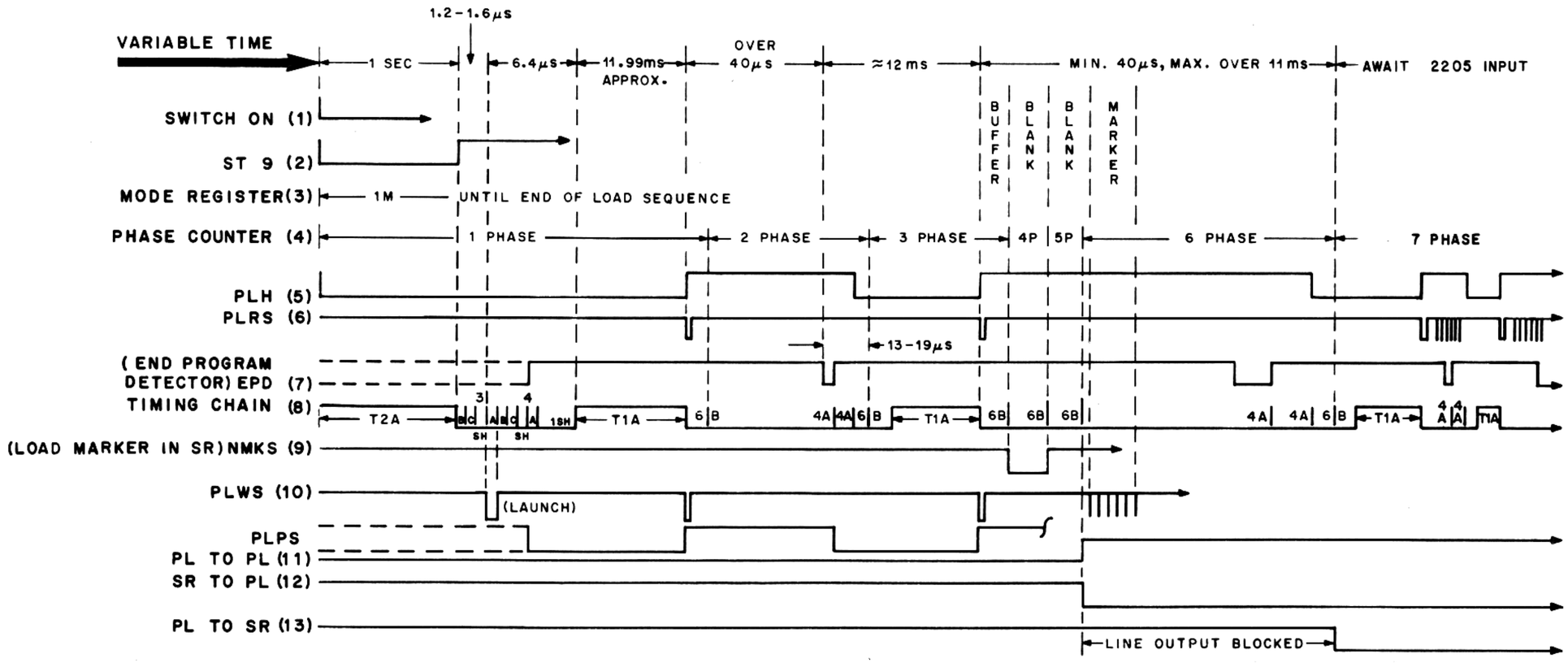
Figure 3-73 illustrates the load process in Flow Chart form.





RE-ENTERING LOAD MODE

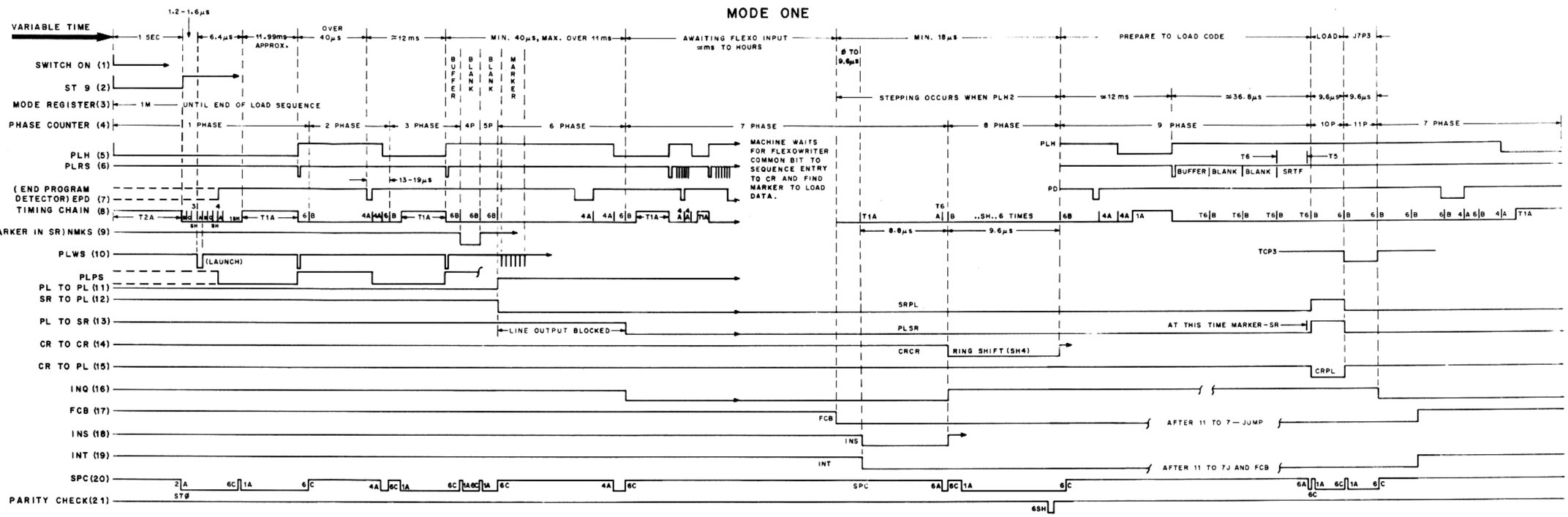
FIGURE 3-70



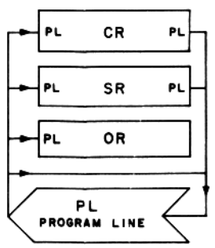
TIMING CHART
LOAD MODE
PRELIMINARY PHASES

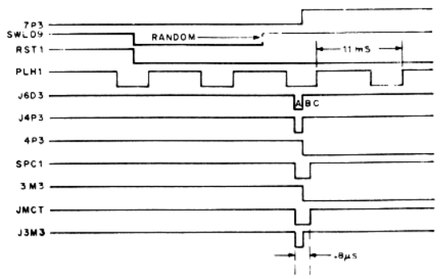
FIGURE 3-71

TP-161SL



A. DATA IS CIRCULATED PL TO PL UNTIL 6 PHASE WHEN THE MARKER IS LOADED.
B. DATA IS CIRCULATED PLSR SRPL FROM 7P UNTIL IN 9P WHEN THE MARKER IS DETECTED.
C. WHEN THE MARKER IS IN SR, THE LINE IS ROUTED PLCR CRPL TO LOAD CODE IN 10P.
D. PLSR SRPL IS RESUMED IN 11P TO WRITE MARKER OUT.
E. A JUMP TO 7P IS DONE AND THE LOGIC WAITS FOR THE NEXT FLEXO ENTRY.



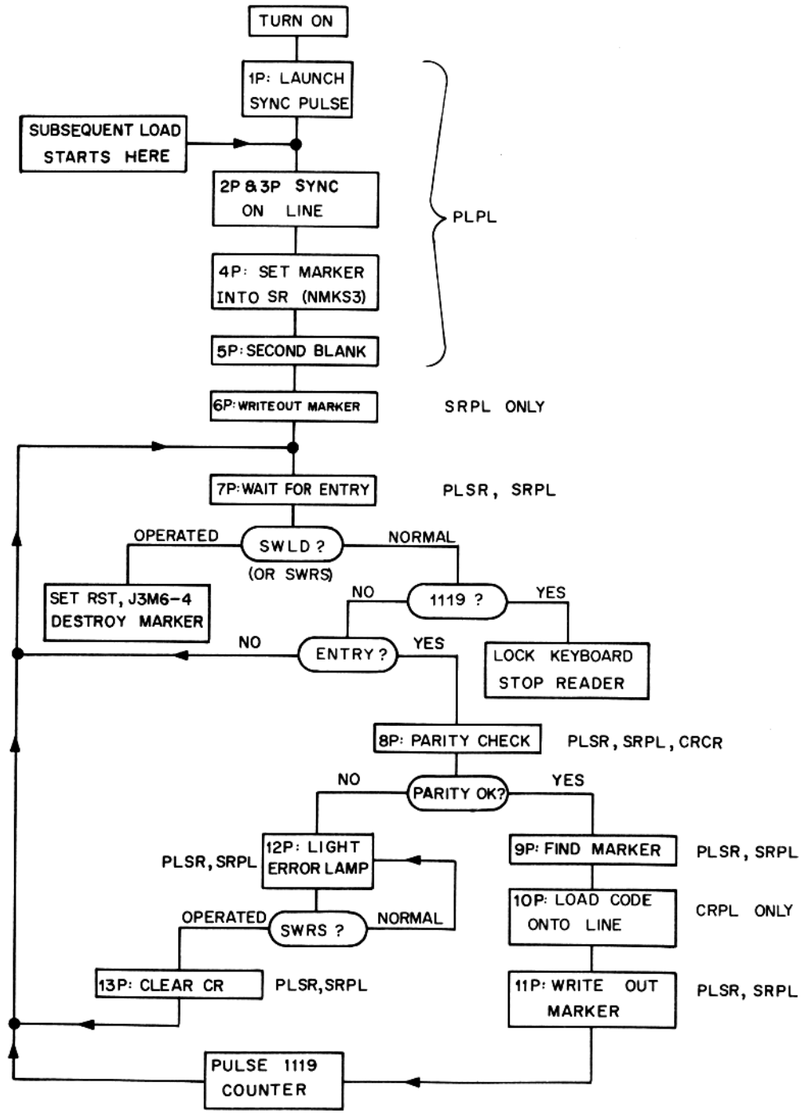


TIMING FOR JUMP TO 3M64 WHEN LOAD SWITCH IS RE-OPERATED.

161-164

TIMING CHART
PROGRAM LOAD

FIGURE 3-72



FLOW CHART
1 MODE
PROGRAM LOAD

FIGURE 3-73

TP-161SL

G. 3 MODE, SEARCH

All searches are conducted in 3 mode. After program load is completed, 3 mode is used to search out and destroy the marker. When a sequence title is entered by the operator, 3 mode is used to locate the beginning of that sequence and insert a marker on the program line before the first instruction in that sequence. Then, after each instruction in that sequence is executed, 3 mode provides the search for the marker that allows the next instruction to be retrieved. Whenever a branch is executed, a 3 mode search locates the new batch of instructions that is to be performed.

1. GENERAL OPERATION

There are basically three different types of searches in 3 mode: Sequence search, Marker search, and Resume search. Each of these searches can be modified, however, when other factors come into play, such as Translate, conditional or unconditional branch, the Reset flip-flop, etc.

a. SEQUENCE SEARCH. Sequence "titles", or the addresses following sequence separators, must be checked when a particular sequence is being sought. The search for a particular sequence can take place under one of four conditions: normal sequence search, unconditional branch sequence search, translate sequence search, or conditional branch sequence search.

(1) Normal Sequence Search. When a sequence is selected by the 2205, 3 mode begins a search of the entire program line for that particular sequence. The 5610 detects all normal sequence separator codes ("H"), and then checks the code immediately following with the code entered via the 2205. When the correct sequence is located, a normal marker is inserted on the program line immediately following the sequence title.

(2) Unconditional Branch Sequence Search. When an unconditional branch instruction is performed, the address, which designates the title of the sequence to which the branch is to be made, is loaded into the common register. A jump to 3 mode is then executed and the search for the new sequence is performed. The 3 mode portion of this search is identical to a normal sequence search except for the fact that the UBR flip-flop is set. This causes the branch marker, instead of the normal marker, to be inserted on the program line after the proper sequence is found.

(3) Translate Sequence Search. Although mode 2 is entitled the Translate mode, its function is simply to allow the entry of three characters instead of only one. The actual search for the translate sequence is accomplished in 3 mode. In this case, 3 mode detects each translate sequence separator ("L"), and compares the next three codes with the three codes entered from the 2205 during 2 mode. When the proper sequence is located a normal marker is inserted on the program line following the third sequence title code.

In each of the above three cases, if the proper sequence title is not loaded on the program line and cannot be found, a search for the error sequence (sequence "space") will be performed. All programs should have a sequence "space" to inform the operator that a non-existent sequence has been selected.

(4) Conditional Branch Sequence Search. When a conditional branch instruction is executed, a search is first conducted to locate the head of the sequence in which program operation was initiated. In this case, when the sequence is found a marker is not written onto the line, because a search for the Resume command must be conducted at this time.

b. MARKER SEARCH. There are two types of markers in the 5610 for which a search is conducted in 3 mode. The normal marker is a tape feed code (1-2-3-4-6-7) and the branch marker is the "degree" symbol code (2-4-7).

(1) Normal Marker. As each instruction is executed, a normal marker is inserted on the program line behind the command. When the instruction is completed a search for the marker is executed in 3 mode so that when the marker is detected, the next instruction may be retrieved.

(2) Branch Marker. When an unconditional branch instruction is executed the normal marker is left on the program line behind the branch instruction. As noted earlier, a branch marker is placed on the program line at the head of the new sequence. As each instruction is executed, the branch marker is moved to a position behind the command. When the instruction is completed a search for the branch marker is conducted in 3 mode in order to detect the next instruction. When in unconditional branch the UBR flip-flop is set, which causes the normal marker to be ignored and only the branch marker to be recognized.

c. RESUMES. As noted earlier, when a conditional branch instruction is executed, a search is first conducted to find the head of the sequence containing the branch instruction. After the sequence is found, a 3 mode search is conducted for the proper Resume code, starting at the beginning of the designated sequence. All Resume codes are detected, and the code following is then compared to the address of the conditional branch instruction. When the correct Resume is located, a marker is inserted on the program line after the Resume title.

2. PHASES

Certain phases in 3 mode always denote that a particular point in the program line has been reached (a particular point is passing through PLR) or that a search for some particular code or type of code is being conducted. The purpose of some phases is modified by some decimal settings, and the settings of the decimal register often dictate the steps to be taken when a particular code is sensed. This will be covered in greater detail shortly.

a. 1 PHASE. This phase is realized only in 0 decimal and denotes that the processor is waiting for a sequence title entry from the 2205.

b. 2 PHASE. During this phase, which is also limited to 0 decimal only, the parity check is performed on the 2205 entry.

c. 3 PHASE. Whenever 3 phase occurs, it serves as the idle condition while waiting for program line home.

d. 4 PHASE. In all decimal settings, 4 phase is used to idle through program line home. When the sync pulse is sensed, it restarts the timing chain, which steps the phase counter to 5 at the first T6B. 3 phase and 4 phase work together to synchronize the phase counter to the program line.

e. 5 PHASE. This phase is true while the first "blank" on the program line is passing through PLR.

f. 6 PHASE. This phase is true while the second "blank" is passing through PLR. At the end of 6 phase a jump to 9 phase is always performed.

g. 7 PHASE. This phase always searches for a 1-address command (a code containing at least 1-6-7). In most cases, if a 1-address code is detected a step to 8 phase is performed; if a 1-address code is not detected, a jump to 9 phase is performed. In normal sequence search (0 or 5 decimal), the presence of an "H" on the program line causes a jump to 12 phase. In 4 decimal (Resume search) an "A" on the program line also causes a jump to 12 phase. During marker search an "H" on the program line causes a step to 8 phase.

h. 8 PHASE. This phase causes an idle through the address of a 1-address command. Such a maneuver is necessary under various circumstances. An example would be when a "Resume H" is on the program line. If the processor is searching for a normal sequence separator (H), all Resume titles are sluffed in 8 phase. If this did not happen, the "H" in the above example would be recognized as a normal sequence separator.

i. 9 PHASE. The search for a translate sequence separator (L) is conducted in 9 phase. If an "L" is detected a step through 10-11-12 phases is performed. If no "L" is found, a jump to 7 phase occurs immediately. This jump is the only one that does not wait for T6B - as soon as the code in PLR differs from an "L" (1-2-7), the jump is made.

j. 10, 11, 12 PHASES. These phases are used in 1, 2, and 3 decimal to compare translate sequence titles with the codes entered via the 2205. In all other decimal settings they are used to sluff the three address codes following each translate sequence separator. An exception to this is found in 12 phase, which is also used to perform the compare when searching for a particular Resume or normal sequence separator. If the compare is successful, a step to 13 phase is allowed to write out the marker: if the compare is unsuccessful, a jump to 9 phase is performed to continue the search.

k. 13 PHASE. The most important use of this phase is to write the marker onto the line after a successful compare has been made. This is done after finding the proper sequence separator in normal sequence search, translate sequence search, and unconditional branch sequence search, and after finding the correct Resume title during conditional branch. 13 phase remains true after a successful compare until program line home.

l. 14 PHASE. This phase is used to initiate the search for the error sequence when program line home is reached without having made a successful compare. It is used only in 0, 1, 2, or 4 decimal.

m. 15 PHASE. This phase is used only in 0 decimal to stop operation in the case of a parity error in the 2205 entry made in 1 phase.

3. DECIMAL SETTINGS

The various decimal register settings in 3 mode dictate exactly what type of search (sequence search, marker search, Resume search) is being conducted.

a. 0 DECIMAL. During 0 decimal a search is being conducted for a specific sequence separator, or letter "H".

b. 1, 2, 3 DECIMAL. These settings are referred to as "Translate Decimal". They indicate that a search for a specific translate sequence separator is in progress.

c. 4 DECIMAL. This setting, used only in conditional branch, indicates that the proper normal sequence separator has been found and that a search for the correct Resume is in progress.

d. 5 DECIMAL. This setting, used only in conditional branch, indicates that a normal sequence search is underway. Normally, the sequence searched out will be the one originally selected by the 2205. However, if the UBR flip-flop is set, the sequence selected will be the one to which the last unconditional branch was made.

e. 6 DECIMAL. This decimal setting is referred to as marker search. If the UBR flip-flop is reset all codes except the normal marker will be ignored. If UBR is set, only the branch marker will be recognized.

4. BASIC SEARCH PATTERNS

All of the different searches in 3 mode are made up of a combination of a few basic search patterns. Some of these patterns are always the same, regardless of the type of search being conducted. Others are modified slightly to make each of the various searches possible.

a. PRECAUTIONS. When a search is being conducted for a specific character in 3 mode, certain steps must be taken so that a command address or sequence title is not recognized as the character being sought. For instance, assume that a search is being conducted for "Resume B", and the program line contains the characters shown below.

H	Sequence separator
A	Sequence title "A"
B	Subtract
1/2	Type out
A	Address "A"
B	Subtract
A	Resume
B	Resume title "B"

Some means must be provided to differentiate between the three "AB's". The first "A" is a sequence title, and the second is the address of a Type command. Neither of these should be recognized as a Resume command.

This problem is compounded by the fact that the program may contain translate sequence separators, each one followed by THREE address characters. Also, the "Resume" is not the only character that can be confused; various titles and addresses can also be recognized as translate sequence separators (L), normal sequence separators (H), and different commands.

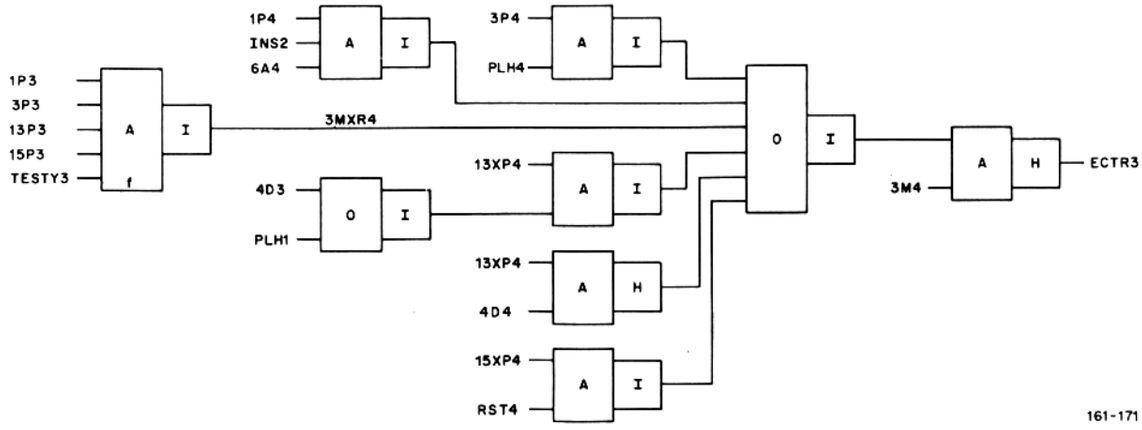
Note that it is not necessary to isolate 2-address codes: both address characters of a 2-address code are always numerics, and cannot be erroneously recognized as sequence separators or Resumes.

The mechanics of identifying the different types of characters are similar for each of the various searches. In many cases it would seem that a particular type of search could be accomplished more easily in a different way. It should be remembered that even though a more direct way may be possible, a roundabout method may have been used in order to utilize existing machine logic. Using the direct way might make additional circuits necessary. In other words, the method which is easiest to understand may not necessarily be the most practical method to use.

b. ENABLING THE PHASE COUNTER. ECTR3 is developed continually in most phases of 3 mode (see Figure 3-74). The only phases requiring special qualifications are 1, 3, 13, and 15 phases. All other phases develop 3MXR4, which, when ANDed with 3M4, produces ECTR3.

When the 5610 operator selects a sequence to be performed, the entry is made in 1 phase. ECTR3 is qualified by INS3 so that 1 phase will remain until the entry is completed.

In 3 phase ECTR3 is qualified by program line home. This enables the phase counter to sync onto the program line for the step to 4 phase. The same thing occurs in 13 phase. Note in Figure 3-74, however, that this operation is different in 4 decimal. In 13 phase and 4 decimal, the phase counter does not jump to 4 phase at program line home, but instead jumps to a different phase after only one character time in 13 phase. This will be explained more fully under Conditional Branch later in this section.



DEVELOPMENT OF ECTR3 IN 3 MODE
(All gates on "C" card unless otherwise noted)

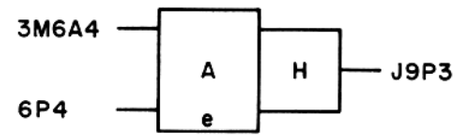
FIGURE 3-74

In 15 phase ECTR3 is qualified by RST4. This enables the phase counter to remain in 15 phase (effectively stopping machine operation) until the Reset switch is operated.

c. PHASE COUNTER PATTERNS. During the preliminary phases (4, 5, and 6) the pattern to be followed is determined by the program line timing chain, or by what portion of the program line data is being sensed. During the latter phases (7 through 13) the pattern used is determined by the data coming off the line.

(1) PRELIMINARY PHASES. The search usually begins at 4 phase during program line home. As noted under "Program Load", 4 phase is realized at T6B time after PLH sets. After a sequence entry is made in 1 phase, the logic automatically progresses from there to 4 phase, which is reached at T6B time during program line home. In other words, 4 phase is true during most of program line home (exception: Resume search).

The T6B that causes the step to 5 phase occurs at the end of the "buffer", after the sync pulse arrives. A step to 6 phase occurs at the next T6B (the end of the first "blank"). At the following T6A time, near the end of the second "blank", a jump to 9 phase is enabled (9 phase is set into the phase hold flip-flops), as shown in Figure 3-75. Then, at T6B time, the phase counter jumps to 9 phase.



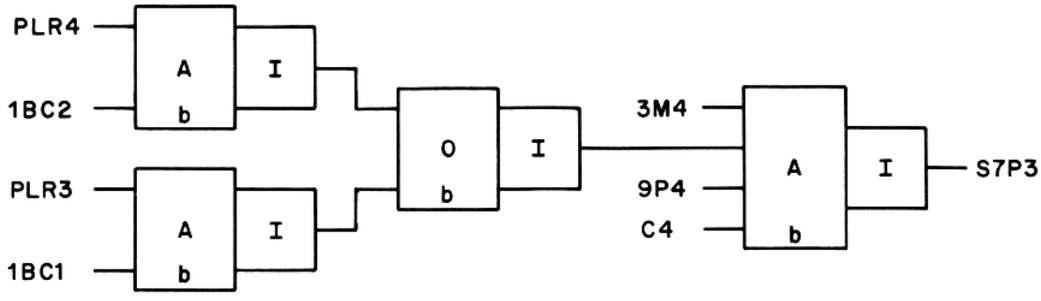
DEVELOPMENT OF J6P3 IN 3M6P

FIGURE 3-75

The actual search begins here in 9 phase. The preceding conditions are true for ALL 3 mode searches: 5 phase and 6 phase occur during the two "blanks" and a jump to 9 phase occurs at the end of the second blank (where the actual program starts) so that the search can begin.

(2) The Actual Search. Each new command or sequence separator is first sensed in 9 phase. By analyzing this code, 3 mode steps through the other phases necessary to sluff addresses or sequence titles, and to compare various codes, until the desired one is located. As each code or group of codes is bypassed, a jump is made back to 9 phase in order to "begin" a new search.

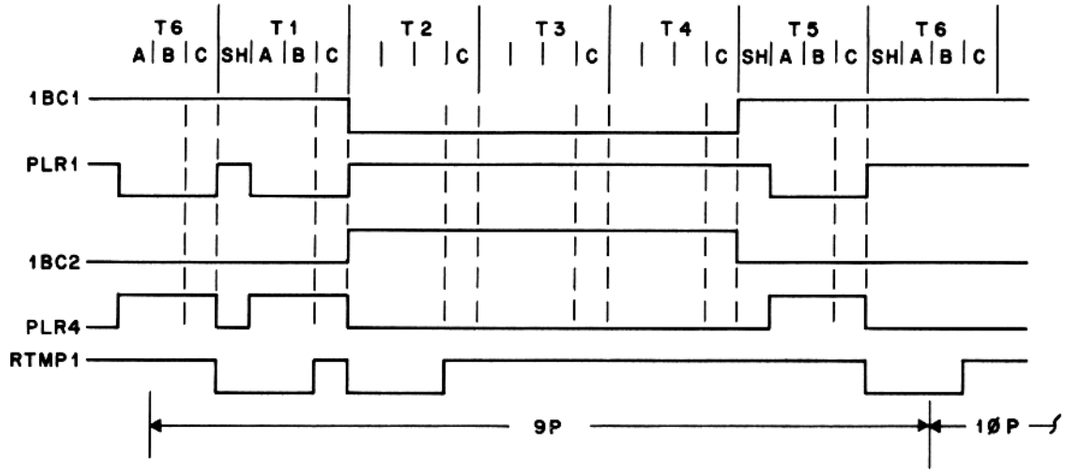
(a) 9-7 Jump. During 9 phase the program line output is monitored for the presence of a translate sequence separator code, letter "L". As soon as the code coming off the line is determined to be something other than an "L", a jump to 7 phase occurs. (This is the only jump that does not wait for T6B time, as explained in the phase counter discussion.) If the code being sensed is an "L", no jump occurs, and the step to 10 phase is made at T6B.



DEVELOPMENT OF S7P3

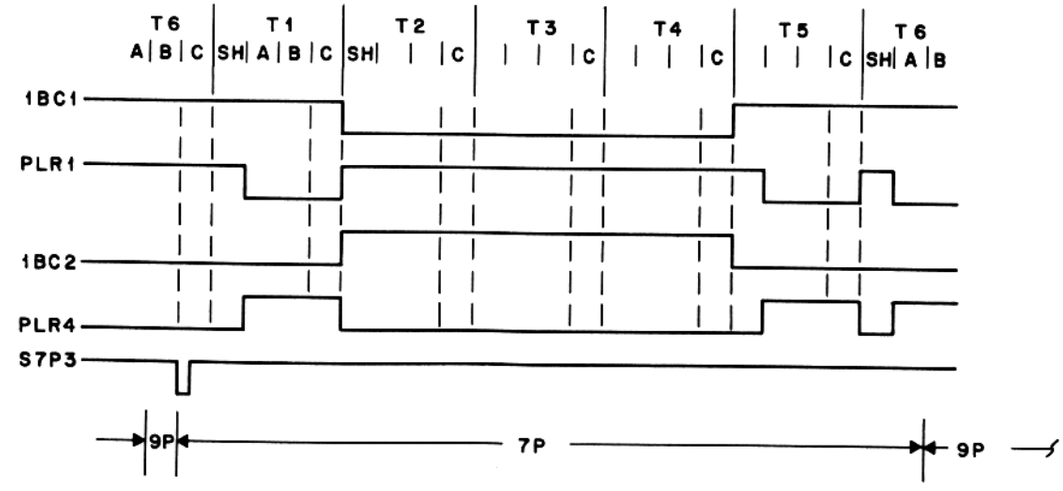
FIGURE 3-76

As explained earlier S7P3 sets the JPC flip-flop and causes an immediate jump to 7 phase. As shown in Figure 3-76, S7P3 can only be developed in 3 mode 9 phase at fractional bit "C" time. By tracing back the fourth leg of the AND gate, it will be seen that this leg becomes false only when the contents of PLR equal something other than an "L" code. Figure 3-77 (a) compares the PLR waveforms when an "L" is sensed with the 1BC waveforms. Note that even though the two flip-flops become set at different times, the comparison is actually made only at "C" time, when the corresponding flip-flop outputs are always opposite. Should they ever be the same at "C" time, the final input to the AND gate in Figure 3-76 would become false, S7P3 would become true, and the jump to 7 phase would occur immediately. For instance, if the code being sensed does NOT contain a 1-bit, PLR1 is false at T6C time, and S7P3 becomes true. In this case, the phase counter action would be as shown in Figure 3-77 (b).



TIMING CHART
DETECTING "L" IN 3M9P

FIGURE 3-77 (a)



TIMING CHART
DETECTING CODE OTHER THAN "L"

FIGURE 3-77 (b)

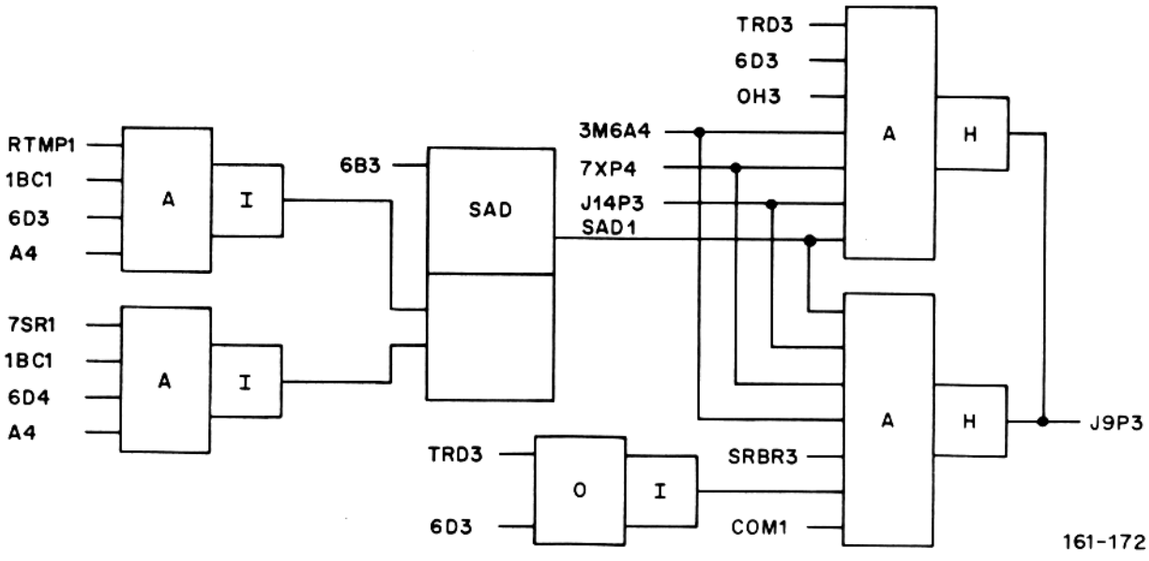
If the code being sensed in Figure 3-77 (b) were a digit "3", (1-2 on the program line) S7P3 would not be developed until T5C time. In other words, S7P3 can be developed at any fractional bit "C" time, T6 through T5, depending upon the code sensed.

What occurs when an "L" is sensed and the phase counter steps to 10 phase depends upon the type of search being conducted. In any case, the phase counter always steps through 11 phase and 12 phase after it leaves 10 phase. This will be elaborated upon shortly. For now it is sufficient to note that the great majority of codes sensed during 9 phase (all codes except "L") will cause a 9 phase to 7 phase jump.

(b) 7-9 Jump vs. 7-8-9 Step. Throughout 9 phase and into 7 phase each code is checked for the presence of the 1-6-7 bits. Those codes not containing the 1-6-7 cause a jump back to 9 phase at T6B, the net result being that the code is merely bypassed during 9 and 7 phases. If the code does contain the 1-6-7 bits, it is a 1-address code, and its address character must be bypassed, so the jump to 9 phase is blocked and a step to 8 phase occurs instead. The address character is sluffed in 8 phase. At T6B during 8 phase, the phase counter steps to 9 phase.

The decimal (1-2-4-6-7) and the normal marker are exceptions: even though they contain the 1-6-7 bits, they do not always block the jump to 9 phase. This will be explained shortly.

The logic element used to detect the presence of the 1-6-7 bits is the Single ADdress flip-flop, SAD. All of the logic directly connected with setting and resetting the SAD flip-flop is shown in Figure 3-78. It sets at every T6B time, and it resets whenever the code sensed is something other than a single address command. In other words, any code containing the 1-6-7 bits will allow SAD to remain set. If SAD resets, a jump to 9 phase occurs at T6B time. If SAD remains set, it blocks J9P3 and allows the phase counter to step to 8 phase. During 8 phase, the code following the single address command (the "address") is sluffed. By being sluffed in 8 phase, the address cannot be recognized as a sequence separator or a Resume, regardless of what code it is.

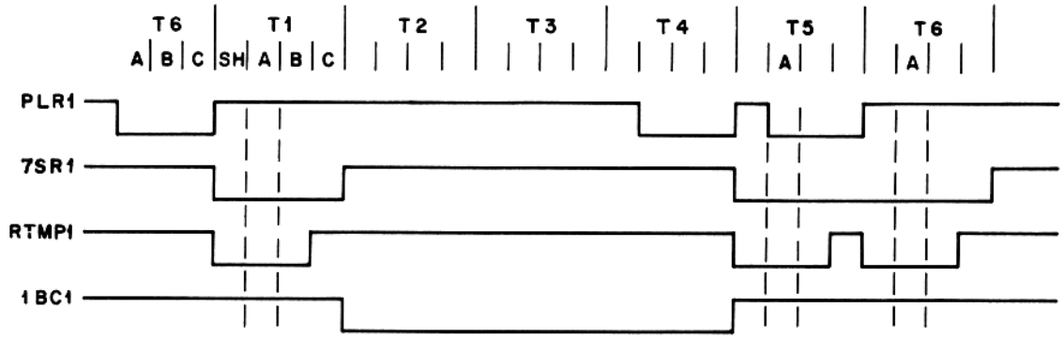


SAD LOGIC
(All on "B" card)

FIGURE 3-78

Refer briefly to the timing charts in Figure 3-77 and note that in the case of S7P3, the 1BC flip-flop was used to check the program line for the 1-2-7 code. In the case of SAD, 1BC is checking for a 1-6-7. How can the same flip-flop check for both of these characters? The answer lies in TIMING. In the case of S7P3, 1BC was compared to PLR at fractional bit "C" time. In the case of SAD, 1BC is compared to either RTMP or 7SR, but at fractional bit "A" time. The key to this difference is the fact that a "SH" occurs between "C" and "A", and the states of all flip-flops can change at this "SH" time.

Figure 3-79 illustrates the waveforms associated with sensing a Resume code with the line routing PLPL and PLSR. As shown previously in Figure 3-78, SAD can only be reset when both 1BC1 and RTMP1, or 1BC1 and 7SR1, are reset at the same time. Figure 3-79 shows that if the code being sensed contains the 1-6-7 bits, SAD will not reset. The presence or absence of the 2-3-4 bits is of no consequence.



TIMING CHART
DETECTING 1-ADDRESS CODE

FIGURE 3-79

In actual practice, the combination of PLPL3 and PLSR3 at the same time is rare. This was used as an example in Figure 3-79 only to avoid the necessity of presenting two different diagrams. Note in Figure 3-78, however, that if 6D3 is true, it blocks the gate comparing RTMP1 and enables the gate comparing 7SR1. If 6D3 is false, the opposite gate is enabled. In other words, SAD will check for a single address character in 6 decimal when the line routing is PLSR. It will check for single address codes in all other decimal settings when the line routing is PLPL.

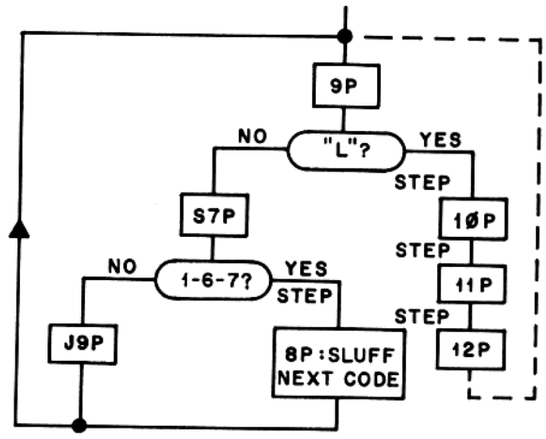
Refer to Figure 3-78 again briefly and note that whenever SAD is set, it prevents J9P3 from being produced by two different gates. TWO gates are necessary for different types of searches: the top gate prevents a jump to 9 phase when an "H" is sensed during normal sequence search, and the bottom gate prevents the jump to 9 phase when an "H" is sensed during marker search. Both of these items will be covered in greater detail shortly. Note now, however, that SAD blocks J9P3 in both cases.

(c) 12-9 Jump vs. 12-13 Step. Addresses are compared in 12 phase. If sequence "D" is being sought, each time an "H" is sensed, the phase counter jumps to 12 phase in order to compare the next code off the delay line to the "D" entered via the 2205. If the next code compares, the phase counter steps to 13 phase; if the two codes do not compare the phase counter jumps back to 9 phase to continue the search for "HD".

Resume titles are also compared in 12 phase, and translate sequence titles are compared in 10, 11, and 12 phases.

The logic element used to perform the compare function is the COMpare flip-flop, COM. The way in which it compares codes varies depending upon the type of search being conducted; this explanation will be covered as the individual types of searching are studied.

d. FLOW CHART. The basic search patterns are illustrated in the Flow Chart in Figure 3-80. Note that the information contained in this chart is very basic, and that there are exceptions not shown on the chart. It is presented here only to summarize the basic patterns: the information to follow will explain the exceptions.



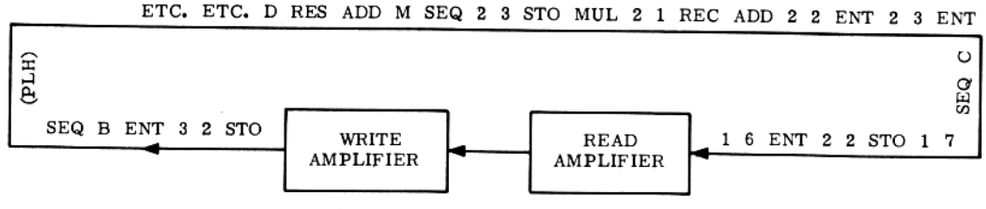
FLOW CHART
BASIC SEARCH PATTERNS

FIGURE 3-80

5. NORMAL SEQUENCE SEARCH

After the program is loaded and the 5610 is ready to operate, it waits for the operator to select the sequence to be performed. The operator does this by striking the 2205 key (or reading a code in the reader) corresponding to the title of the sequence to be performed. Once the sequence title is selected, a search of the program line is conducted to locate the sequence. When the desired sequence is found, a marker is written onto the line to "mark" the spot. This process is referred to as normal sequence search.

a. BASIC OPERATION. While the processor is waiting for the sequence title entry, the line routing is PLPL, as shown in Figure 3-81 (a).



BLOCK DIAGRAM
NORMAL SEQUENCE SEARCH

FIGURE 3-81 (a)

When the operator selects a sequence, the code representing the sequence title is set into the common register in parallel, as is done during program load. Almost immediately the parity check occurs (CRCR) while the line routing remains PLPL. See Figure 3-81 (b).

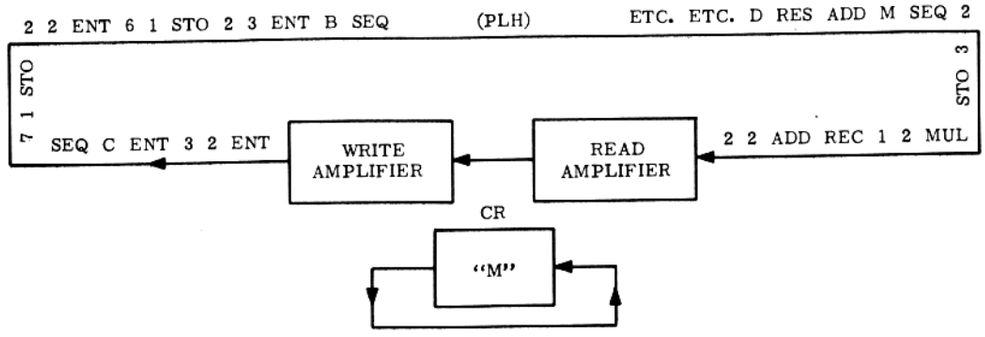


FIGURE 3-81 (b)

Next the line routing changes slightly to PLPL, PLOR. The output of the read amplifier is shifted into the operation register. As each code is shifted in, the previous code is shifted out and lost. Meanwhile, the program is maintained on the delay line by virtue of the PLPL line routing. See Figure 3-81 (c).

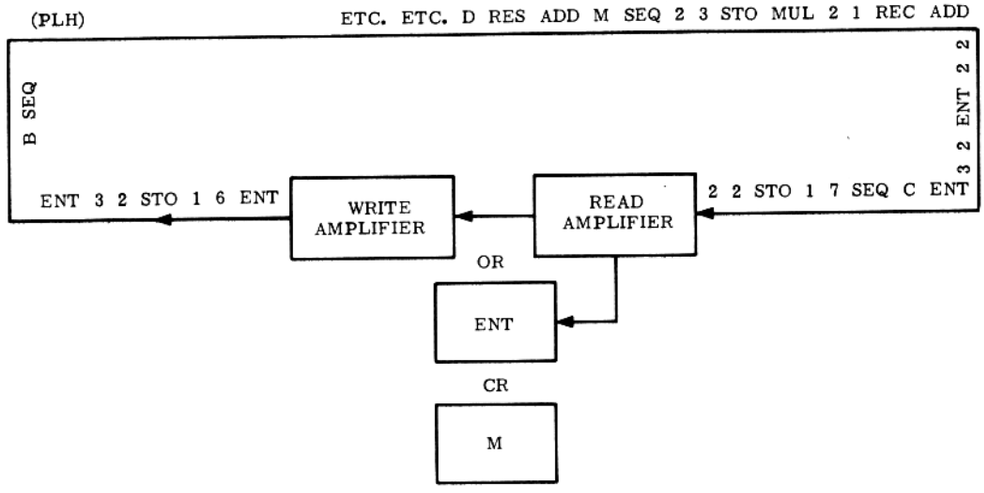


FIGURE 3-81 (c)

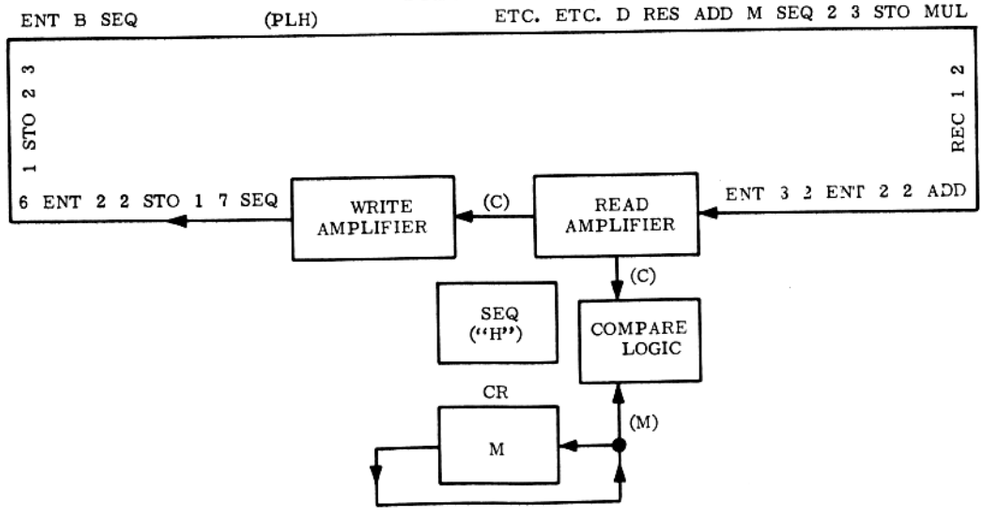


FIGURE 3-81 (d)

This line routing remains until a normal sequence separator, "H", is detected in the operation register. As soon as this occurs, the line routing changes to PLPL, CRCR, so that the next code off the line can be compared to the code in the common register. The ring shift (CRCR) is necessary for this check in order to retain the code in the common register for subsequent compares, should this one fail. This is shown in Figure 3-81 (d).

If the two codes are different, the PLPL, PLOR line routing resumes, searching for the next normal sequence separator, as shown in Figure 3-81 (e).

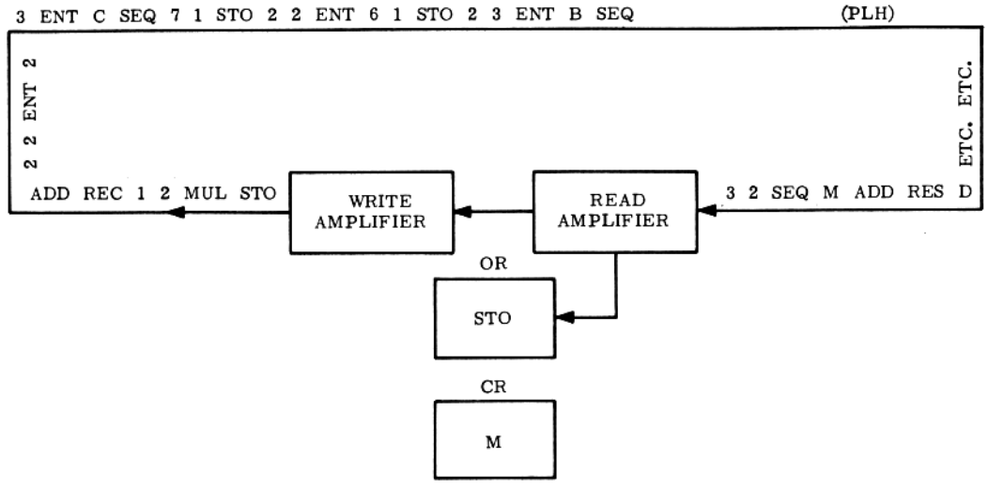


FIGURE 3-81 (e)

Each sequence separator, "H", detected in the operation register causes a compare to occur during the next character time. When the proper sequence is eventually located (Figure 3-81 (f)), a marker is immediately set into the search register.

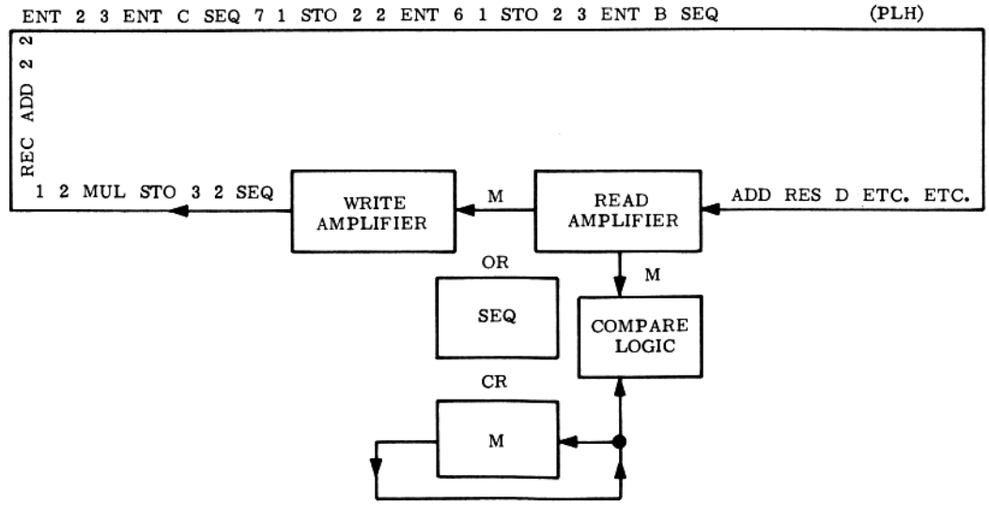


FIGURE 3-81 (f)

TP-161SL

The line routing changes to PLSR, SRPL, to write the marker out onto the line, as shown in Figure 3-81 (g). PLOR is also true at this time for another operation. This will be covered in Section 3K, Conditional Branch.

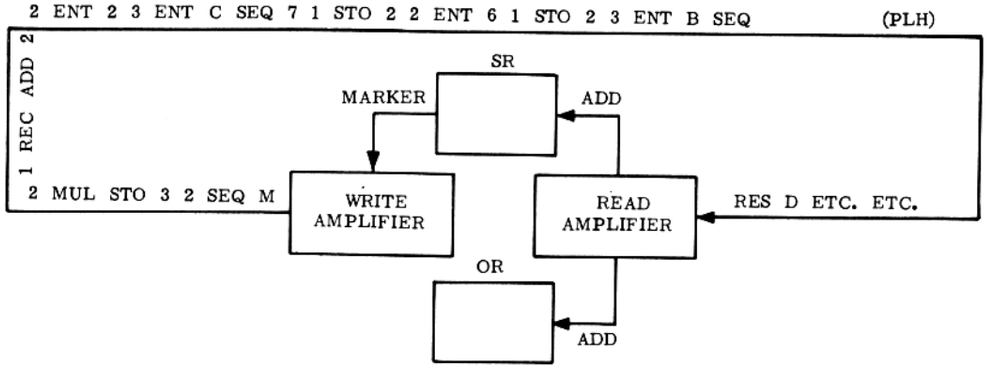


FIGURE 3-81 (g)

Once the marker is written out, the read amplifier output must be routed through the search register (PLSR, SRPL) in order to "open up" the program line to make room for the marker. Note in Figure 3-81 (h) that the marker is written out immediately in front of the first command in the sequence. After program line home is reached, a search for the marker is initiated. This will be covered shortly.

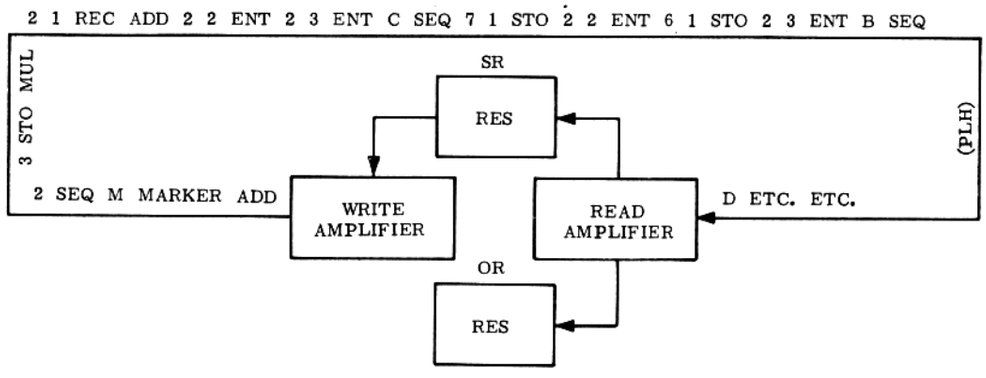


FIGURE 3-81 (h)

When the marker is detected, the command immediately following it is retrieved and the marker is rewritten onto the program line BEHIND this command. Each instruction in the sequence is performed in order. After the last instruction is performed, the marker is destroyed, and the processor returns to 3M0,1 to wait for the entry of the next sequence title.

b. PHASE COUNTER ACTION. Normal sequence search begins in 3 mode, 0 decimal, 1 phase, with the entry of the desired sequence title into the common register via the 2205. After the entry, the parity check is performed in 2 phase. If there is no parity error, a step to 3 phase occurs. The phase counter steps to 4 phase at program line home, and to 5 phase at the end of the "buffer". During 5 phase, the code in the common register is written onto the program line in the first "blank". This is necessary for use later during conditional branch. The code is also retained in the common register for subsequent use in the compare. The phase counter then steps to 6 phase for the second blank, and at the end of the second blank, as always, it jumps from 6 phase to 9 phase.

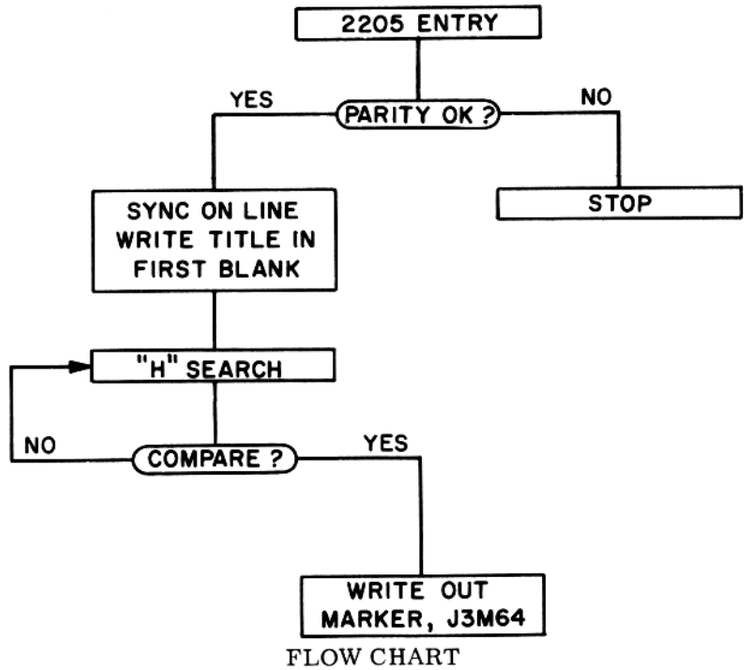
In 9 phase the actual search begins. Each code is sensed, and when an "H" is detected, a jump to 12 phase occurs so that the next code off the line can be compared to the code in the common register. If the codes do not compare, the phase counter jumps back to 9 phase and the search for the next "H" begins. When codes DO compare in 12 phase, a step to 13 phase occurs, where a marker is written onto the line. At program line home a jump to 3M6,4 occurs and marker search begins.

During the actual search, starting in 9 phase, each "L" that is detected will cause the phase counter to step through 10, 11, and 12 phases to sluff the 3-character translate sequence title. All other codes cause a jump from 9 phase to 7 phase. Each code sensed that contains the 1-6-7 bits causes a step to 8 phase, where the address or title character is sluffed. All other codes cause a jump from 7 phase back to 9 phase.

14 phase is not normally used. When an invalid sequence is selected and the entire program line is searched without a successful compare, 14 phase is used to clear the common register and initiate the search for the "error" sequence.

15 phase is used only after a parity error is detected in 2 phase. In this case, 15 phase remains true until the operator depresses the Reset switch. This causes a jump back to 1 phase to wait for another entry.

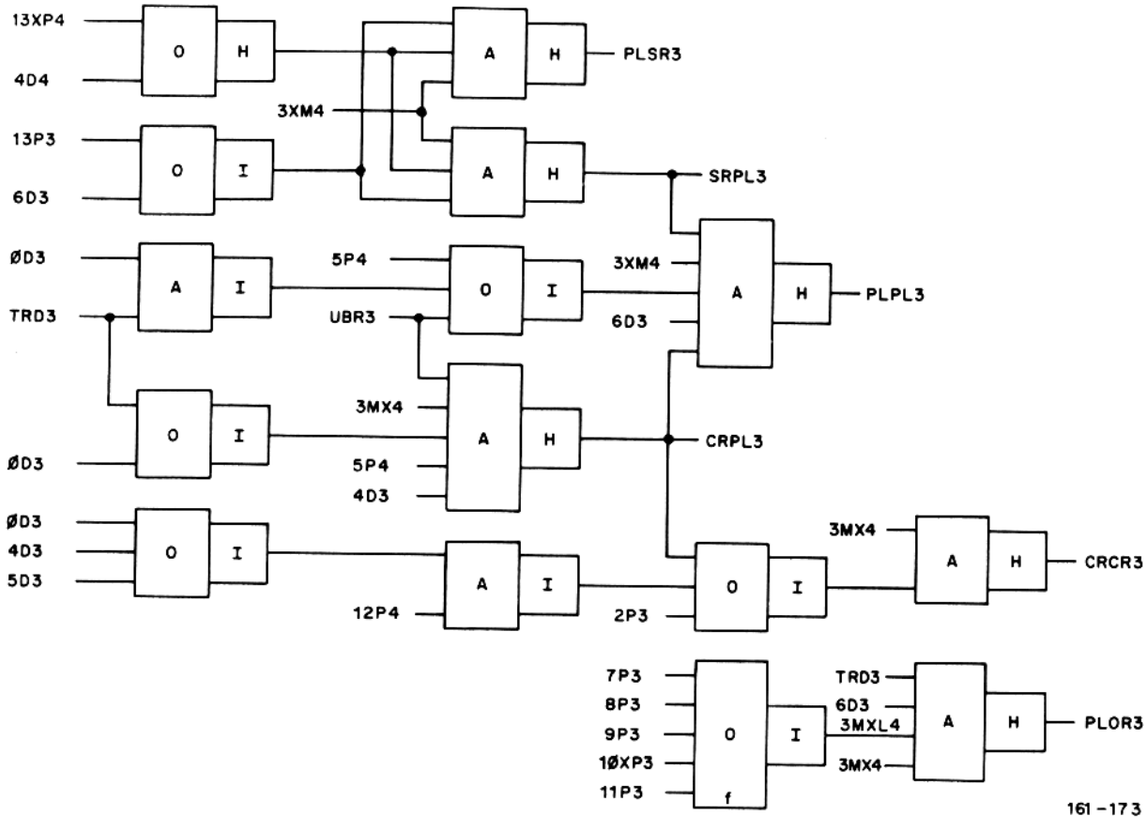
The simplified Flow Chart in Figure 3-82 summarizes the basic operation of the normal sequence search.



FLOW CHART
NORMAL SEQUENCE SEARCH

FIGURE 3-82

c. LINE ROUTINGS. All of the 3 mode 0 decimal line routings are pictured in Figure 3-83. During phases 1 through 4 no data is changing on the program line, so the line routing remains PLPL. The parity check in 2 phase requires a common register ring shift, CRCR3, in addition.



LINE ROUTINGS IN 3 MODE 0 DECIMAL
(All on "C" card unless otherwise noted)

FIGURE 3-83

CRPL3 is developed in 5 phase to write the code in the common register onto the program line. It also develops CRCR3 in order to retain this code in the common register. At the same time, CRPL3 inhibits PLPL3, thus blocking the program line output and effectively destroying the code (if any) previously located in the first blank. The line routing returns to PLPL in 6 phase.

PLPL3 remains true during the actual search, in phases 7 through 11. At this time PLOR3 is also developed so that the "Op Decoder" outputs may be utilized when needed.

CRCR3 is developed again in 12 phase to perform the compare.

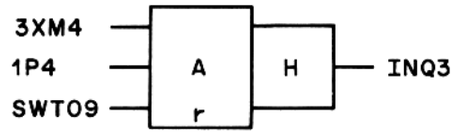
13 phase is the only phase other than 5 phase during which PLPL3 is not true. At this time PLSR3 and SRPL3 are developed in order to write out the marker and to delay the balance of the program one character time (to make room for the marker). PLOR3 also occurs at this time. This is necessary only during conditional branch, but it is allowed in 0 decimal because it does no harm.

The list below summarizes the different line routings used in 0 decimal.

PHASE	LINE ROUTING
1	PLPL
2	PLPL, CRCR
3, 4	PLPL
5	CRPL, CRCR
6	PLPL
7 - 11	PLPL, PLOR
12	PLPL, CRCR
13	PLSR, SRPL, PLOR
14, 15	PLPL

d. PRELIMINARY PHASES. While phases 7 through 12 can occur any number of times after the actual search begins, the first six phases occur only once. These preliminary phases cover the sequence title entry, parity check, and synchronization with the program line.

In 3M0,1 the processor is waiting for the entry to be made on the 2205. INput Qualification, INQ3, is developed as shown in Figure 3-84. When the entry is made, the INput Signal flip-flop, INS, becomes set, and INS2 enables the step to 2 phase. (Refer to Section 3F, Program Load, and Section 8, Electronic Circuits, for more information on INS.)



DEVELOPMENT OF INQ3 IN 3M0,1

FIGURE 3-84

The parity check occurs in 2 phase. If an even-bit code was entered, a jump to 15 phase occurs: this will be covered later in this section. If no parity error is detected, a step to 3 phase occurs.

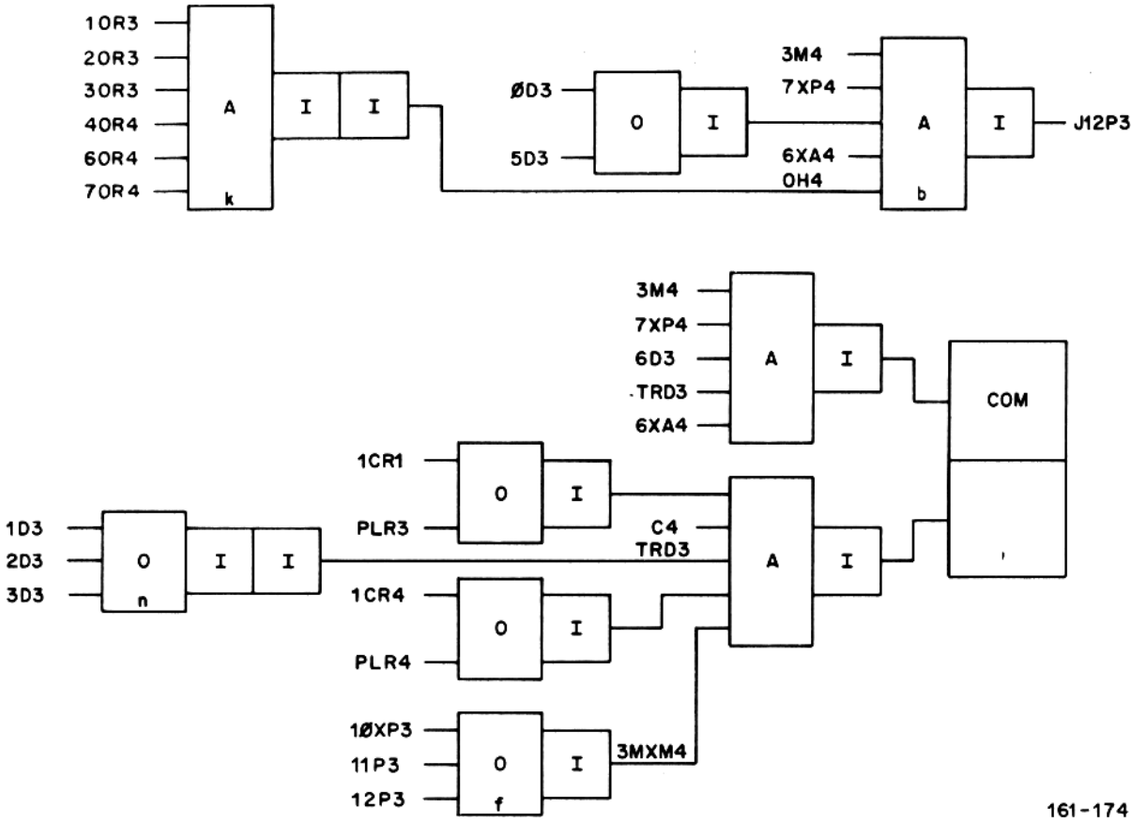
As noted earlier, 3 phase remains until program line home, at which time 4 phase becomes true. The phase counter cannot step to 5 phase until the timing chain restarts after the sync pulse is sensed. Therefore, 4 phase remains true through program line home and until the end of the "buffer".

5 phase becomes true at the beginning of the first "blank". CRPL causes the code in the common register to be written into this blank, and CRCR causes it to be retained in the common register. The reason for writing this code into the blank will become evident later when conditional branch is studied. It is retained in the common register so that it can be used later in comparing sequence titles.

At the end of the first "blank" (which now contains the title of the sequence being sought), the phase counter steps to 6 phase. At the end of the second blank the phase counter jumps from 6 phase to 9 phase, where the actual search begins.

e. THE SEARCH. The basic search patterns operate in phases 7 through 11. Every code other than an "L" causes an immediate jump from 9 phase to 7 phase. In 7 phase any code not containing the 1-6-7 bits (except the "H") causes a jump back to 9 phase. Codes with the 1-6-7 bits cause a step to 8 phase where the next code is bypassed.

(1) Compare Logic. As noted earlier, the output of the program line is shifted through the operation register during the search phases. Whenever an "H" is shifted into the operation register, OH4 becomes false and enables a jump to 12 phase. At the same time (T6A) the COM flip-flop becomes set. See Figure 3-85.

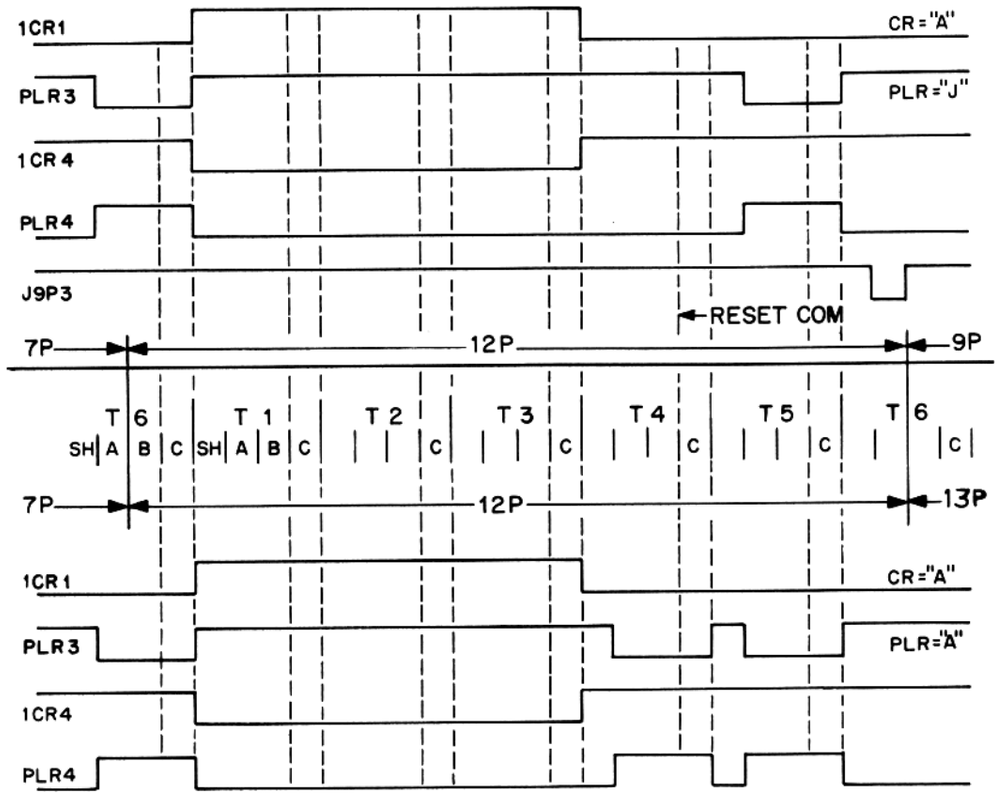


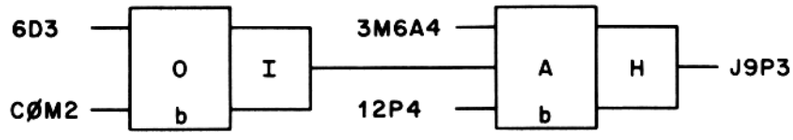
COMPARE LOGIC
(All on "B" card unless otherwise noted)

FIGURE 3-85

In 12 phase a ring shift (CRCR) is performed, and the 1CR output is compared to the program line output. This is also shown in Figure 3-85. If the two codes are the same, COM remains set; if they do not compare, COM resets. (TRanslate Decimal, TRD3, disables the reset of COM through this gate during 1, 2, and 3 decimal.)

An example of each is illustrated in Figure 3-86. In the top drawing, an "A" in the common register is compared to a "J" on the program line. COM resets at T4C time and J9P3 is developed at T6A. In the bottom half of Figure 3-86 both the common register and the delay line contain an "A", so COM remains set and the step to 13 phase occurs.



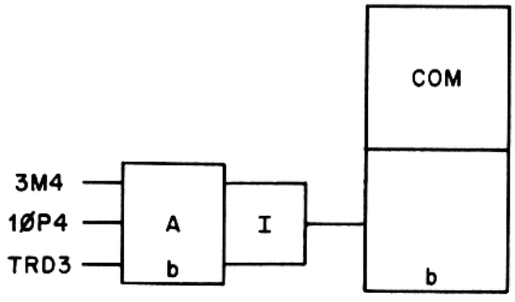


TIMING CHART
COMPARE IN 12 PHASE

FIGURE 3-86

(2) Translate Sequence Separator. If an "L" is sensed in 9 phase, it prevents the jump to 7 phase, and a step to 10 phase occurs. The three-character translate sequence title is then bypassed in 10, 11, and 12 phases.

The COM flip-flop is set at the end of 9 phase (T6A), just before the step to 10 phase. This is necessary during translate search, and will be covered later. At this time, however, COM cannot remain set: if it is set at the end of 12 phase it blocks the jump back to 9 phase. For this reason, COM is automatically reset whenever 10 phase is reached, except during translate search. See Figure 3-87.

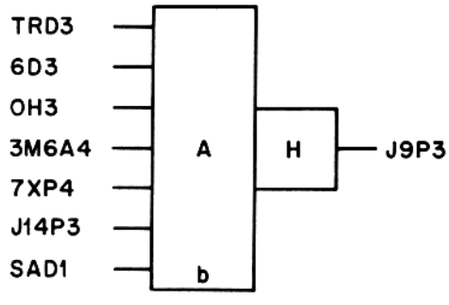


RESET OF COM IN 10P

FIGURE 3-87

The phase counter steps through 10, 11, and 12 phases as the translate sequence title is sensed and rewritten onto the line. At the end of 12 phase a jump back to 9 phase occurs as shown previously in Figure 3-86.

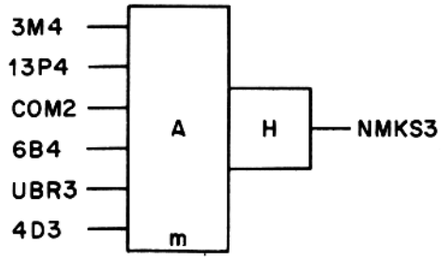
(3) Other Codes. As explained previously, all codes other than an "L" cause an immediate jump from 9 phase to 7 phase, and all codes sensed in 7 phase that do not contain the 1-6-7 bits (with the exception of the "H") cause another jump back to 9 phase. With an "H" in the operation register, OH3 blocks this jump to 9 phase, as shown in Figure 3-88. The SAD flip-flop detects all single address codes and causes their addresses to be sluffed in 8 phase.



OH3 BLOCKING J9P3

FIGURE 3-88

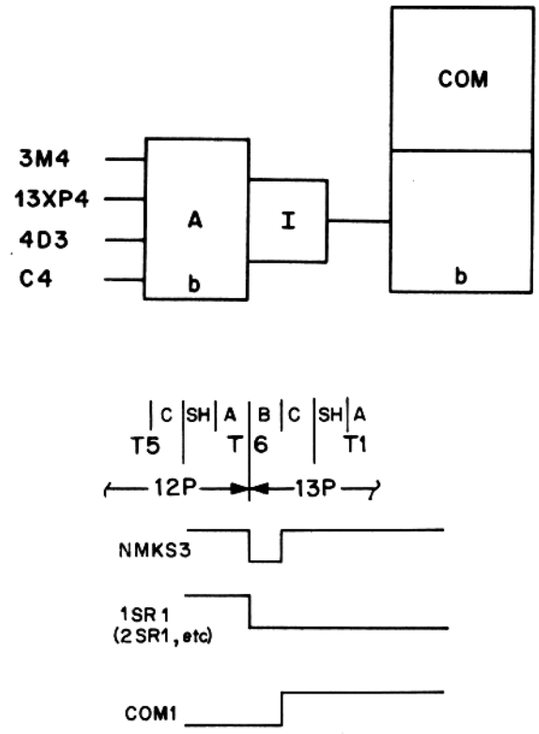
f. AFTER COMPARE. After a successful compare has occurred, the phase counter steps from 12 phase to 13 phase where a marker is immediately manufactured and written out onto the line. The line routing changes to PLSR, SRPL for the duration of 13 phase. NMKS3 sets all six SR flip-flops, and the marker is shifted out of the search register onto the program line.



DEVELOPMENT OF NMKS3 IN 3M13P

FIGURE 3-89

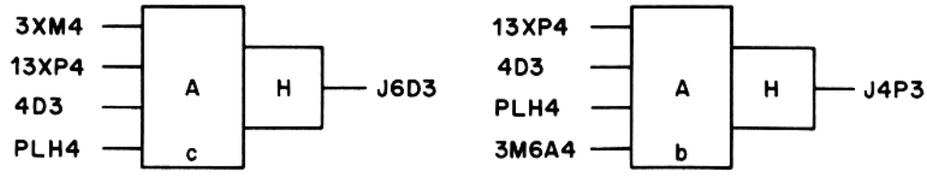
Immediately after NMKS3 is produced, COM is reset by the gate shown in Figure 3-90. This is necessary to prevent subsequent true signals on NMKS3 as 13 phase progresses. Representative waveforms are illustrated.



TIMING CHART
SUCCESSFUL COMPARE

FIGURE 3-90

13 phase remains true until program line home, at which time J6D3 and J4P3 are developed. See Figure 3-91. This condition, 3 mode, 6 decimal, and 4 phase (3M6,4) is the beginning of normal marker search.



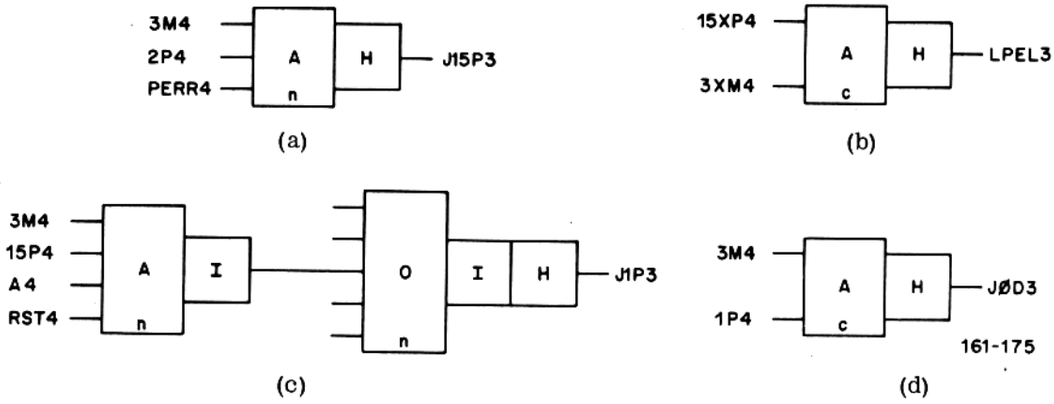
DEVELOPMENT OF J5D3 & J4P3 in 3M13P

FIGURE 3-91

g. ERROR CONDITIONS. There are two types of error conditions for which the 5610 is equipped to take corrective action. These are parity errors upon entry of the sequence title, and selection of a sequence title that is not loaded on the program line.

(1) Parity Error. When a parity error is detected in 2 phase, a jump to 15 phase is enabled. Once in 15 phase, the Light Parity Error Lamp signal, LPEL3, goes true and causes the error lamp to be turned on. See Figure 3-92 (a) and (b).

The phase counter remains in 15 phase until the Reset switch is operated. This sets the RST flip-flop and causes a jump to 1 phase (Figure 3-92 (c)). An insurance circuit (Figure 3-92 (d)) provides 0 decimal whenever 3M1 is reached. J1P3 also sets the CRR flip-flop, which resets the erroneous code out of the common register. The processor then waits in 3M0,1 for another sequence entry. See Figure 3-92.

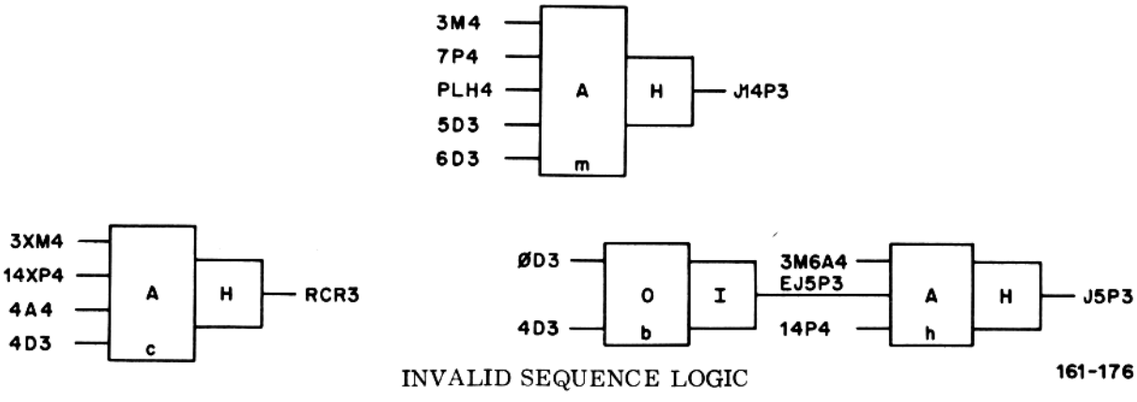


PARITY ERROR LOGIC

FIGURE 3-92

(2) Invalid Sequence Entry. When the entire program line has been searched without making a successful compare, the sequence title entered via the 2205 is determined to be invalid. In this case the processor searches out the error sequence (sequence "Space") and performs the instructions contained therein.

Refer to Figure 3-93. Just before program line home is reached in the search for the invalid sequence, the PLR flip-flop is sensing the absence of data on the program line, and the phase counter is alternating between 9 phase and 7 phase. When PLH sets, a jump to 14 phase is enabled, and 14 phase becomes true at the T6B during program line home.



INVALID SEQUENCE LOGIC

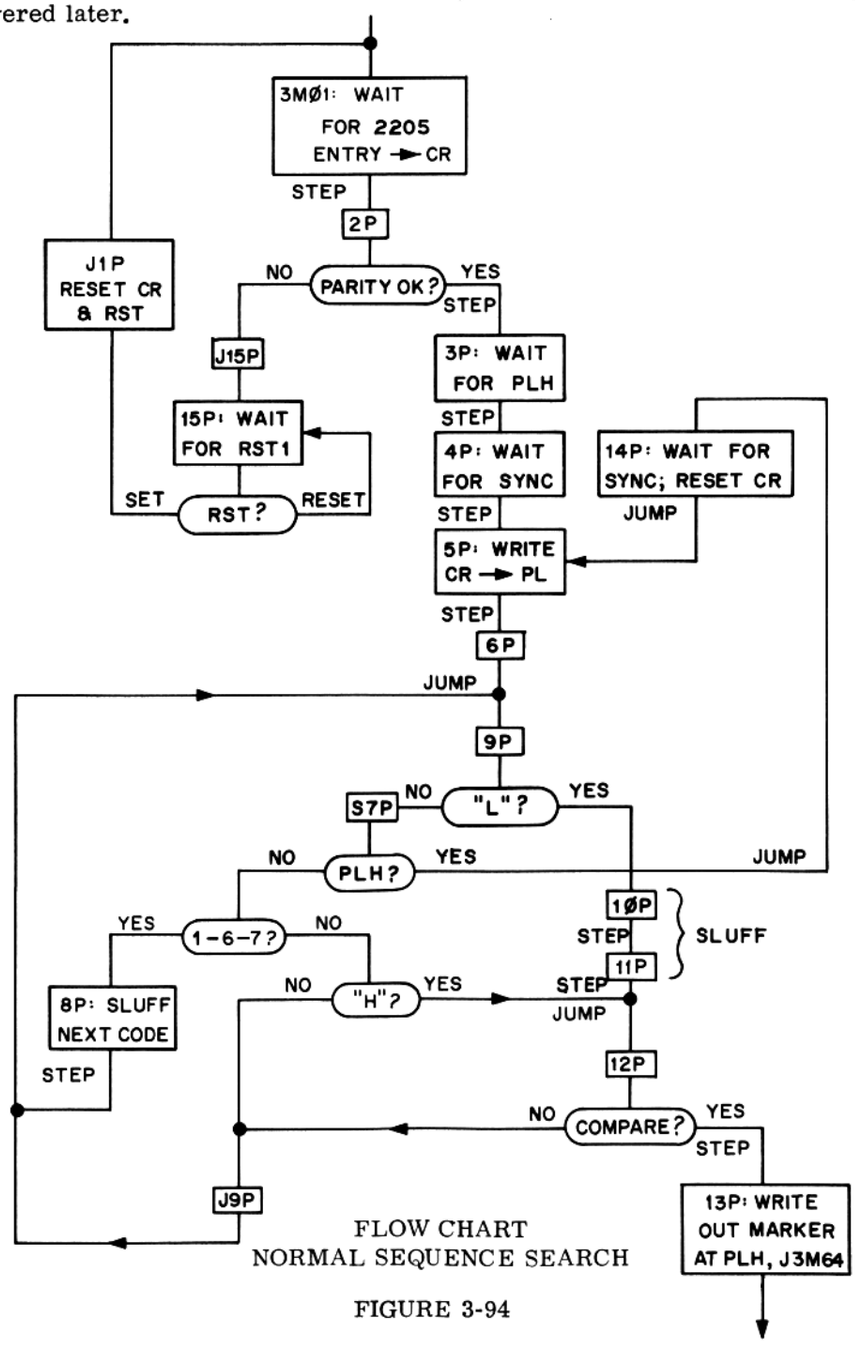
FIGURE 3-93

When the timing chain restarts after the sync pulse is sensed, RCR3 is developed at T4A time to set the CRR flip-flop and clear the common register. At the following T6A time, J5P3 is developed, and at the end of the buffer, the phase counter jumps to 5 phase.

The balance of the operation is the same as before. Note, however, that with the common register now clear, a space (blank) is written into the first blank on the program line, and the search is conducted for sequence "Space".

It is imperative that every program contain an error sequence. If an invalid sequence title is entered and the program contains no error sequence, the processor will never come out of the search for the "H-Space" and it will be necessary for the operator to turn off the 5610 to get out of this condition.

h. FLOW CHART. All conditions possible during normal sequence search are covered in the Flow Chart in Figure 3-94. Do not use this chart for sequence search during unconditional branch, for there are some differences caused by the set condition of the UBR flip-flop. These will be covered later.



FLOW CHART
NORMAL SEQUENCE SEARCH

FIGURE 3-94

TP-161SL

6. MARKER SEARCH

After the program is loaded and a sequence is chosen and located, a marker is written onto the program line after the sequence address and before the first command. At the next program line home, a jump to 3 mode, 6 decimal, and 4 phase occurs. This 3M6,4 setting is the beginning of marker search. After the marker is located, a jump to 4 mode occurs so that the instruction following the marker may be retrieved and finally executed. The rewriting (or the destroying) of the marker occurs in 4 mode, and will be explained later.

Marker search also occurs (the 3M6,4 setting becomes true) at program line home after each instruction is executed, so that the marker may again be located and the next instruction retrieved.

During unconditional branch, the normal marker is ignored and the branch marker is searched out instead.

a. GENERAL OPERATION. The 3M6,4 condition is true during program line home. The phase counter steps to 5 phase at the end of the buffer, and to 6 phase at the end of the first "blank". A jump to 9 phase occurs at the end of the second blank, and the actual search begins.

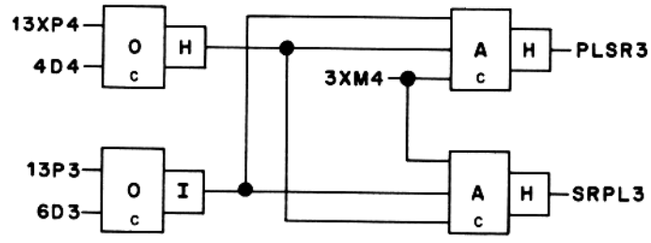
The basic search patterns operate during marker search. Sensing an "L" in 9 phase causes a step through 10, 11, and 12 phases to bypass the translate sequence title. Any code other than an "L" sensed in 9 phase causes an immediate jump to 7 phase.

In 7 phase any code containing the 1-6-7 bits allows the SAD flip-flop to remain set. This causes a step to 8 phase, where the next code (address) is bypassed.

There are two exceptions to the above: the decimal and the marker both contain the 1-6-7 bits and do not reset SAD. However, special gates cause a jump to 9 phase when the decimal is sensed, and to 4 mode when the marker is sensed.

The COM flip-flop is used during 9-7 phases to detect normal sequence separators (letter "H"). When an "H" is sensed, COM blocks the jump to 9 phase so that the sequence title can be bypassed in 8 phase.

b. LINE ROUTINGS. The only line routing used in 3 mode and 6 decimal is PLSR, SRPL, as shown in Figure 3-95. This routing changes only when the marker is located and the jump to 4 mode occurs.

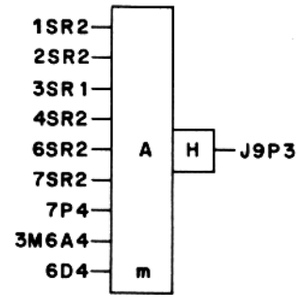


LINE ROUTINGS IN MARKER SEARCH

FIGURE 3-95

c. BASIC SEARCH PATTERNS. The basic search patterns function as previously explained with few exceptions. The phase counter operation from 4 phase and program line home to 9 phase after the second "blank" is identical to that of all other searches. Note, however, that because the line routing is PLSR, SRPL, no data is changed on the program line. The S7P3 logic also operates the same as during other searches.

The SAD flip-flop operates the same as during other searches, except that its effect on the phase counter is nullified when the decimal or the normal marker is sensed. The latter will be covered shortly. When the decimal code is sensed in 7 phase, SAD remains set, but the jump to 9 phase is enabled through a different gate, shown in Figure 3-96.

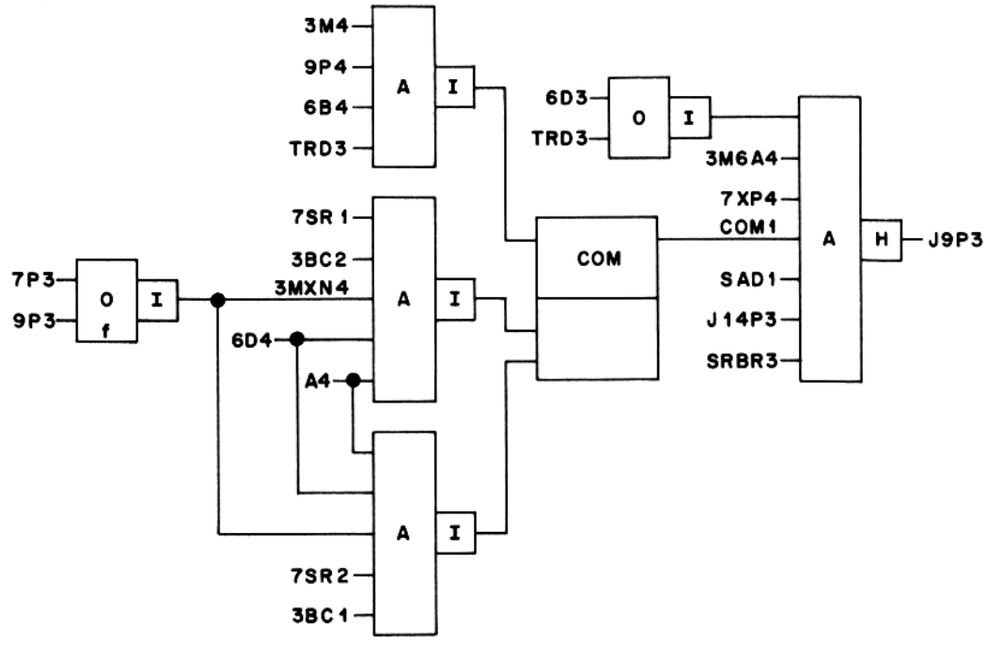


J9P3 CAUSED BY DECIMAL IN MARKER SEARCH

FIGURE 3-96

d. "COM" FLIP-FLOP OPERATION. During marker search the COMpare flip-flop checks for the presence of a normal sequence separator. This is the only time the COM flip-flop is used for a purpose other than comparing. Whenever a normal sequence separator is sensed, its address must be bypassed in 8 phase. Since the PLOR line routing is false at this time, the Op Decoder outputs cannot be used to sense the presence of the "H", so the COM flip-flop is used instead.

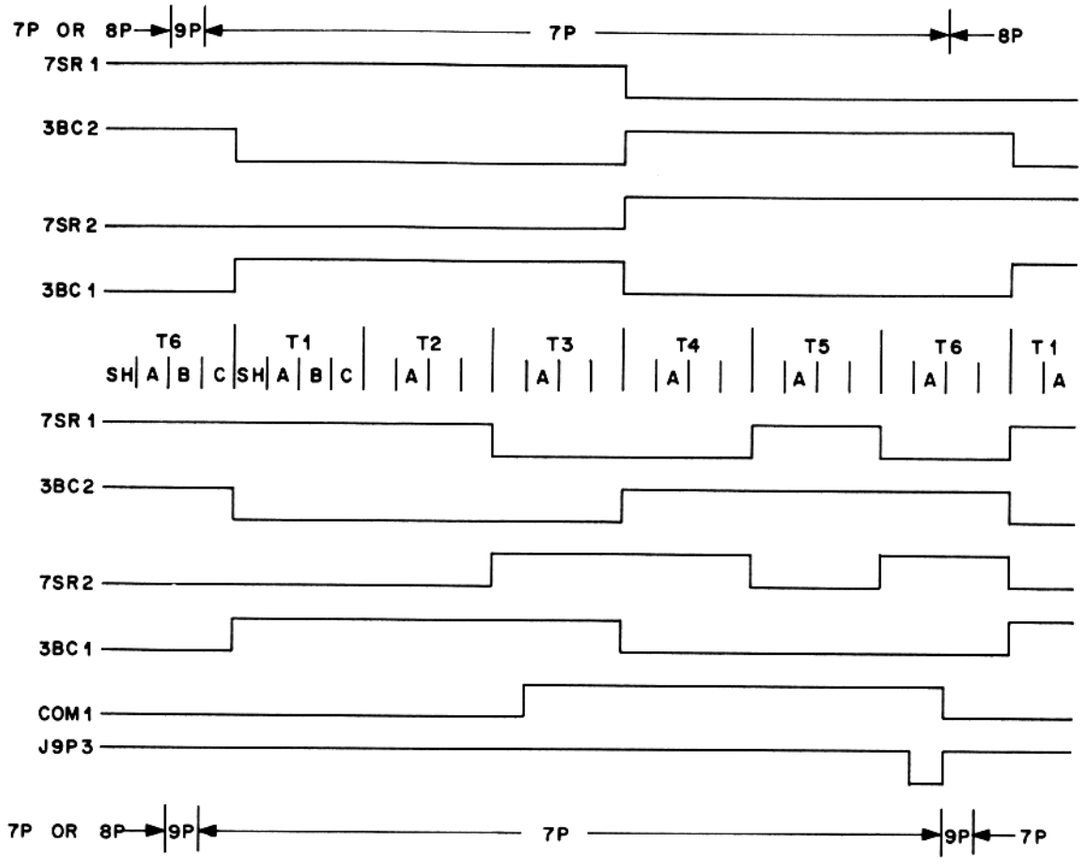
Figure 3-97 shows that COM is set each time 9 phase is reached in 3 mode. It is reset by any code other than an "H" that is shifted through the search register. If an "H" is sensed, COM remains set, blocking the jump to 9 phase and allowing a step to 8 phase, where the sequence title is bypassed. COM remains set during 8 phase and is set at the beginning of 9 phase to check the code following the sequence address. If some code other than an "H" is sensed, the jump to 9 phase (from 7 phase) occurs and a "new" search begins.



COM FLIP-FLOP OPERATION IN 3 MODE, 6 DECIMAL
(All gates on "B" card, unless otherwise noted)

FIGURE 3-97

Refer now to Figure 3-98. The top half illustrates the waveforms involved when an "H" is detected as it is being shifted into the search register. COM remains set and the phase counter steps to 8 phase to "sluff" the next code.



TIMING CHART
COM FLIP-FLOP ACTION DURING MARKER SEARCH

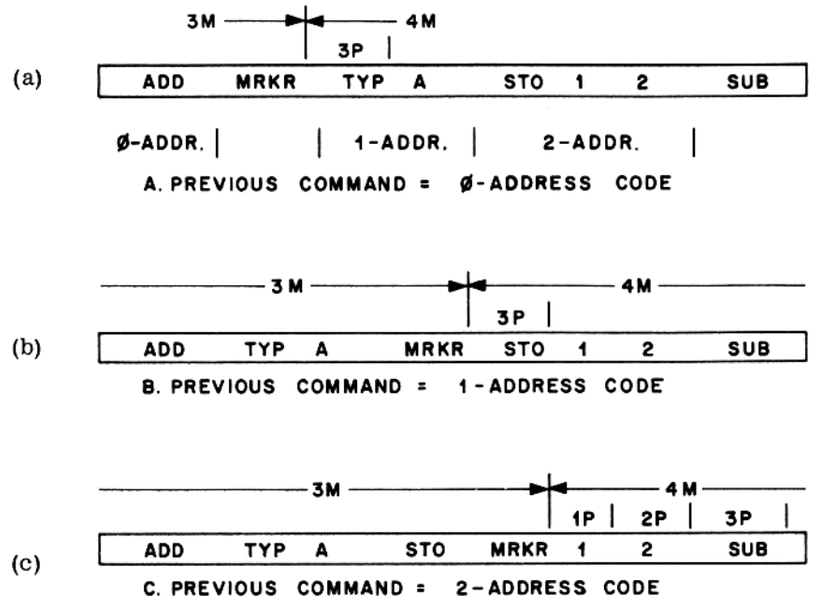
FIGURE 3-98

In the bottom part of Figure 3-98 a 3-4-7 code is sensed (represented by the 3-4-6 bits on the program line). COM becomes reset at T3 time, when the code being sensed first differs from an "H". Note that COM can reset during any T time, depending upon the code sensed. Regardless of when it resets, however, J9P3 does not occur until T6A time.

e. JUMP TO 4 MODE. The searching process just described, encompassing phases 7 through 12, continues until the normal marker is shifted into the search register. When this occurs, a jump to 4 mode is enabled.

The entry into 4 mode can be in either 1 phase or 3 phase; the choice is determined by the previous instruction performed (in the same program sequence). It will be seen later, in Section 3H, Retrieve, that if the last instruction performed had a 0- or 1-address command, the marker is placed immediately in front of the next command. In marker search, when the marker is found, a jump to 4 mode and 3 phase occurs, where the next command is retrieved. If, however, the previous instruction had a 2-address command, the marker is inserted between the command and its two address digits. In this case, when the marker is found, a jump to 4 mode, 1 phase occurs. The two address digits are then bypassed in 1 and 2 phases, and the

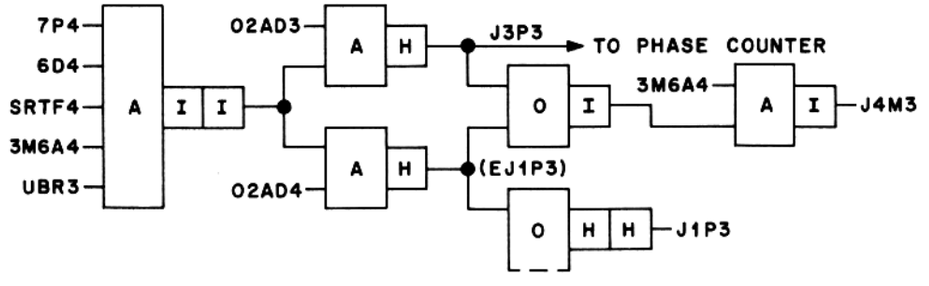
next command is sensed in 3 phase. In this way, all new commands to be retrieved are first sensed in 4 mode and 3 phase. This is illustrated in Figure 3-99. (Note in Figure 3-99 that the contents of the program line are shown as sensed at the read amplifier: the data sensed in Figure 3-99 (a) would be rewritten as shown in Figure 3-99 (b), etc.)



BLOCK DIAGRAM
JUMP TO 4 MODE

FIGURE 3-99

The gates used to develop the jump to 4 mode are illustrated in Figure 3-100. SRTF4 becomes false when all six search register flip-flops are set. The Operation 2 ADdress signal, O2AD, then determines whether the jump will be to 1 phase or 3 phase. In either case, the jump to 4 mode is enabled by causing J4M3 to become true.

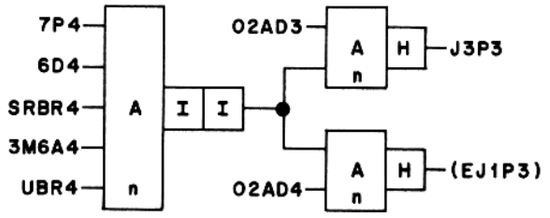


JUMP TO 4 MODE AFTER LOCATING MARKER
(All gates on "N" card unless otherwise noted)

FIGURE 3-100

Note that when the marker is sensed prior to executing the first instruction in a sequence, the operation register is clear. This is equal to a 0-address code, so when the marker is found, the jump occurs to 4 mode 3 phase.

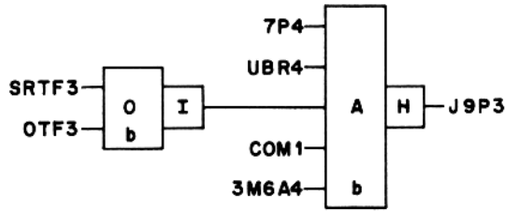
f. MARKER SEARCH DURING UNCONDITIONAL BRANCH. Whenever the Unconditional BRanch flip-flop, UBR, is set, the normal marker is ignored and the branch marker (2-4-7) is recognized instead. Figure 3-100 shows how UBR3 disables the jump to 4 mode when the normal marker is sensed during unconditional branch. Figure 3-101 illustrates a similar circuit, used when UBR is set, to produce J3P3 or Enable Jump to 1 Phase, EJ1P3, depending upon the previous command. The Search Register BRanch marker signal, SRBR4, becomes false when 2SR, 4SR, and 7SR are set.



JUMP TO 4 MODE AFTER LOCATING BRANCH MARKER

FIGURE 3-101

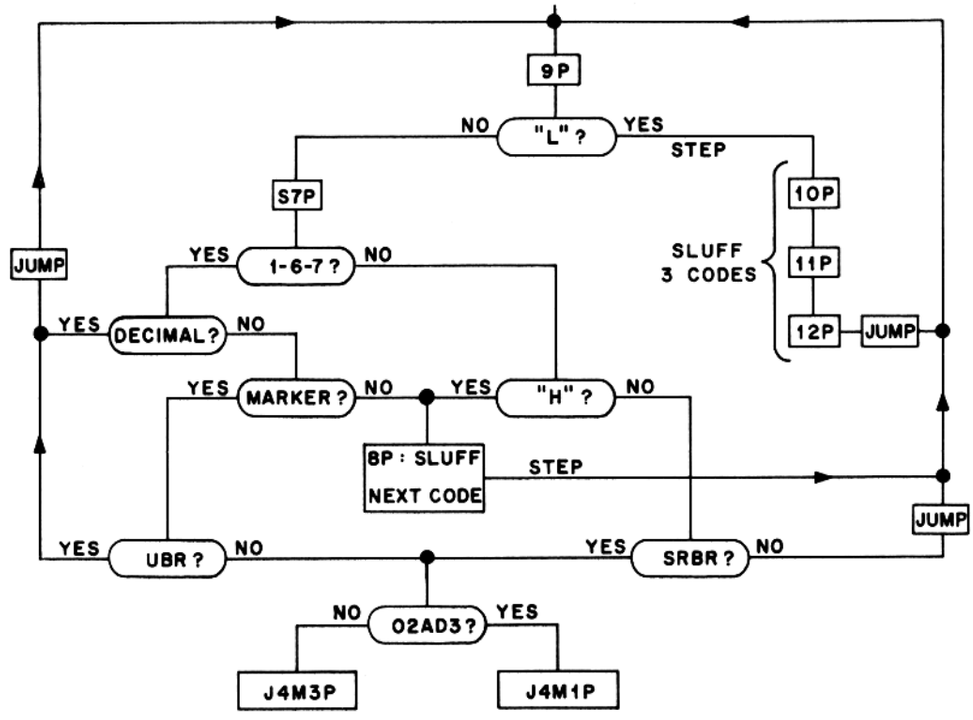
The UBR flip-flop has one additional function during marker search: when the normal marker is sensed, SAD does not reset, so a special gate is added to allow the jump back to 9 phase. This is pictured in Figure 3-102.



J9P3 CAUSED BY MARKER WITH UBR SET

FIGURE 3-102

g. FLOW CHART. Figure 3-103 summarizes the variations possible in marker search.



FLOW CHART
MARKER SEARCH

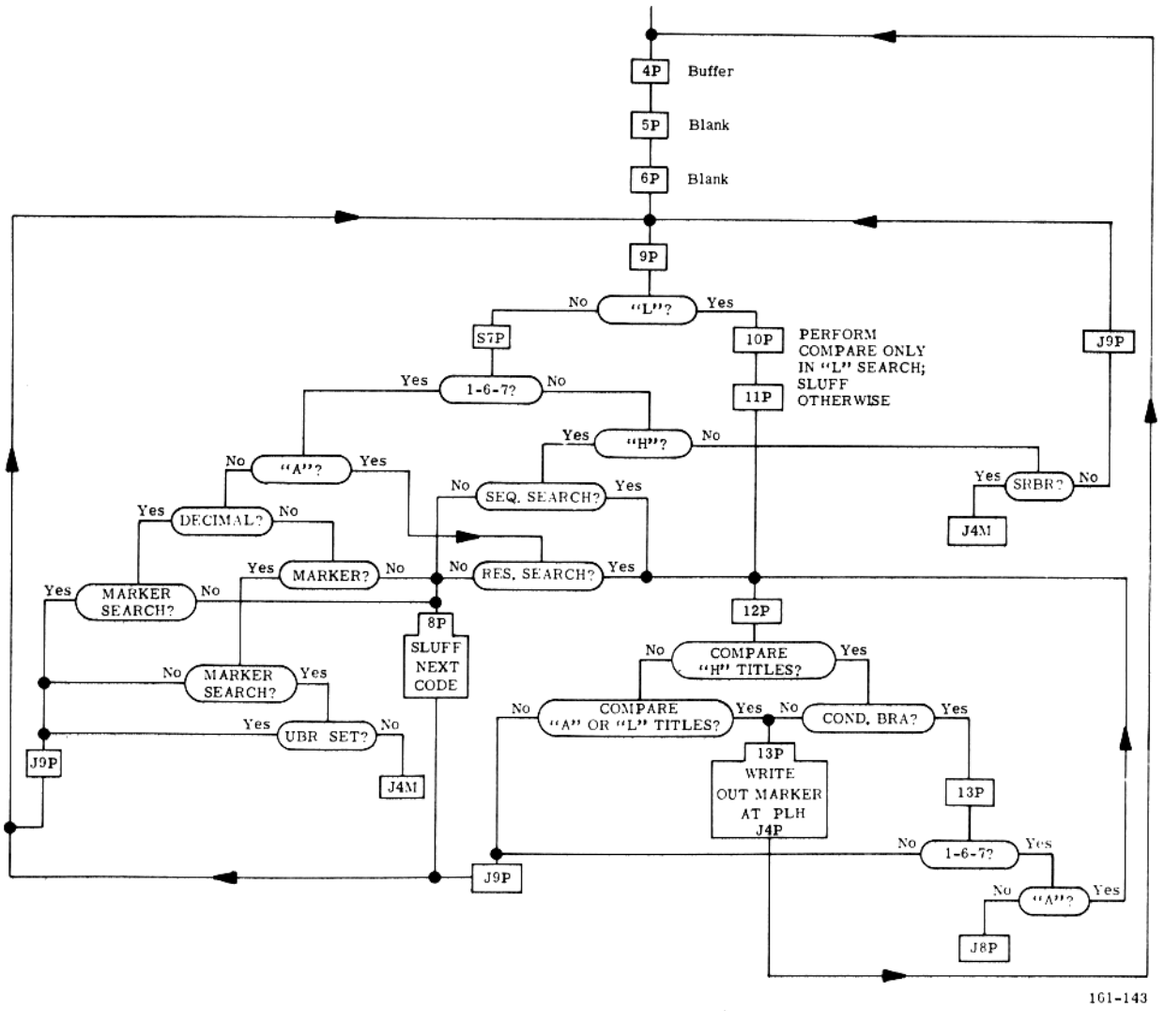
FIGURE 3-103

7. OTHER TYPES OF SEARCH

The remaining types of searches not already covered (Resume search, unconditional and conditional branch sequence search) will be discussed in Sections 3I, Unconditional Branch and 3K, Conditional Branch.

8. FLOW CHART

The Flow Chart in Figure 3-104 summarizes the complete 3 mode search process.



FLOW CHART
3 MODE SEARCH

FIGURE 3-104

H. 4 MODE, INSTRUCTION RETRIEVAL

The main purpose of 4 mode is to retrieve the next instruction to be performed (both the command AND its address) from the program line. It then develops the logic signals necessary for the eventual execution of the instruction (the "master" signals). 4 mode occurs after marker search locates the marker in 3 mode: this is the only way 4 mode can be realized. Usually, the logic jumps from 4 mode to 5 mode, but it can go to 3 mode or 6 mode under some circumstances.

After retrieval, the command portion of each instruction is stored in the operation register. The address portion may be stored in the common register, search register, and/or decimal register, depending upon the nature of the command.

Another important function of 4 mode is to rewrite the marker back onto the program line in the correct position, after the instruction has been retrieved. If circumstances call for the marker to be destroyed, 4 mode first decides this, then destroys the marker, and "closes up" the program line to compensate for removal of the marker.

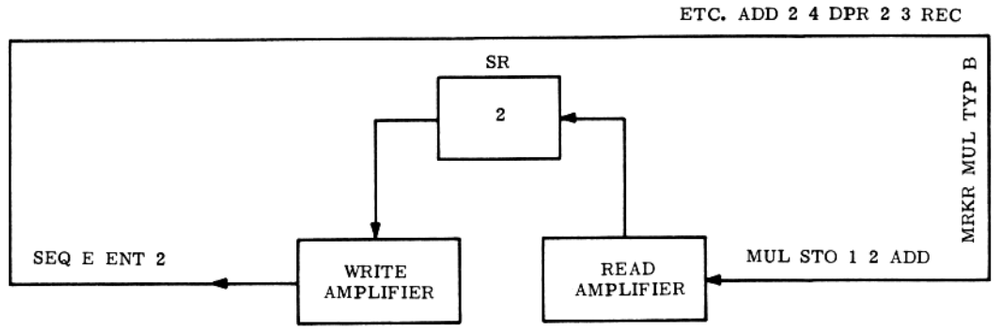
4 mode operation during branching and constants will be covered in more detail later.

1. GENERAL OPERATION

The point of entry into 4 mode (either 1 phase or 3 phase) is determined by the instruction previously performed. This was mentioned under "Marker Search", and will be more fully explained shortly. In any case, the command to be performed next is shifted into both the operation register and the common register in 3 phase. Certain "Op Decoder" outputs are immediately developed to determine where the logic will go from 3 phase; it may go either 4, 5, 6, 9, 10, 11, 12, or 13 phase. This path is determined by the new command in the operation register.

a. "REGULAR" INSTRUCTIONS. Most instructions follow one of three different patterns depending upon whether they contain a 0-, 1-, or 2-address command.

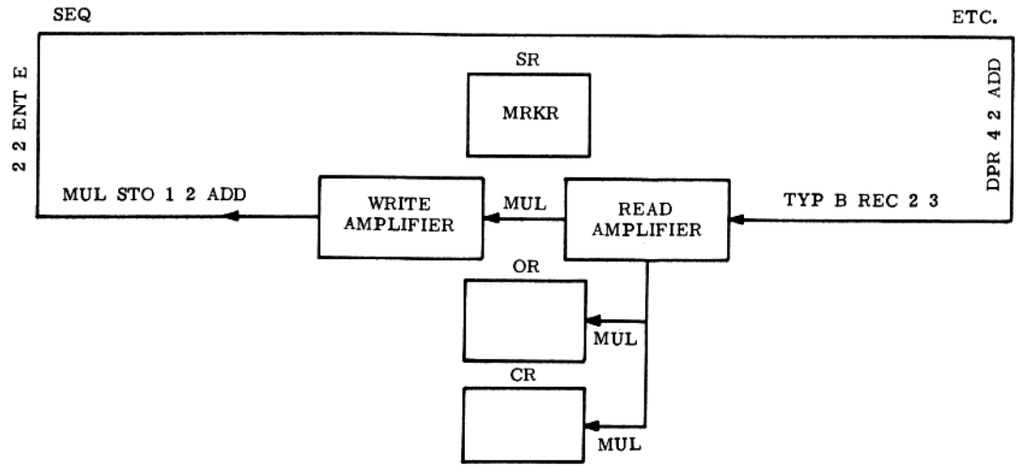
(1) 0-Address Commands. Assume that an Add instruction has just been completed and the 3 mode search for the normal marker is underway. The line routing is PLSR, SRPL: all data from the read amplifier is passing through the search register, as shown in Figure 3-105 (a).



MARKER SEARCH AFTER EXECUTING A 0-ADDRESS COMMAND

FIGURE 3-105 (a)

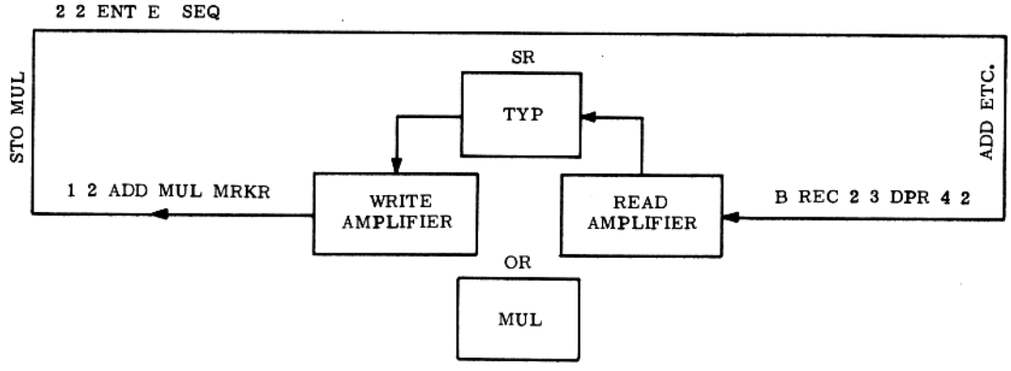
As soon as the marker is shifted into the search register, a jump to 4 mode 3 phase occurs, and the line routing changes to PLPL, PLOR, PLCR. The PLOR routing shifts the new command into the operation register, and PLPL writes it onto the program line. Since SRPL is not true at this time, the marker remains in the search register. See Figure 3-105 (b). (The PLCR routing is necessary only for Constants and some writing machine functions, but occurs in 3 phase regardless.)



LINE ROUTING, 4 MODE 3 PHASE; 0-ADDRESS COMMAND

FIGURE 3-105 (b)

When the new command is completely shifted into the operation register, it is recognized as a 0-address command. This causes a jump to 5 phase, where the line routing changes back to PLSR, SRPL, and the marker is written back onto the program line AFTER the new command. See Figure 3-105 (c). This line routing remains until program line home, so there is no further change in the contents of the program line.



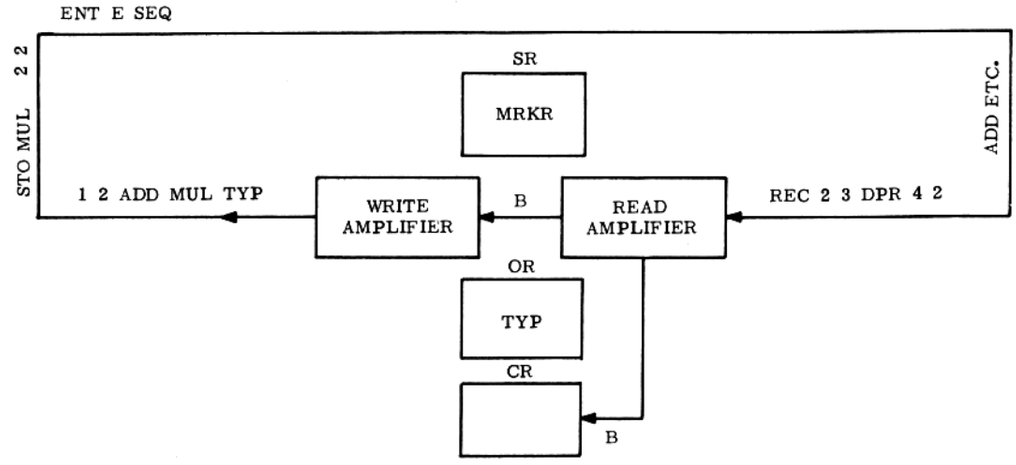
LINE ROUTING, 4 MODE 5 PHASE

FIGURE 3-105 (c)

The PLPL line routing in 3 phase is necessary to "close up" the program line where the marker was removed. The new command takes the place of the marker, and the marker takes the place of the new command. At program line home, a jump to 5 mode occurs and the Multiply instruction is executed.

(2) 1-Address Commands. Unconditional branch commands sensed while UBR is set, and all conditional branch commands cause different operation than do the other 1-address commands. These exceptions will be treated individually later.

All other 1-address commands sensed in 4 mode 3 phase inhibit the phase counter jumps and allow a step to 4 phase. In 4 phase the line routing remains PLPL, PLCR. The PLOR routing is dropped so as to maintain the command just retrieved in the operation register. See Figure 3-106 (a).

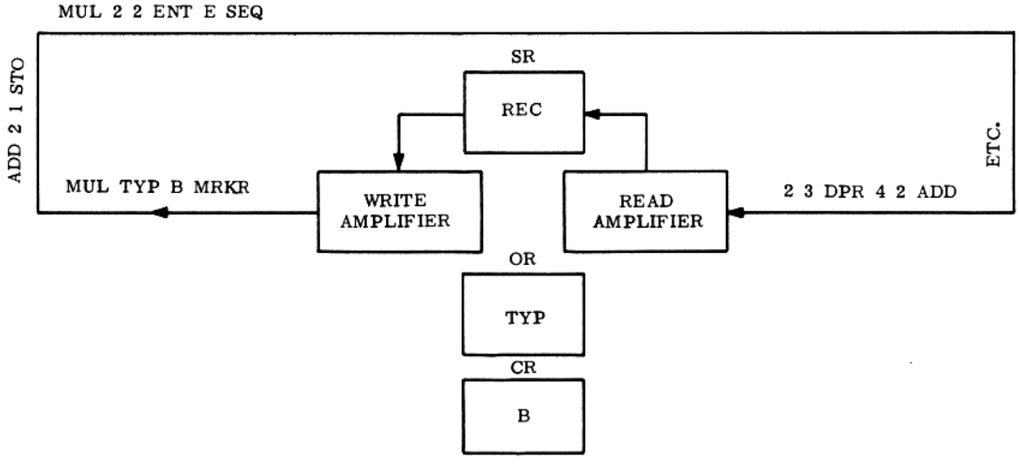


LINE ROUTING, 4 MODE 4 PHASE

FIGURE 3-106 (a)

The PLCR routing in 4 phase shifts the address into the common register. The PLPL routing maintains the address on the delay line.

After the address is shifted into the common register, a step to 5 phase occurs, and the line routing returns to PLSR, SRPL. This shifts the marker back onto the delay line, as shown in Figure 3-106 (b).



LINE ROUTING, 4 MODE 5 PHASE

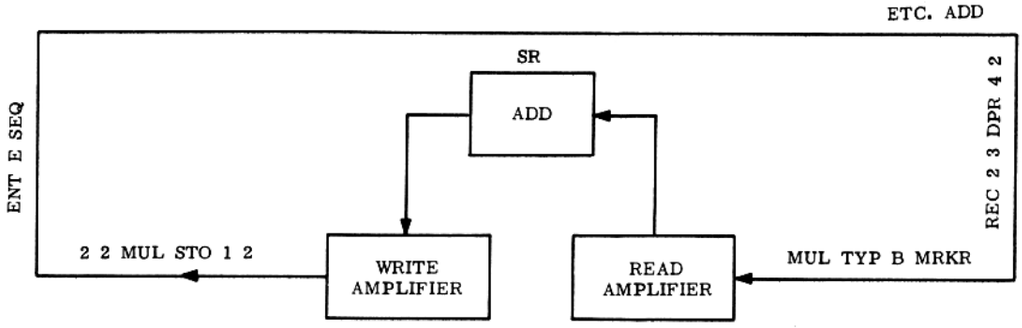
FIGURE 3-106 (b)

5 phase remains until program line home, so the balance of the program line data is routed through the search register and back onto the program line. Note that the marker has been shifted TWO character positions on the program line, and it is again located immediately in front of the next command.

At program line home, a jump from 4 mode to 5 mode again takes place, and the Type instruction is executed: the "B" is sent from the common register to the 2205. (During 5 mode the line routing is PLPL only.) When the Type instruction is complete, a jump to 3M6,4 occurs and another marker search is initiated to locate the next instruction.

(3) 2-Address Commands. The handling of the marker is somewhat unusual when a 2-address command is retrieved. The marker is inserted BETWEEN the command and its address digits, instead of AFTER the address digits. In this case, when the command is completed and the NEXT command is to be retrieved, special allowances must be made to bypass the two address digits after the marker is located. This was mentioned briefly in the Marker Search discussion, and will be covered in more detail here.

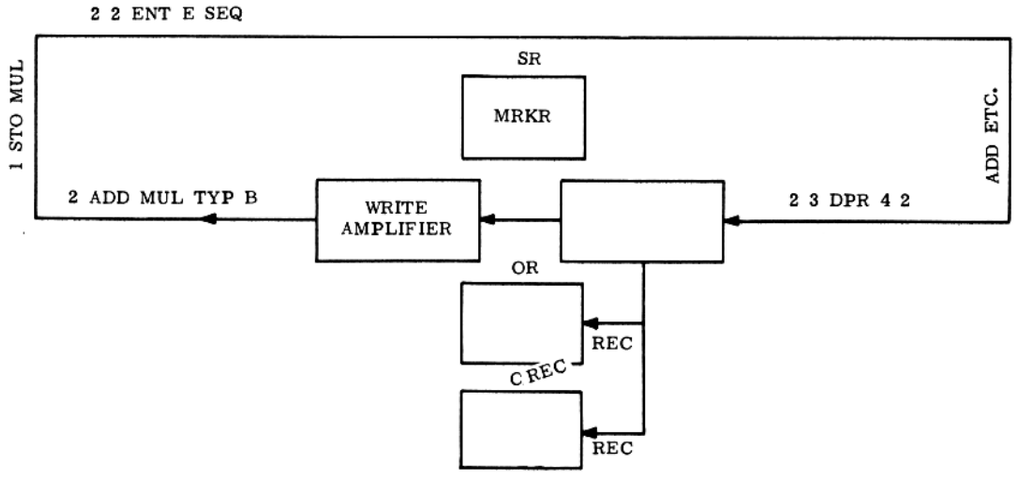
(a) Retrieval of 2-Address Command. Assume that a Type instruction has just been completed and marker search is underway, as shown in Figure 3-107 (a).



MARKER SEARCH AFTER EXECUTING A 1-ADDRESS COMMAND

FIGURE 3-107 (a)

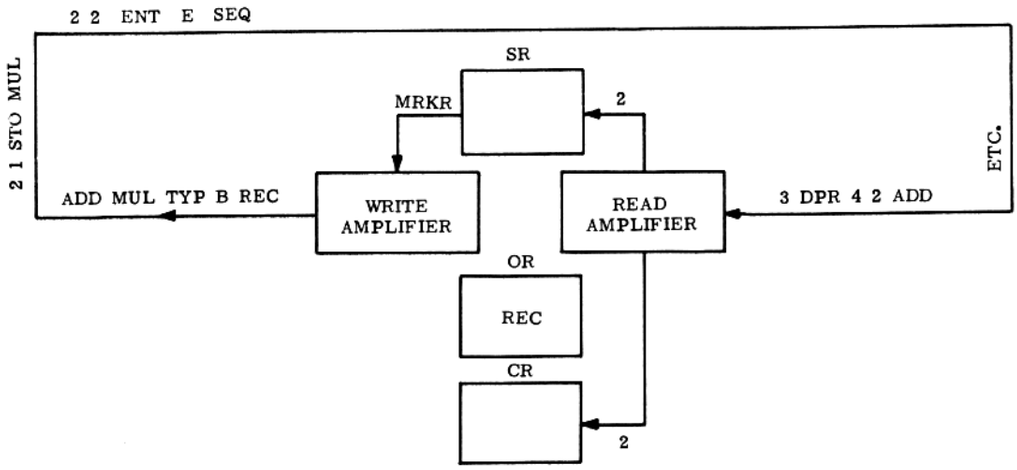
When the marker is located, a jump to 4 mode 3 phase occurs and the next command is shifted into both the operation register and the common register (in addition to being rewritten onto the line), as shown in Figure 3-107 (b).



LINE ROUTING, 4 MODE 3 PHASE, SENSING A 2-ADDRESS COMMAND

FIGURE 3-107 (b)

When this command is determined to be a 2-address command, a jump to 6 phase occurs, where the line routing changes to PLSR, SRPL, PLCR. The marker is written out onto the line immediately behind the command, and the first address digit is shifted into both the common register and the search register. See Figure 3-107 (c).

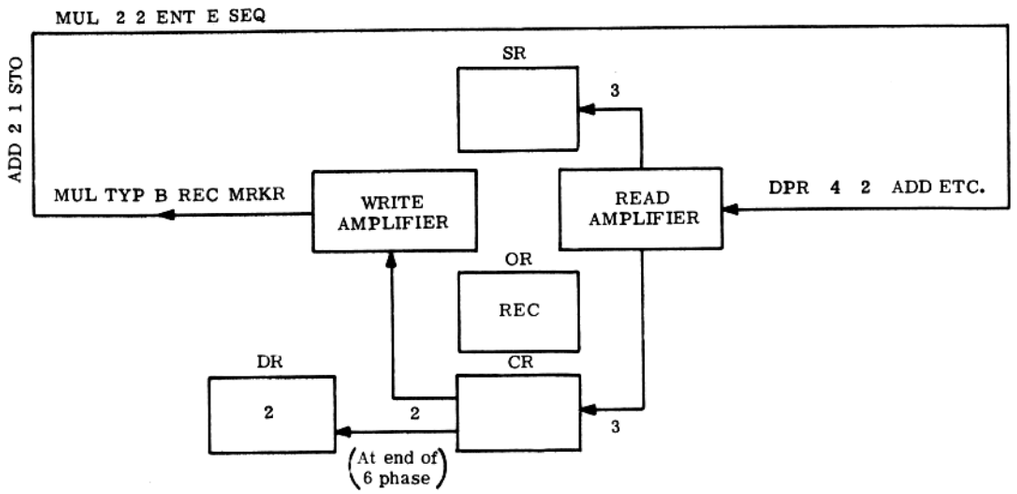


LINE ROUTING, 4 MODE 6 PHASE

FIGURE 3-107 (c)

At the end of 6 phase, the code in the common register is set into the decimal register in parallel. This action is abbreviated "CRDR", even though it is not really a "line routing". (CRDR also occurs in 4 phase to set the address of the Truncate command into the common register.)

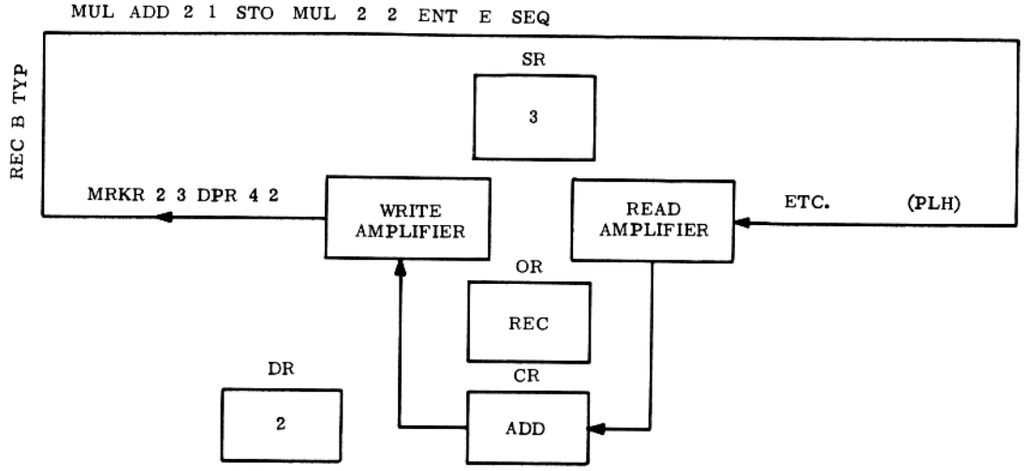
Next a step to 7 phase occurs and the line routing becomes PLCR, CRPL, and PLSR. The first address digit is written back onto the program line from the common register as the second digit is shifted in. The second digit is also shifted into the search register for later use during 5 mode. See Figure 3-107 (d).



LINE ROUTING, 4 MODE 7 PHASE

FIGURE 3-107 (d)

At the end of 7 phase the first address digit is stored in the decimal register and the second in the search register. A step to 8 phase occurs next. Here the PLSR routing is dropped, but PLCR, CRPL remains. The balance of the program passes through the common register in 8 phase, as shown in Figure 3-107 (e). This unusual line routing is necessary to maintain the balance of the program on the line. Normally the balance of the program data would be shifted through the search register, but in this case the least significant digit of the address is stored there; the common register is used to maintain the program because it is "free" at this time.



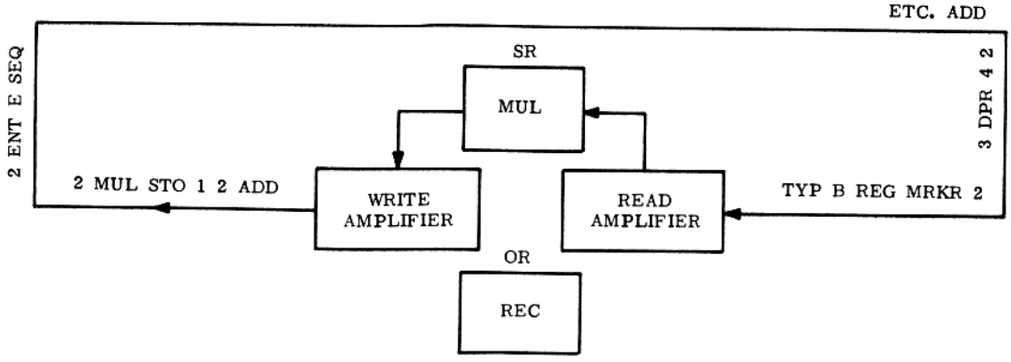
LINE ROUTING, 4 MODE 8 PHASE

FIGURE 3-107 (e)

At program line home a jump to 5 mode occurs, where the Recall instruction is executed. When this jump takes place, the line routing changes to PLPL, which is the only line routing used in 5 mode.

(b) Retrieval of a Command Following the Execution of a 2-Address Command. When a 0- or 1-address command is executed, the marker is left immediately in front of the next command on the program line. However, after a 2-address command has been completed, the marker is still located between the command and its address digits, as shown in Figure 3-107 (e). As the next instruction is sought out, the fact that the previous 2-address command is still in the operation register causes the two address digits to be bypassed before the following command is actually retrieved.

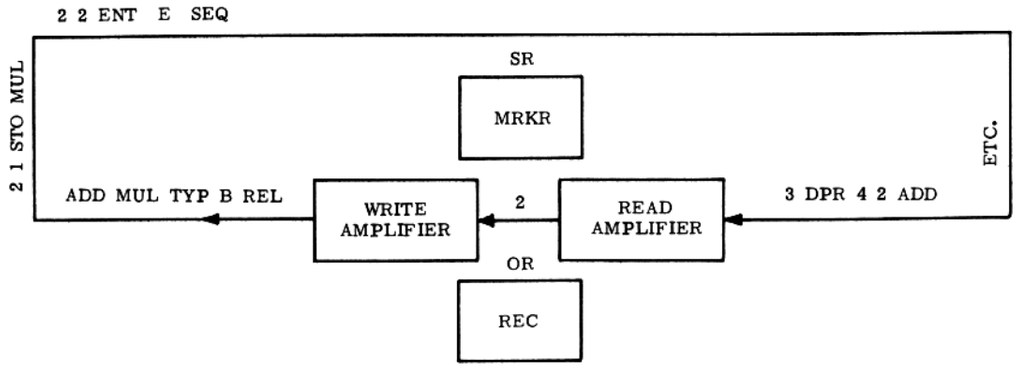
Assume that the Recall instruction has been executed and that marker search is again underway. Since it is no longer necessary to store the address digits of the previous instruction, the search register may be used to look for the marker. See Figure 3-108 (a).



MARKER SEARCH AFTER EXECUTING A 2-ADDRESS COMMAND

FIGURE 3-108 (a)

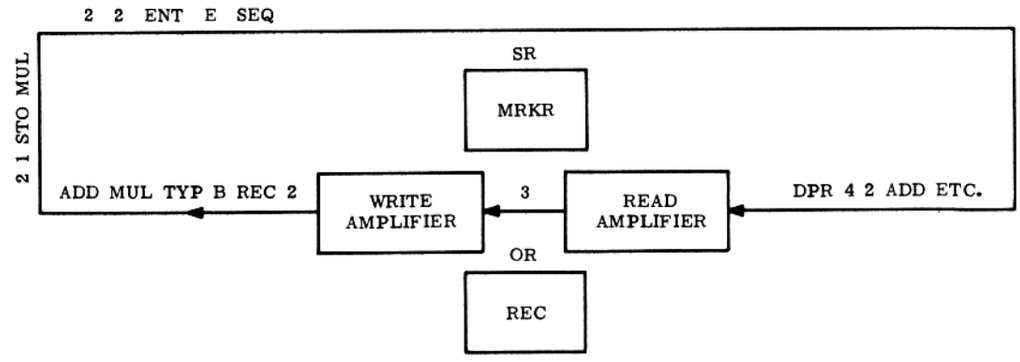
When the marker is located, the Operation 2 ADdress signal, O2AD3, causes a jump to 4 mode 1 phase instead of 4 mode 3 phase, as explained previously in "Marker Search". In 4 mode 1 phase the PLPL line routing causes the first address digit of the preceding instruction to be bypassed. See Figure 3-108 (b).



LINE ROUTING, 4 MODE 1 PHASE

FIGURE 3-108 (b)

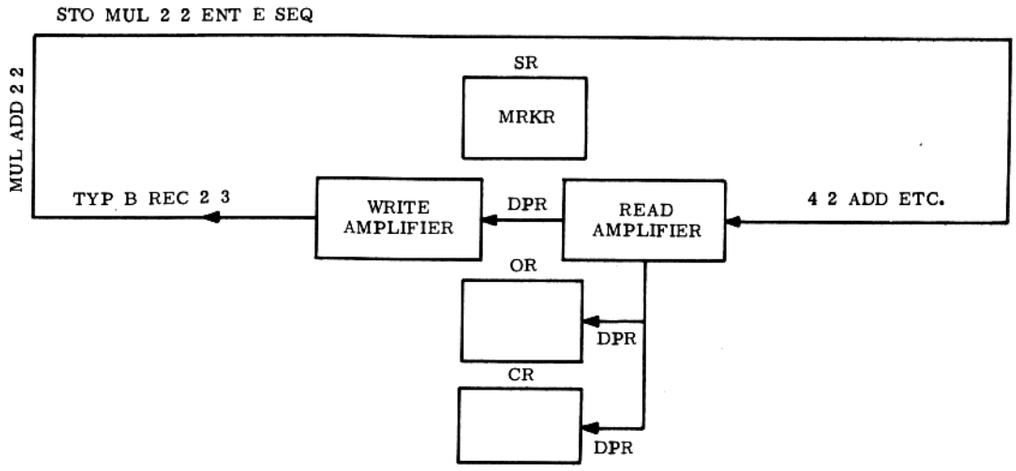
Next a step to 2 phase occurs. The line routing remains PLPL and the second address digit is bypassed, as shown in Figure 3-108 (c).



LINE ROUTING, 4 MODE 2 PHASE

FIGURE 3-108 (c)

Then a step to 3 phase occurs. The line routing changes to PLPL, PLOR, and PLCR and the next command is retrieved by shifting it into both the operation register and the common register, as shown in Figure 3-108 (d). The continuing PLPL line routing writes the new command onto the line.



LINE ROUTING, 4 MODE 3 PHASE,
AFTER EXECUTING A 2-ADDRESS COMMAND

FIGURE 3-108 (d)

Subsequent operation is the same as described previously. In the example shown in Figure 3-108, a jump to 6 phase occurs first, followed by steps to 7 and 8 phase. Had the next command been a 1-address command instead of a Decimal Print, it would have been retrieved in 3-4-5 phases instead. A 0-address command would have caused a jump from 3 phase to 5 phase.

Utilizing the preceding method, any new command is retrieved in 3 phase, regardless of the command that precedes it.

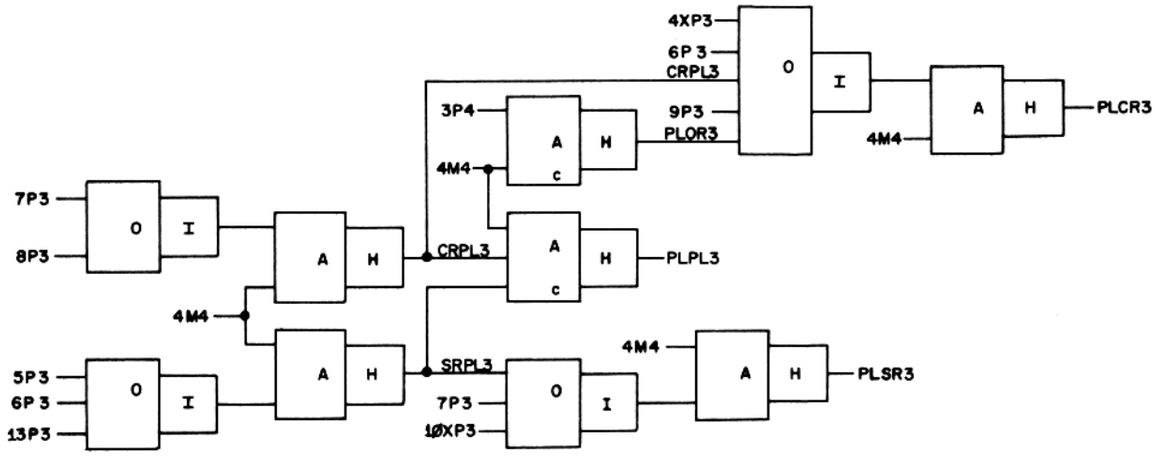
b. SPECIAL INSTRUCTIONS. Most branch commands, and a few other commands, require variations in the "normal" procedure just described. The 4 mode operation of these commands will be covered shortly, although full details of their operation will be presented later.

c. SUMMARY OF OPERATION BY PHASES. The following list briefly summarizes the purpose(s) of each phase in 4 mode.

a. Phase 1. Bypass first address digit of previous 2-address command.
b. Phase 2. Bypass second address digit of previous 2-address command.
c. Phase 3. Retrieve commands and constant factor digits (and characters).
d. Phase 4. Retrieve address of 1-address commands.
e. Phase 5. Sync on program line home, set up proper jumps for 0- and 1-address commands.
f. Phase 6. Retrieve first address digit of 2-address commands.
g. Phase 7. Retrieve second address digit of 2-address commands.
h. Phase 8. Sync on program line home, set up proper jumps for 2-address commands.
i. Phase 9. Retrieve unconditional branch address when UBR is previously set.
j. Phase 10. Retrieve conditional branch address.
k. Phase 11. Sync on program line home, set up jumps for branches using 9 or 10 phase, and for end of sequence. Destroy marker.
l. Phase 12. End of sequence during unconditional branch: destroy branch marker, reset UBR, set RST, set up jump to 3M6,4.
m. Phase 13. Sync on program line home during constant factor digit retrieval.
n. Phase 14. Not used.
o. Phase 15. Wait for writing machine before entry, either in Enter instruction or at end of sequence (subsequent entry of new sequence title).

2. LINE ROUTINGS

The development of all of the line routings used in 4 mode is illustrated in Figure 3-109. Note that PLPL will be true whenever both CRPL and SRPL are false. CRPL is true during 7 and 8 phases, and SRPL is true during 5, 6, and 13 phases. This leaves PLPL true during 1, 2, 3, 4, 9, 10, 11, 12, and 15 phases. PLSR is true during 7 and 10 phases, and also during 5, 6, and 13 phases, when SRPL is true. PLOR is true only in 3 phase. PLOR drives PLCR true during 3 phase, and CRPL drives it true during 7 and 8 phases. It is also true during 4, 6, and 9 phases.



DEVELOPMENT OF LINE ROUTINGS IN 4 MODE
(All on "F" card unless otherwise noted)

161-9

FIGURE 3-109

Note that the decimal settings have no effect on the line routings in 4 mode.

The chart below summarizes the line routings by phase.

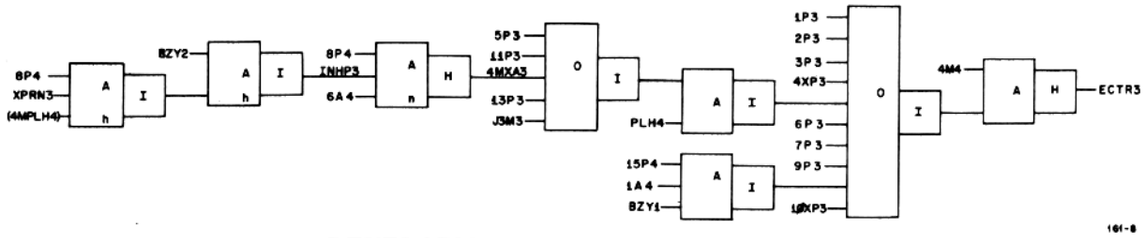
PHASE	ROUTING
1&2	PLPL
3	PLPL, PLCR, PLOR
4	PLPL, PLCR
5	PLSR, SRPL
6	PLSR, SRPL, PLCR
7	PLCR, CRPL, PLSR
8	PLCR, CRPL
9	PLPL, PLCR
10	PLPL, PLSR
11&12	PLPL
13	PLSR, SRPL
15	PLPL

The reasons for the different line routings will not be covered yet: they will become apparent shortly as the various operations are treated in more detail.

3. ENABLING THE PHASE COUNTER

The development of ECTR3 is automatic in most phases of 4 mode, as illustrated in Figure 3-110. The exceptions are 5, 8, 11, 12, 13, and 15 phases. In 5, 11, and 13 phases, the only requirement is program line home. The same is true for 12 phase, which develops the Jump to 3 Mode signal J3M3. The requirement in 15 phase is that the 2205 not be "busy". A special

case is found in 8 phase. Most instructions simply require PLH to be set. If, however, the command retrieved is one of the two print commands, the logic remains in 8 phase until BOTH requirements are satisfied. That is, 8 phase remains true until the 2205 is no longer busy and PLH is set. Remember that 14 phase is not used.



DEVELOPMENT OF ECTR3 IN 4 MODE
(All on "F" card, unless otherwise noted)

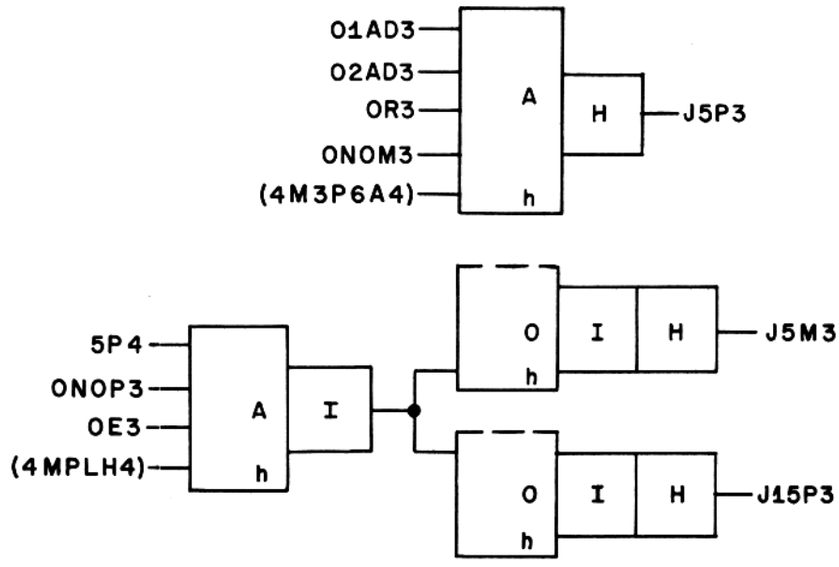
FIGURE 3-110

4. STANDARD OPERATION

The path taken by the logic in 4 mode is determined by the Op Decoder signals produced by the command as it is retrieved in 3 phase. Most instructions follow one of three different paths, determined by whether the command is a 0-, 1-, or 2-address command. Other special commands cause different operation, and will be discussed later.

a. 0-ADDRESS COMMANDS. Most 0-address commands sensed in 3 phase simply cause a jump to 5 phase, as shown in Figure 3-111. The J5P3 signal will be true at T6A time in 4 mode 3 phase if the operation register does NOT contain a 1-address code, a 2-address code, a Return command, or a "no-marker" (end of sequence) code ("H", "L", or space). The latter will be covered shortly.

The logic remains in 5 phase until program line home, where a jump to 5 mode 15 phase occurs, also shown in Figure 3-111. Note that an unconditional branch command in the operation register (OE3) will block this jump, as will the "Op No OP" signal, ONOP3. This will also be explained shortly.

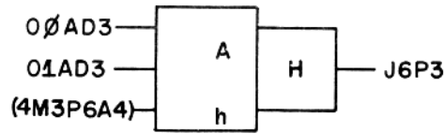


0-ADDRESS COMMAND RETRIEVAL

FIGURE 3-111

b. 1-ADDRESS COMMANDS. Conventional 1-address commands simply block the jump to 5 phase provided by the 0-address commands and allow a step to 4 phase instead. In 4 phase the address portion of the instruction is shifted into the common register. Then a step to 5 phase takes place and subsequent operation is the same as for 0-address commands: a jump to 5 mode 15 phase occurs at program line home. Refer to Figure 3-111 and note how the 1-address signal, O1AD3, blocks the jump to 5 phase.

c. 2-ADDRESS COMMANDS. All 2-address commands cause a jump from 3 phase to 6 phase (Figure 3-112).



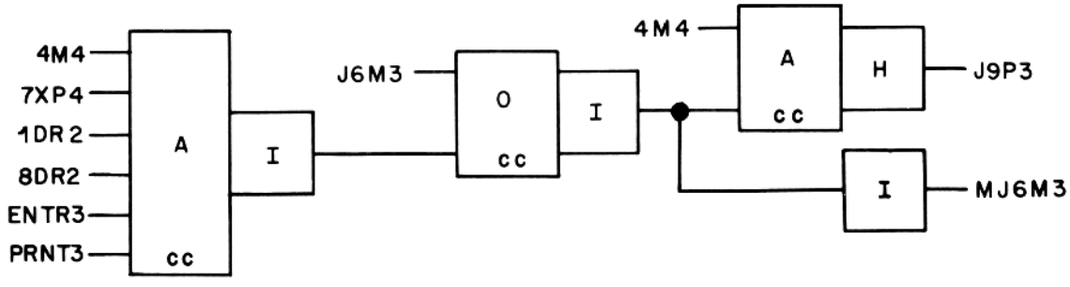
DEVELOPMENT OF J6P3 DURING 4M3P

FIGURE 3-112

In 6 phase the PLSR, SRPL routing writes the marker back onto the line immediately behind the command. The PLCR routing shifts the first address digit into the common register. At the end of 6 phase (T6A time) this digit is duplicated into the decimal register. (This will be explained shortly.)

Next a step to 7 phase occurs, where the PLCR, CRPL line routing serves to maintain the address digits on the program line, and the PLSR routing shifts the second address digit into the search register, where it will be used in 5 mode.

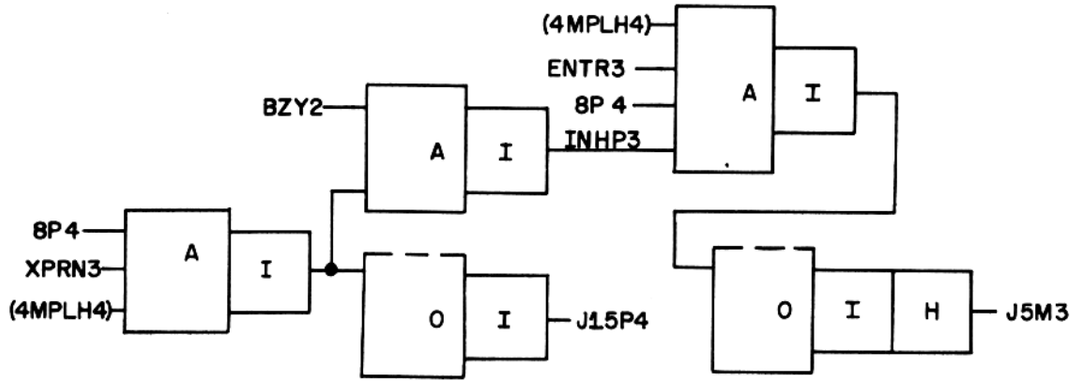
If the instruction being retrieved is a Store, Recall, or Accumulate instruction that will utilize the Address Modifier feature, the address retrieved in 6 and 7 phases will be "99". If a 9 was retrieved and set into the decimal register in 6 phase, a jump to 6 mode 9 phase is enabled in 7 phase, as shown in Figure 3-113. In 6 mode 9 phase the line routing remains PLCR, CRPL. See Section 9 for a complete description of the Address Modifier.



JUMP TO 6M6P DURING ADDRESS MODIFIER INSTRUCTION
(All on "CC" card)

FIGURE 3-113

For 2-address codes not using the Address Modifier, a step to 8 phase occurs, where the program is maintained on the line via the PLCR, CRPL line routing. At program line home a jump occurs, depending upon the command, to either 4 mode 15 phase, 5 mode 15 phase, or 5 mode 9 phase. Refer to Figure 3-114 and note that if the operation register contains an Enter command, a jump to 4 mode 15 phase will occur. If one of the two print commands has just been retrieved, a jump to 5 mode will occur, but only after BZY2 again becomes true (when the 2205 is no longer busy). In this case, the jump to 15 phase is blocked, so a step to 9 phase occurs instead. Any other command in the operation register will cause a jump to 5 mode 15 phase.

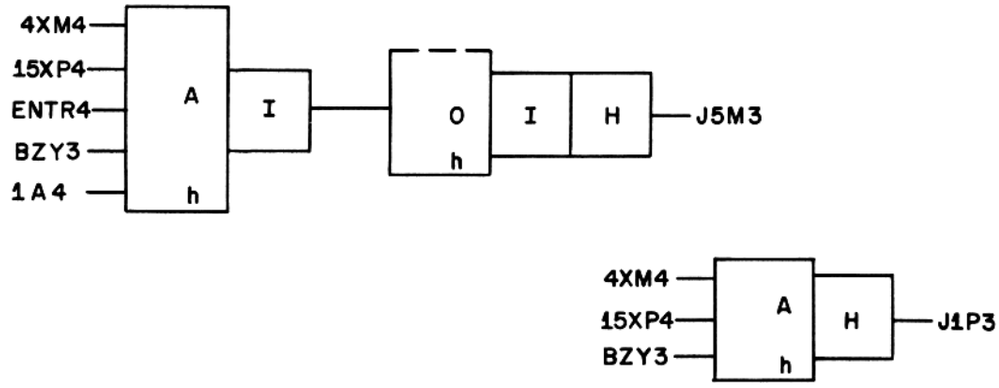


JUMPS FROM 4M8P CAUSED BY 2-ADDRESS COMMANDS
(All on "H" card)

FIGURE 3-114

Summarizing Figure 3-114 another way, it can be said that any 2-address command other than an Enter will cause a jump to 5 mode, and any 2-address command other than a print will cause a jump to 15 phase.

The Enter command causes the logic to wait in 15 phase until the 2205 is no longer busy, at which time a jump to 5 mode 1 phase occurs. See Figure 3-115.



"ENTER" COMMAND ACTION IN 4M15P

FIGURE 3-115

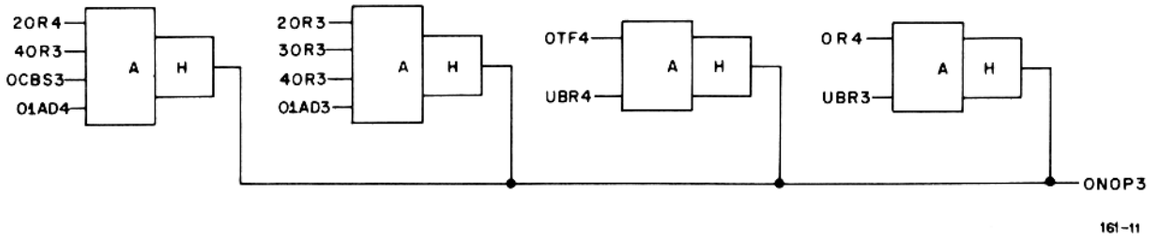
5. SPECIAL OPERATION

Most commands that do not follow the 4 mode operation outlined previously are 1-address commands. There are no special 2-address commands, and only a few special 0-address commands. The treatment of constants and branch oriented instructions will be brief at this time; they will be covered in greater detail later on.

a. "NO-OP" COMMANDS. Some commands, when sensed in the operation register under certain circumstances, become "No Operation" commands, and develop the "Op No OP" signal ONOP3. A Positive Branch command retrieved while R1 is negative, or a Negative Branch retrieved while R1 is positive, will produce ONOP3. Any time a Resume command is retrieved

in 4 mode 3 phase, it produces ONOP3. (The Resume command has a purpose ONLY in 3 mode, during conditional branch search: in the absence of a conditional branch command, the Resume code is superfluous).

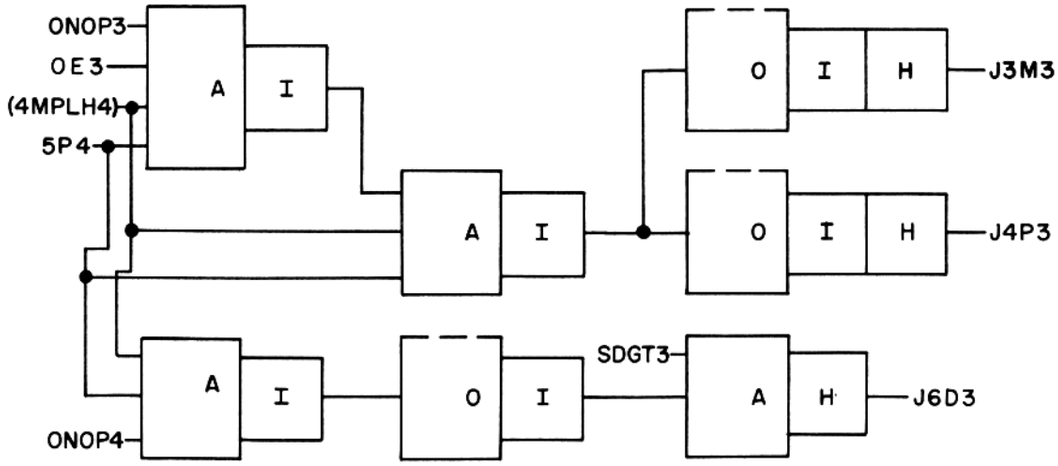
The preceding commands mentioned are all 1-address commands. One 0-address command, the Return, also produces ONOP3 if it is sensed when not in an unconditional branch condition.



DEVELOPMENT OF ONOP3
(All on "J" card)

FIGURE 3-116

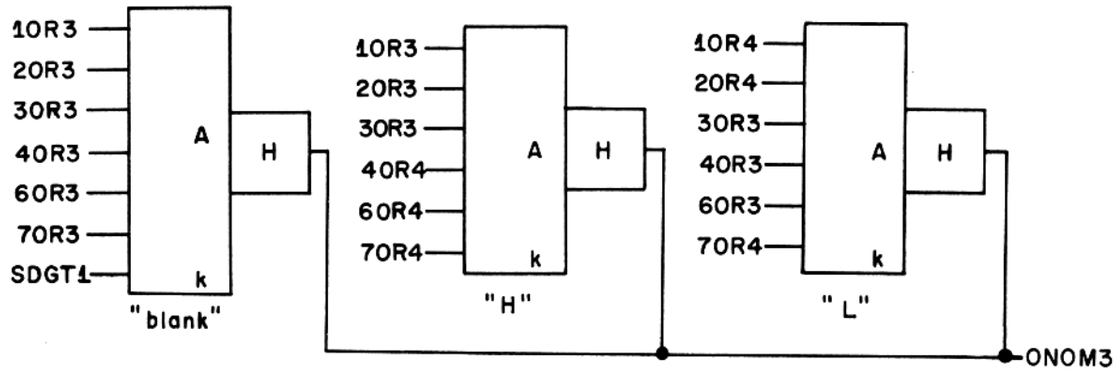
Figure 3-116 illustrates the development of ONOP3. The Op Conditional Branch Signal OCBS3, is produced when a PBR code is retrieved with R1 positive, and when an NBR code is retrieved with R1 negative. The development of OCBS3 will be covered in the conditional branch discussion. When a "No-Op" is sensed in 3 phase, the normal course is followed to 5 phase (a jump to 5 phase for 0-address commands, or a step through 4 phase for 1-address commands) in order to allow the marker to be rewritten in the correct position. At program line home in 5 phase, however, a jump to 3M6,4 occurs instead of a jump to 5 mode, and marker search begins to locate the next command. Figure 3-117 illustrates the development of the jump to 3M6,4.



JUMP TO 3M6,4 CAUSED BY ONOP IN 4M5P
(All on "H" card)

FIGURE 3-117

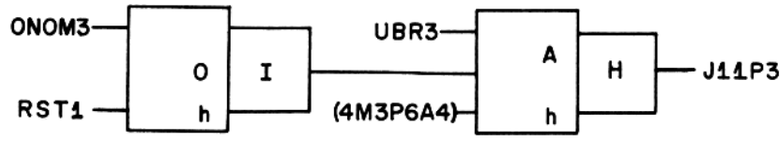
b. "NO MARKER" CODES. Other codes, when retrieved in 4 mode, inhibit the rewriting of the marker, which, in effect, destroys it. Retrieval of a sequence separator ("H" or "L") denotes the end of a sequence: after the last instruction in a sequence has been performed, the ensuing marker search results in the retrieval of the sequence separator heading up the next sequence loaded on the program line. When this occurs, the marker is destroyed and a jump to 3M0,1 occurs to wait for the entry of the title of the next sequence to be performed. A space (blank) retrieved in 3 phase has the same effect; should the sequence just completed be the last one loaded on the program line, no data follows it, and a blank will be retrieved. (The SDGT flip-flop becomes set to allow retrieval of a space during constant factor instructions.) All of these codes produce the Operation NO Marker signal, ONOM3, as shown in Figure 3-118.



DEVELOPMENT OF ONOM3

FIGURE 3-118

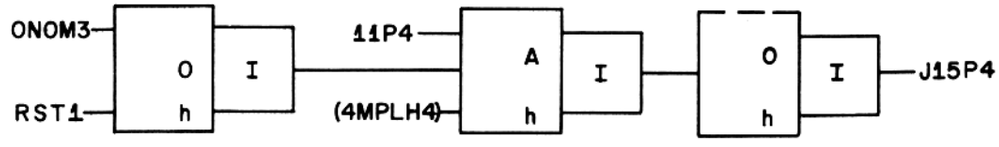
(1) Normal Conditions. At all times except during unconditional branch ONOM3 causes a jump from 3 phase to 11 phase, as shown in Figure 3-119.



JUMP TO 11P CAUSED BY ONOM IN 4M3P

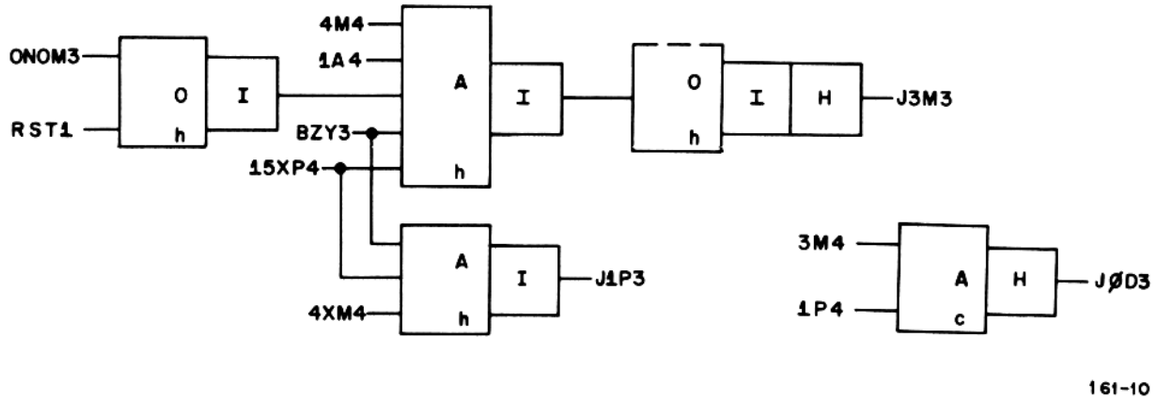
FIGURE 3-119

In 11 phase the logic waits for program line home, and then jumps to 15 phase, as shown in Figure 3-120.



JUMP TO 15P CAUSED BY ONOM IN 4M15P
FIGURE 3-120

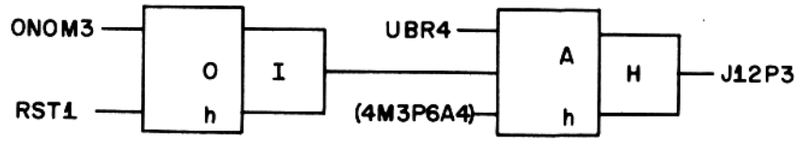
The logic waits in 15 phase until the 2205 is no longer busy. Then a jump to 3 mode 1 phase occurs. As soon as 3 mode 1 phase is realized, J0D3 is produced. See Figure 3-121. This 3M0,1 condition then remains until a new sequence title is entered via the writing machine.



JUMP TO 3M0,1 FROM 4M15P

FIGURE 3-121

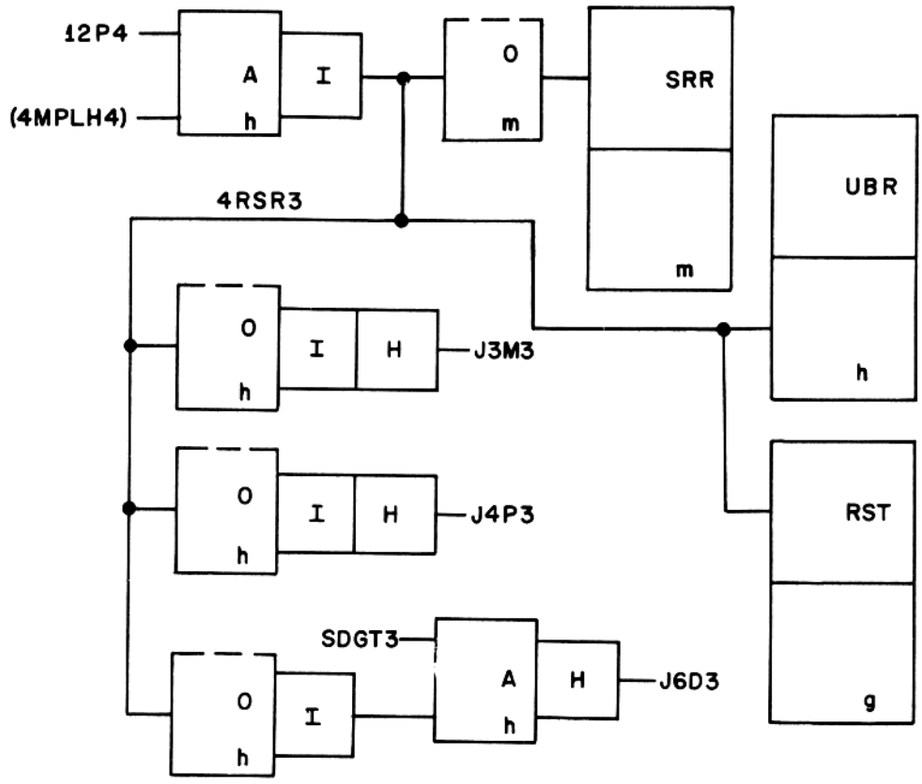
(2) During Unconditional Branch. When ONOM3 is produced during unconditional branch, the jump from 3 phase goes to 12 phase instead of 11 phase. See Figure 3-122.



JUMP TO 12P CAUSED BY ONOM3 IN 4M3P DURING UBR1

FIGURE 3-122

In 12 phase the logic waits for program line home, at which time 4RSR3 is developed. This resets UBR, sets SRR (to reset the branch marker out of the search register), sets the RST flip-flop, and enables the jump to 3M6,4. See Figure 3-123.



EFFECT OF 4RSR3 IN 4M12P

FIGURE 3-123

When the sync pulse restarts the timing chain, the 3M6,4 setting causes a search for the marker. When the marker is located, the fact that RST is set causes the marker to be destroyed, the same as ONOM3 did previously. After the marker is destroyed, the logic returns to 3M0,1 to wait for the entry of the next sequence title. Refer to Figure 3-119 through 3-122 and note that RST being set has the same effect as ONOM3 being true. Note also that this procedure is the same as that followed when terminating program load.

c. UNCONDITIONAL BRANCH COMMAND. The operation caused by retrieval of the Branch command varies, depending upon whether the UBR flip-flop was set previously.

(1) Normal Conditions. If UBR is reset, retrieval of a Branch command causes the same operation in 4 mode as any other 1-address command, except that the jump from 5 phase to 5 mode 15 phase is replaced by a jump to 3M0,4. In this case, the marker is written onto the program line immediately behind the branch address, so that after a Return command is executed sometime later, the subsequent marker search and instruction retrieval will locate the next command in the sequence.

(2) During Unconditional Branch. When a Branch command is sensed with the processor already in a branch condition, no marker is written: instead, the branch is completed to the new sequence as instructed by the latest command, where a new branch marker is written out. (The original normal marker is not affected.) In this case, a jump from 3 phase to 9 phase takes place, where the address of the new sequence is shifted into the common register, the same as in 4 phase. Then a jump to 11 phase takes place, and the logic waits for program line home, where a jump to 3M0,4 occurs.

d. RETURN COMMAND. As explained earlier, a return command that is retrieved when the UBR flip-flop is reset becomes a "No-op". On the other hand, if UBR is set, the Return command must destroy the branch marker and reset UBR before initiating another marker search. The Return uses 11 phase to do these things while waiting for program line home.

e. CONDITIONAL BRANCH COMMANDS. As explained earlier, a Positive Branch command retrieved when R1 is negative, or a Negative Branch retrieved when R1 is positive, does not develop the Op Conditional Branch Signal, OCBS3 (to be covered later). Instead, these codes function the same as any other 1-address "No-op", causing the logic to step through 4 phase to 5 phase, and then jump to 3M6,4.

Those conditional branch commands that do develop OCBS3 cause a jump from 3 phase to 10 phase, where the PLPL line routing not only maintains the program on the line, but also "closes up" the line to compensate for removal of the marker. The PLSR routing shifts the branch address into the search register. As soon as this is completed, a step to 11 phase takes place. In 11 phase, OCBS3 inhibits the reset of the search register, thus retaining the address. At program line home, the jump to 3 mode and 4 phase again takes place. At this time, however, a jump to 5 decimal occurs. Conditional branch operation from this point will be covered later; only the 4 mode portion is treated here.

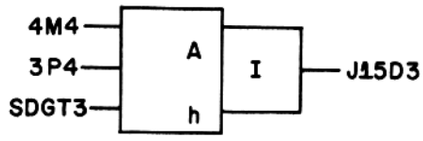
f. CONSTANTS. When the Constant command is retrieved in 4 mode 3 phase it is treated the same as any other 0-address command. However, after the jump to 5 mode occurs, the 5 mode logic sets the Significant DiGiT flip-flop, SDGT. Then marker search resumes in order to retrieve the first digit of the constant factor. Each time a digit is retrieved, the marker moves down the line one space. Each digit is shifted into the common register in 4 mode and transferred to the arithmetic line in 5 mode.

With SDGT set, each digit is shifted into the operation register and the common register in 4 mode 3 phase. Then a jump to 13 phase occurs, where the PLSR, SRPL line routing resumes, and continues until program line home, at which time a jump to 5 mode 15 phase occurs.

In 5 mode the constant factor digit is transferred into R1 and the marker search is set up for the next digit. This operation continues until a space or a dash is retrieved. Either of these will reset SDGT in 5 mode, thus allowing regular 4 mode operation to resume.

g. COMMON REGISTER-TO-DECIMAL REGISTER. The code in the common register is duplicated into the decimal register at the end of 4 phase, in the case of 1-address codes, and at the end of 6 phase, for 2-address codes. In the case of 2-address commands, it is always necessary to store the first of the two address digits for later use in 5 mode. In the case of 1-address commands, this action is necessary only for the Truncate command, but it occurs anyway for all commands using 4 phase.

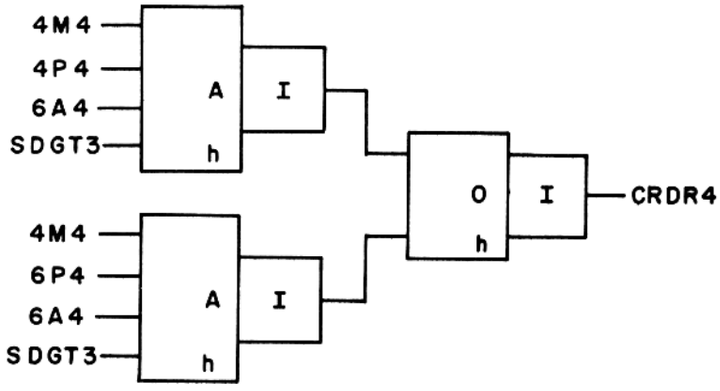
This action is accomplished by first setting all decimal register flip-flops (DR=15) in 3 phase. This occurs during retrieval of any code except constant factor digits, as shown in Figure 3-124. For most 1-address codes, this has no effect, since other conditions are set into the decimal register upon leaving 4 mode.



SETTING DR TO 15 IN 4M3P

FIGURE 3-124

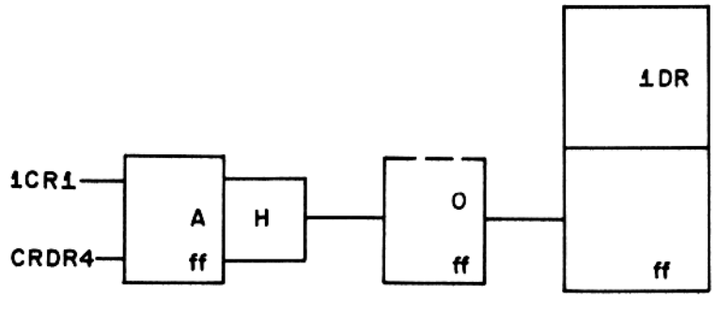
The Common Register-to-Decimal Register signal, CRDR4, becomes false at the end of 4 phase or at the end of 6 phase, as shown in Figure 3-125.



DEVELOPMENT OF CRDR4

FIGURE 3-125

Immediately, CRDR4 is ANDed with 1CR1 to reset 1DR, as shown in Figure 3-126, and with 2CR3 to reset 2DR, etc.



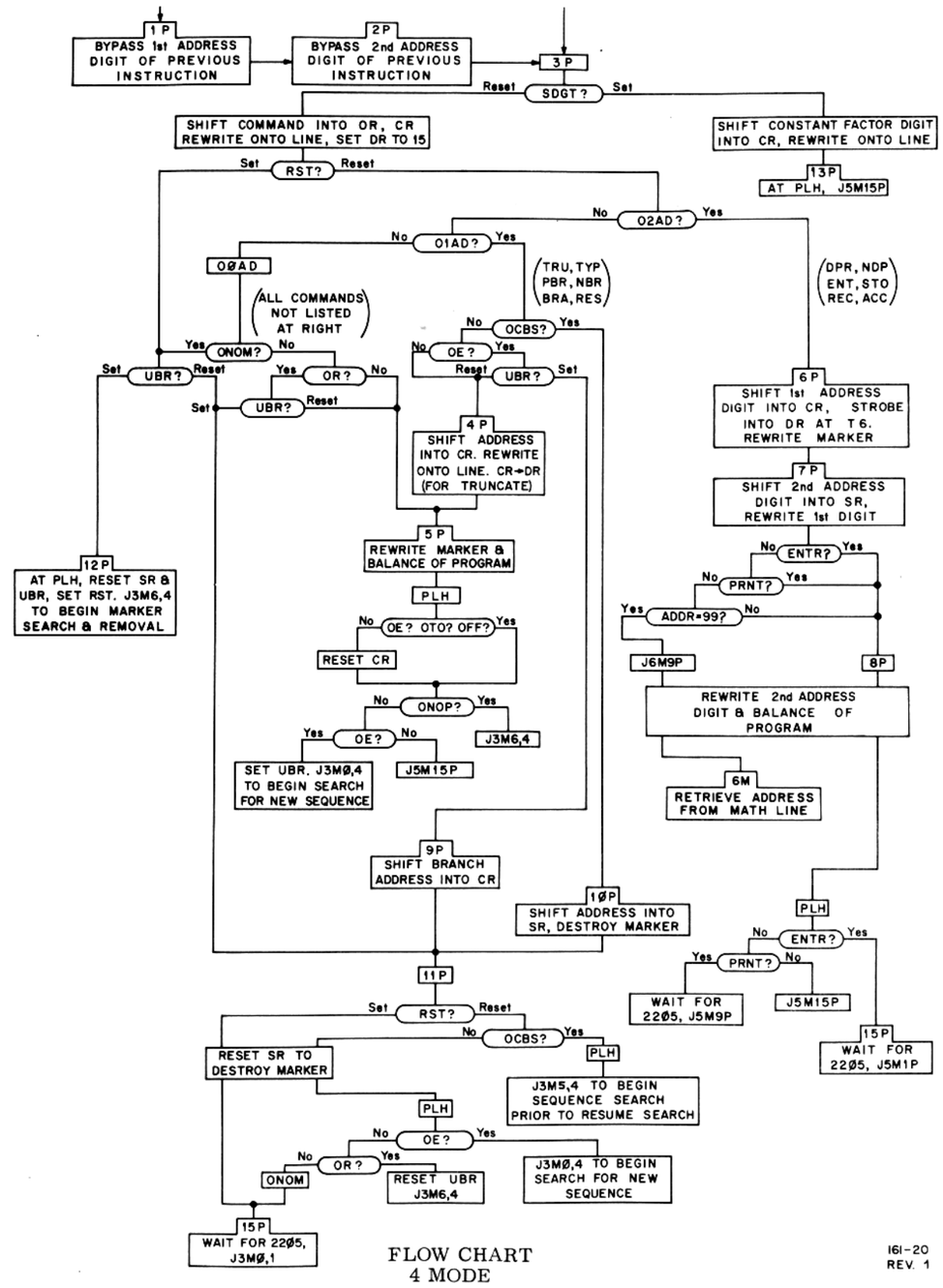
CRDR ACTION ON 1DR

FIGURE 3-126

Thus, if a common register flip-flop is reset, its corresponding decimal register flip-flop becomes reset; if a common register flip-flop is set, it blocks the reset of the corresponding decimal register flip-flop. In this way the digit in the common register is duplicated into the decimal register, and the common register is then free to be used for another code. Note, however, that this function does not clear the common register: the original code is retained. This fact is important in 6 phase, because immediately following this CRDR action, 7 phase causes the code to be shifted out of the common register back onto the program line. Notice also that CRDR is valid only for digits, since only the first four bits can be transferred to the decimal register.

6. SUMMARY

The flow chart in Figure 3-127 summarizes the entire retrieve operation in 4 mode. In addition, the basic operations involved in the retrieval of each instruction are reviewed later as the individual instructions are covered in more detail.



FLOW CHART
4 MODE

161-20
REV. 1

FIGURE 3-127

I. UNCONDITIONAL BRANCH

When a Branch instruction is executed, the processor interrupts the sequence of instructions being performed and jumps to a new sequence. This new sequence has an address identical to the address of the Branch instruction. The instructions in the new sequence are then executed.

When executing a Branch instruction, the processor automatically leaves a normal marker on the program line immediately behind the Branch address, and sets the UBR flip-flop. While performing the new sequence, it ignores the normal marker and recognizes only the branch marker. When a Return instruction is executed, the branch marker is destroyed, UBR is reset, and operation resumes back at the normal marker.

If a second Branch instruction is executed with no Return command intervening (a Branch within a branch), no marker is left at that point. Instead, the processor merely jumps to the newer sequence, leaving the original marker undisturbed.

If a Branch instruction specifies a non-existent sequence, a branch to the error sequence occurs. Branches can be performed to points either ahead of or behind the point in the program at which the Branch instruction is located. There is no limit to the number of branches that can be made, either independently or within other branches. (Limits are determined only by total program storage capacity.) Note, however, that regardless of the number of successive branches, execution of the Return instruction will always return the logic to a point immediately behind the first Branch instruction of the group: the point at which the normal marker was left.

1. GENERAL OPERATION (See Figure 3-128)

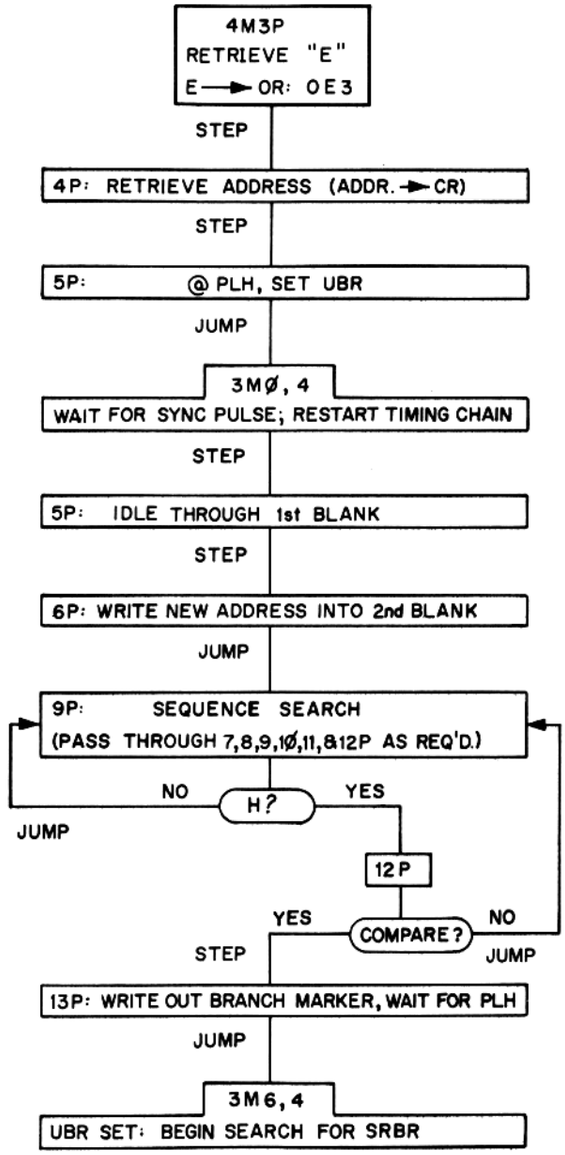
When a Branch command is retrieved in 4 mode 3 phase, it is treated the same as any other 1-address command: the logic steps to 4 phase, where the address is shifted into the common register, and then to 5 phase, where the marker is written out. The balance of the program is shifted through the search register before being rewritten onto the line. At program line home, a jump to 3M0,4 occurs, and the UBR flip-flop becomes set.

When the sync pulse is sensed and the timing chain restarts, the phase counter steps to 5 phase at the end of the buffer, and to 6 phase at the end of the first blank. During 6 phase the new sequence address (Branch address) is written into the second blank on the program line. At the end of the second blank a jump to 9 phase occurs and a sequence search begins. Each normal sequence separator sensed causes a jump to 12 phase where its address is compared to the code in the common register (the Branch address). If they do not compare, a jump back to 9 phase occurs and the search continues. When a compare is made, a step to 13 phase occurs, where the marker is written out (a BRANCH marker, because UBR is set). At program line home, a jump to 3M6,4 occurs and marker search begins. Because UBR is set, the normal marker will be ignored, and only the branch marker will be recognized.

If a Branch command is retrieved with the processor already in the branch condition (UBR set), the logic jumps from 4 mode 3 phase to 9 phase, instead of stepping to 4 phase. The address of the Branch instruction is retrieved in 9 phase, and a jump to 11 phase takes place. In 11 phase the marker is reset out of the search register and at program line home a jump to 3M0,4 occurs. Subsequent 3 mode operation is the same as described previously.

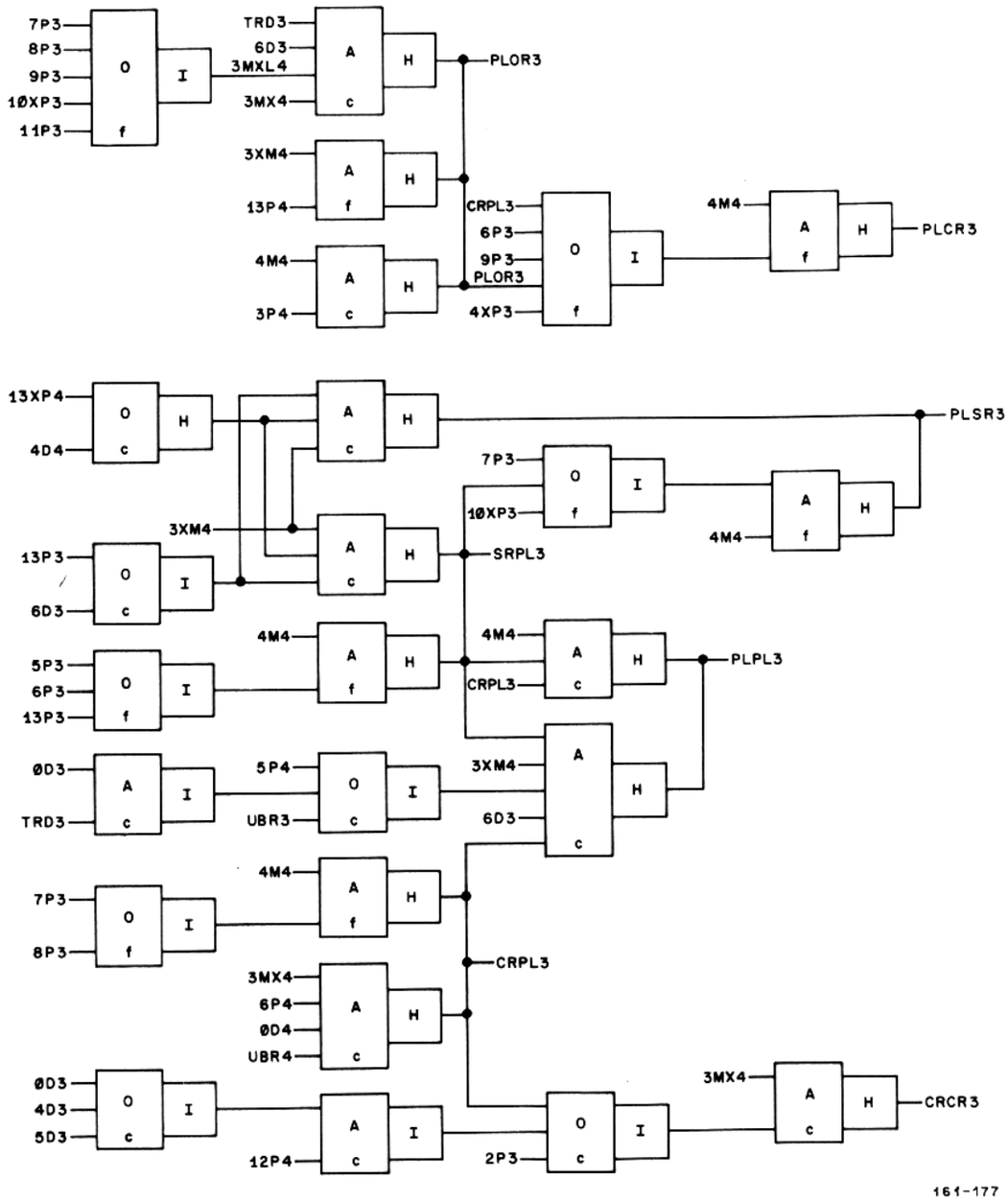
2. LINE ROUTINGS

The development of all of the line routings used in unconditional branch is illustrated in Figure 3-129. Most of these gates have been used earlier in Sections 3G, Search, and 3H, Instruction Retrieval. Note, however, that there are some differences. In 3 mode, the UBR3 signal no longer blocks PLPL3 in 5 phase. Instead, UBR4 develops CRPL3 in 6 phase, and CRPL3 blocks PLPL3. This writing of the new sequence title into the second blank instead of the first is one of the two major differences between unconditional branch sequence search and normal sequence search; the other is the fact that a branch marker, instead of a normal marker, is written out once the new sequence is located. The reason for writing the address into the blank will be covered in Section 3K, Conditional Branch. The functions of the various line routings used in unconditional branch are listed in Figure 3-130.



FLOW CHART
UNCONDITIONAL BRANCH - Basic Operation

FIGURE 3-128



4 MODE		3 MODE (0 DECIMAL, UBR1)	
3P	PLPL, PLOR, PLCR	4P, 5P	PLPL
4P	PLPL, PLCR	6P	CRPL
5P	PLSR, SRPL	7P to 11P	PLPL, PLOR
9P	PLPL, PLCR	12P	PLPL, CRCR
11P	PLPL	13P	PLSR, SRPL, PLOR
		14P, 15P	PLPL

DEVELOPMENT OF LINE ROUTINGS
USED IN UNCONDITIONAL BRANCH

FIGURE 3-129

4 MODE	PLPL	PLSR	SRPL	PLOR	PLCR	CRPL	CRCR
3 PHASE	Maintain Program			BRANCH Develops OE3	Necessary Only for Constants & 2205 functions		
4 PHASE	Maintain Program				Retrieve BRANCH Address		
5 PHASE		Maintain Program	Write Out Marker				
3 MODE							
4 PHASE 5 PHASE	Maintain Program						
6 PHASE						Write BRANCH Address	
7 to 11 PHASE	Maintain Program			Search for "H"			
12 PHASE							Compare
13 PHASE		Maintain Program	Write Out Branch Marker	Necessary During OCBS3			
4 MODE	(Retrieve BRANCH in 4M3P with UBR already set; leave no marker.)						
9 PHASE	Maintain Program				Retrieve BRANCH Address		
11 PHASE							
3 MODE	(Invalid Sequence Given as Branch Address)						
14 PHASE 15 PHASE	Maintain Program						
	PLPL	PLSR	SRPL	PLOR	PLCR	CRPL	CRCR

LINE ROUTINGS USED IN UNCONDITIONAL BRANCH

FIGURE 3-130

3. RETRIEVAL OF THE BRANCH INSTRUCTION

The operation caused by retrieval of the Branch command varies, depending upon whether the UBR flip-flop was set previously.

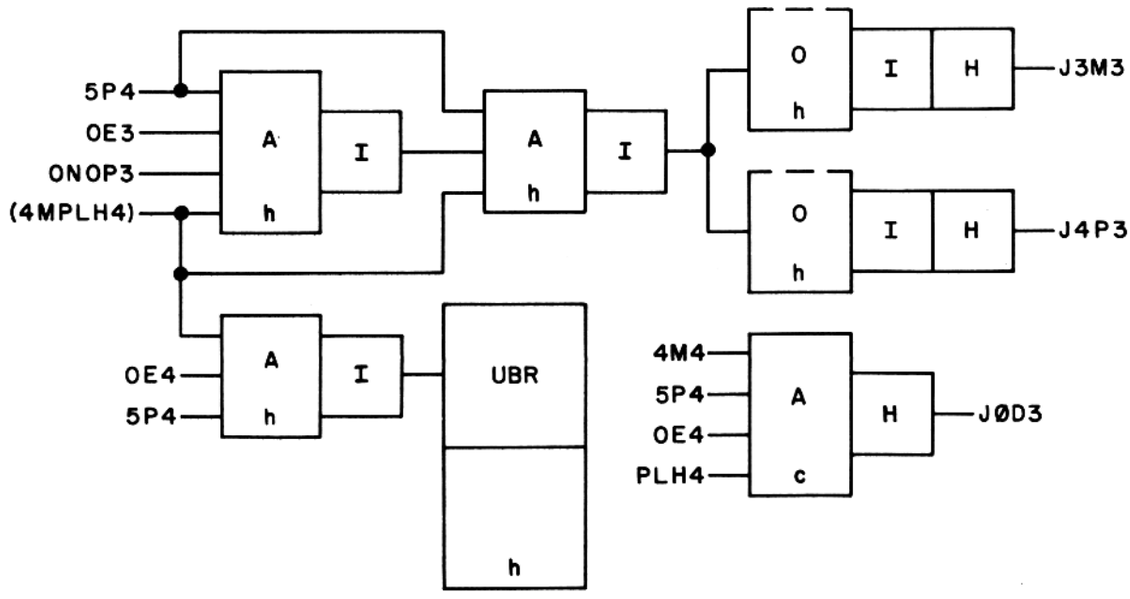
a. NORMAL CONDITIONS. If UBR is reset, retrieval of a Branch command causes normal 1-address command operation in 4 mode: the logic steps through 4 phase to 5 phase, where it waits for program line home, and then jumps to 3M0,4 where the search for the new sequence begins.

In 3 phase the PLOR line routing shifts the Branch command into the operation register. The PLPL routing writes the Branch back onto the program line in place of the normal marker. The PLCR routing is not necessary at this time, but it occurs anyway.

A step to 4 phase occurs at the next T6B time. In 4 phase the PLPL line routing writes the address onto the program line immediately behind the Branch command, and the PLCR routing shifts the address into the common register. Then a step to 5 phase occurs.

In 5 phase the line routing changes to PLSR, SRPL, which first writes the normal marker onto the program line behind the address of the Branch instruction. This line routing also serves to again "open up" the line to make room for the marker. The net change in the contents of the program line after phases 3, 4, and 5 is that the normal marker has moved two character positions, from a position in front of the Branch command to a spot directly behind its address.

At program line home the UBR flip-flop sets and a jump occurs from 5 phase to 3M0,4. This is illustrated in Figure 3-131.

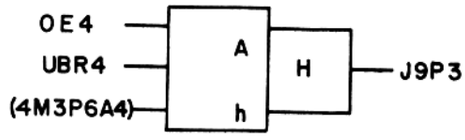


JUMP TO 3M0,4 CAUSED BY "BRANCH" IN 4M5P

FIGURE 3-131

b. DURING UNCONDITIONAL BRANCH. When a Branch command is sensed with the processor already in a branch condition, no marker is written: instead, the branch is completed to the new sequence as dictated by the latest instruction, and the original marker is retained.

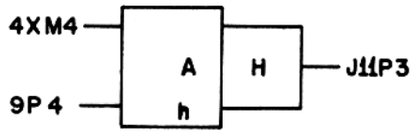
As before, the Branch command is shifted into the operation register (and the common register) in 3 phase, and PLPL3 writes it onto the program line in place of the marker. Then, because UBR is already set, a jump from 3 phase to 9 phase takes place, as shown in Figure 3-132.



DEVELOPMENT OF J9P3 IN 4M3P

FIGURE 3-132

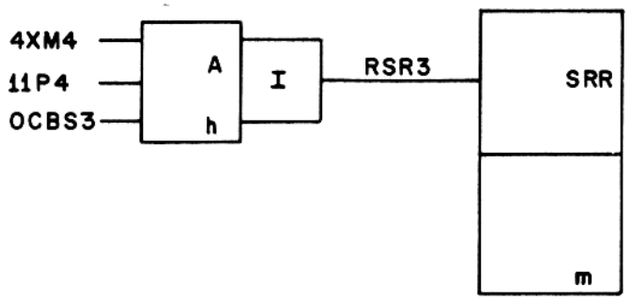
The line routing in 9 phase is exactly the same as that during 4 phase; the Branch address is shifted into the common register and written onto the program line. However, when 9 phase is completed, the logic goes to 11 phase instead of to 5 phase. See Figure 3-133.



DEVELOPMENT OF J11P3 IN 4M9P

FIGURE 3-133

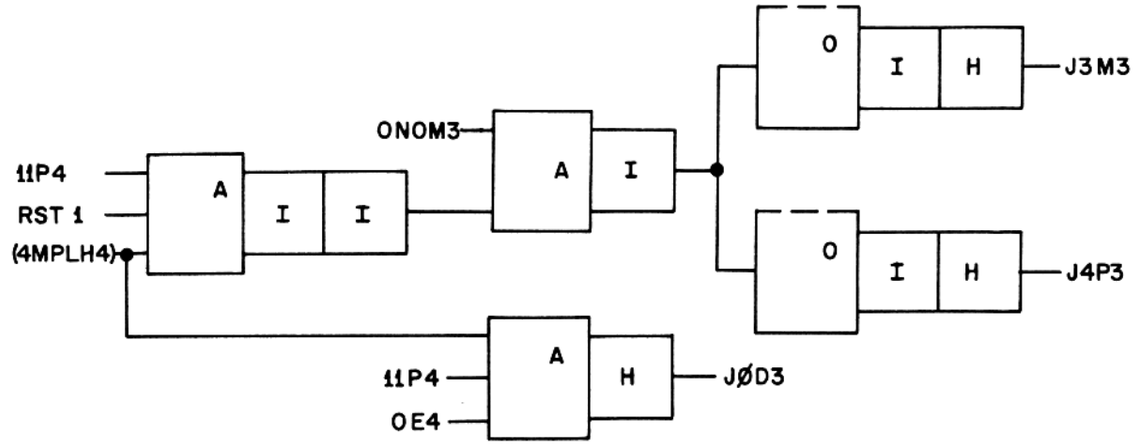
The logic waits for program line home in 11 phase, the same as it would in 5 phase. The important difference lies in the line routings. In 5 phase the line routing was PLSR, SRPL, and the line was "opened up" to allow the rewriting of the marker. In 11 phase the routing is PLPL only, and the balance of the data is rewritten direct. The marker (in this case it is a branch marker) is destroyed by clearing the search register. This is accomplished by developing the Reset Search Register signal, RSR3 (shown in Figure 3-134), which sets SRR to clear the search register.



DEVELOPMENT OF RSR3

FIGURE 3-134

At program line home the jump to 3M0,4 occurs as shown in Figure 3-135.



JUMP TO 3M0,4 FROM 4M11P
(All on "H" card)

FIGURE 3-135

4. UNCONDITIONAL BRANCH SEQUENCE SEARCH

In 3 mode a sequence search is performed to locate the sequence corresponding to the title in the common register. This search is identical to a normal sequence search except for the fact that it is possible for a normal marker to be detected at this time. When the correct sequence is located, a branch marker is inserted into the program immediately behind the title of the new sequence.

Because UBR is set before 3 mode is reached, the unconditional branch sequence search is the same regardless of whether it is the first Branch instruction or not.

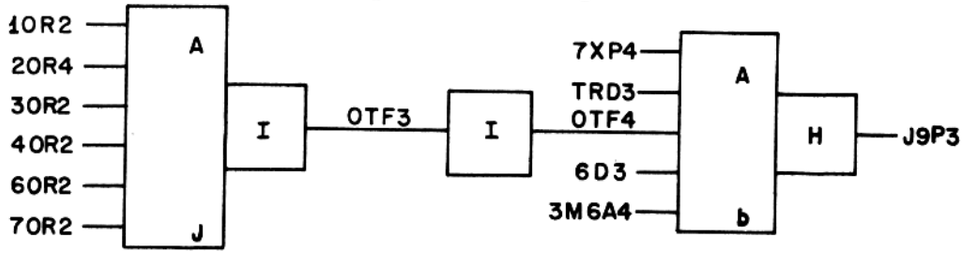
If an invalid sequence title is specified as the Branch address, the error sequence will be located instead, the same as during normal sequence search. (Note, however, that if the error sequence contains a Return command, execution of the Return will cause operation to resume back at the normal marker.) Refer to Section 3G, Search, for review.

a. ENABLING THE COUNTER. ECTR3 remains true throughout the search, until a valid compare is made, unless an invalid sequence title is specified. Refer again to "Search" if necessary.

b. GENERAL OPERATION. Unconditional branch sequence search begins in 3M0,4 at program line home. When the sync pulse is sensed, the timing chain restarts and steps to 5 phase at the end of the buffer, and to 6 phase at the end of the first "blank". The title of the new sequence is written into the second "blank" during 6 phase, and at the end of the second "blank" the phase counter jumps to 9 phase. Phases 7 through 12 are the same as in normal sequence search except for the fact that a normal marker may be sensed at this time. One additional gate eliminates any problem this might cause. When a valid compare is made in 12 phase, a step to 13 phase occurs and a branch marker is written out. At program line home a jump to 3M6,4 occurs and marker search begins. In this case the search will be for the branch marker because the UBR flip-flop is set.

c. PRELIMINARY PHASES. When the sync pulse is sensed in 3M0,4 the timing chain restarts and the phase counter steps to 5 phase at the end of the buffer. With PLPL3 still true, the code in the first blank (the title of the sequence containing the Branch instruction) is simply rewritten. At the end of the first blank, a step to 6 phase occurs. This drives both CRPL3 and CRCR3 true, and the title of the new sequence is written into the second blank. At the end of the second blank, the phase counter makes its usual jump from 6 phase to 9 phase.

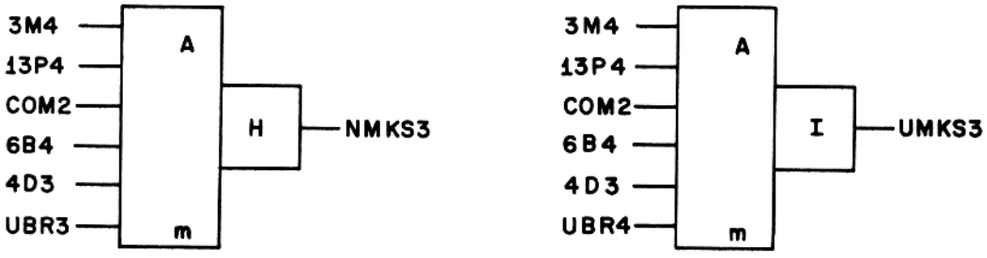
d. BASIC SEARCH PATTERNS. The "basic search patterns" operate in phases 7 through 12 until a valid compare is made in 12 phase, or until program line home is reached (invalid sequence). There is, however, one additional item to be considered at this time: the possibility that a normal marker may be detected. (Note that a normal marker can be sensed in 3M0,7 ONLY with UBR set: if UBR is reset there can be no normal marker on the program line.) This could cause a problem, because the normal marker contains the 1-6-7 bits and does not reset SAD. With SAD set, the normal jump from 7 phase to 9 phase is blocked. However, when the marker is in OR, the Operation Tape Feed signal, OTF3, becomes true, and develops a jump to 9 phase through a different gate, shown in Figure 3-136.



DEVELOPMENT OF J9P3 CAUSED BY NORMAL MARKER DURING UNCONDITIONAL BRANCH SEQUENCE SEARCH

FIGURE 3-136

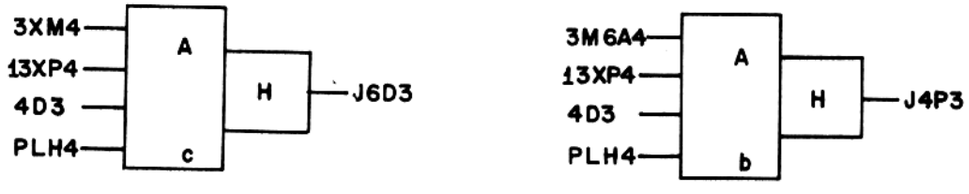
e. WRITING OUT THE BRANCH MARKER. When a valid compare is made in 12 phase, a step to 13 phase occurs. At the beginning of 13 phase, NMKS3 is blocked and UMKS3 is developed instead, as shown in Figure 3-137. UMKS3 sets 2SR, 4SR, and 7SR, and resets the other SR flip-flops. Then, as the line routing changes to PLSR, SRPL, the branch marker is written onto the line.



DEVELOPMENT OF UMKS3

FIGURE 3-137

13 phase remains true until program line home, at which time a jump to 6 decimal and 4 phase occurs, as illustrated in Figure 3-138. This 3M6,4 setting is the beginning of marker search, and with UBR set, the branch marker will be located instead of the normal marker.



JUMP TO 3M6,4 AT END OF BRANCH SEQUENCE SEARCH

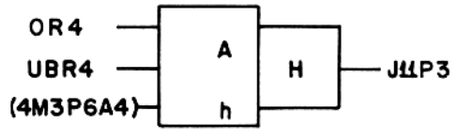
FIGURE 3-138

The Flow Chart in Figure 3-157 (located at the end of this section) summarizes all unconditional branch operations.

J. "RETURN" COMMAND

Retrieval of a Return command with the UBR flip-flop reset develops ONOP3 and the command is simply bypassed. On the other hand, if UBR is set, retrieval of a Return command both resets UBR and destroys the branch marker. The marker search which follows causes operation to resume at the point after the original Branch instruction, where the normal marker was left.

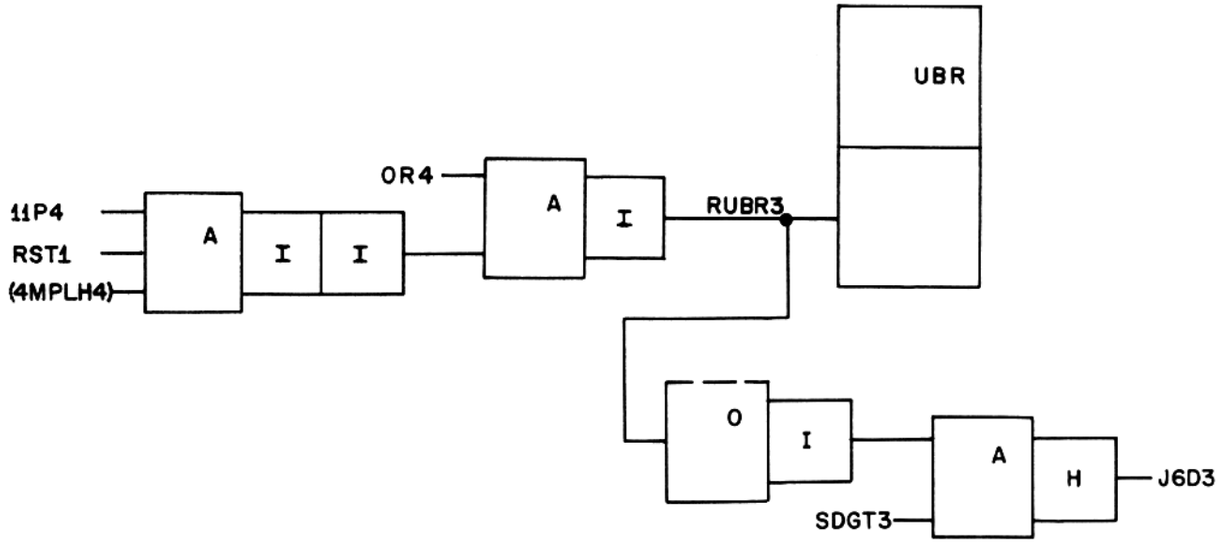
When a Return command is retrieved with UBR set, a jump to 11 phase occurs, as shown in Figure 3-139.



JUMP TO 11P CAUSED BY "RETURN" IN 4M3P

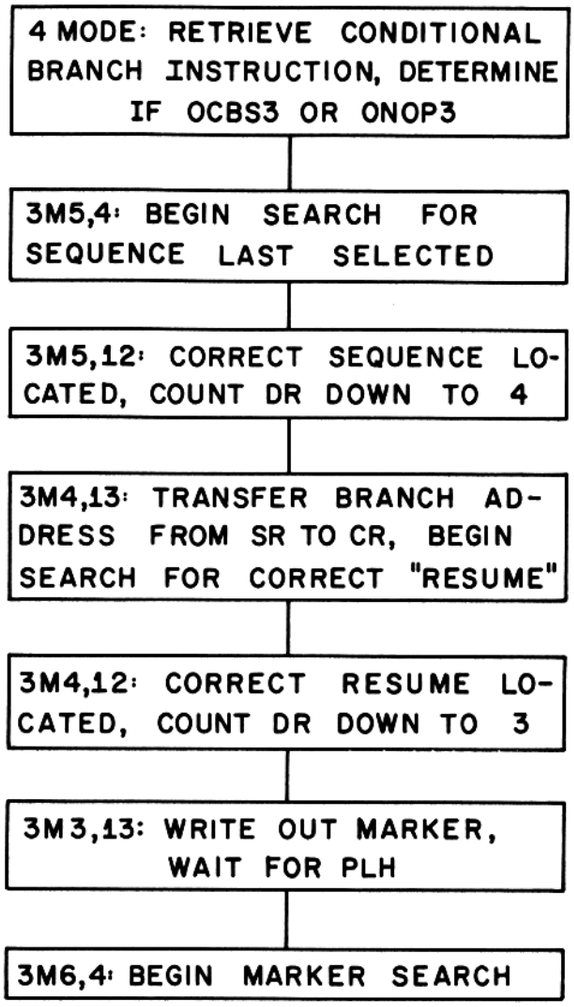
FIGURE 3-139

In 11 phase the PLPL line routing prevents the rewriting of the branch marker, and RSR3 is developed (refer to Figure 3-134), which sets SRR to clear the branch marker out of the search register. Then, at program line home, RUBR3 is developed (see Figure 3-140), which resets UBR and develops the jump to 6 decimal. The jump to 3 mode and 4 phase is the same as shown earlier in Figure 3-135.



DEVELOPMENT OF RUBR3 AND J6D3 IN 4M11P
(All on "H" card)

FIGURE 3-140



FLOW CHART
CONDITIONAL BRANCH - Basic Operation

FIGURE 3-141

K. CONDITIONAL BRANCH

When a conditional branch instruction (either Positive Branch or Negative Branch) is retrieved, the processor first compares the command to the sign of Register 1. If the sign and the command are different, the instruction is treated as a "No-Op" and is bypassed. If the sign of R1 and the command agree, however, the processor develops the "Op" Conditional Branch Signal, OCBS3, interrupts the normal sequence of instructions being performed, and jumps to a point in the program specified by the address of the conditional branch instruction. This point in the program is identified by a Resume instruction having an address identical to that of the conditional branch. For example, if the sign of R1 is positive, the "Positive Branch A" instruction will cause the processor to jump to a point in the program immediately behind "Resume A".

Also, should there be more than one "Resume A", the processor will give preference to the first one located in the sequence containing the conditional branch command. In other words, once the instruction has been initiated, a search for "Resume A" is conducted, starting at the

beginning of the sequence containing the conditional branch command. Operation then restarts at the first "Resume A" sensed. If there is no "Resume A" located in this sequence, the search continues through the balance of the program until a "Resume A" is found.

The Flow Chart in Figure 3-141 illustrates the basic action involved in a conditional branch.

There are two major differences between conditional branch and unconditional branch. First, as denoted by the names, Positive Branch and Negative Branch can occur only if Register 1 contains the proper sign, while unconditional branch occurs regardless of sign. Second, conditional branch instructions do not automatically return the program to the original branching point after their completion, as occurs during unconditional branch when a Return command is sensed.

Other differences are that the conditional branch instructions cause operation to resume at a specified point in the program, not necessarily at the beginning of a sequence. Once a conditional branch instruction is executed and operation re-commences at the proper Resume instruction, the conditional branch is finished; there is no flip-flop similar to UBR specifying that a conditional branch has occurred. In other words, operation is back to normal once the conditional branch instruction is complete.

A conditional branch can occur within an unconditional branch. In this case, the Resume search will start at the beginning of the sequence branched to during the unconditional branch, and not at the head of the original sequence selected.

An invalid Resume title cannot be specified as the address of the conditional branch instruction; if this should occur the processor will stall in an endless search for the non-existent Resume, and it will be necessary to completely restart the machine in order to get out of this condition.

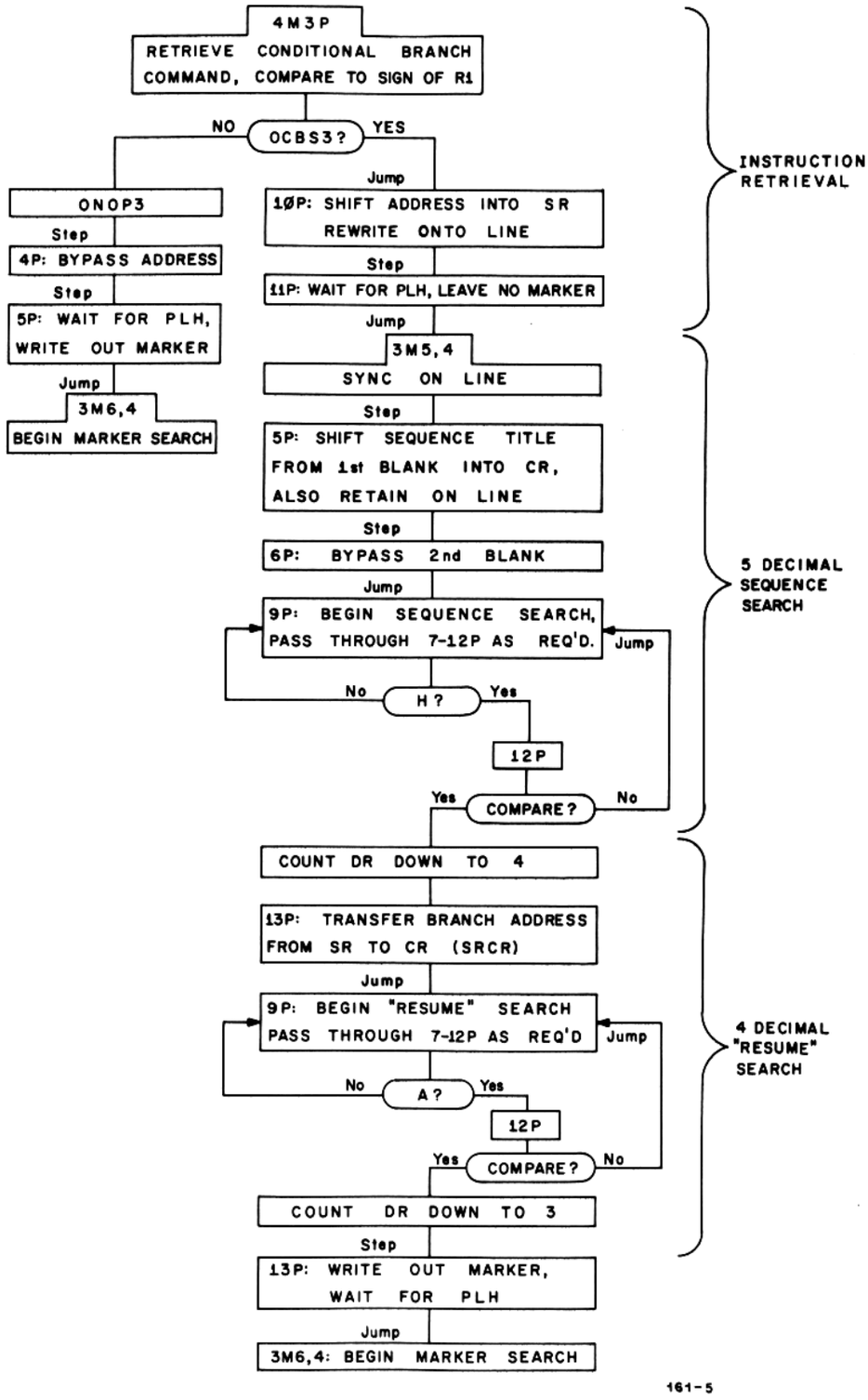
1. GENERAL OPERATION

When a conditional branch is retrieved in 4 mode 3 phase the logic immediately compares the command to the sign of R1. If the command and the sign are different, the command is treated as a "No-Op", and is bypassed as explained in Section 3H, Instruction Retrieval. If the command and the sign of R1 agree, however, a jump to 10 phase occurs, where the address is shifted into the search register. Next the logic steps to 11 phase to wait for program line home, at which time a jump to 3M5,4 occurs.

The 3 mode 5 decimal setting is used to search for the beginning of the sequence being executed. The title of the sequence to be located is shifted into the common register from the first "blank" on the program line. For example, if sequence "D" was selected by the operator, a "D" was loaded into the first blank during normal sequence search. If a conditional branch is then executed, the "D" will be shifted into the common register from the first blank and then a search for sequence "D" will take place. Note, however, that once sequence "D" is located, a marker is not written out, as would normally be the case. Instead, a search begins immediately for the Resume instruction having an address the same as the conditional branch instruction.

The 3M5,4 setting remains true until after the sync pulse is sensed. At the end of the buffer, a step to 5 phase occurs and the address is shifted from the program line into the common register. (It is also rewritten onto the line at the same time.) Next, a step to 6 phase occurs, and at the end of the second "blank" the logic jumps to 9 phase.

In 9 phase the basic search patterns take over and the logic passes through phases 7, 8, 9, 10, 11, and 12 as required until a valid compare is made in 12 phase. This denotes the beginning of the original sequence, and immediately the decimal register is counted down to 4. A step to 13 phase occurs, where SRCR is produced to transfer the conditional branch address from the search register to the common register.



FLOW CHART
CONDITIONAL BRANCH - Normal Operation

FIGURE 3-142

Next, a jump to 9 phase occurs, where the 4 decimal setting conducts a "Resume" search: the basic search patterns are slightly modified to cause a jump to 12 phase when each Resume command (letter "A") is sensed. In 12 phase the Resume address on the program line is compared to the branch address in the common register. If they are not the same, a jump back to 9 phase occurs and the search continues. When the correct address is located and the two addresses do compare, a step to 13 phase occurs, where a marker is immediately manufactured and written out. 13 phase remains true until program line home, where a jump to 3M6,4 occurs and marker search begins.

The Flow Chart in Figure 3-142 illustrates the normal operation of the conditional branch instruction.

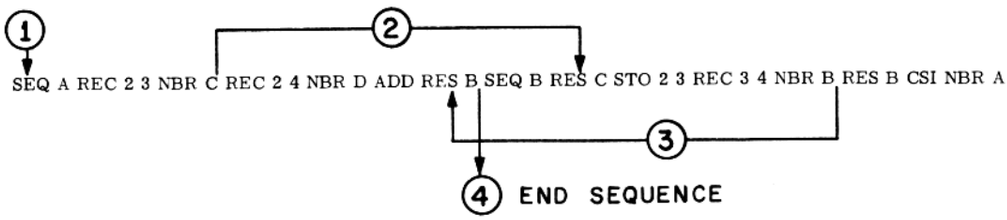
If program line home should be reached during the search for the Resume, a jump to 14 phase occurs. 14 phase remains true until after the sync pulse; a jump to 5 phase occurs at the end of the buffer. The two "blanks" cause successive steps through 5 phase and 6 phase, and at the end of the second "blank" a jump to 9 phase occurs and the "Resume" search begins again. In this case, no data changes in the first two blanks or in the common register. This operation is necessary when the Resume instruction being sought is located on the program line in a sequence loaded ahead of the sequence containing the conditional branch instruction.

Conditional branch search is only slightly different if the UBR flip-flop is set. In this case the sequence address searched for is shifted into the common register from the second "blank" instead of the first. Thus, the Resume search will start at the beginning of the sequence to which the last unconditional branch was made.

It is important to note that the Resume search does not always start in the sequence that contains the conditional branch command. For example, assume that a conditional branch command causes the program to jump to a Resume in a different sequence. If a second conditional branch command is then executed, the Resume search will start at the beginning of the "first" sequence, because its title is still stored at the head of the program line.

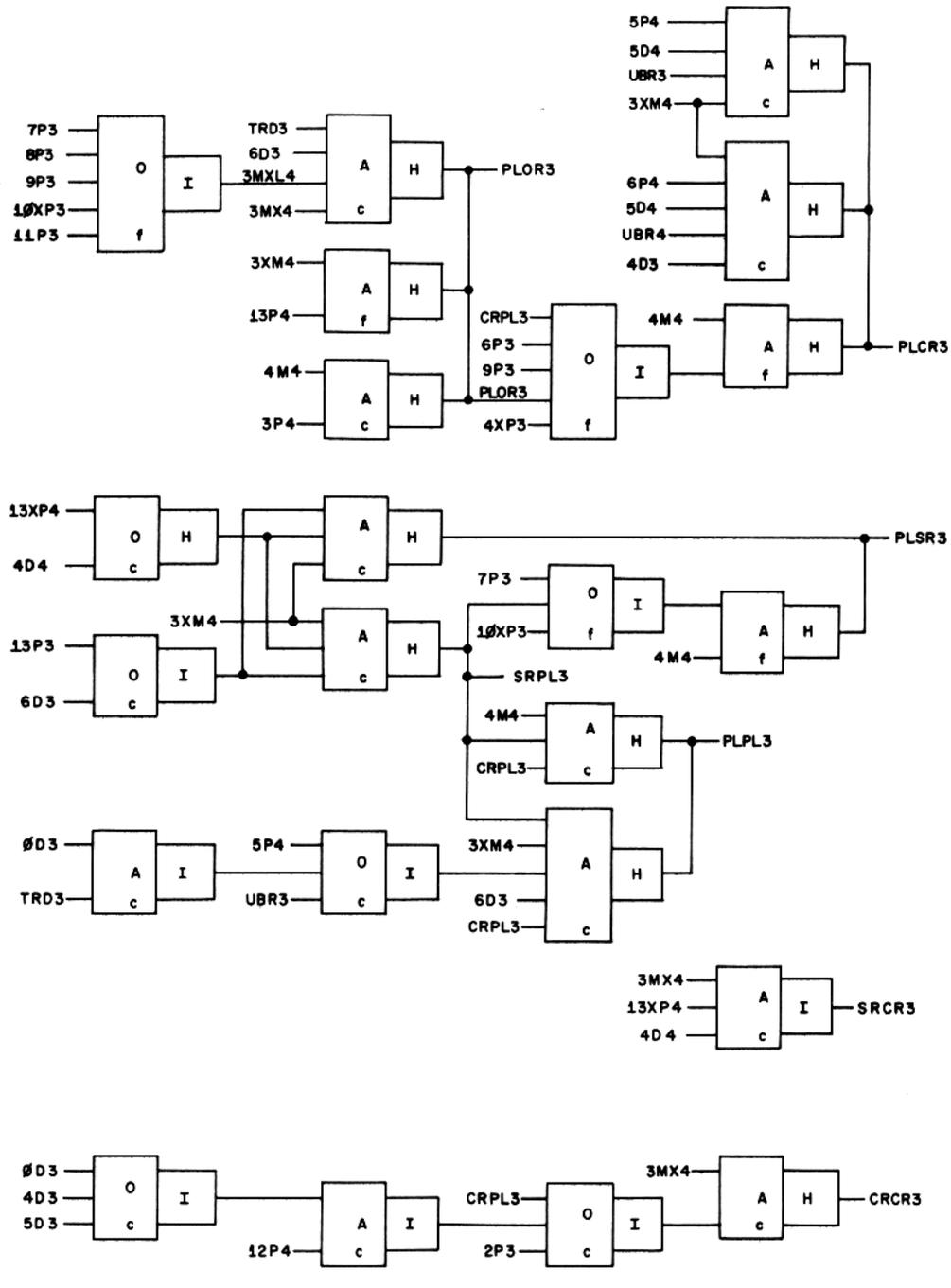
An example is shown in Figure 3-143. The order of events is as follows:

(1) Sequence "A" is selected.
(2) NBR C causes a jump to RES C in SEQ B.
(3) NBR B does not recognize the very next instruction (RES B), since the Resume search starts at the beginning of SEQ A, because the "A" is stored in the first "blank" at the head of the program line.
(4) With no more instructions following RES B, the sequence ends.



SAMPLE PROGRAM
"RESUME" NOT IN SAME SEQUENCE

FIGURE 3-143



4M3P	PLPL, PLOR, PLCR	3M4,13	PLPL, PLOR	161-7
10P	PLPL, PLSR	4-11	PLPL, PLOR	
11P	PLPL	12	PLPL, CRCR	
3M5,4	PLPL	3M3,13	PLSR, SRPL	
5	PLPL, PLCR*			
6	PLPL*			
7-11	PLPL, PLOR			
12	PLPL, CRCR			

*With UBR set, PLCR occurs in 3M5,6 instead of 3M5,5

LINE ROUTINGS USED IN CONDITIONAL BRANCH

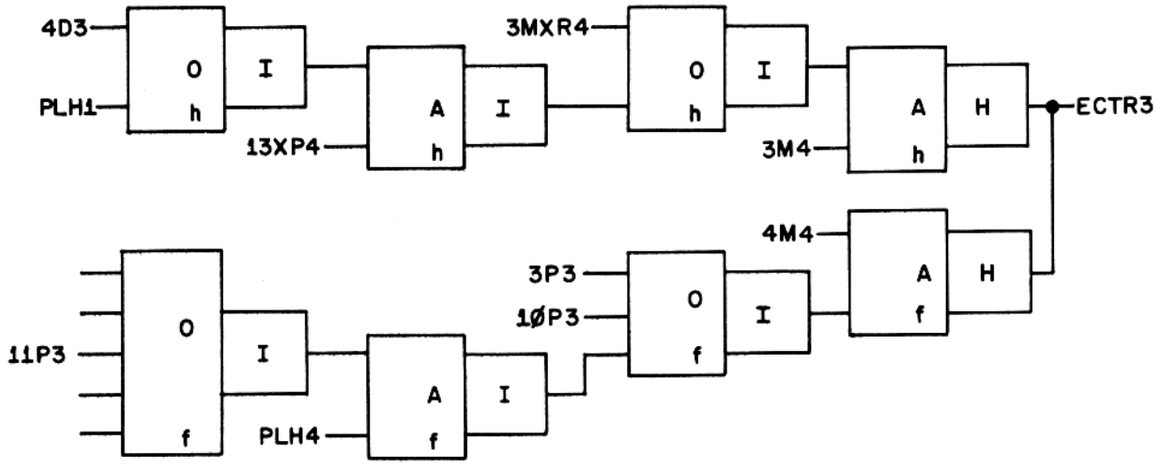
FIGURE 3-144

2. LINE ROUTINGS

The development of all of the line routings used in conditional branch is illustrated in Figure 3-144. Most of the gates shown have been used before, but the importance of some of them now becomes more apparent. For instance, it can now be seen that 4 decimal must disable PLSR and SRPL in 13 phase to allow the "Resume" search to function. Note also that the gate developing SRCR3 uses an "I" inverter, denoting that this is the only time SRCR is used. Last, note the two gates that produce PLCR3 in either 5 phase or 6 phase of 3 mode, depending upon the state of UBR: the UBR flip-flop determines whether the sequence title is taken from the first or the second blank.

3. ENABLING THE COUNTER

All of the gates used to develop ECTR3 in conditional branch are illustrated in Figure 3-145. ECTR3 is automatic in all phases except 3 mode 13 phase and 4 mode 11 phase. In 13 phase, ECTR3 is automatic only if 4D3 is true: the phase counter waits for program line home if the decimal register is at some other setting (3, in this case). The phase counter also waits for program line home before stepping off of 11 phase in 4 mode.



DEVELOPMENT OF ECTR3 IN CONDITIONAL BRANCH

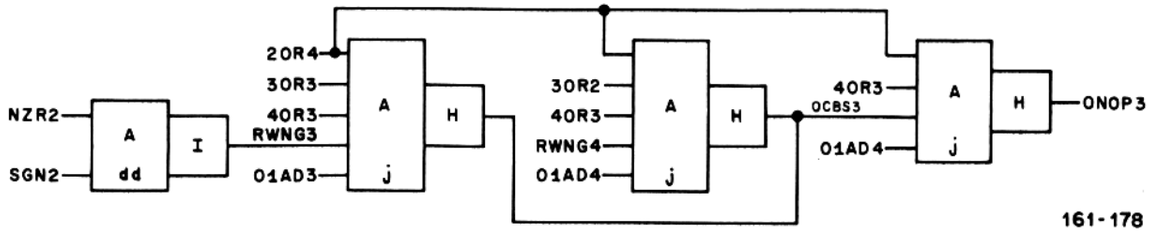
FIGURE 3-145

4. RETRIEVING THE CONDITIONAL BRANCH INSTRUCTION

The retrieval of the conditional branch instruction actually involves three important steps: retrieval of the command, determining whether or not the branch will occur, and retrieval of the address.

a. COMMAND RETRIEVAL. The conditional branch command is shifted into both the operation register and the common register in 4 mode 3 phase, the same as any other command. Again, it is not necessary to have the command in the common register, but it does no harm, so it is not prevented.

b. COMPARING COMMAND TO SIGN OF R1. As soon as the command is completely shifted into the operation register, a test is performed to determine whether OCBS3 or ONOP3 will become true. OCBS3 is produced as shown in Figure 3-146. The R1 Negative signal, RWNG3, will be true if R1 is negative and contains data, and RWNG4 will be true if R1 is positive or equal to zero. The action of NZR2 and SGN2 in developing RWNG3 is covered near the end of Section 4L, Multiply or Divide.



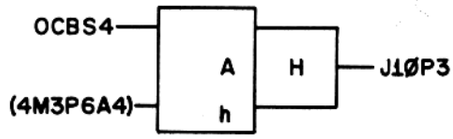
SIGNALS PRODUCED BY RETRIEVAL OF PBR AND NBR

FIGURE 3-146

Assume that R1 is positive and that a Positive Branch command (1-2-5-6-7) has just been shifted into the operation register, setting 1OR, 2OR, 6OR, and 7OR. This makes all of the inputs to the first large AND gate false, and OCBS3 becomes true. Similarly, it can be seen that a Negative Branch command (1-2-3-6-7) will produce OCBS3 if RWNG4 is false (R1 negative).

If neither gate produces OCBS3, the command becomes a "No-op", producing ONOP3. Subsequent operation in this case is the same as explained in Section 3H, Instruction Retrieval. (4 mode).

c. ADDRESS RETRIEVAL. With OCBS3 produced in 4 mode 3 phase, a jump to 10 phase is enabled as shown in Figure 3-147.



DEVELOPMENT OF J10P3 BY CONDITIONAL BRANCH

FIGURE 3-147

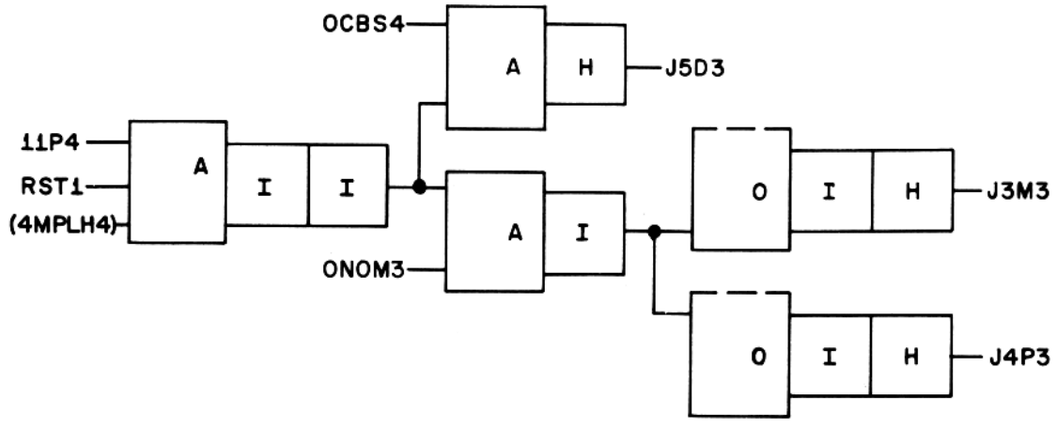
In 10 phase the PLSR line routing shifts the address into the search register and the PLPL routing maintains the address on the program line. As the address is shifted into the search register, the marker is shifted out and lost.

At the next T6B the phase counter steps to 11 phase and recirculates the program directly (PLPL) while waiting for program line home. Note that no marker is written out. The reset of the search register (RSR3) which normally occurs in 11 phase is inhibited at this time by OCBS3. Refer to Figure 3-134.

At program line home in 11 phase, with OCBS3 true, a jump to 3M5,4 occurs, as shown in Figure 3-148.

5. SEQUENCE SEARCH

The 3M5,4 setting remains until the sync pulse is sensed and the timing chain restarts. At the end of the buffer a step to 5 phase occurs and the PLCR line routing shifts the code from the first "blank" on the program line into the common register. Note that as this sequence title is shifted into the common register, the command, which was shifted in during 4 mode 3 phase, is shifted out. PLPL maintains the new sequence address on the line in the first "blank".



JUMP TO 3M5,4 CAUSED BY CONDITIONAL BRANCH
(All on "H" card)

FIGURE 3-148

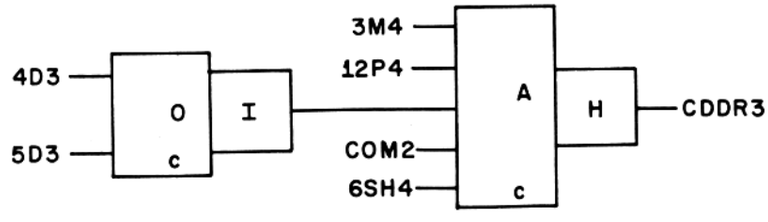
At the end of the "blank" the logic steps to 6 phase and idles through the second "blank". At the end of the second "blank" the familiar jump to 9 phase occurs.

The basic search patterns take over in 9 phase and guide the phase counter through phases 7, 8, 9, 10, 11, and 12 as required by the program data. This portion of the search is identical to normal sequence search: each normal sequence separator causes a jump to 12 phase where the next code off the program line is compared to the code in the common register. If the codes are different, a jump back to 9 phase occurs and the search continues.

When the proper sequence is finally located, the operation is slightly different from that in normal sequence search. First, the decimal register is counted down, and then, instead of writing out a marker, a search for the proper Resume instruction is initiated.

6. COUNT DOWN DECIMAL REGISTER

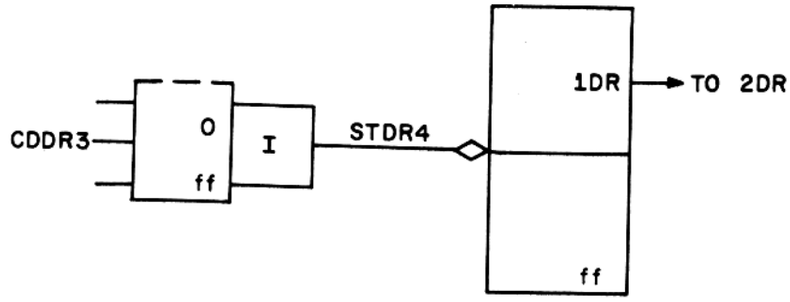
When a valid compare is made in 12 phase, the Count Down Decimal Register signal, CDDR3, is produced as shown in Figure 3-149. Note that CDDR3 is not produced unless COM is still set at 6SH time in 12 phase.



DEVELOPMENT OF CDDR3 DURING CBR SEARCH

FIGURE 3-149

CDDR3 is applied to an OR gate and causes STep Decimal Register, STDR4, to become false. See Figure 3-150. When CDDR3 then goes false (at the end of 6SH time), STDR4 again returns to true and counts the decimal register down from 5 to 4.



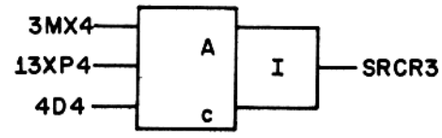
CDDR3 ACTION ON THE DECIMAL REGISTER

FIGURE 3-150

7. "RESUME" SEARCH

At the next T6B time, the phase counter steps to 13 phase. Here two operations occur simultaneously: the conditional branch address in the search register is shifted into the common register (SRCR), and the search for the Resume command is initiated.

Figure 3-151 shows the development of SRCR3 in 3M4,13. This signal is applied to the inputs of the common register and to the shifting elements of both the common register and the search register. As the program line timing chain steps through 13 phase, each SH3 transfers a bit from the search register into the common register. Since there is no input to the search register, it is left reset at the end of 6 shifts.

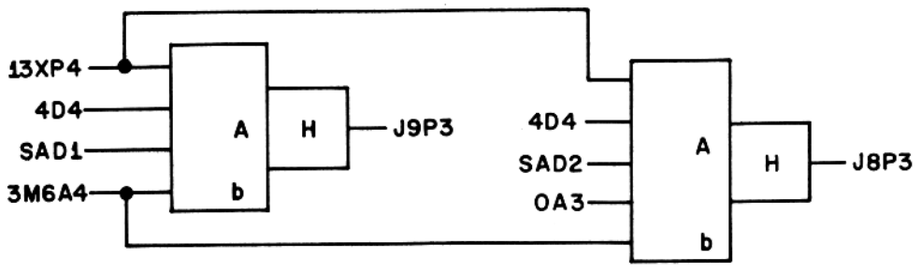


DEVELOPMENT OF SRCR3

FIGURE 3-151

Note that SRCR is not actually a "line routing", because data is not being transferred to or from a delay line. However, the operation is controlled by the program line timing chain, and for all practical purposes it can be considered as a "line routing".

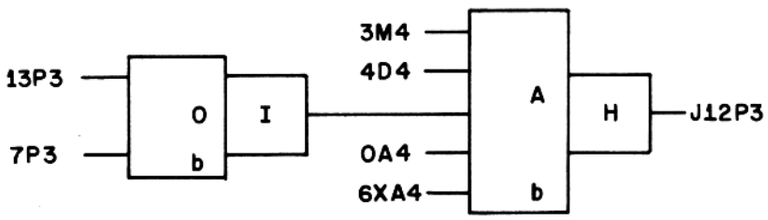
At the end of 13 phase a jump is made back to 9 phase (or 8 phase) to begin the search for the Resume command. Normally, J9P3 will be produced, but it is possible for the code sensed in 13 phase to be a 1-address code. In this case, the jump would be to 8 phase in order to bypass the address digit: a 1-address code would leave SAD set, inhibiting J9P3 and enabling J8P3. The development of these two signals is illustrated in Figure 3-152.



LEAVING 13 PHASE AND 4 DECIMAL

FIGURE 3-152

Notice also in Figure 3-152 that OA3 inhibits the jump to 8 phase. This is in case the first command in the sequence (the code sensed in 13 phase) happened to be a Resume. In this case, the jump would not be to 8 phase or 9 phase, but to 12 phase. This is illustrated in Figure 3-153.



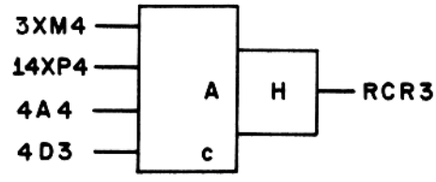
J12P3 CAUSED BY SENSING A "RESUME" IN 13P

FIGURE 3-153

The case just mentioned is the reason for requiring the PLOR line routing in 13 phase. This was mentioned in the normal sequence search discussion, but no reason for it was given at that time. Note, then, that PLOR3 is required in 13 phase only due to the possibility that a Resume command can be the first command in the sequence. Note also that in such a case, J9P3 is blocked by the SAD flip-flop (Figure 3-152), which is not reset by the Resume.

If a Resume is not detected immediately, the basic search patterns take over and conduct a search similar to normal sequence search. However, a normal sequence separator no longer causes a jump to 12 phase because of the 4 decimal setting. Note that OH3 still blocks J9P3, so when a normal sequence separator is sensed the logic steps to 8 phase to bypass the sequence address. Also, each Resume command sensed does cause a jump to 12 phase (refer to Figure 3-153), where the next code off the line is compared with the branch address in the common register. If COM becomes reset in 12 phase, a jump back to 9 phase occurs and the search continues.

Should program line home be reached before locating the correct Resume command, a jump to 14 phase occurs, the same as in normal sequence search. In 14 phase and 4 decimal, however, 4D3 blocks the development of RCR3 (see Figure 3-154) and the common register does NOT become reset.

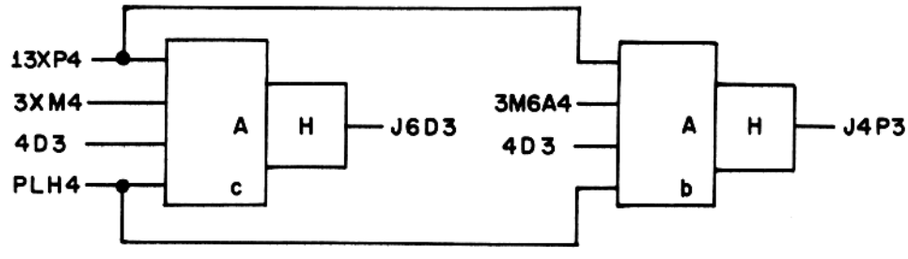


DEVELOPMENT OF RCR3 IN 3M14P

FIGURE 3-154

At the first T6B after the timing chain restarts, the phase counter jumps to 5 phase, as in normal sequence search. The logic steps to 6 phase for the second "blank". Note that the line routing remains PLPL throughout phases 14, 5, and 6, so no change is made in the data at the head of the line. At the end of the second "blank" the jump to 9 phase occurs again and the "Resume" search continues.

When the correct Resume instruction is located in 12 phase, the COM flip-flop remains set and the decimal register is again counted down, the same as before. The logic then steps to 13 phase, and with 4D3 no longer true, a marker is immediately set into the search register and the PLSR, SRPL line routing becomes true to write the marker out onto the line. 13 phase remains true until program line home, when a jump to 3M6,4 occurs to initiate marker search. However, the gates are illustrated here again (Figure 3-155) to point out the preventive action of 4D3, should program line home be reached before locating the proper Resume, as explained previously.



JUMP TO 3M6,4 AFTER COMPLETING CONDITIONAL BRANCH

FIGURE 3-155

Note that once marker search begins, the system is back in normal operation, with nothing to tell it that a conditional branch has occurred. Note also that many parts of the two foregoing searches are similar to normal sequence search; for gates not pictured here, refer to that section.

8. EFFECTS OF UBR

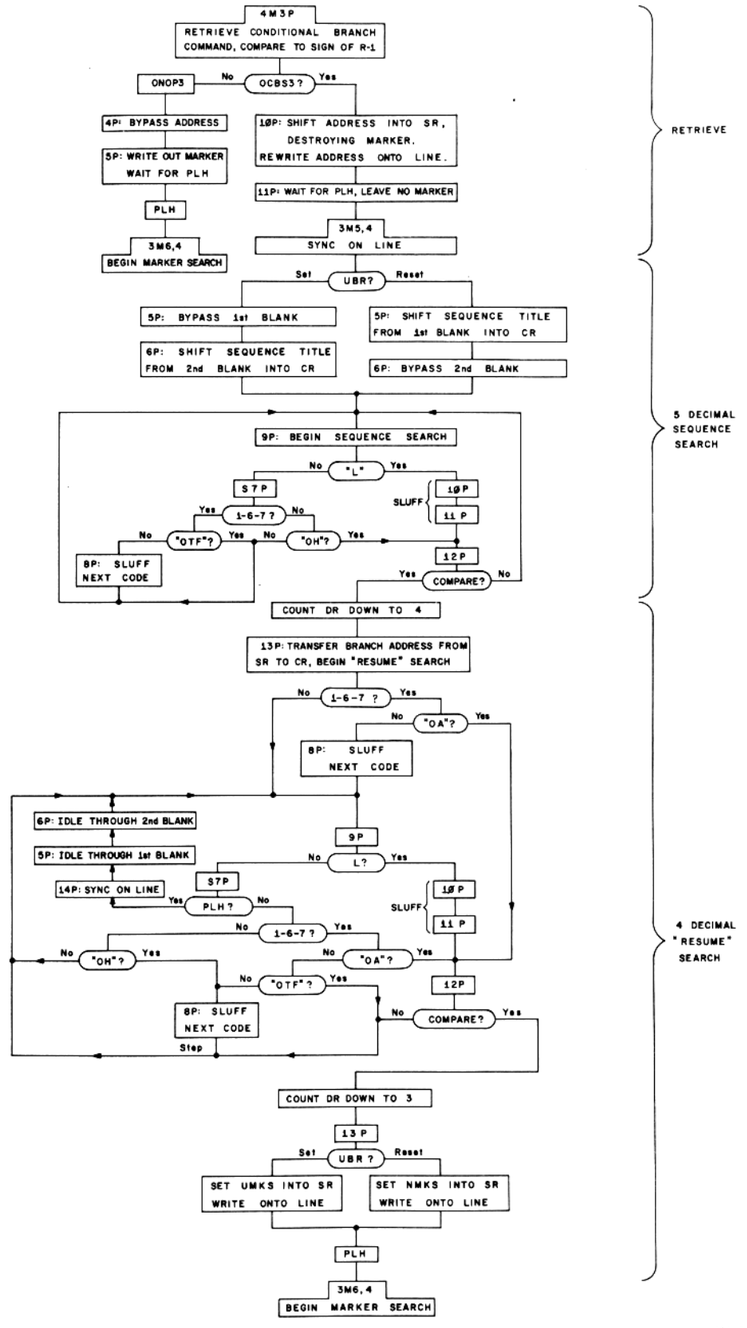
If a conditional branch instruction is executed while in an unconditional branch condition (UBR set), there are only two minor differences in the conditional branch operation. First, the sequence title sought out is not taken from the first "blank" on the program line, but from the second "blank". This is so the sequence already branched to will be looked at first when searching for the Resume. Second, when the correct Resume is located, a branch marker, instead of a normal marker, is set into the search register and written onto the line.

9. "RESUME" COMMAND

It was stated earlier that the Resume command, when retrieved in 4 mode 3 phase, is always treated as a "No-op", developing ONOP3. Now that conditional branch has been covered, the reason for saying this is much clearer. The Resume only has a function in 3 mode; any time it is retrieved in 4 mode, it must be ignored.

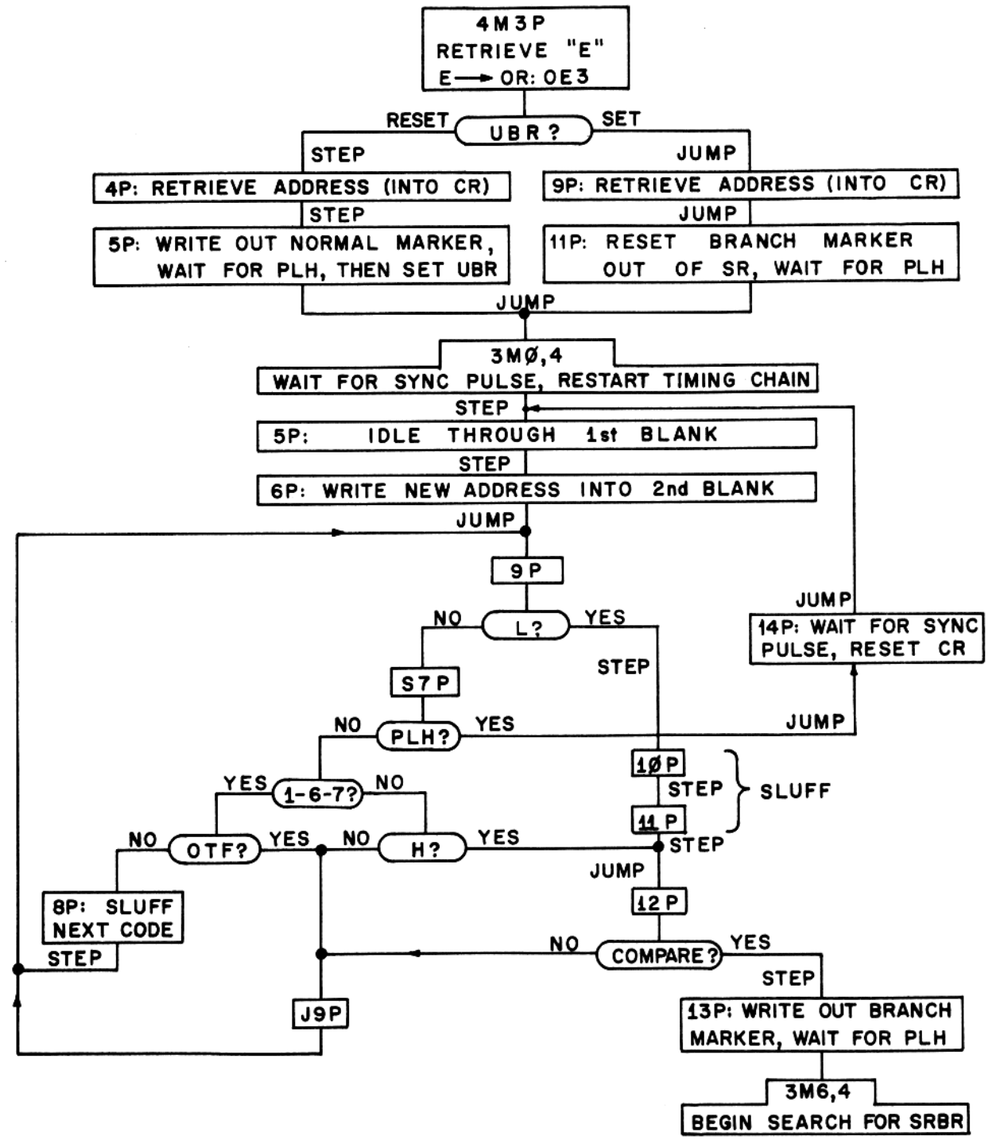
10. FLOW CHART

The Flow Chart in Figure 3-156 illustrates the complete operation of the conditional branch instruction.



FLOW CHART
CONDITIONAL BRANCH - Complete Operation

FIGURE 3-156



FLOW CHART
UNCONDITIONAL BRANCH

FIGURE 3-157

TP-161SL

TABLE OF CONTENTS

Page

SECTION 4, ARITHMETIC LINE LOGIC
ARITHMETIC DELAY LINE 4-1
Data Organization 4-1
Arithmetic Timing Chain 4-3
General Operation 4-9
Clear R0 4-17
Short and Long Data Paths 4-18
Shift Up/Shift Down 4-21
Shift Left 4-23
ACTION STEP COUNTER 4-27
DOST Flip-flop 4-27
Decoded Outputs 4-30
Two or Five Steps 4-30
Development of the Action Step Shift Pulse 4-33
Summary 4-36
Arithmetic Operations Chart 4-36
TRUNCATE 4-39
General Operation 4-39
4 Mode Operation 4-39
5 Mode Operation 4-39
Timing Chart 4-42
Flow Chart 4-43
DELETE 4-47
General Operation 4-47
4 Mode Operation 4-47
5 Mode Operation 4-47
DUPLICATE 4-51
General Operation 4-51
4 Mode Operation 4-51
5 Mode Operation 4-52
Timing Chart and Flow Chart 4-54
CHANGE SIGN 4-57
General Operation 4-57
4 Mode Operation 4-57
5 Mode Operation 4-57
Illustrations 4-58

	Page
ENTER	4-63
General Operation	4-63
Enabling the Phase Counter	4-64
Retrieval of the Enter Instruction	4-65
Preliminary 5 Mode Operations	4-66
Repetitive Operations	4-66
Decimal Point	4-72
Right-of-Decimal	4-72
Terminating Enter	4-72
Variations	4-77
Flow Chart	4-91
PRINT	4-95
General Operation	4-95
Retrieval of the Print Instruction	4-96
Preliminary 5 Mode Operations	4-96
Repetitive Operations	4-101
Zero to Space Conversion	4-106
Decimal Considerations	4-106
Printing Right-of-Decimal	4-109
Terminating Print	4-109
Flow Chart	4-112
CONSTANTS	4-117
General Operations	4-117
Constant Command	4-117
Constant Factor Digits	4-119
Terminating the Constant Factor	4-124
Summary	4-128
BASIC MATHEMATICAL CONCEPTS	4-129
Binary Arithmetic	4-129
Binary Coded Decimal System	4-132
ADD OR SUBTRACT	4-135
General Operation	4-135
Retrieval of the "AOS" Instruction	4-135
Preliminary 5 Mode Operation	4-136
Summary	4-137
Important Concepts	4-138
Carry/Borrow Logic	4-144

Page

ADD OR SUBTRACT (Cont.)
- Summary of Timing 4-146
- Flow Chart 4-149
- Correction 4-149
- Overflow 4-154
- Supplementary Instructional Aids 4-162

MULTIPLY OR DIVIDE 4-169
- General Operation 4-169
- Instruction Retrieval. 4-171
- Action Step Counter Operation. 4-172
- Preliminary 5 Mode Operation. 4-172
- The Multiply/Divide "Loop" 4-174
- Terminate Multiply or Divide 4-182
- Prevent Dividing by Zero 4-183
- Flow Charts 4-185

SECTION 4, ARITHMETIC LINE LOGIC

The arithmetic line logic is very important to the operation of the 5610 because it is associated with so many instructions. Not only are the computational functions of add, subtract, multiply, and divide performed by the arithmetic logic, but the data control instructions (Truncate, Delete, etc.), the input/output instructions, and the storage-oriented instructions rely heavily on this part of the machine. The arithmetic line timing chain is also used to count program characters during load, and the arithmetic logic determines the positive and negative conditions as related to branching.

A. ARITHMETIC DELAY LINE

The arithmetic delay line is a short line, about 3 feet long, which recirculates the data once every 350 μsec. (approx.). Its read and write amplifiers are the same as those used with the other delay lines. (See Section 8 for information on these amplifiers.) The only other physical difference is the fact that the arithmetic line read and write transducers use only one nickel ribbon as opposed to the two ribbons used by the other transducers.

1. DATA ORGANIZATION

Figure 4-1 illustrates the layout of the arithmetic line in the way it is most often discussed. The registers are depicted in a "stack". Each register, R0 through R4, contains eight digits left-of-decimal and 5 digits right-of-decimal. The algebraic sign for each register is found in the A1 time slot. Note that TIME RUNS FROM RIGHT TO LEFT.

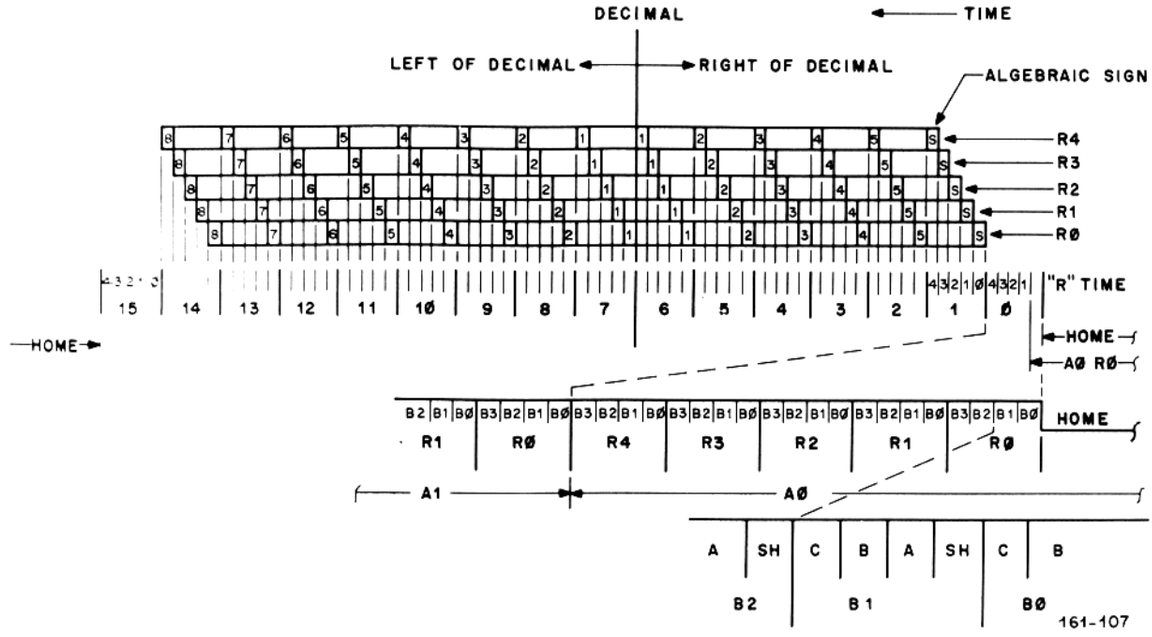
Figure 4-2 is basically a mirror-image of Figure 4-1. It illustrates the same arithmetic line composition with time running from left-to-right, the way most technicians are used to seeing it. Either drawing may be used in becoming acquainted with the layout of the arithmetic line, but if the concept of time moving from right to left can be accepted, Figure 4-1 will be easier to follow in other respects later on.

a. COLUMNS. The arithmetic line can be thought of as being divided into sixteen parts, or columns, numbered A0 through A15. A1 through A15 are equal in length - twenty microseconds. A0 includes, in addition, the "home" time of the arithmetic line, whose length varies slightly between machines due to mechanical differences in delay line length and/or oscillator speed. Columns 2 through 14 make up the thirteen digit columns in each stack register. Column zero is used only to recirculate the sync pulse. Column one contains the algebraic sign of each register, and column fifteen is used to hold intermediate answers in multiply and divide.

b. REGISTERS. Each column is divided into five equal register times, R0, R1, R2, R3 and R4. These register times correspond to the "operating stack registers". Each register, then, has its own time slot in every column.

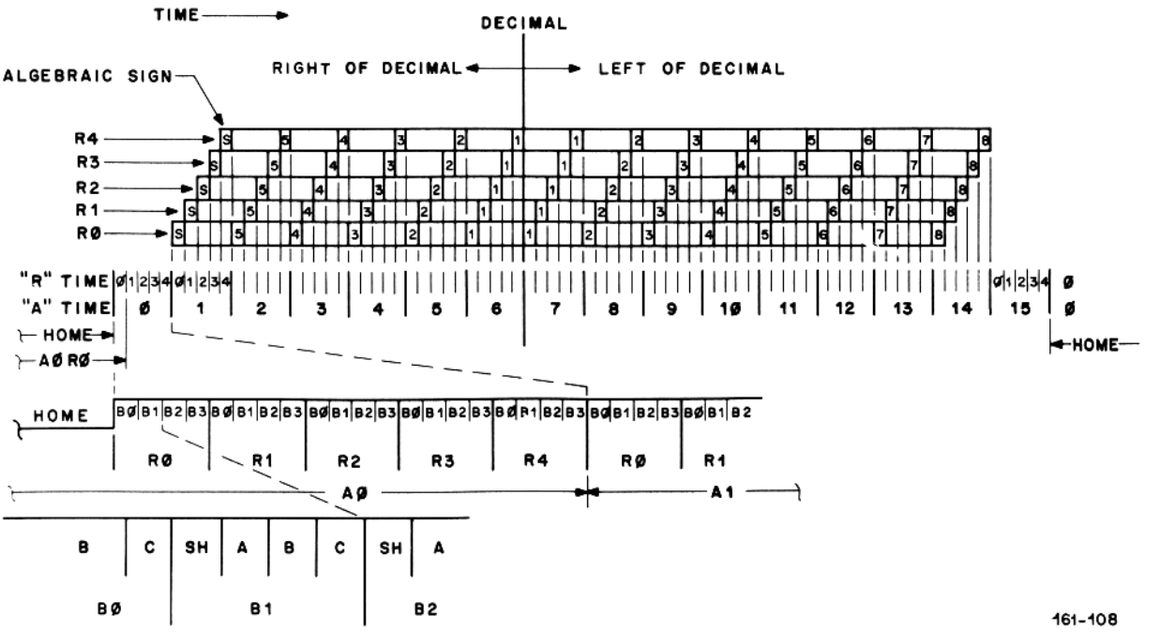
c. BITS. Each register time is further divided into four equal "bit" times, B0, B1, B2 and B3. A digit on the arithmetic line is coded into the bit times. B0 has a binary weight of one. B1 has a binary weight of two, B2 has a weight of four, and B3 has a weight of eight. Pulses at B0 time and at B2 time would represent the digit 5.

d. FRACTIONAL BITS. The bit times are further divided into four fractional bit times, SH (shift), A, B, and C. These fractional bit times are important in changing data from one bit time to the next, as in add, subtract, etc.



ORGANIZATION OF ARITHMETIC LINE

FIGURE 4-1



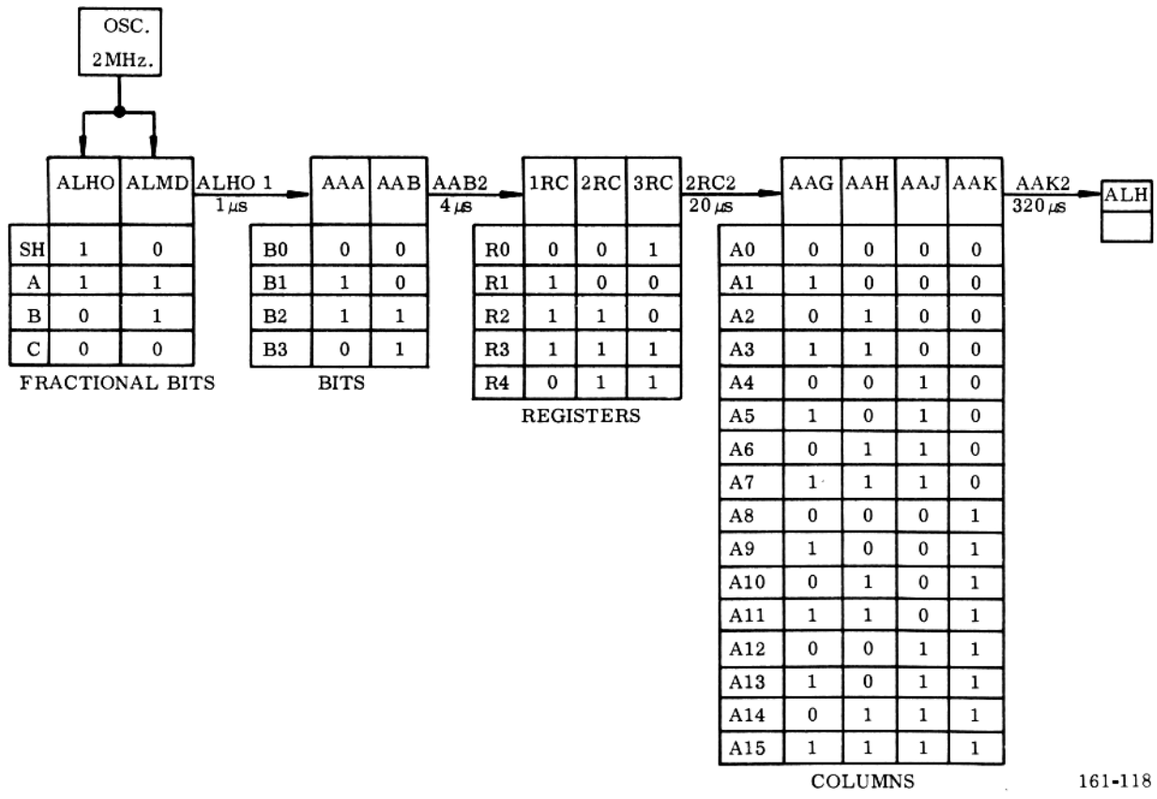
ORGANIZATION OF ARITHMETIC LINE

FIGURE 4-2

2. ARITHMETIC TIMING CHAIN

The important point to remember in this discussion is that data pulses on the delay line are meaningless unless they can be sensed by the read amplifier at a specific time. The TIME at which a pulse is sensed is what determines the value of the pulse and its position in the registers.

All of the times referred to previously are determined by a group of counters, called the arithmetic line timing chain (see Figure 4-3). There are eleven flip-flops arranged in four groups. The ALHO and ALMD flip-flops make up the fractional bit counter, the AAA and AAB flip-flops make up the bit counter, the 1RC, 2RC, and 3RC flip-flops make up the register counter, and the AAG, AAH, AAJ, and AAK flip-flops make up the column counter. The fractional bit counter, the bit counter, and the register counter are series shift counters, and the column counter is a regular binary counter.



ARITHMETIC LINE TIMING CHAIN

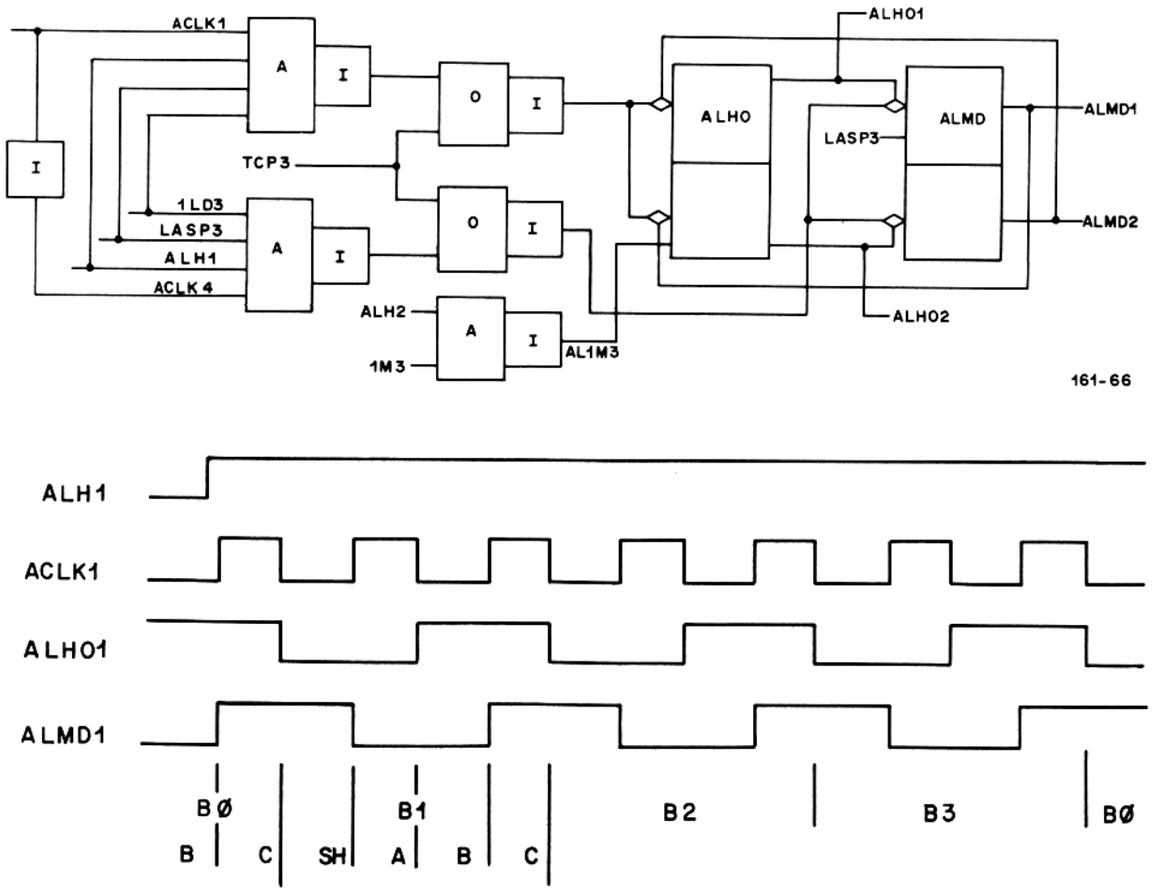
FIGURE 4-3

The entire arithmetic timing chain is driven by a two MHz oscillator. The Arithmetic Line Half Oscillator (ALHO) and the Arithmetic Line MiD oscillator (ALMD) flip-flops are driven by this crystal oscillator. Refer to the truth tables in Figure 4-3 and note that as ALHO1 goes true once every microsecond ("SH" time) it drives the bit counter. As AAB2 becomes true (at B0 time) every four microseconds, it pulses the register counter. 2RC2 becomes true (R0 time) every 20 microseconds and drives the column counter. As the counters leave their "highest" state (A15, R4, B3 time), AAK2 goes true and sets the Arithmetic Line Home flip-flop, ALH.

a. FRACTIONAL BIT COUNTER. ALHO and ALMD are both driven by the 2 MHz oscillator in such a manner as to produce two 1 MHz square waves, the ALHO cycle leading the ALMD cycle by slightly less than 300 nsec. Corresponding waveforms are shown in Figure 4-4.

Figure 4-4 also shows that any one of three signals can prevent the stepping of ALHO and ALMD. ALH1 stops them during arithmetic line home time; LASP3 stops them while the arithmetic line sync pulse is being launched; 1LD3 stops them from being driven (in this manner) during program load. If all of these signals are false, the 2 MHz oscillator will drive the two AND gates alternately, thus alternately toggling the two flip-flops. The outputs of one flip-flop gate the setting and resetting of the other.

The Timing Chain Pulse (TCP3) is produced during program load to allow the arithmetic timing chain to keep count of the number of characters loaded onto the program line. (As noted above, the oscillator cannot drive the timing chain at this time due to 1LD3.)



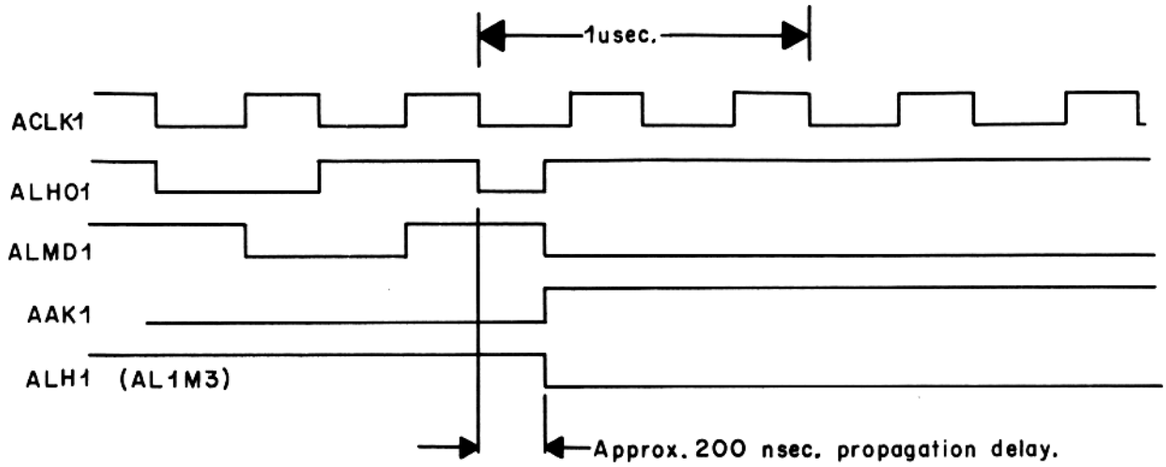
FRACTIONAL BIT COUNTER
(All on "MM" card)

FIGURE 4-4

The oscillator output actually resembles a sine wave, but is shown as a square wave for ease of understanding.

As shown previously (Figure 4-3) when ALHO sets, ALHO1 drives the bit counter. When ALHO sets after A15R4 time it will reset the bit counter. This resets the register counter, which in turn resets the column counter. As the last flip-flop in the chain, AAK, resets, AAK2 sets the

Arithmetic Line Home flip-flop, ALH. However, there is a slight delay from the time ALHO sets to when ALH sets. This is caused by the fact that it takes a finite length of time (approx. 10-20 nsec.) to change the state of a flip-flop. Since a total of eleven flip-flops are setting or resetting at this time, ALH actually sets about 200 nsec. after ALHO sets. This delay as the entire timing chain is resetting is termed "propagation delay", and is illustrated in Figure 4-5.



TIMING CHAIN ACTION AT ALH

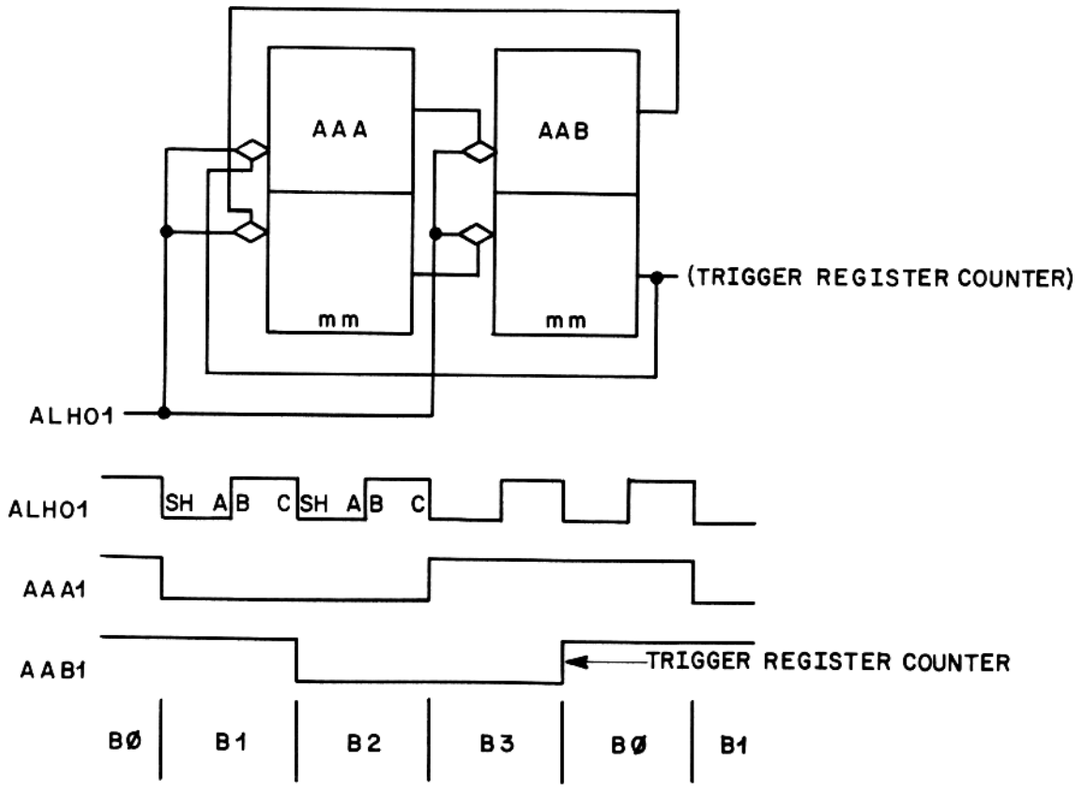
FIGURE 4-5

When ALH sets, ALH1 being applied to the two left-hand AND gates (Figure 4-4) drives the triggers of both ALHO and ALMD true. Since the ALHO trigger was previously true (this is what set ALHO at the end of A15R4), ALHO is not affected by this. However, the ALMD trigger was false, so now ALMD is triggered. Since ALHO has been set for about 200 nsec., the ALMD set gate is enabled, allowing ALMD to set.

Again referring to Figure 4-5, note that ALHO resets as arithmetic line home is reached. This is not caused by the normal trigger, but by AL1M3 (Arithmetic Line home, not 1 Mode). This is illustrated in Figure 4-4.

ALH1 holds the triggers to both ALHO and ALMD true so no counting takes place during arithmetic line home. Since ALHO is reset and ALMD is set, fractional bit "B" time is true during this time. Then, when the sync pulse is sensed by the read amplifier, ALRS3 resets ALH and counting again resumes as shown in Figure 4-4.

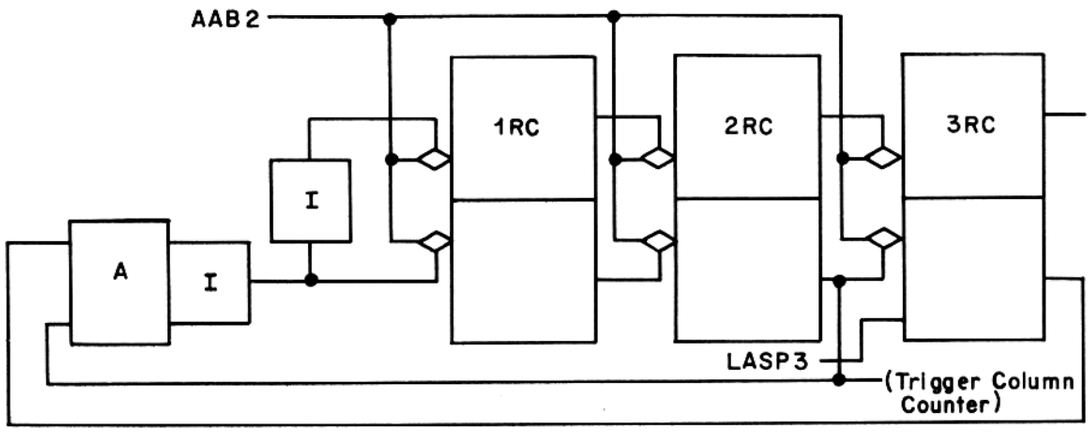
b. BIT COUNTER. The bit counter, composed of the AAA and AAB flip-flops, is a simple two-stage series shift register with the outputs of AAA gating the inputs to AAB and the outputs of AAB gating the opposite inputs to AAA. It operates the same as the program line bit counter except that it has only 4 states, while the program line bit counter has 6. Both flip-flops are driven by ALHO1. At the beginning of Bit 0 time, AAB2 becomes true and drives the register counter. The operation of the bit counter is summarized in Figure 4-6.

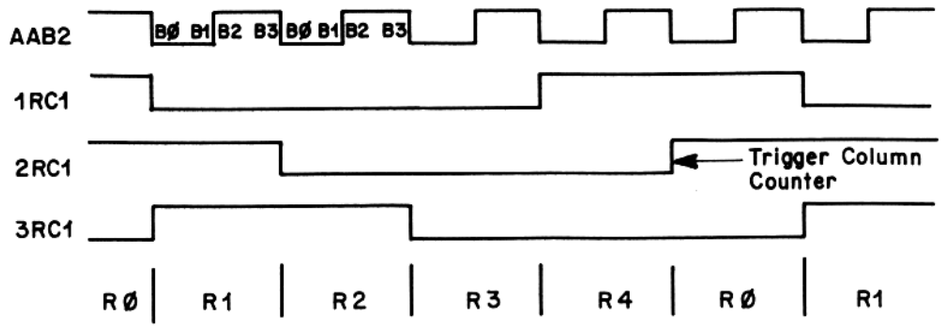


BIT COUNTER

FIGURE 4-6

c. REGISTER COUNTER. The register counter is composed of the 1RC, 2RC, and 3RC flip-flops. It is also a series shift counter. Its operation is the same as that of the program line bit counter except that its gating has been modified to allow it only five different states instead of six. It is driven by the bit counter through AAB2, and it drives the column counter when 2RC2 becomes true at R0 time. LASP3 resets 3RC when the arithmetic sync pulse is first launched. This will be discussed shortly. Figure 4-7 illustrates the operation of the register counter. Note that all three flip-flops are never reset at the same time.

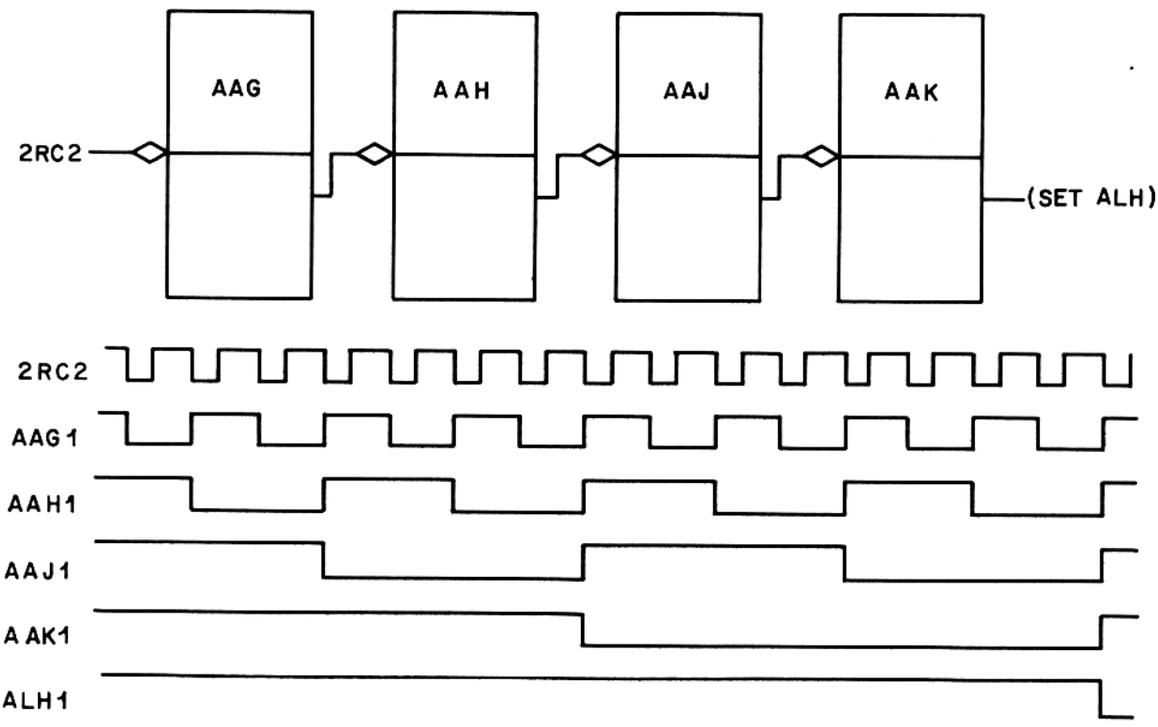




REGISTER COUNTER
(All on "MM" card)

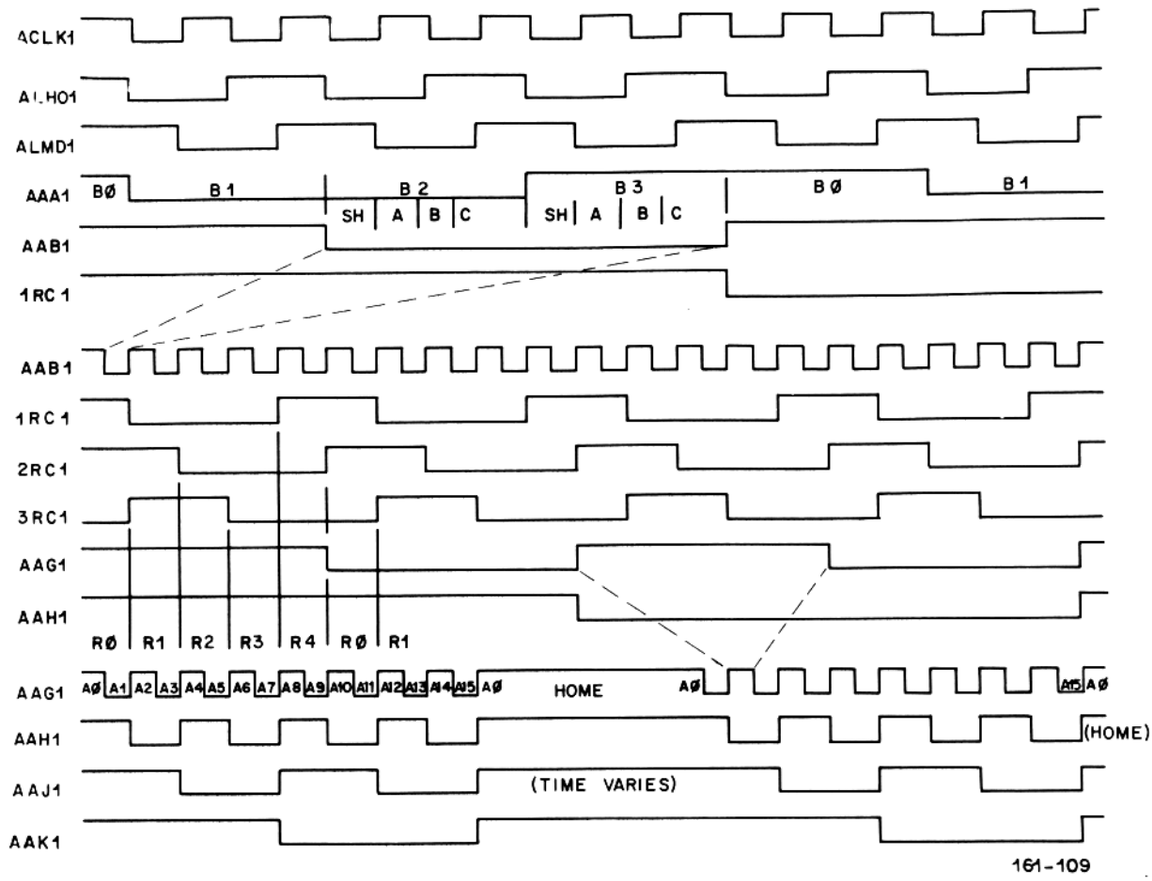
FIGURE 4-7

d. COLUMN COUNTER. The column counter is a regular four-stage binary counter made up of the AAG, AAH, AAJ, and AAK flip-flops. It is driven by 2RC2, at R0 time. At the end of column 15 AAK2 becomes true and sets the ALH flip-flop. AAG is set during program load to enable the timing chain to correctly count up to 1119 program characters. This will be discussed later. Figure 4-8 illustrates the operation of the column counter.



COLUMN COUNTER
(All on "MM" card)

FIGURE 4-8



ARITHMETIC LINE TIMING CHAIN

FIGURE 4-9

A comparison of all timing chain flip-flop waveforms is presented in Figure 4-9.

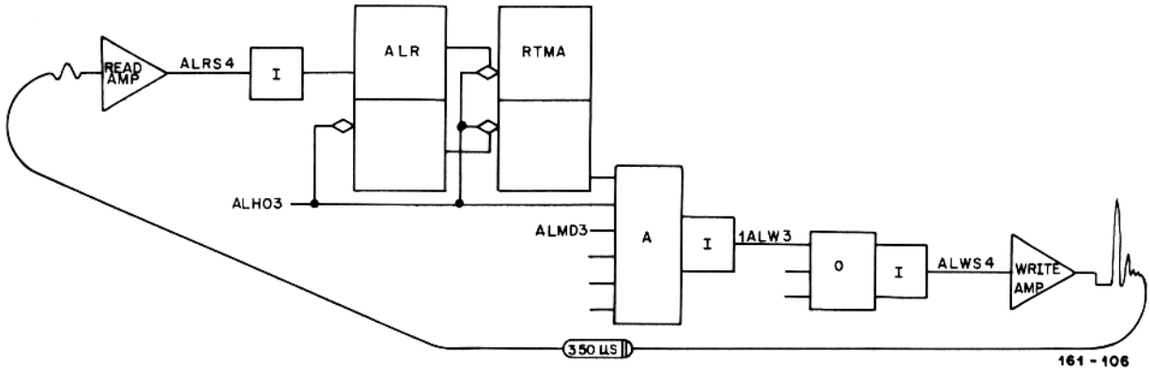
3. GENERAL OPERATION

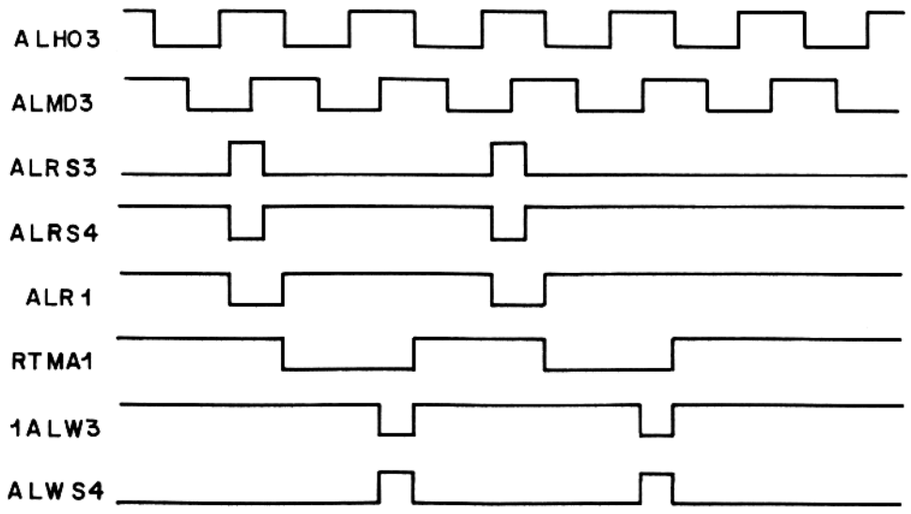
Certain arithmetic line functions must be understood before the individual commands can be studied: These include basic arithmetic line operation, operation during program load, and the launching of the sync pulse after program load.

a. BASIC READ/WRITE OPERATION. The output of the arithmetic line read amplifier, called the Arithmetic Line Read Signal (ALRS) is normally true, so it receives the "4" designation, ALRS4. When a data pulse is sensed by the read amplifier, ALRS4 becomes false. This signal is inverted and ALRS3, now true, sets the Arithmetic Line Read flip-flop, ALR. The outputs of ALR gate the inputs to the Re-TiMe Arithmetic flip-flop, RTMA, which is triggered by ALHO3. If ALR is set, the next ALHO3 will shift the data from ALR to RTMA.

ALR is set whenever a data pulse comes off the line, independent of the timing chain: RTMA is set by ALHO3, so it is set in sync with the timing chain. Thus even though the data is read randomly in relation to the timing chain, it is shifted into RTMA synchronized with the timing chain. All data coming off of the line passes through these two flip-flops in order to be re-timed. The binary value of a data pulse and its position in the arithmetic registers is determined by the state of the timing chain when the data is in RTMA.

Figure 4-10 illustrates the basic arithmetic line data flow. It is necessary for ALWS4 to become false in order to write data onto the line. Note that this occurs when ALHO3 and ALMD3 are both false (fractional bit "C" time) and RTMA is set. Note also that data is sensed randomly to the timing chain, but is rewritten in sync with the timing chain. The same ALHO3 that sets RTMA also resets ALR. The next ALHO3 resets RTMA if ALR does not become set in the meantime.

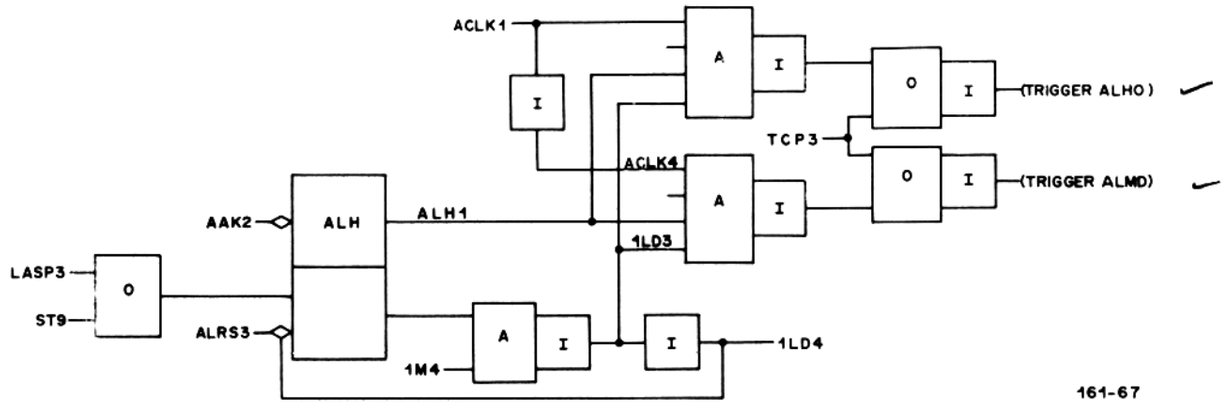




BASIC ARITHMETIC LINE DATA FLOW
(All on "JJ" card)

FIGURE 4-10

b. OPERATION DURING PROGRAM LOAD. When the 5610 is first turned on ST9 holds ALH reset so the entire arithmetic timing chain cycles continually for the first second. When ST9 becomes false the timing chain continues to run until AAK resets and sets ALH. As soon as ALH sets, both ALH1 and 1LD3 prevent further timing chain operation by the 2 MHz oscillator (see Figure 4-11). (Remember that the 5610 is automatically in 1 mode when it is first turned on.) ALHO does not reset at this time because 1M3 holds AL1M3 false (refer to Figure 4-4 if necessary). However, ALMD does become set by virtue of ALH1 driving the ALMD trigger true, the same as during a normal arithmetic line home. Thus program load starts out with both ALHO and ALMD set (fractional bit "A" time).



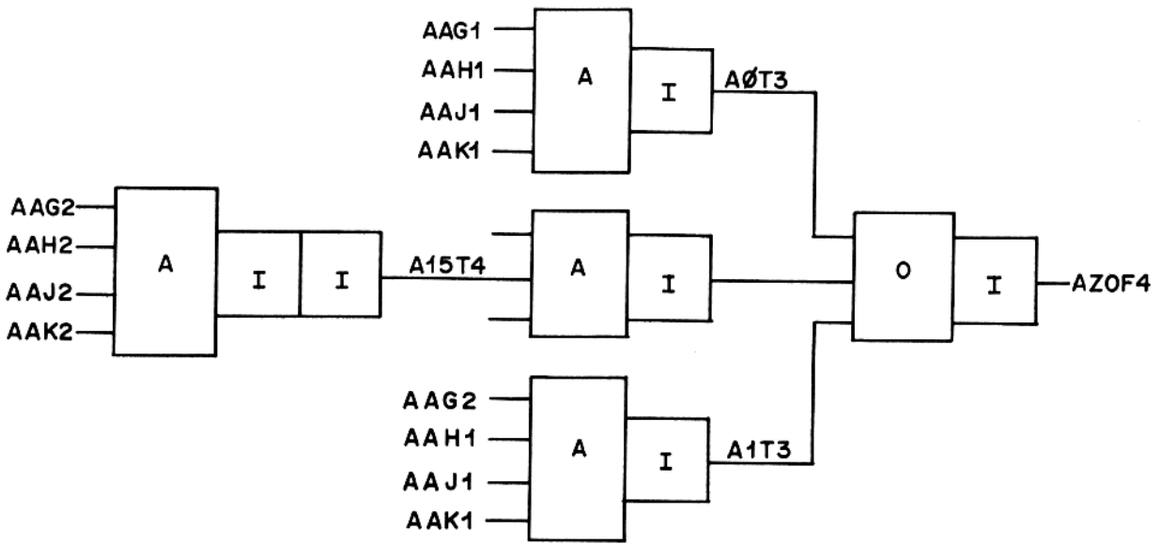
FRACTIONAL BIT COUNTER DRIVE
(All on "MM" card)

FIGURE 4-11

Subsequent loads affect the arithmetic timing chain in the same way: 1M3 becomes true at some random time in relation to the timing chain. As soon as ALH sets, 1LD3 stops the oscillator from driving the timing chain and subsequent operation is identical. Since the timing chain is stopped, data coming off of the arithmetic line cannot be re-written, so it is lost.

Each time a program character is loaded, the Timing Chain Pulse (TCP3) is developed in 1 mode 11 phase, and thus drives the triggers of ALHO and ALMD, which reset and then set alternately. The first TCP3 resets ALHO, the second resets ALMD, and the third sets ALHO. As ALHO sets, it pulses the bit counter. As more program characters are loaded, the fractional bit counter cycles, and each time ALHO sets (every four characters) it pulses the bit counter. Thus, after 15 characters are loaded the bit counter becomes reset and pulses the register counter. The register counter then receives another pulse every 16 characters.

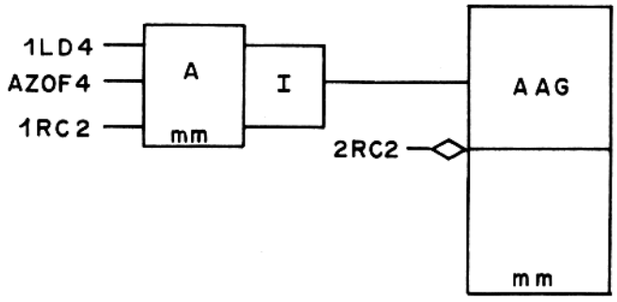
The operation of the fractional bit counter, the bit counter, and the register counter is merely a repeat of this action as more program characters are loaded. The column counter operation is a little different at the beginning however. Refer to Figure 4-12 and trace the development of AZOF4 (A-Zero, One, Fifteen).



DEVELOPMENT OF AZOF4
(All on "MM" card)

FIGURE 4-12

At the beginning of program load the entire timing chain is reset (except for ALHO and ALMD). Since A0T3 is true at this time, AZOF4 is false. The first time the register counter is pulsed (after 15 characters), 1RC2 becomes false, and since 1LD4 and AZOF4 are both false, the AAG flip-flop becomes set, as shown in Figure 4-13. As soon as AAG sets, A0T3 becomes false, but A1T3 becomes true, thus holding AZOF4 false.

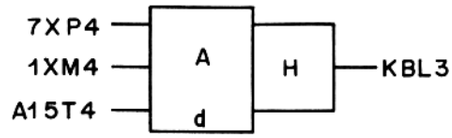


AAG ACTION DURING LOAD

FIGURE 4-13

After 64 more characters are loaded, 2RC2 becomes true and resets AAG, which sets AAH. The gate in Figure 4-13 is no longer effective because AZOF4 is now true. This gate can again operate when the column counter gets to A15T3, but in this case it will have no effect because AAG will already have been set.

As additional characters are loaded ALHO1 pulses the bit counter for every four characters, AAB2 pulses the register counter for every 16 characters, and 2RC2 pulses the column counter for every 80 characters. If a total of 1119 characters are loaded, the column counter will have just reached A15 time, and the KeyBoard Lock (KBL3) signal will have been produced (see Figure 4-14). This signal drives INQ3 (INput Qualification) false, and locks the 2205 keyboard, preventing further entries.



DEVELOPMENT OF KBL3

FIGURE 4-14

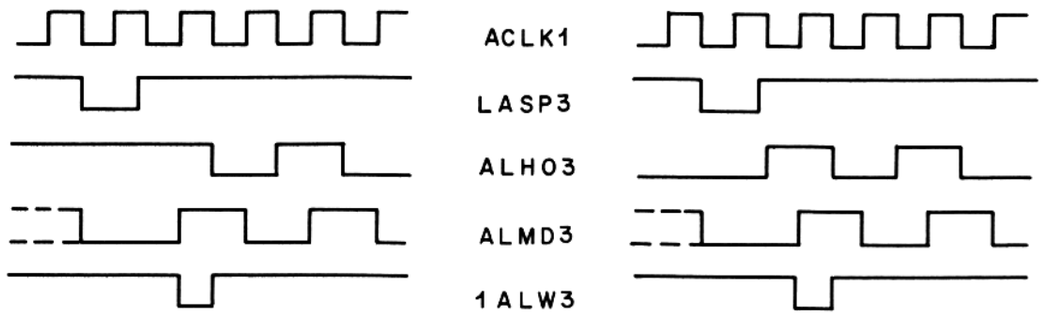
Figure 4-15 illustrates the state of the arithmetic timing chain after certain given numbers of program characters have been loaded.

SH	A	B	C		
	0	1	2	B0 R0 A0	
3	4	5	6	B1	
7	8	9	10	B2	
11	12	13	14	B3	
15	16	17	18	B0 R1 A1	(set AAG)
19	20	21	22	B1	
23	24	25	26	B2	
27	28	29	30	B3	
31	32	33	34	B0 R2	
35	36	37	38	B1	
39	40	41	42	B2	
43	44	45	46	B3	
47	48	49	50	B0 R3	
51	52	53	54	B1	
55	56	57	58	B2	
59	60	61	62	B3	
63	64	65	66	B0 R4	
67	68	69	70	B1	
71	72	73	74	B2	
75	76	77	78	B3	
79	80	81	82	B0 R0 A2	
83	84	85	86	B1	
87	88	89	90	B2	
91	92	93	94	B3	
95	96	97	98	B0 R1	
		etc.			

ARITHMETIC TIMING CHAIN ACTION
IN COUNTING PROGRAM CHARACTERS LOADED

FIGURE 4-15

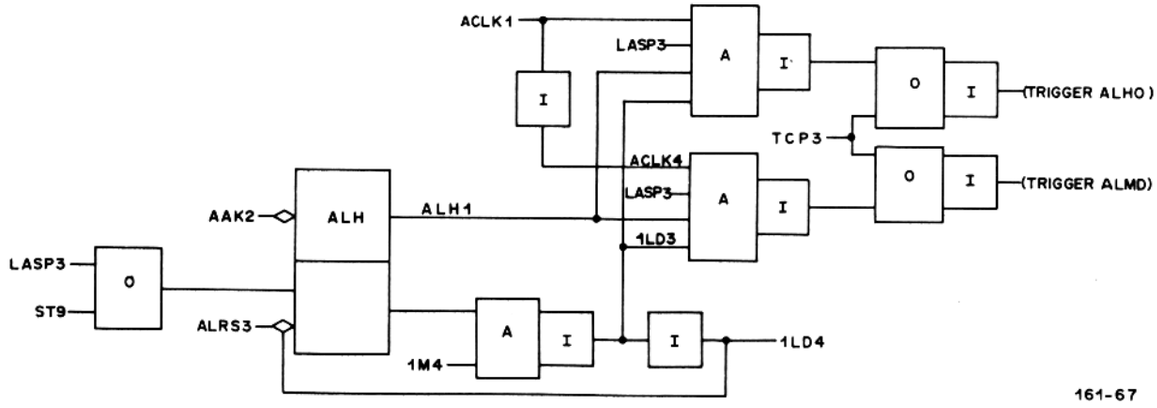
c. LAUNCH ARITHMETIC SYNC PULSE. At the end of program load, LASP3 (Launch Arithmetic Sync Pulse) artifically sets a bit (the sync pulse) into RTMA. As soon as LASP3 becomes false the timing chain restarts, and when ALMD resets ("C" time) the sync pulse is written onto the line. Representative waveforms are shown in Figure 4-16. Note that ALHO can be in either condition when LASP3 is generated, with no effect on the writing of the sync pulse.



TIMING CHART, LASP3 ACTION

FIGURE 4-16

Figure 4-17 illustrates the effect of LASP3 on the timing chain drive. First, LASP3 resets ALH so that timing chain action can resume. Second, by being applied to the two large AND gates, LASP3 prevents the timing chain from operating until it again becomes false.

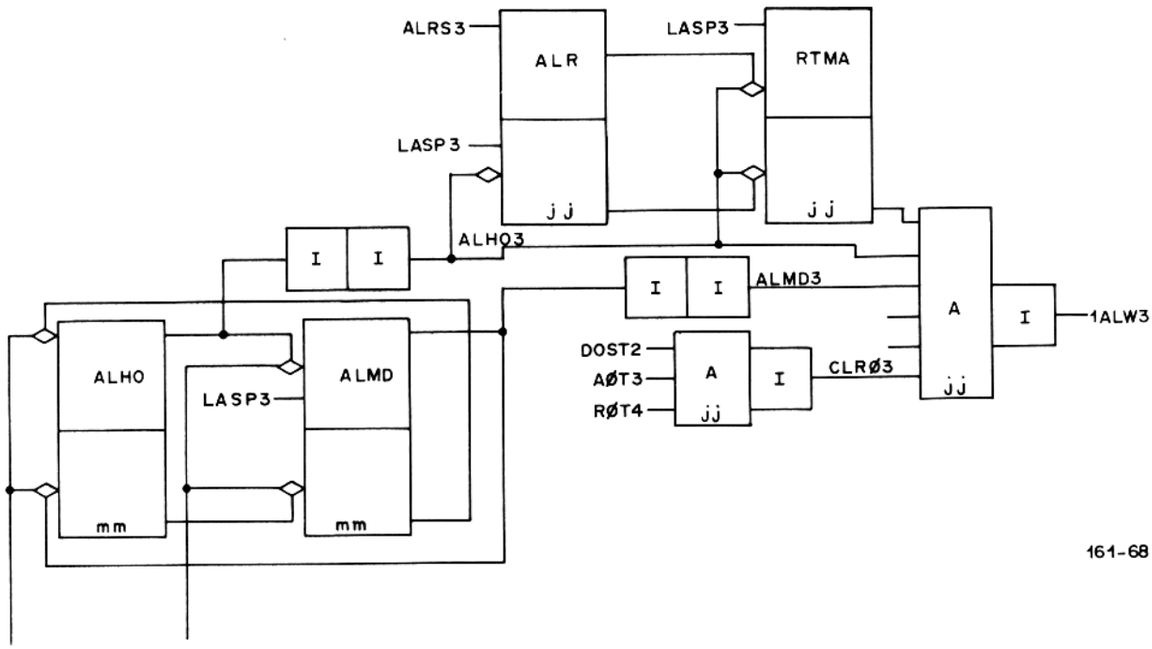


TIMING CHAIN DRIVE
(All on "MM" card)

FIGURE 4-17

Figure 4-18 illustrates the action of LASP3 on the basic arithmetic line write logic. First, LASP3 sets the newly manufactured sync pulse into RTMA. Second, it sets ALMD so that there will be a slight delay before fractional bit "C" time is realized. If this were not the case, it would be possible for ALHO and ALMD to both be set due to normal counting of the program characters. Then, when LASP3 became false (randomly in relation to the oscillator) the timing chain would start and "C" time could be too short to reliably write the sync pulse. Third, LASP3 resets ALR. This is merely for insurance: if ALR were set (by noise), the net result would be the writing out of two sync pulses.

Still referring to Figure 4-18, notice the AND gate which produces CLear R0 (CLR03). R0 is normally cleared (except in A0 time) whenever the DOST flip-flop is set, and DOST will always be set at the time LASP3 is generated. (DOST can only be reset during 5 mode.) If CLR03 is true, it prevents writing via 1ALW3.

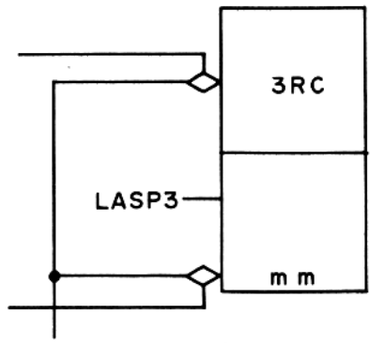


EFFECT OF LASP3 ON ALW LOGIC

FIGURE 4-18

If at least 15 characters have been loaded A0T3 will be false. It is also possible for the register counter to be in the R0 state (R0T4 false) due to the counting of program characters. Thus the writing of the sync pulse would be blocked.

To prevent this, LASP3 was added as a reset input to 3RC. This insures that the register counter will be in either the R1 or R2 state when the sync pulse is launched, as shown in Figure 4-19.



LASP3 PREVENTING R0 CONDITION

FIGURE 4-19

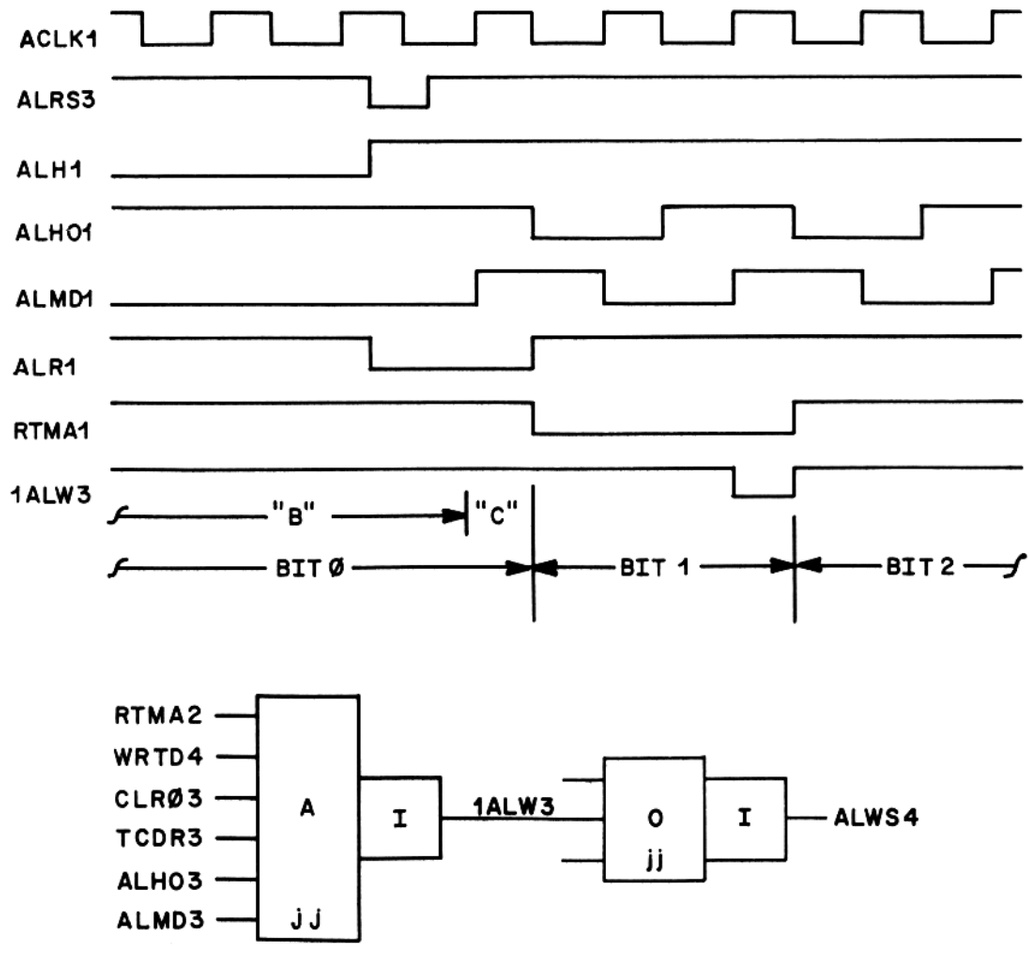
Once the timing chain is restarted (from any timing chain setting) it runs normally until AAK resets, thus setting ALH. The timing chain then stops, waiting for the sync pulse to be sensed by the read amplifier, and restarts when the sync pulse resets ALH. From this point on, timing chain operation is normal, taking 320 μsec. for each full cycle.

d. RE-WRITE SYNC PULSE. Whenever the arithmetic line is idling, it is merely recirculating the data, keeping the existing time relationship between all pulses. This action can be termed ALAL, and likened to the PLPL operation of the program line. When data is being routed in this manner, "write direct" is said to be true.

Since the arithmetic line home time and the oscillator speed vary between machines, it is next to impossible to have the data come off the line precisely synchronized with the timing chain. Therefore, the data must be re-timed before it is written back onto the line. This is one of the functions of the Arithmetic Line Read flip-flop, ALR, and the Re-Time Arithmetic flip-flop, RTMA. This re-timing of data is initiated at A0R0B0 time after the timing chain has restarted. (Remember that A0R0B0 time exists throughout arithmetic line home.) The actual sequence of events is as follows (refer to Figure 4-20):

(1) The sync pulse, (ALRS) resets ALH and sets ALR.
(2) ALH being reset allows the arithmetic timing chain to resume counting.
(3) ALR set and ALHO going true at the end of the bit time set the sync pulse into RTMA. ALHO resets ALR.
(4) ALHO and ALMD (false) and RTMA set allow the sync pulse to be rewritten back onto the line in the last quarter of B1 ("C" time), using ALWS4.
(5) ALHO resets RTMA at the shift to bit 2 time.

To summarize, the sync pulse is read and ALR sets. In the following bit time the pulse transfers to RTMA and is written back onto the line, this time in sync with the timing chain.



REWRITE ARITHMETIC LINE SYNC PULSE (ALAL)

FIGURE 4-20

Referring to Figure 4-20 again, note that the sync pulse is rewritten onto the delay line roughly one bit time after it appears as ALRS. All data following will be written with the same delay, so that when the line pass is completed, all data will be on the line in exactly the same relationship as it was previously. Note also that even though the sync pulse was written at B1 time, the next time it appears, it is in B0 time, since the timing chain is preset in B0 state. All flip-flops in the timing chain except ALMD are reset throughout arithmetic line home.

The only purpose of A0 time is to allow the sync pulse to be rewritten onto the line at the proper time. No data ever appears in A0.

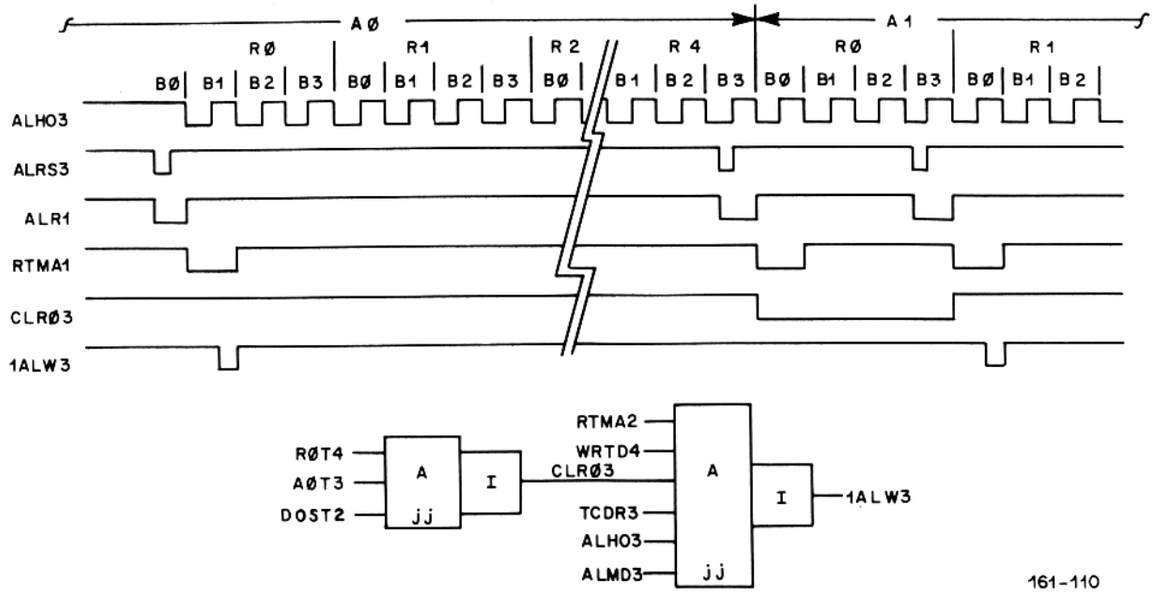
4. CLEAR R0

Register 0 is automatically cleared of all data whenever the arithmetic logic is "idling". If no functions utilizing the arithmetic line are being performed, the DOST flip-flop is set, and CLR03 is developed at every R0 time, except in A0, to inhibit writing on the delay line. Writing is not inhibited during A0 so that the sync pulse can be rewritten.

Since the data path is ALAL, write direct is true and WRTD4 is false. If 1ALW3 is not inhibited, all data will be written onto the line during the same bit time in which it appears in RTMA.

In Figure 4-21, an arithmetic instruction has just been completed, DOST has just been set, and there is still data in R0. On subsequent line passes R0 will be clear, but this procedure will continue as long as DOST is set. Note the following concerning Figure 4-21:

(1) The arithmetic line is "idling". The data path is short (ALAL).
(2) 1ALW3 is inhibited at R0 time to clear R0. The binary ONE previously in A1R0 is not rewritten onto the line. The binary ONE in A1R1 is rewritten the same as the sync pulse.
(3) 1ALW3 is not inhibited during A0 to allow the sync pulse to be rewritten.



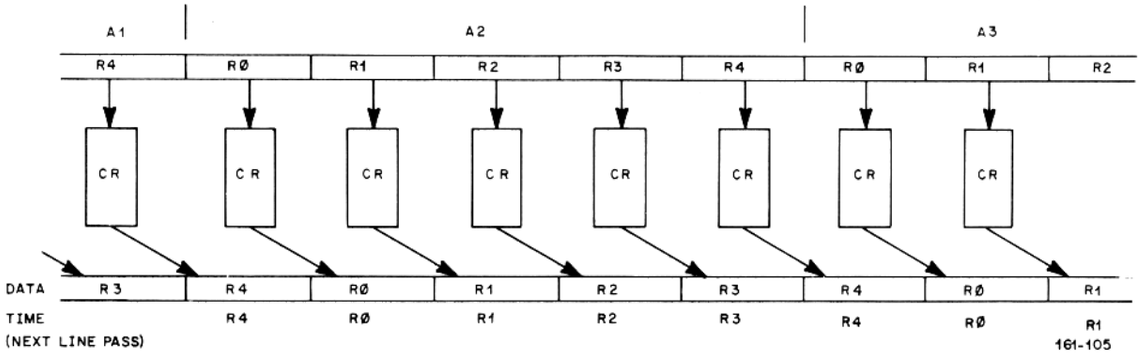
CLEAR R0 IN A1-A15, ALAL

FIGURE 4-21

5. SHORT AND LONG DATA PATHS

As explained previously, when the arithmetic logic is "idling", data is rewritten onto the arithmetic line approximately one bit time after it is read, or in the same bit time as it appears in RTMA. This is called the short data path. At this time it is said that "write direct" is true, and the arithmetic line routing is ALAL, similar to the PLPL condition of the program line.

At times it is also necessary to provide a long path for the data, which results in the writing of the data one register time later, or one R-time after it appears in RTMA. See Figure 4-22. When this occurs, the data passes through the common register before it is written back onto the delay line. In this case the data path is ALCR, CRAL. Note that data can be transferred to and from the arithmetic line ONLY via the common register, whereas the program line functions with either the common register, the search register, or the operation register.



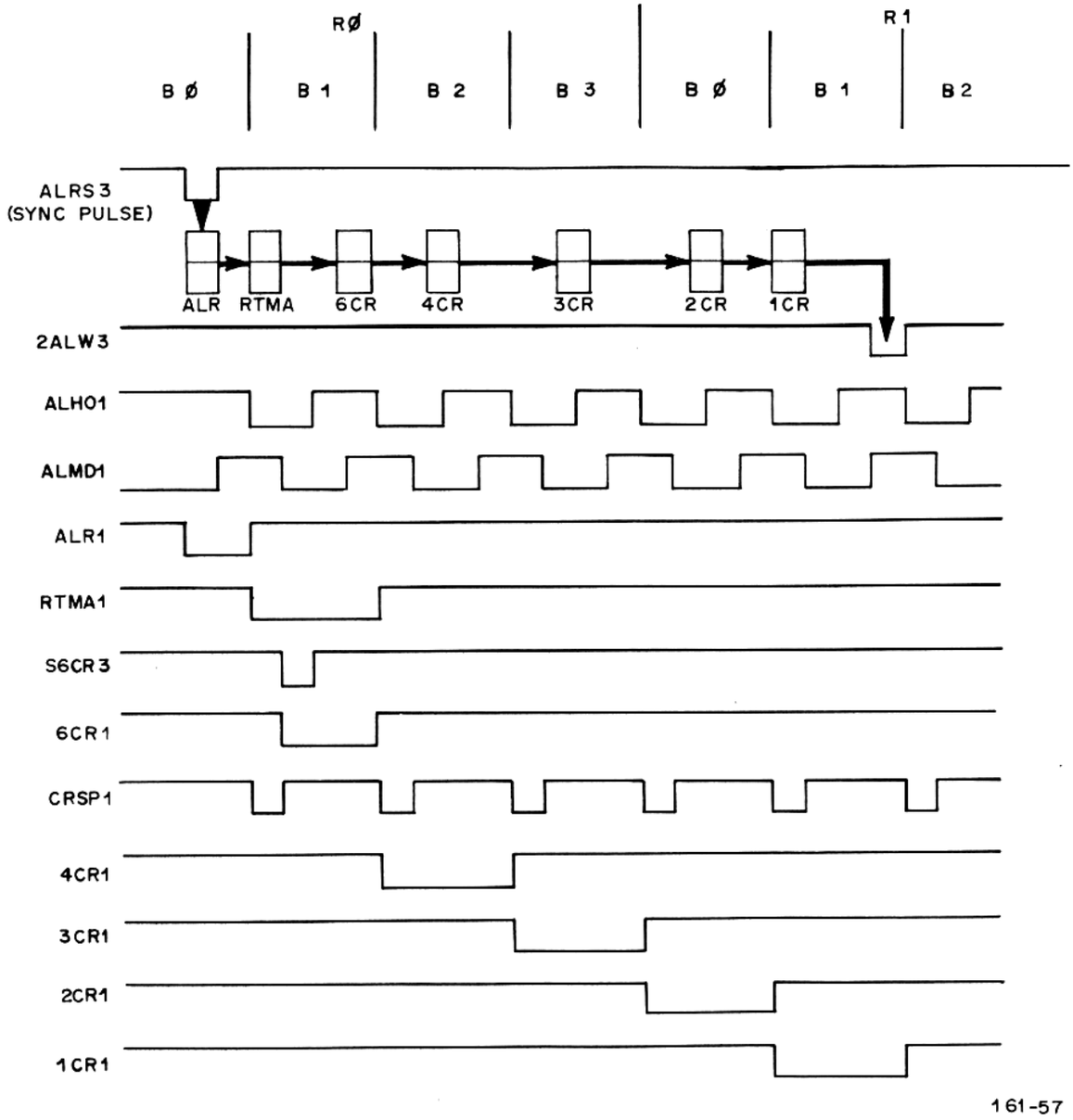
BASIC CONCEPT, DATA LONG

FIGURE 4-22

The signal which allows this long path is Data LoNG, DLNG, and is developed at specific times during all commands which deal with the arithmetic logic. The basic purpose of DLNG is to enable (1) the shifting up or down of the stack registers as part of Enter, Delete, etc., (2) the shift left of R0 during operations such as Recall and Accumulate, (3) the shift left of R1 during operations such as Enter and Constants, and (4) the shift left of R2 in Multiply and Divide.

Refer now to Figure 4-23 and note that the first part of the long data path is the same as the beginning of the short path: data comes off the line into ALR and at the next bit time it is shifted into RTMA. Once the data is in RTMA, however, the paths differ. In "write direct", the data passes directly from RTMA back onto the delay line. In data long, however, the data goes from RTMA into 6CR. It is then shifted through the common register, a shift occurring at the beginning of each bit time, and rewritten onto the line one R-Time later than in write direct. ALHO3 sets the Common Register Shift Pulse fllp-flop, CRSP, which shifts the data through the common register.

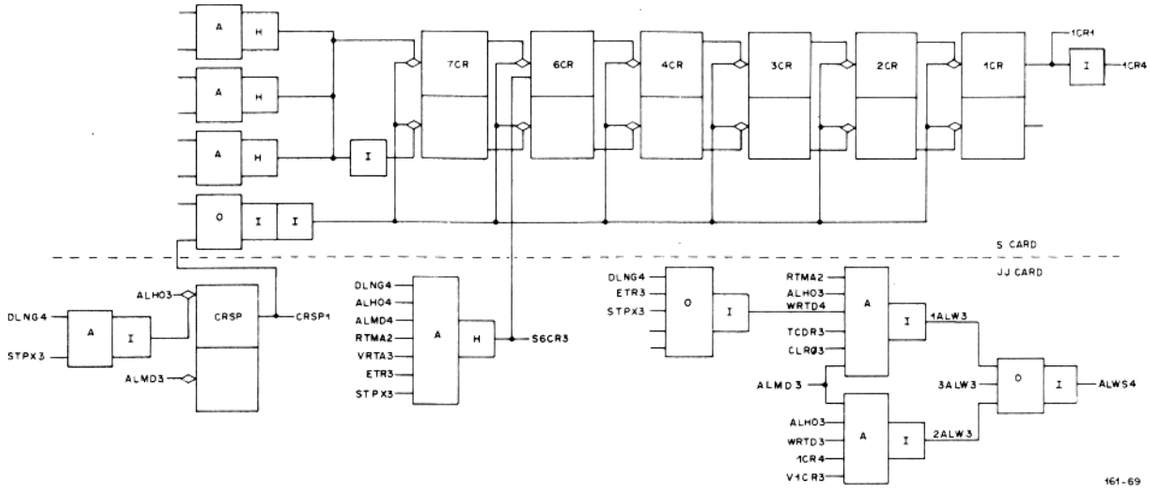
It is important to note at this point that the "4" signals used in this area of the logic can be confusing. For example, when WRTD4 is driven false, WRiTe Direct becomes true. Also, when DLNG4 becomes false, Data LoNG becomes true. In this manual, when a signal or condition is mentioned and no number (polarity indicator) follows, it is assumed that the "3" state is the one being discussed. In other words, if the statement is made, "write direct is true", it automatically means that WRTD4 is false.



REWRITE ARITHMETIC SYNC PULSE
DATA LONG PATH (ALCR, CRAL)

FIGURE 4-23

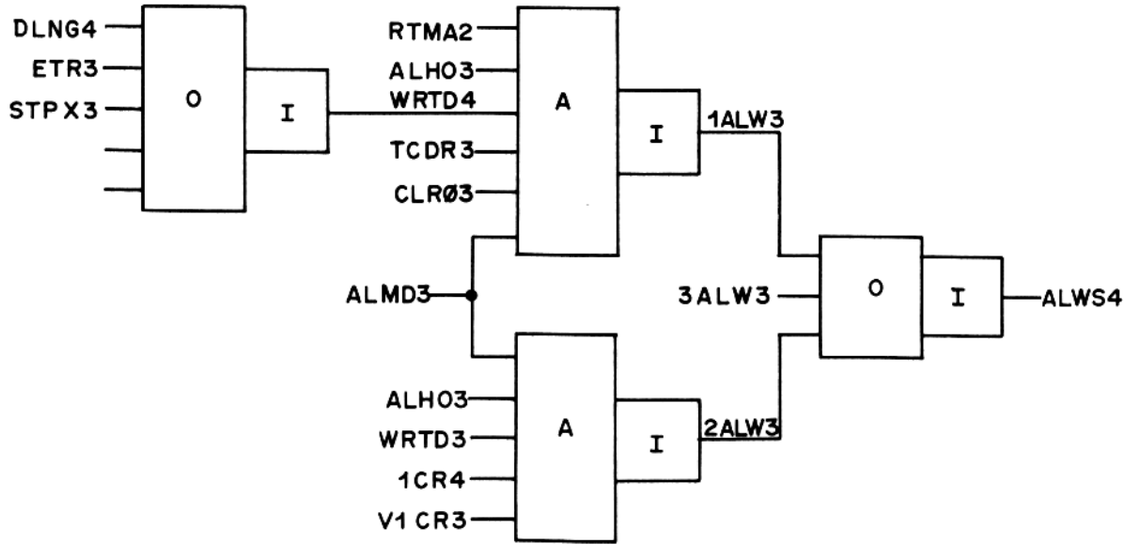
Figure 4-24 illustrates the development of S6CR3 and the operation of the CRSP flip-flop. As long as DLNG4 is false, CRSP will shift the common register. In most cases, the contents of RTMA will be shifted into 6CR and through the rest of the common register. Note, however, that VRTA3 (used mostly in arithmetic functions) and ETR3 (used in Store, Recall, and Accumulate) can block S6CR3, thus preventing RTMA data from being shifted into 6CR. (When this occurs, S6CR3 may be produced by other gates, so that some other data may be shifted in.) In other words, DLNG4 does not necessarily guarantee the ALCR line routing, but ALCR is impossible without DLNG4 being false. MOST OF THE TIME that DLNG4 is false, ALCR is true.



ALCR, CRAL LOGIC

FIGURE 4-24

After the data is shifted through the common register and reaches 1CR, it is usually written back onto the delay line via 2ALW3 and ALWS4. See Figure 4-25. This is the normal "data long" path. However, if "write Direct" is made true, WRTD3 blocks 2ALW3, and WRTD4 enables 1ALW3. In this case, the state of RTMA is copied onto the delay line via 1ALW3. Note that there are other signals present on the OR gate, in addition to DLNG4, that may become true and drive WRTD4 false. For example, it would be possible for DLNG4 to be false and ETR3 to be true at the same time. This would cause the writing to be "direct" even though "data long" is true (DLNG4 is false). This situation will be covered in more detail later on, when the individual instructions are discussed.



WRITE DIRECT/DATA LONG LOGIC
(All on "JJ" card)

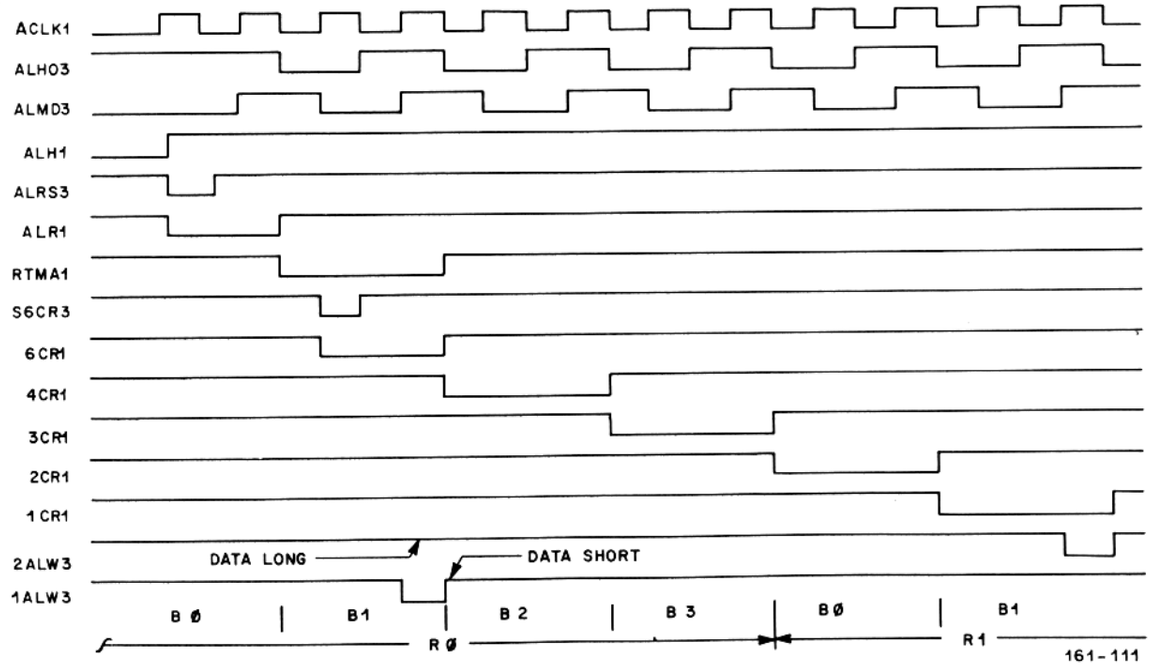
FIGURE 4-25

Still referring to Figure 4-25, note that still another signal, V1CR3, can inhibit 2ALW3 and thus block CRAL. When this occurs, 3ALW3 is usually enabled instead. This will also be covered more thoroughly later on.

In summary, the following statements can be made:

(1) WRTD4 being false insures "write direct", or the short data path (ALAL).
(2) DLNG4 being false insures only that the common register will be shifted. However, it USUALLY means that CRAL will be true (unless blocked by WRTD3 or V1CR3) and it ALMOST ALWAYS means that ALCR will be true (unless blocked by VRTA3 or ETR3).

The timing chart in Figure 4-26 compares the rewriting of the sync pulse, using both the long and short paths.



REWRITE SYNC PULSE
COMPARISON BETWEEN DATA LONG & DATA SHORT

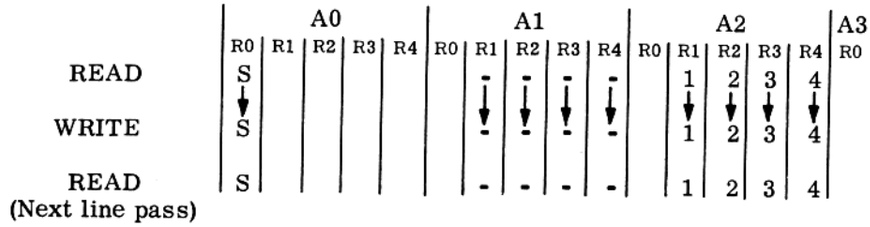
FIGURE 4-26

6. SHIFT UP/SHIFT DOWN

All of the instructions dealing with the arithmetic delay line employ some of the principles involved in the basic shifting up or down of the "stack" registers. Most instructions employ either a modified shift up or shift down.

The basic action involved in shifting the registers up or down is the manipulation of the sync pulse in relation to the other data. By advancing or delaying the sync pulse in relation to the other data, the data can be shifted up or down, respectively. The movement of the sync pulse in relation to the data is accomplished by changing the arithmetic line data flow from long to short and back again at the proper time.

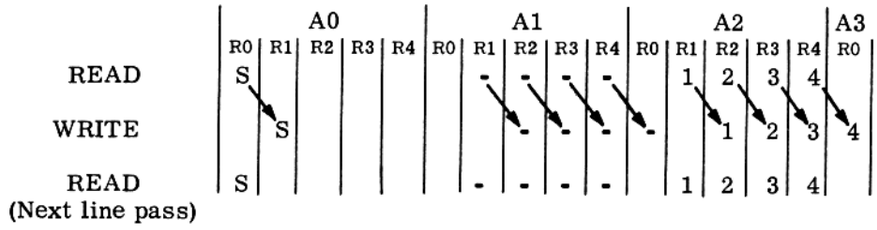
Refer now to Figure 4-27, which illustrates a portion of the arithmetic line before and after a line pass in which all information, both sync pulse and data, is written short (ALAL). For ease of understanding, R1 has been loaded with all "1's", R2 with "2's", etc. Note that on the next line pass, no data has changed.



ARITHMETIC DATA FLOW
DATA SHORT (ALAL)

FIGURE 4-27

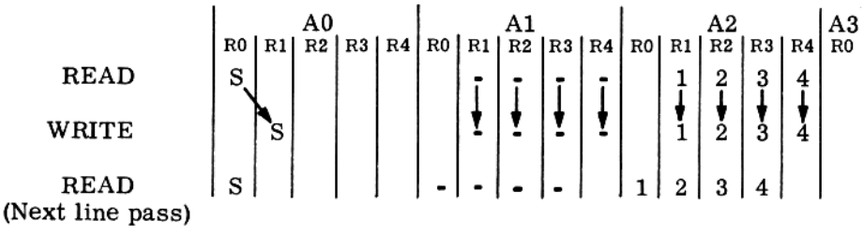
The same is true in Figure 4-28: the only difference is that all data has been written long (ALCR, CRAL). There is still no change in the relationship of the data to the sync pulse.



ARITHMETIC DATA FLOW
DATA LONG (ALCR, CRAL)

FIGURE 4-28

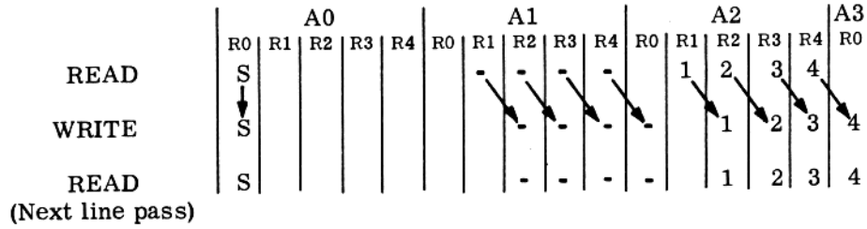
Refer now to Figure 4-29. In this case, only the sync pulse was written long: all other data was written short (direct). The net effect produced is to make the sync pulse appear "closer" to the data, so that on the next line pass the data is seen earlier in relation to the timing chain. (Because the timing chain always starts at the same place when the sync pulse is sensed, it can be said that the sync pulse restarts the timing chain later in relation to the data.) This, then, has the effect of shifting the stack down, moving the previous R4 data into R3, the previous R3 data into R2, etc.



BASIC SHIFT DOWN

FIGURE 4-29

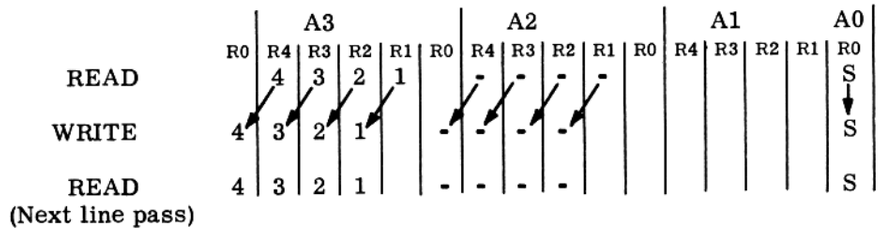
Exactly the opposite action is required to shift the stack up: the data must come later in relation to the sync pulse, so the sync pulse is written direct (short) and the balance of the data is written long. This is illustrated in Figure 4-30. Note that on the next line pass the previous R0 data appears in R1, the previous R1 data is in R2, etc. This effectively shifts the stack up.



BASIC SHIFT UP

FIGURE 4-30

Figure 4-31 is a mirror image of Figure 4-30. The only purpose in presenting it at this time is to show that this type of information can be illustrated both ways: with time running from left to right, as one is used to seeing it, or with time running from right to left. Again, this latter method is useful when discussing those arithmetic instructions utilizing the "shift left" function, as will be seen in the material that follows.



BASIC SHIFT UP

FIGURE 4-31

7. SHIFT LEFT

Most arithmetic instructions utilize the "shift left" function in one of its forms. During a shift left, ONE of the registers is shifted "left" in relation to the contents of the other registers. Figure 4-32 illustrates the operating "stack" both before and after the data in R1 has been shifted left one time (one column).

A14														A14													A2
8	7	6	5	4	3	2	1	.5	4	3	2	1		R4	8	7	6	5	4	3	2	1	.5	4	3	2	1
8	7	6	5	4	3	2	1	.5	4	3	2	1		R3	8	7	6	5	4	3	2	1	.5	4	3	2	1
8	7	6	5	4	3	2	1	.5	4	3	2	1		R2	8	7	6	5	4	3	2	1	.5	4	3	2	1
8	7	6	5	4	3	2	1	.5	4	3	2	1		R1	7	6	5	4	3	2	1	5	.4	3	2	1	Ø
8	7	6	5	4	3	2	1	.5	4	3	2	1		RØ	8	7	6	5	4	3	2	1	.5	4	3	2	1

BEFORE AFTER

STACK MOVEMENT DURING SHIFT LEFT

FIGURE 4-32

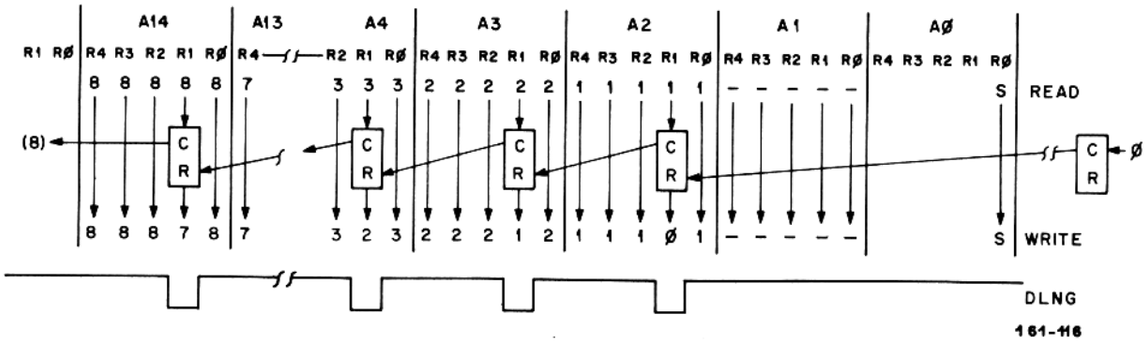
The code standing in the common register at the beginning of the shift left is shifted into the LSD position of R1, and the code from the MSD position remains in the common register at the end of the shift left. What happens to this code next depends upon the instruction being performed. Similarly, the instruction being executed determines the source from which the code initially in the common register is obtained.

The instruction being performed also dictates which of the registers will be shifted left. R0 will be shifted left during Store, Recall, and Accumulate. R1 is shifted during Multiply, Divide, Enter, Constants, and the two print commands. R2 is also shifted during Multiply and Divide. R3 and R4 are never shifted left.

Arithmetic column 1, containing the sign bit, is shifted only when R0 is being shifted left. When R1 or R2 is being shifted, A1 is bypassed. A15 is used only in Multiply and Divide when R2 is shifted and in Divide when R1 is shifted.

a. BASIC OPERATION. It was noted previously that manipulation of the data and the sync pulse in relation to each other allowed the shift-up or shift-down function to occur. The basic operation of the shift left may be stated as follows: THE DATA TO BE SHIFTED LEFT IS WRITTEN LONG: THE SYNC PULSE AND ALL OTHER DATA IS WRITTEN DIRECT.

Refer to Figure 4-33. In this example of shifting R1 left, the prior conditions show all registers containing a sign bit in column 1, all registers contain a "1" in column 2, all have a "2" in column 3, and so on. "Data long" becomes true at R1 time in A2 through A14. (It remains false in A1 so as not to disturb the sign bit.) When data long first becomes true, the "1" is shifted out onto the line. Data long then returns to false and the balance of the A3 data is written direct.



BASIC SHIFT LEFT (R1)

FIGURE 4-33

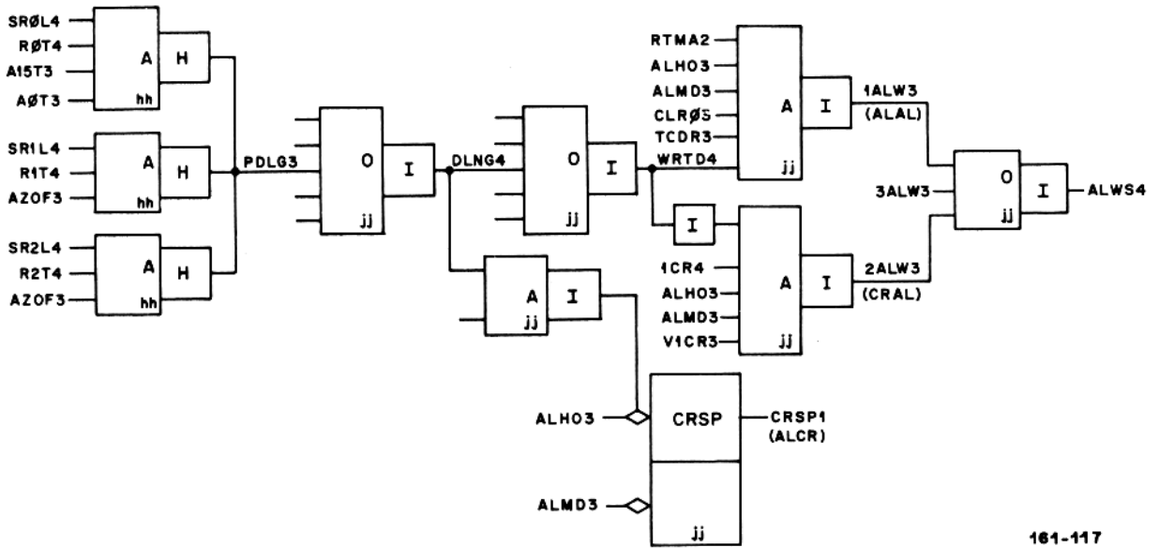
This procedure continues until the end of the line pass, at which time the digit from A14R1 is in the common register. As stated previously, the disposition of this digit depends upon the instruction being performed. Another shift left may occur during the next line pass, or it may be delayed for one or more line passes, again depending upon the instruction. Likewise, the source of the code in the common register before the shift left begins is also determined by the instruction being executed.

b. LOGIC. The signal used in shifting left is called Pulse Data LonG, PDLG3. This signal becomes true at the proper time(s) during Add, Subtract, Multiply and Divide. PDLG3 is also used at other times, and it does not always produce "pulses"; its other uses will be explained when the various instructions are covered.

Refer to Figure 4-34. When PDLG3 becomes true, it drives DLNG4 false. This makes WRTD4 true, blocking 1ALW3 (ALAL) and enabling 2ALW3 (CRAL). DLNG4 becoming false also allows the Common Register Shift Pulse flip-flop, CRSP, to set at each bit time, thus enabling ALCR.

Figure 4-34 also shows the development of PDLG3 in a simplified manner. Note that when the Shift R0 Left signal, SR0L4, is false, PDLG3 will become true at each R0 time, except in columns A0 and A15. If SR1L4 is false, PDLG3 will be true in each R1 time, except during A0, A1, and A15. If SR2L4 is false, PDLG3 will be true each R2 time, except during A0 and A1. (Note that AZOF3 is false at A15R2 time, thus allowing data to be shifted into A15R2 during Multiply. During Divide, AZOF3 is held false throughout A15 time, allowing data to be shifted into A15R1 and A15R2.) Thus, data long (ALCR, CRAL) is used for the register being shifted left, and data short (ALAL) is used at all other times.

The development of SR0L4, SR1L4, and SR2L4 will be covered as the individual instructions are examined.



BASIC SHIFT LEFT LOGIC

FIGURE 4-34

B. ACTION STEP COUNTER

The action step counter is a three-stage series shift couunter which operates during all commands dealing with the arithmetic line. The truth table (Figure 4-35), illustrating the states of the three flip-flops during each of the six action steps, shows a non-standard configuration or code. The reasons for using this odd code will be apparent after the operation of the counter is understood.

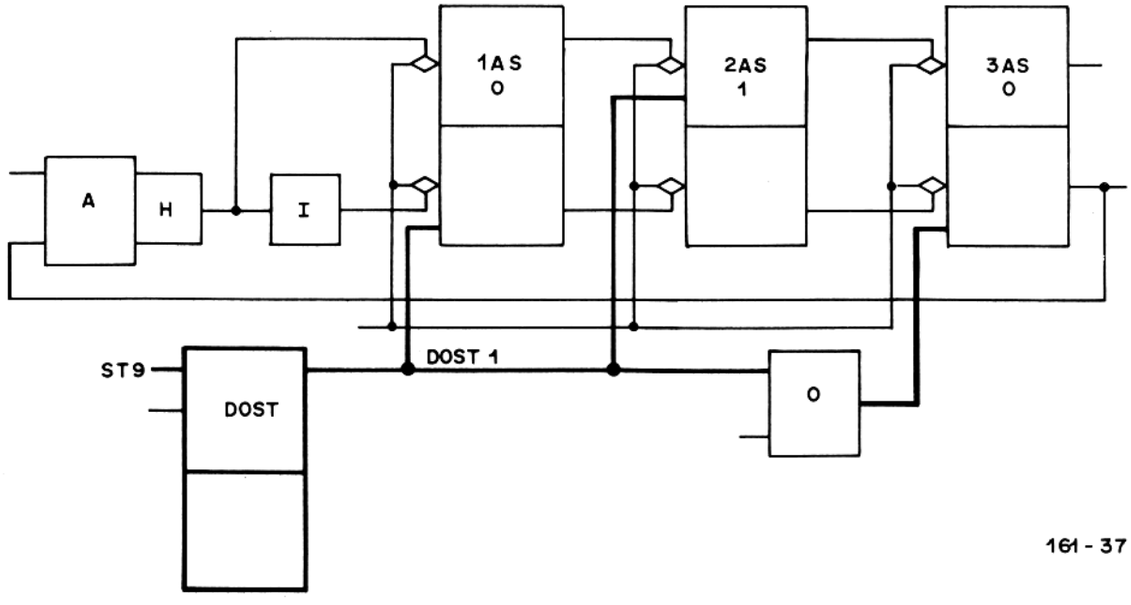
STEP NO.	CODE		
	1AS	2AS	3AS
0	0	1	0
1	0	0	1
2	1	0	0
3	1	1	0
4	0	1	1
5	1	0	1

TRUTH TABLE FOR ACTION STEP COUNTER

FIGURE 4-35

1. DOST FLIP-FLOP

Action step 0 is not an active step. Rather, the counter is in step 0 when the math processor is idling. Action step 0 can be considered the counter's reset state. When the 5610 is first turned on, ST9 sets the Data Operation STop flip-flop (DOST), which in turn drives the action step counter to action step 0, the 010 configuration. This is illustrated in Figure 4-36.

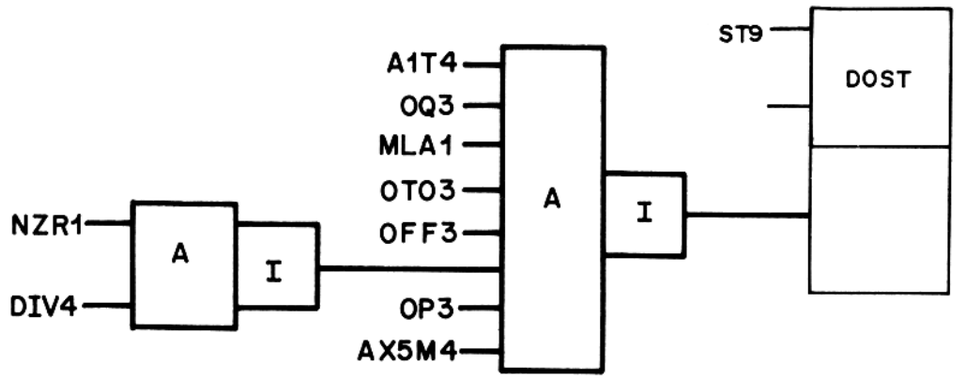


ACTION STEP COUNTER, RESET STATE
(All on "KK" card)

FIGURE 4-36

Before the action step counter can step, DOST must be reset. Note the conditions necessary to reset DOST in Figure 4-37. The following signals will prevent the reset of DOST:

1. A1T4 -- DOST can only be reset at A1 time.
2. OQ3 -- the operation register contains the Halt command.
3. MLA1 -- the Math LAtch flip-flop, when set, prevents DOST from resetting.
4. OTO3 -- the operation register contains the Type command.
5. OFF3 -- the operation register contains a Flexowriter Function.
6. OP3 -- the operation register contains the Print Sign command.
7. NZR1 & DIV4 -- stops math action in division if R1 = 0 (prevents dividing by 0).
8. AX5M4 -- the program logic must be in 5 mode before DOST can be reset. This means DOST cannot be reset by any of the branching instructions or sequence separator codes, because in the operation caused by these codes, 5 mode is never realized.



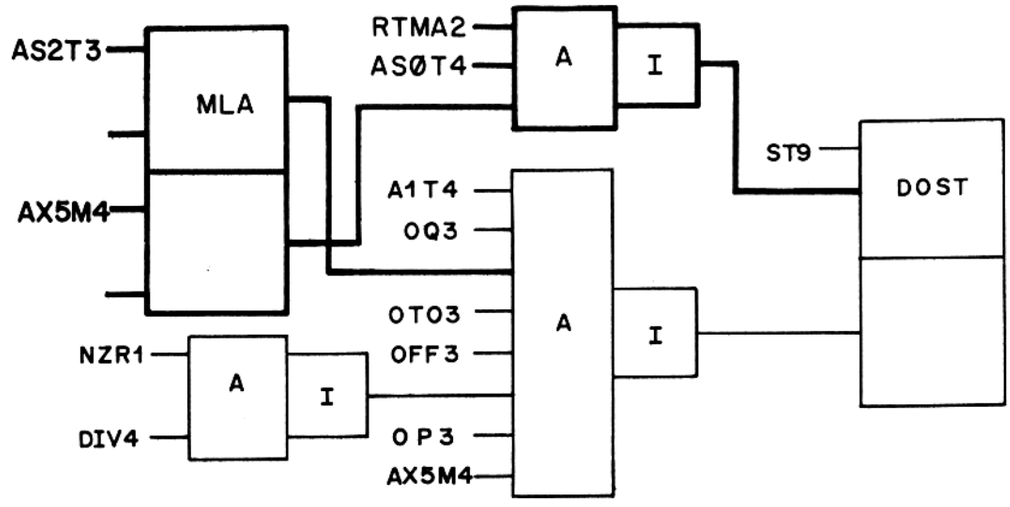
RESETTING DOST
(All on "KK" card)

FIGURE 4-37

With the above in mind, it can be seen that DOST is prevented from becoming reset during all commands except those dealing directly with the arithmetic logic. Therefore, only those commands utilizing the arithmetic line will allow the action step counter to operate.

When in an arithmetic function in 5 mode, two conditions must be satisfied before the program line logic can leave 5 mode. One is program line home, which occurs every 11 ms. The other is DOST3, which will again become true when the arithmetic function is complete. The shorter arithmetic functions (add, subtract, change sign, etc.) will be finished long before the first program line home, because these functions require only one or two ms. of time to be completed. However, longer multiplication and division problems may take so long to complete that the program line makes many cycles in 5 mode. Eventually, though, the arithmetic function will be completed at some random time in relation to the program line. The program line logic will wait until program line home occurs and then step off of 5 mode.

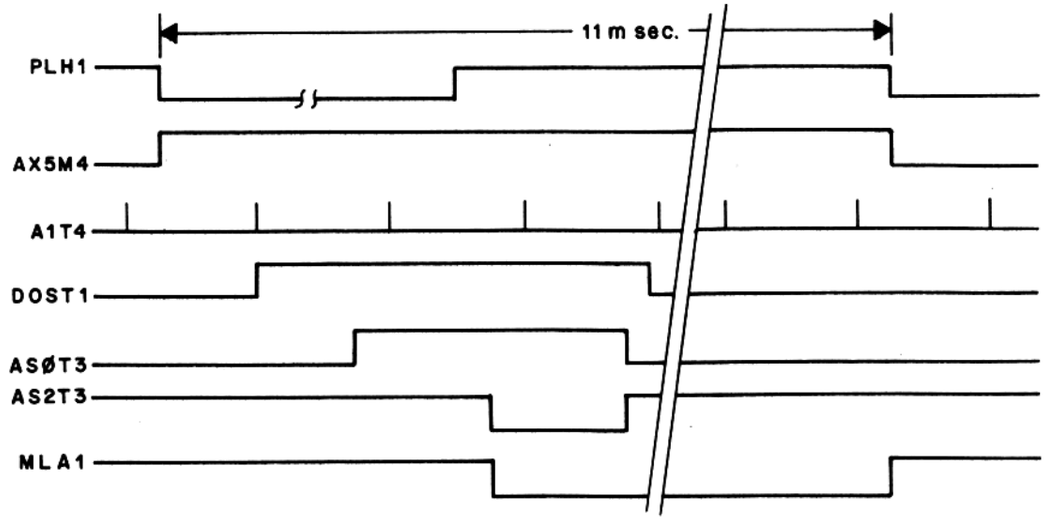
However, DOST is vulnerable during this time between the end of the arithmetic function and program line home. As the arithmetic function is completed the action step counter returns to action step 0, and the next arithmetic sync pulse sets DOST (see Figure 4-38). Since the program line logic is still in 5 mode, waiting for program line home, the next A1T4 going false would again reset DOST and allow the action step counter to step. To prevent this, the Math LAtch flip-flop (MLA) qualifies the reset of DOST. Refer to Figure 4-38 and note that MLA will be reset before 5 mode is reached by AX5M4 (Arithmetic 5 Mode). It will be set by AS2T3



MATH LATCH EFFECT ON DOST
(All on "KK" card)

FIGURE 4-38

during 5 mode, thus disabling the reset of DOST. MLA stays set until the end of 5 mode, when it is again reset to prepare for the next arithmetic function. The other set and reset inputs to MLA are used only during Enter, and are explained in Section 4G. The timing chart in Figure 4-39 helps to explain this operation of MLA.

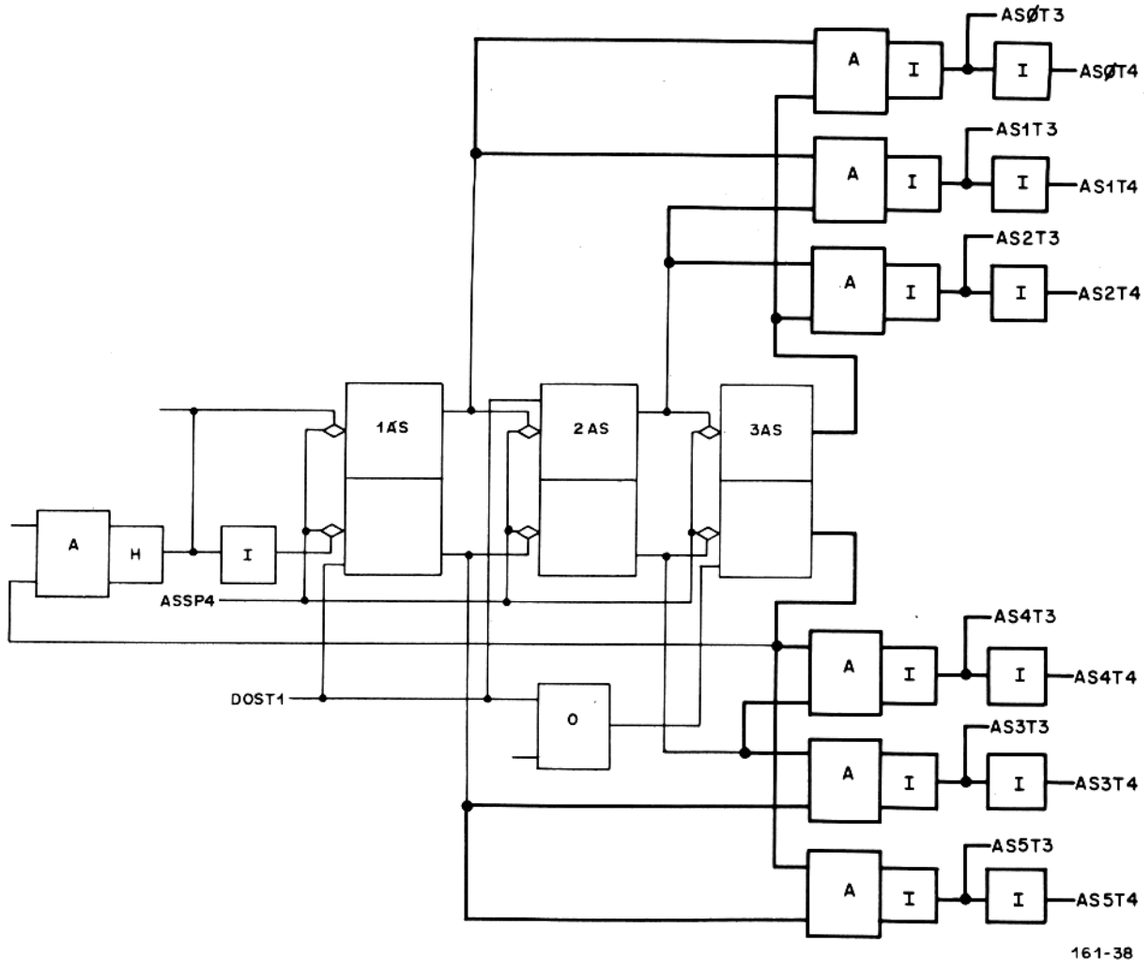


MATH LATCH OPERATION

FIGURE 4-39

2. DECODED OUTPUTS

The outputs of the action step counter are decoded as shown in Figure 4-40. These outputs are routed throughout the arithmetic logic to gate the particular arithmetic functions. For example, it has already been shown that AS2T3 sets MLA. Similarly, the other decoded outputs of the action step counter gate other arithmetic functions. Refer to the "Arithmetic Operations Chart", Figure 4-51, for a summary of action step counter control of each individual function.



DECODING THE ACTION STEPS
(All on "KK" card)

FIGURE 4-40

3. TWO OR FIVE STEPS

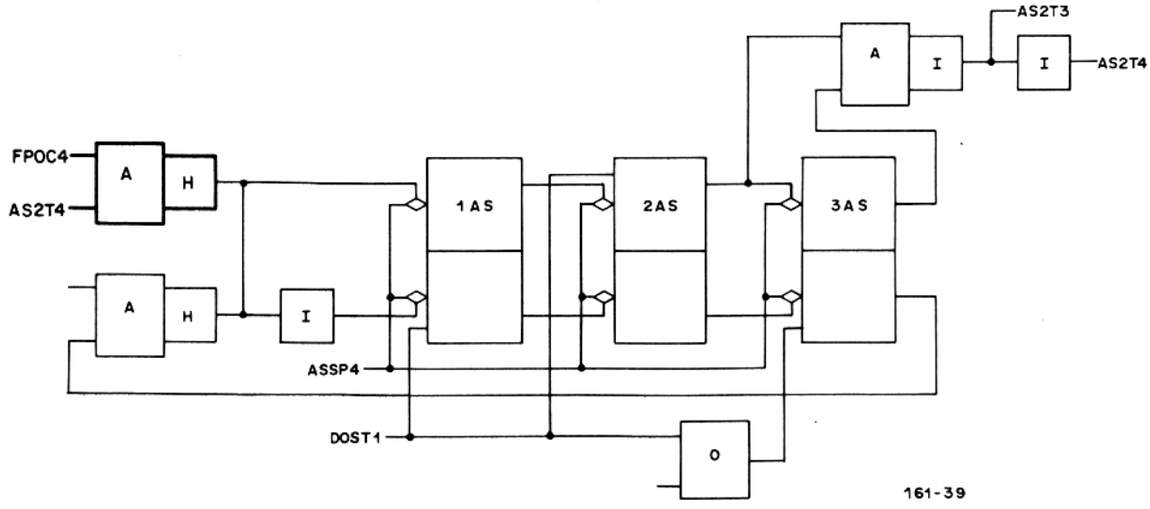
Many math commands require only two action steps (some require only one, but idle through action step 2 ALAL). Others (Enter, Print, Accumulate, Store, Multiply, and Divide) require the action step counter to cycle through all action steps. In order to increase the processor speed for the less complex math commands, the action step counter has the capability of returning to action step 0 from action step 2, instead of cycling through action steps 3, 4, and 5. When necessary, however, all action steps will be utilized.

a. TWO STEPS. Refer again to Figure 4-40. If the counter is at action step 0, 2AS is set, and 1AS and 3AS are reset. If the Action Step Shift Pulse (ASSP4) goes true, the set condition of 2AS will be shifted into 3AS, the reset condition of 1AS will shift into 2AS, and the reset condition of 3AS will shift into 1AS. This 001 configuration thus decodes into action step 1, and permits the various arithmetic functions gated by action step 1 to occur.

A second ASSP4 will shift the counter one more time, shifting the reset condition of 1AS and 2AS into 2AS and 3AS respectively, and shifting the set condition of 3AS into 1AS. This 100 configuration of the counter causes AS2T3 to become true, allowing those processor functions gated by action step 2 to occur.

A third shift pulse will now reset 1AS, set 2AS, and leave 3AS reset. The counter is now back to its original condition, 010, or action step 0. Note, as shown in Figure 4-38, that after the counter returns to action step 0, DOST becomes set the next time that RTMA sets. This occurs when the sync pulse is sensed at the beginning of the next line pass. DOST being set clamps the action step counter in the 010 configuration. Should any stray ASSP4 signals occur at this time, they would have no effect on the counter because it is clamped at action step 0. No further stepping can occur until DOST is again reset.

b. FIVE STEPS. The longer arithmetic functions, which require the action step counter to step through all action steps, develop the Five Phase Op Code signal, FPOC. This signal is applied to the gates controlling 1AS. Refer to Figure 4-41 and note that during action step 2 time with FPOC, the output of the upper H-inverter goes true, so that when ASSP4 occurs 1AS will remain set. During action step 2 the counter contained the 100 configuration. After ASSP4 occurs 1AS and 2AS will both be set and 3AS will remain reset. This configuration (110) decodes into action step 3.



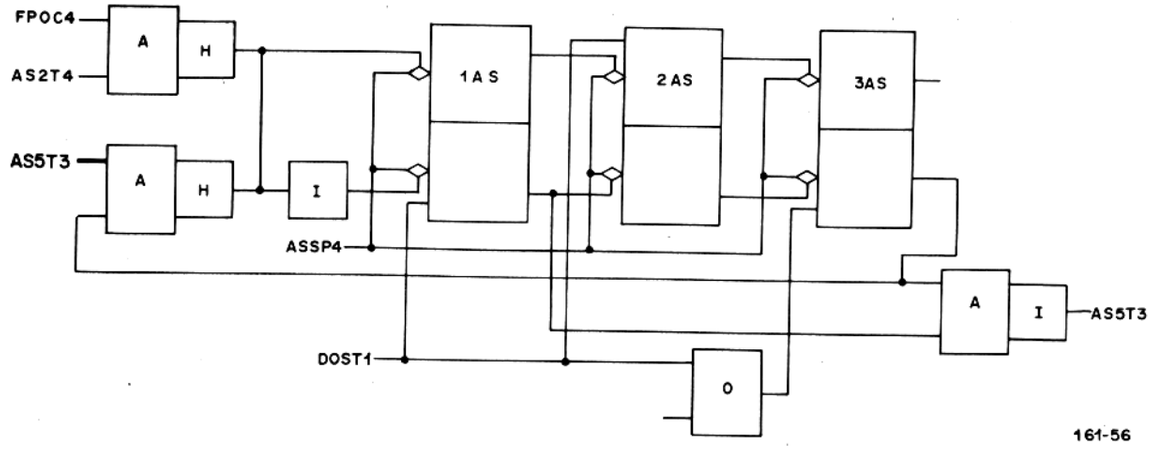
FIVE ACTION STEPS
(All on "KK" card)

FIGURE 4-41

When action step 3 is finished another ASSP4 will shift the counter so it contains the 011 configuration, action step 4. Still another shift pulse will put the action step counter into action step 5, the 101 configuration.

When in action step 5 another shift pulse would return the counter to action step 3, the 110 configuration. However, a shift pulse will occur during action step 5 only if the arithmetic function is complete, and if this is the case the action step counter must return to step 0, not to step 3.

This is accomplished by adding AS5T3 as an inhibit to the set of 1AS, as shown in Figure 4-42. With action step 5 blocking the set of 1AS in this manner, ASSP4 will set the counter back to step 0, the 010 configuration.



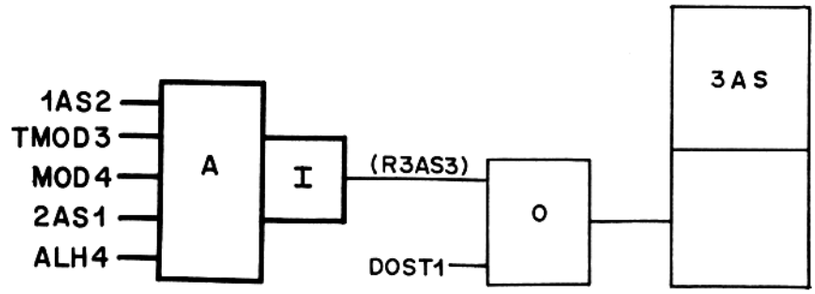
RETURNING TO A.S. 0 FROM STEP 5
(All on "KK" card)

FIGURE 4-42

c. REPEATING STEPS. One other aspect of action step counter operation must be studied before moving on. In the 5610, multiplication and division are treated as they are in most other computing machines; a series of additions and subtractions, respectively. At a certain point in both of these operations it is necessary to return to step 2 after step 4 has been completed. For instance, if some number is to be multiplied by 25, the number will be added two times during step 3. The action step counter will then step to 4 and return to step 2, where the 25 is shifted one place to the left. The counter then steps to 3, and allows the number to be added five more times. After the addition is complete the counter steps to 4, 5, and then back to 0. (This explanation is very brief and leaves a great deal unsaid: it is included here only to point out the reason for the action step 4 to 2 shift.)

This shift cannot take place until action step 4 has been completed, and when this happens the counter automatically tries to step to 5. The shift actually takes place from step 5 to step 2, but the counter is in step 5 for such a short time that for all practical purposes it can be said that the shift occurs at the end of action step 4.

Refer now to Figure 4-43. If the 5610 is in Mult Or Divide, MOD4 is false. If the Terminate Mult Or Divide signal has not yet been developed, TMOD3 will be false. (This signal being true will inhibit the jump back to action step 2 when the Mult or Divide problem is complete.) If the action step counter is in action step 5 (101), 1AS2 and 2AS1 will be false.



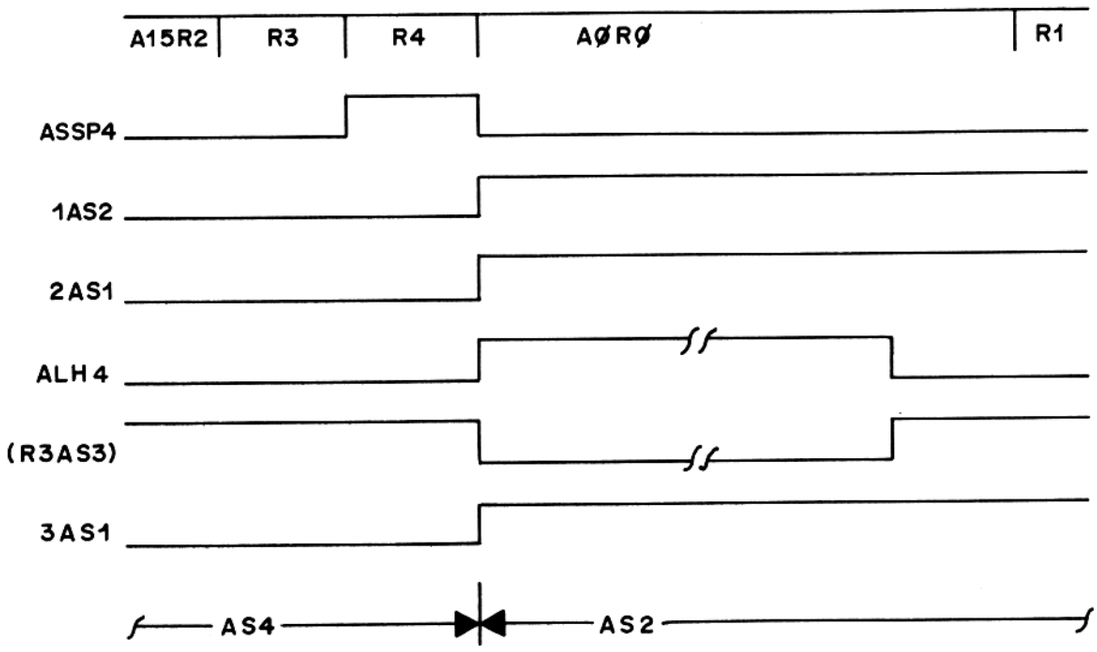
RETURNING TO A.S. 2 FROM STEP 5
(All on "KK" card)

FIGURE 4-43

Then, at arithmetic line home, ALH4 will become false. This will allow the output of the inverter to become true and reset the 3AS flip-flop. When 3AS resets, the 101 (action step 5) configuration changes to 100, which decodes into action step 2.

As will be seen in the discussion which follows, the ASSP4 signal is developed at A15R4 time and the action step counter is stepped to action step 5 at the end of A15R4. Arithmetic line home occurs simultaneously and ALH4 removes the other inhibit from resetting 3AS.

Figure 4-44 shows the timing relationships involved in this jump back to action step 2 during Mult or Divide.



TIMING CHART, RETURNING TO A.S. 2 FROM STEP 5

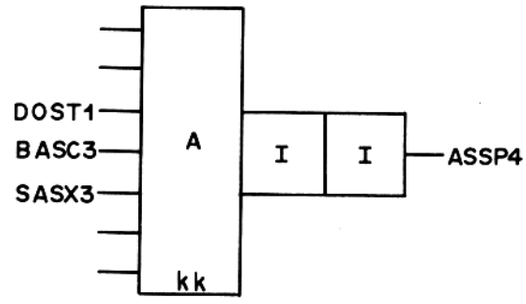
FIGURE 4-44

As stated previously, when the multiply or divide problem is near completion, TMOD3 will be developed, thus inhibiting the jump back to action step 2. The action step counter will then stay in step 5 until the following ASSP4 returns it to step 0.

4. DEVELOPMENT OF THE ACTION STEP SHIFT PULSE

The final factor to be considered in discussing the action step counter is the development of the Action Step Shift Pulse, ASSP4. Like many shift pulses in the 5610, this signal is true most of the time. However, it must be remembered that the shift actually takes place on the true-going flank of the signal. Since the normal state of this signal is true, it must first go false, and then true again, before the action step counter can be incremented.

Refer to Figure 4-45 and note that a true signal on any input to the gate will hold ASSP4 true. The four unmarked inputs hold ASSP4 true at certain times during the Enter and Print commands, and during some arithmetic functions, such as during an overdraft in subtract.

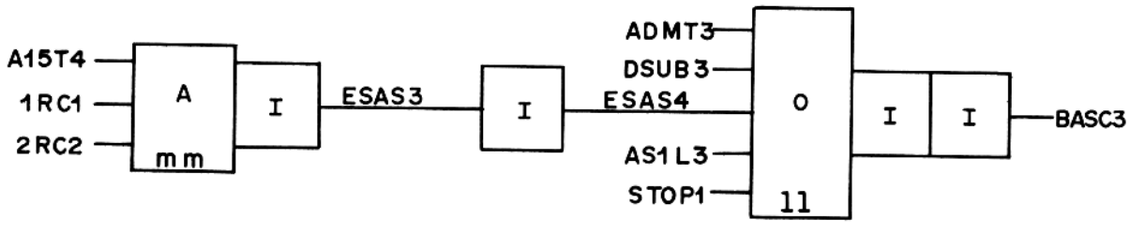


BASIC DEVELOPMENT OF ASSP4

FIGURE 4-45

As stated earlier, DOST must be reset before an arithmetic function can take place. Therefore, DOST1 holds ASSP4 true during all non-arithmetic commands. SASX3 (Storage Action Stop eXtra) is developed at times to hold ASSP4 true during those commands dealing with the storage line (Store, Recall, and Accumulate).

During all other arithmetic commands the final true signal to be removed from the gate is BASC3 (Block Action Step Counter). Refer to Figure 4-46 and trace the development of BASC3. ESAS4 (Enable Shift of Action Step) will be true at all times except during A15R4 time. If all other inputs to the BASC3 gate are false, BASC3 will become false during A15R4 time and allow ASSP4 to become false. All of the other inputs in Figure 4-46 should be considered as clamps to ASSP4. In other words, ESAS4 will allow ASSP4 to become false at A15R4 time if no other clamp signals are true at that time. At the end of A15R4 time BASC3 again becomes true, causing ASSP4 to return to true and increment the action step counter.

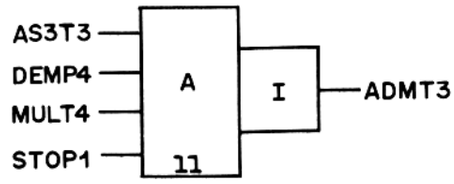


DEVELOPMENT OF BASC3

FIGURE 4-46

The other signals which will produce BASC3, and thus clamp ASSP4, are as follows:

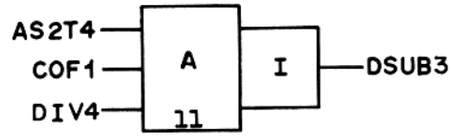
1. ADMT3 (ADd in MulT, Figure 4-47). Since mult is a series of additions, the processor must stay in action step 3 until a given number of additions is completed. In the example given previously, when multiplying a number by 25, the number is added twice during action step 3. Then the counter is incremented and the jump made back to action step 2, where the 25 is shifted one place to the left. When back in action step 3 the number is added five more times. Since it takes one complete line pass to add a number once, ADMT3 prevents the action step counter from incrementing after A15R4 time until the number has been added twice, in the first case, and five times, in the second case.



DEVELOPMENT OF ADMT3

FIGURE 4-47

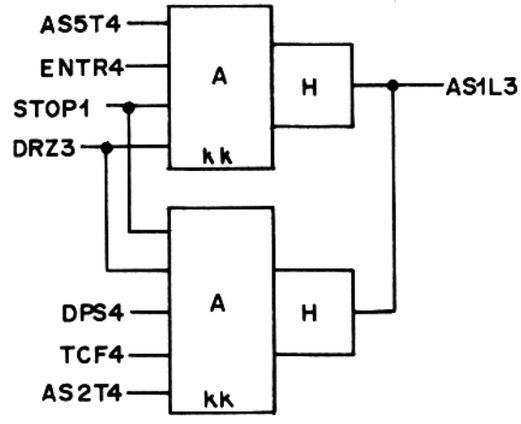
2. DSUB3 (Division SUBtract, Figure 4-48). This can also be called subtract in divide, and likened to add in mult. It prevents the action step counter from stepping in divide until a series of subtractions is completed.



DEVELOPMENT OF DSUB3

FIGURE 4-48

3. AS1L3 (Align Shift 1 Left, Figure 4-49). This signal is produced at certain times during the Enter and Constant commands to achieve decimal alignment.



DEVELOPMENT OF AS1L3

FIGURE 4-49

4. STOP1. This is a test signal which enables the technician to stop the machine operation when a particular action step is reached.

All signals controlling ASSP4 are shown in Figure 4-50. It should be remembered that the final true signal to be removed in all operations except those using the storage line is ESAS4. In other words, ESAS4 can be considered as the "permiss", and all other signals thought of as clamps. The development of these clamp signals will be studied in greater detail as the individual commands are reviewed. However, the way in which these signals prevent the action step counter from stepping should be thoroughly understood before moving on.

5. SUMMARY

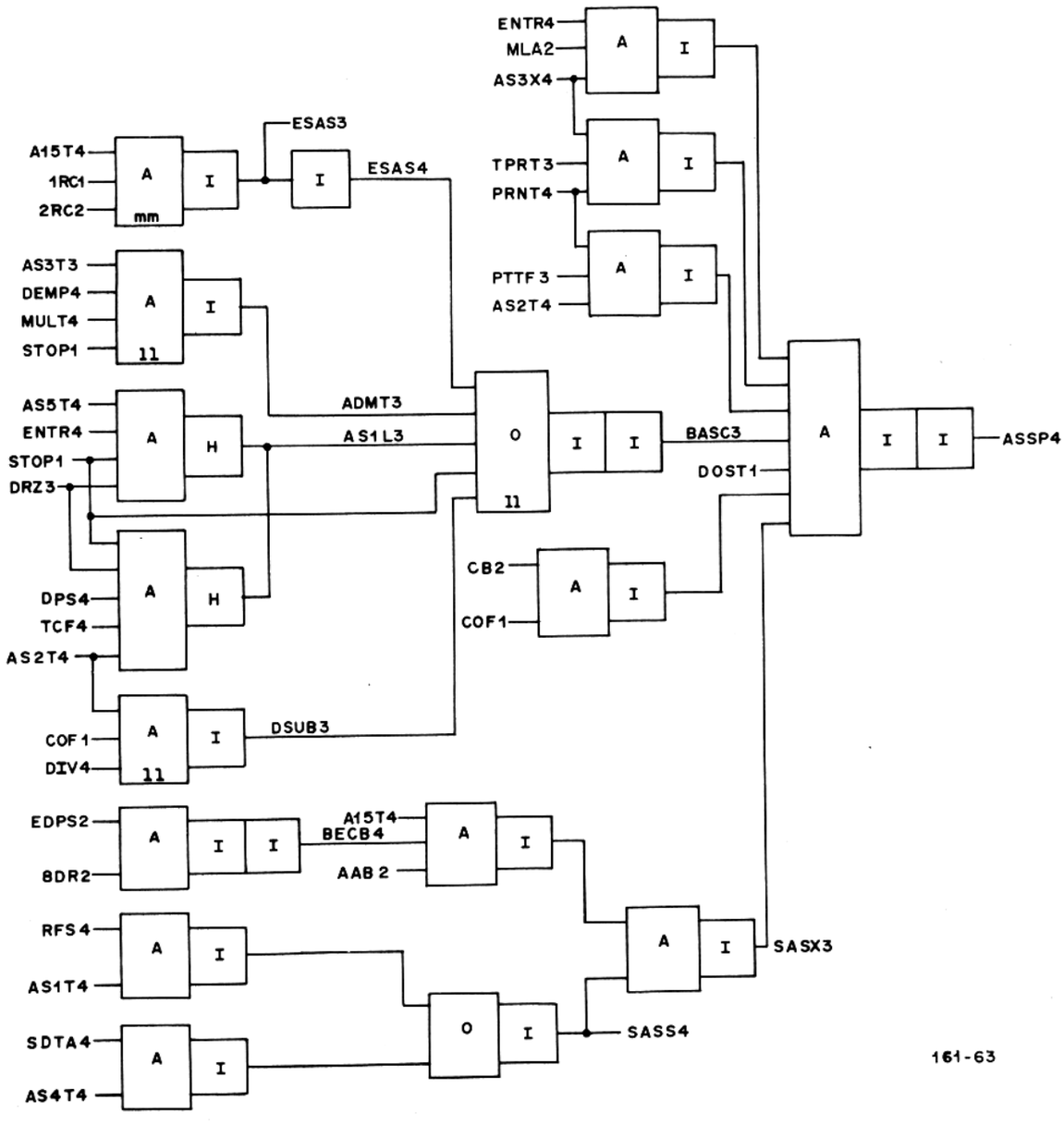
In short arithmetic functions the action step counter will count from step 0 to 1 and to 2 and then back to 0. The more complex arithmetic functions develop FPOC, which allows ASSP4 (during action step 2) to set 1AS and cause a step to 3 instead of to 0. Subsequent shift pulses step the counter through action steps 4 and 5. AS5T3 is applied to the gate controlling 1AS so that when ASSP4 occurs during action step 5 the counter returns to 0 instead of to 3.

In multiply and divide it is necessary to jump back to action step 2 from step 4 in order to complete a series of additions or subtractions. This is accomplished by resetting the 3AS flip-flop coincident with the step from 4 to 5, so that step 5 becomes step 2 instead. When the multiplication or division is completed, the gate used to reset 3AS is disabled and the step to 5 occurs normally.

The Action Step Shift Pulse is normally true, but shifts the counter only on its true-going flank. In order to shift the counter, ASSP4 must first go false and then go true again. When ASSP4 is false, it returns to true at the end of A15R4 time, which is coincident with ALH1. Therefore, the action step counter always steps coincident with arithmetic line home.

6. ARITHMETIC OPERATIONS CHART

The chart in Figure 4-51 summarizes the various arithmetic functions that occur during each of the action steps.



DEVELOPMENT OF ASSP4
(All gates on "KK" card unless otherwise noted)

FIGURE 4-50

ASC	1 ØØ1	2 1ØØ	3 11Ø	4 Ø11	5 1Ø1	Ø Ø1Ø
ENTER		SHIFT UP RØ → R4	ALLOW ENTRY SHIFT R1 LEFT	CHANGE SIGN IF NEC.	DECIMAL ALIGN	WRITE RØ TO ZERO
CONSTANT FACTOR FOLLOWS						
C.F. DIGIT OR DECIMAL	PERFORM ENTRY SHIFT LEFT					
CHANGE SIGN	CHANGE SIGN					
TERMINATE C.F.	CHANGE SIGN IF OP = CSGN(Minus)	DECIMAL ALIGN				
TRUNCATE	WRITE ZEROS INTO LSD					
DELETE	SHIFT DOWN R1 → R4					
NON DEC PRINT	SHIFT R1 INTO RØ RETAIN R1	SLUFF OFF EXCESS DIGITS	SHIFT R1 LEFT, PRINT DIGIT, REPEAT UNTIL FINISHED		INVERT R1, RØ (SHIFT R1 → RØ, RØ → R1)	
DECIMAL PRINT			PRINT DECIMAL			
DUPLICATE		SHIFT UP RØ → R4				
RECALL	SLC: SHIFT DATA INTO DTR. ALSO RETAIN. SHIFT DTR → CR; RØ LEFT VIA CR WRITING RECOVERED STORAGE INTO RØ					
ACCUMULATE		ALGEBRAICALLY ADD OR SUBTRACT RØ AND R1	SHIFT DOWN R1 → R4	SHIFT RØ LEFT, MSD INTO CR SHIFT CR → DTR SHIFT DTR → SLC		A1 - A15
STORE						
ADD	SHIFT DOWN R1 → R4					
SUBTRACT						
MULTIPLY	SHIFT R1 → RØ, ADD SIGNS; SET DR TO 13	SHIFT R2 LEFT	ADD RØ & R1. ADD CARRIES INTO R2. STAY UNTIL FINISHED	SHIFT R1 LEFT COUNT DOWN DR	SHIFT DOWN R1 → R4	
DIVIDE	SET DR TO 14	SUB RØ FROM R1 ADD 1 TO R2. STAY UNTIL O'DRAFT ADD RØ TO R1	SHIFT R2 LEFT			
ASC	1 ØØ1	2 1ØØ	3 11Ø	4 Ø11	5 1Ø1	Ø Ø1Ø

161-13 Rev. 1

ARITHMETIC OPERATIONS CHART

FIGURE 4-51

C. TRUNCATE

The purpose of the Truncate instruction is to "clear" the least significant digits out of R1. Any number of digits, from 1 to 13, can be replaced by zeros, depending upon the address of the Truncate instruction. However, the number of digits cleared is always counted from the right, or least significant digit.

The Truncate command is represented by the letter "I" code, 1-4-5-6-7. The chart in Figure 4-52 lists the addresses possible and the number of digits each address converts to zero.

NO. OF DIGITS REPLACED	ADDRESS CODE
1-9	1-9
10	+
11	F1
12	F2
13	F8

TRUNCATE ADDRESSES

FIGURE 4-52

1. GENERAL OPERATION

The Truncate command is retrieved in 4 mode 3 phase and its address is retrieved in 4 phase. At the end of 4 phase the address is set into the decimal register. The logic then steps to 5 phase where it waits for program line home, at which time a jump to 5 mode 15 phase occurs. As soon as the DOST flip-flop resets, the phase counter jumps to 6 phase, where it waits for the Truncate instruction to be completed.

With DOST reset, the action step counter goes to 1, where a signal (TCDR3) is developed each R1 time to block the writing onto the arithmetic line. This signal also counts the decimal register down once for each R1 time, and when the decimal register reaches zero, further inhibiting of the arithmetic line write signal is prevented. Thus, a number of digits in R1 time, equal to the address of the Truncate instruction, are converted to zero.

The Truncate is complete after the first arithmetic line pass in action step 1, so the logic merely idles through action step 2. Then the action step counter returns to 0 and DOST again becomes set, allowing the phase counter to step to 7 phase. At program line home a jump to 3M6,4 occurs and the search for the next instruction begins.

2. 4 MODE OPERATION

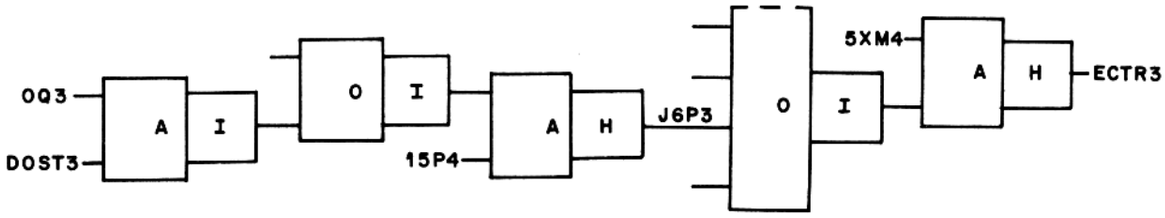
When a Truncate command is retrieved in 4 mode 3 phase, it is recognized as a 1-address command, and the logic steps to 4 phase where the address is shifted into the common register. At the end of 4 phase the address is set into the decimal register when CRDR is produced. Refer to Section 3G, Retrieve, for review. This is the only time that CRDR is really necessary in 4 phase.

The logic then steps to 5 phase where the marker is written out (behind the Truncate address) and the balance of the program data passes through the search register. When program line home is reached, a jump to 5 mode 15 phase takes place.

3. 5 MODE OPERATION

The only "Op Decoder" output developed by the Truncate command is the O1AD3 signal, so nothing prevents the reset of DOST at A1 time in 5 mode 15 phase. At this point two independent actions occur: the action step counter steps from 0 to 1 at the end of the current arithmetic line pass, and the jump to 6 phase is enabled.

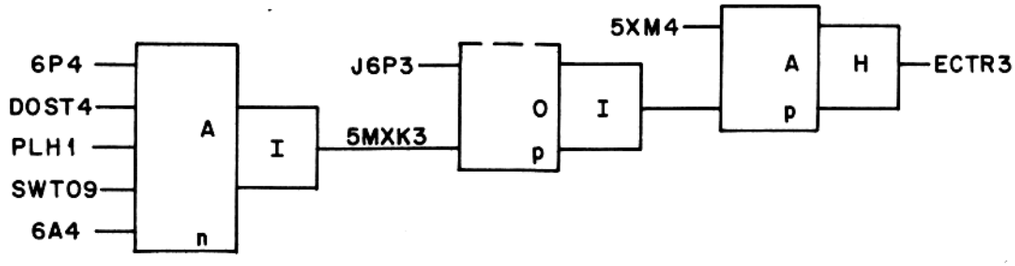
a. PROGRAM LINE ACTION. Because the phase counter jumps to 15 during program line home, the next T6B, which causes the jump to 6 phase, does not occur until after the timing chain restarts. This causes 15 phase to remain true during program line home. Note, however, that 6 phase is set into the phase hold flip-flops and ECTR3 is developed as soon as DOST resets. Both of these actions occur as a result of developing J6P3, as shown in Figure 4-53. Because both of these actions occur during program line home, T1A3 is true and "A" time, along with ECTR, sets the SPC flip-flop. Thus, the phase counter is primed for a jump to 6 phase, awaiting only a T6B to trigger the jump.



DEVELOPMENT OF J6P3 AND ECTR3 IN 5M15P
(All on "P" card)

FIGURE 4-53

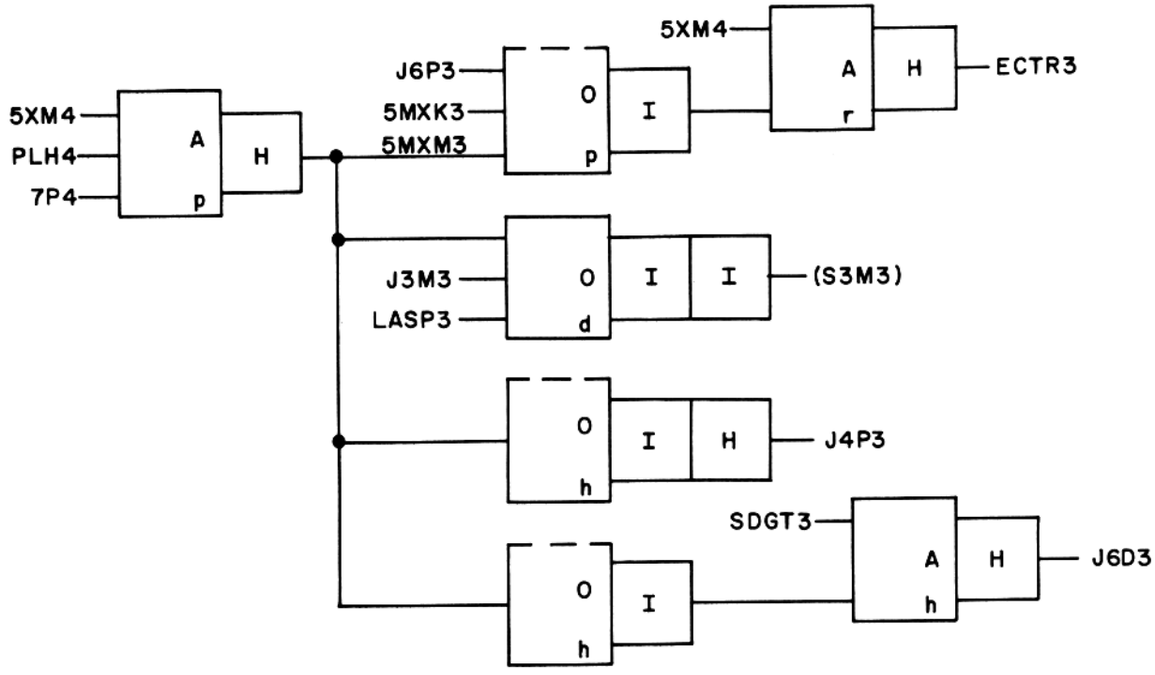
Due to differences in programs, the length of program line home is variable: when the timing chain finally restarts, the next T6B may occur during the ensuing arithmetic line action, or it may not occur until well after the Truncate action is complete. If the Truncate action is complete when the jump to 6 phase finally takes place, the next T6B causes a step to 7 phase. If the Truncate is not complete, however, the step to 7 phase waits until DOST is again set. In either case, the step to 7 phase will not occur until DOST sets, after the truncate is complete. This is illustrated in Figure 4-54.



DEVELOPMENT OF ECTR3 IN 5M6P

FIGURE 4-54

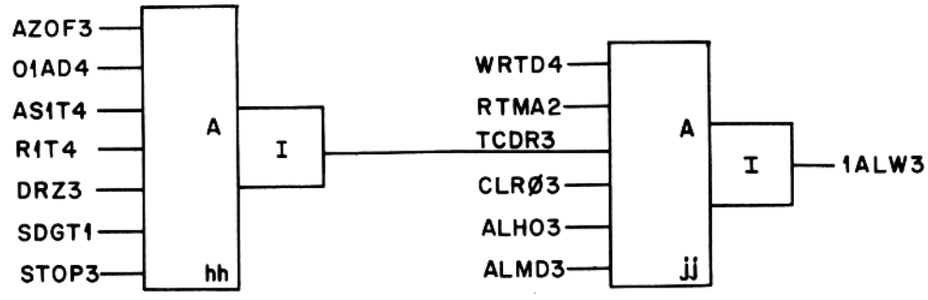
In 7 phase the phase counter waits for program line home. Then 5MXM3 is produced, which enables the jump to 3M6,4 to initiate marker search. See Figure 4-55.



JUMP TO 3M6,4 FROM 5M7P

FIGURE 4-55

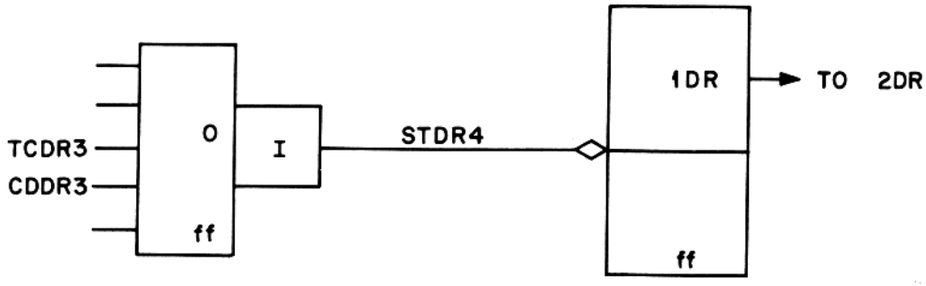
b. ARITHMETIC LINE OPERATION. When action step 1 is realized is 5 mode, TCDR3 is produced during R1 time starting in arithmetic column 2. See Figure 4-56. Writing onto the arithmetic line is still direct, via 1ALW3, so each time TCDR3 goes true, 1ALW3 is blocked and the digit is changed to a zero on the line. This is also shown in Figure 4-56.



DEVELOPMENT OF TCDR3

FIGURE 4-56

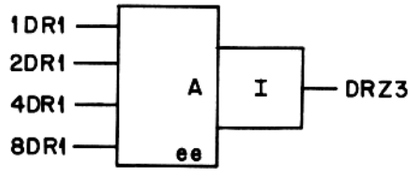
When TCDR3 returns to false (at the end of R1 time) the decimal register is counted down, as shown in Figure 4-57.



EFFECT OF TCDR ON DECIMAL REGISTER

FIGURE 4-57

This process repeats, with TCDR3 blocking the writing onto the arithmetic line and counting down the decimal register, until the decimal register is counted down to zero. The Decimal Register Zero signal, DRZ3, is then produced, as shown in Figure 4-58.



DEVELOPMENT OF DRZ3

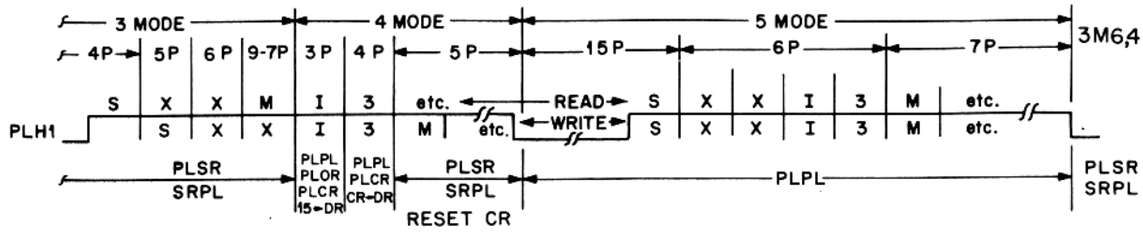
FIGURE 4-58

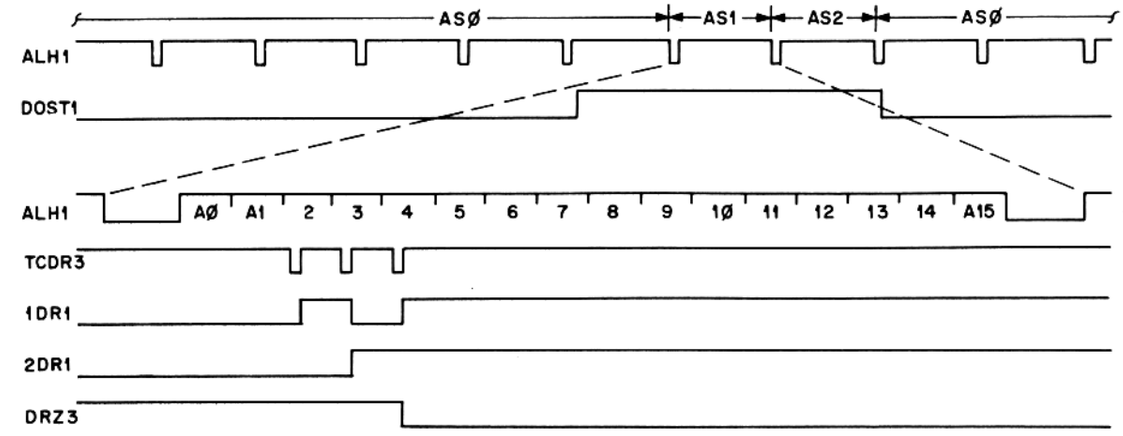
DRZ3 immediately prevents any further development of TCDR3 (refer to Figure 4-56), and the balance of the R1 data is unaffected as the line pass is completed.

At the end of the line pass, the action step counter goes to 2 and the next line pass causes no further change in data. At the end of the second line pass the action step counter returns to 0, and DOST becomes set at the beginning of the third arithmetic line pass. As noted earlier, the phase counter then waits for the timing chain to restart for the jump to 6 phase, or steps to 7 phase if the timing chain is running and the jump to 6 phase has already occurred.

4. TIMING CHART

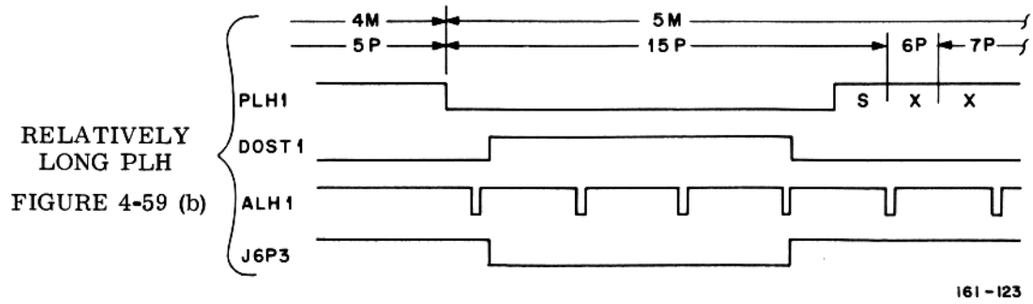
The timing chart in Figure 4-59 (a) represents normal action when a "Truncate 3" instruction is retrieved and the program is fairly long, causing a relatively short program line home. Figure 4-59 (b) illustrates the variations caused by a shorter program and a correspondingly longer program line home.





RELATIVELY
SHORT PLH

FIGURE 4-59 (a)



RELATIVELY
LONG PLH

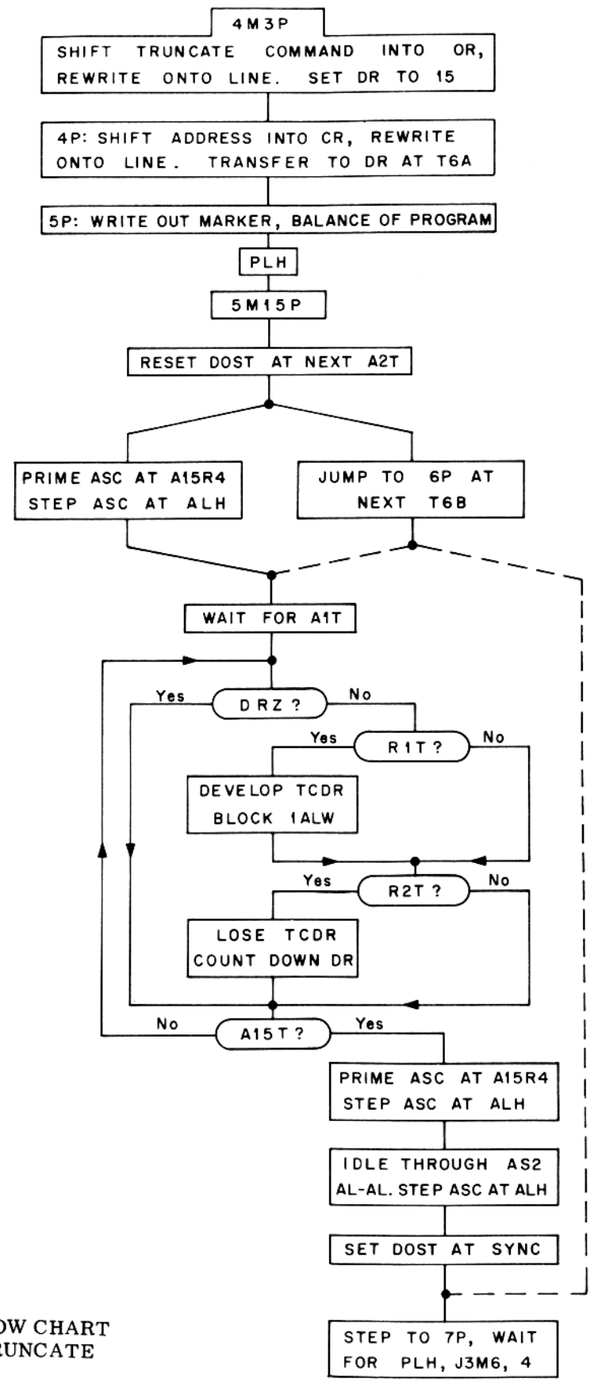
FIGURE 4-59 (b)

TRUNCATE OPERATION

FIGURE 4-59

5. FLOW CHART

The flow chart in Figure 4-60 depicts normal Truncate action. The dotted line after "jump to 6 phase" denotes the fact that phase counter action is dependent upon DOST, and not upon the action step counter. The next T6B may occur anywhere between the points bracketed by the two dotted lines.



FLOW CHART
TRUNCATE

FIGURE 4-60

161-19

D. DELETE

Execution of the Delete instruction shifts the operating stack down, destroying the contents of R1 and duplicating the contents of R4 into both R3 and R4. The Delete command (letter D, 3-6-7 code) is a 0-address command, its address (the entire operating stack) being implied by the nature of the command.

1. GENERAL OPERATION

The Delete command is retrieved in 4 mode and at the next program line home a jump to 5 mode 15 phase occurs. DOST resets at the next A1 time, allowing a jump to 6 phase at the next T6B. (Unless program line home is very short, the delete operation will be complete by the time the program line timing chain restarts.)

With DOST reset the action step counter steps to 1, where the stack is shifted down. The processor idles through action step 2 and then returns to 0, where a step to 7 phase is enabled. After 7 phase is reached, the next program line home allows a jump to 3M6,4.

Refer to Figure 4-61 and note that the former R1 data, having been shifted down in action step 1, is in R0 during action step 2. It will remain there until DOST again sets and R0 is cleared.

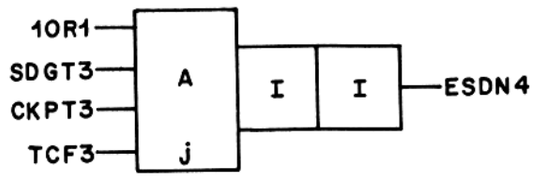
	AS0	AS2	AS0
R4	0000012345610	0000012345610	0000012345610
R3	0000000000014	0000012345610	0000012345610
R2	0000000657922-	0000000000014	0000000000014
R1	0000000000021	0000000657922-	0000000657922-
R0	0000000000000	0000000000021	0000000000000

DELETE EXAMPLE

FIGURE 4-61

2. 4 MODE OPERATION

As soon as the Delete command is retrieved in 4 mode 3 phase, Enable Shift DowN, ESDN, is developed as shown in Figure 4-62. Being a 0-address command, the Delete then causes a jump to 5 phase, and at program line home a jump to 5 mode 15 phase occurs.



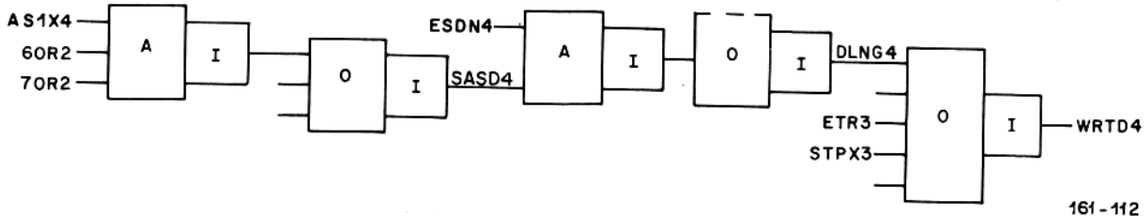
DEVELOPMENT OF ESDN4

FIGURE 4-62

3. 5 MODE OPERATION

DOST resets at the first A1 time in 5 mode 15 phase. This allows the action step counter to go to step 1 at the end of the arithmetic line pass. DOST being reset also enables a jump to 6 phase, which occurs at the first T6B after the timing chain restarts. The program line and phase counter action that occurs at this time is covered in detail in the previous section (4C) on Truncate: refer to that section for review.

In action step 1 the fact that the 6OR and 7OR flip-flops are set develops Storage and Arithmetic Shift Down, SASD, which combines with ESDN4 to drive DLNG4 false. This is shown in Figure 4-63. With DLNG4 false, the output of the arithmetic line is shifted into the common register.

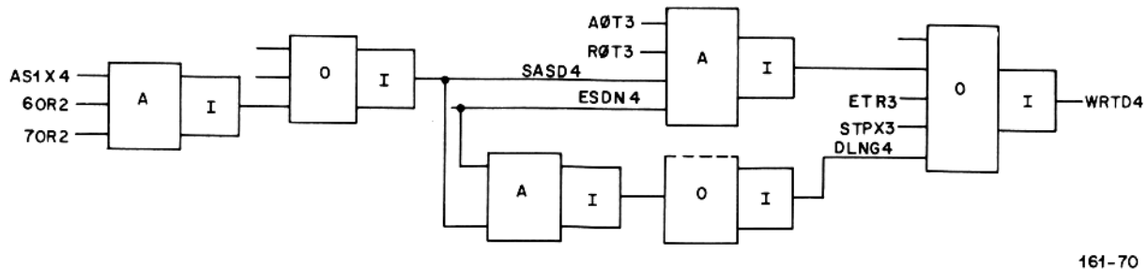


DEVELOPMENT OF DLNG4 AND WRTD4

FIGURE 4-63

As was noted earlier, the normal "idle" condition sees DLNG4 as the only true signal holding WRTD4 false, thus making "write direct" true. Unless some other input to the WRTD4 gate in Figure 4-63 is driven true, WRTD4 will go true when DLNG4 goes false. If WRTD4 goes true (and DLNG4 is false) the line routing will be ALCR, CRAL.

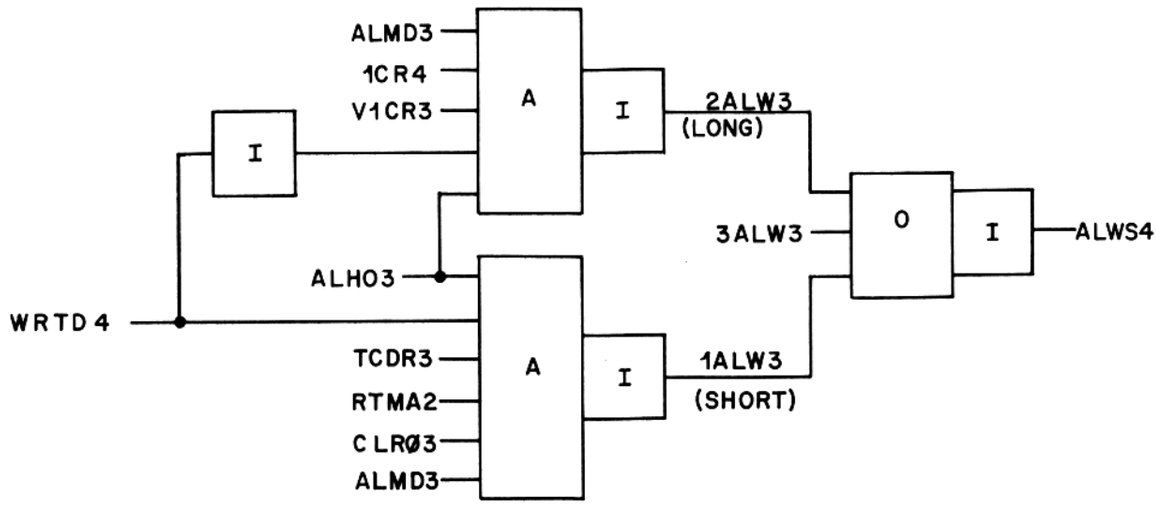
During Delete, one of the two unmarked inputs in Figure 4-63 goes true at certain times to allow writing to be direct, even though "data long" is true. This happens during R1, R2, R3, and R4 in A1 through A15. Refer now to Figure 4-64 and note that the output of the larger AND gate is true in action step 1 at all times except during A0 and during each R1 time. Thus "write direct" is true (WRTD4 is false) and data follows the short path back onto the delay line (ALAL) during R1, R2, R3, and R4, except in column 0.



WRITING DIRECT WHILE "DATA LONG" IS TRUE

FIGURE 4-64

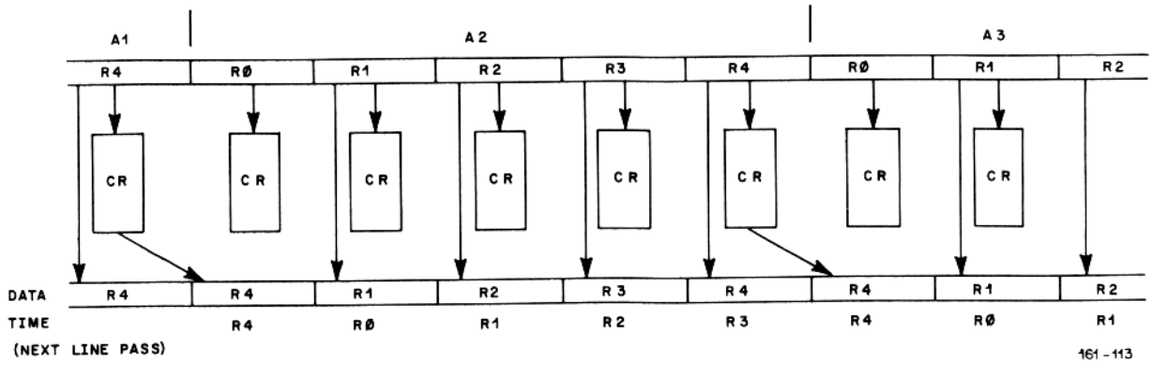
While WRTD4 is false, data follows the short path and is rewritten via 1ALW3. See Figure 4-65. Throughout A0 and during each R0, however, WRTD4 becomes true and blocks 1ALW3. At the same time 2ALW3 is enabled. Notice that 1ALW3 allows the contents of RTMA to be written onto the line (the short path); 2ALW3 allows the contents of 1CR to be written out (the long path).



TWO SOURCES OF DATA TO BE WRITTEN
(All on "JJ" card)

FIGURE 4-65

It is necessary to block write direct during A0 time in order to write the sync pulse "long". The rest of the data being written short will then cause the stack to move down. However, "write direct" is also false (WRTD4 is true) during each R0 time. The reason for this is to write the R4 data a second time. During R4 time the data coming off the line follows two paths; the short path back onto the line and the long path into the common register. In the ensuing R0 time the same data is written onto the line from the common register. (This writing of R4 data twice is sometimes referred to as "constants from heaven".) The R0 data flows into the common register, but it is not rewritten during the next R1 time because writing then goes back to the direct path. Thus, the R0 data is lost. (This is of no consequence since R0 had been clear when the Delete instruction was initiated.) This process is illustrated in Figure 4-66.

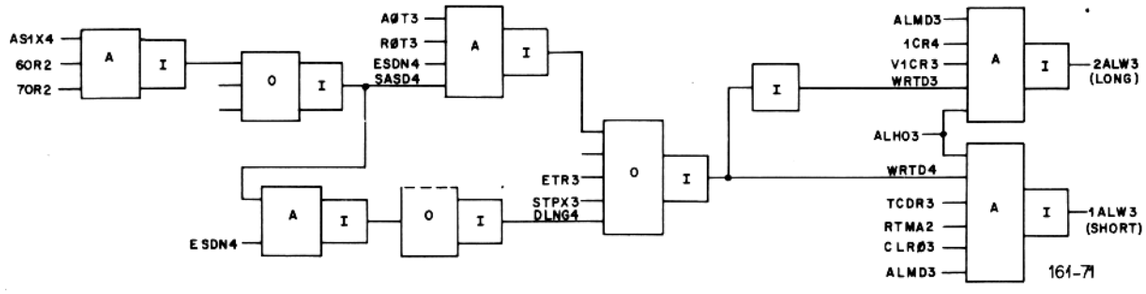


DELETE
BASIC DATA FLOW

FIGURE 4-66

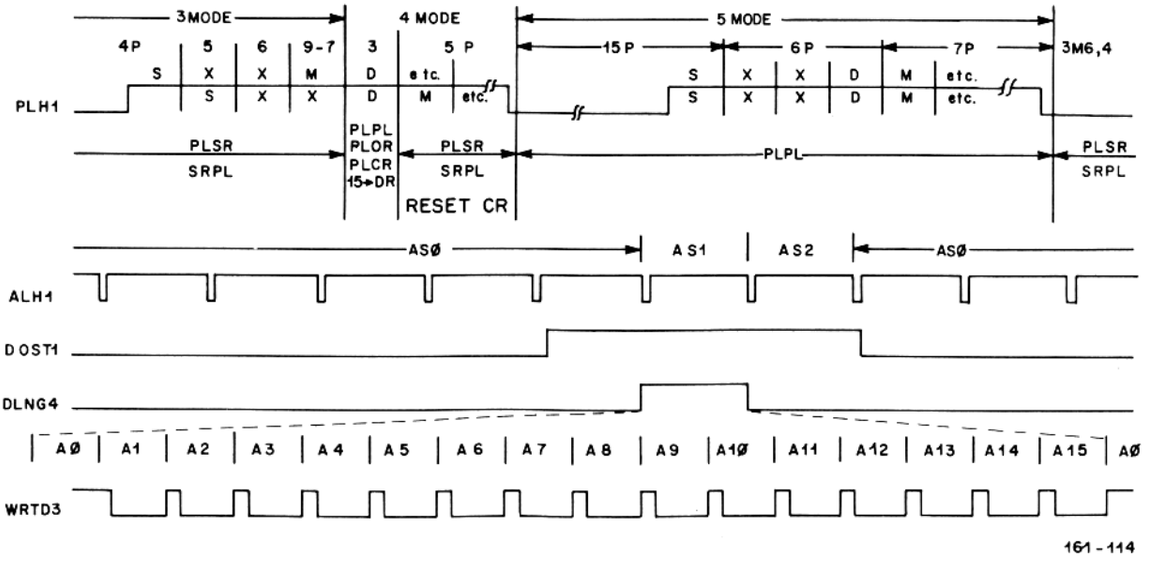
At the end of the delete line pass the action step counter steps to 2 and one more line pass occurs with all data being written direct. Since the sync pulse is now "delayed" in respect to the rest of the data, the stack has moved "down" in relation to the timing chain, with the former R1 data now in R0, etc. At the end of this line pass the action step counter returns to 0, DOST sets at the next arithmetic sync pulse, and the step to 7 phase occurs at the next T6B. At program line home a jump to 3M6,4 occurs. Also, the "clear R0" routine occurs after DOST sets.

All of the logic used in the delete operation is brought together in Figure 4-67, and the timing chart in Figure 4-68 summarizes the entire process.



DELETE LOGIC
(All on "JJ" card)

FIGURE 4-67



TIMING CHART
DELETE

FIGURE 4-68

E. DUPLICATE

When the Duplicate instruction is executed, the contents of R1 are duplicated into R2. The operating stack shifts up, the contents of R2 moving into R3 and the contents of R3 moving to R4. R4 data is lost.

1. GENERAL OPERATION

The overall operation of the Duplicate instruction is quite similar to that of the Delete instruction or any other 0-address arithmetic instruction. The Duplicate command is retrieved in 4 mode and at the next program line home a jump to 5 mode 15 phase occurs. DOST then resets, enabling a jump to 6 phase, which occurs at the next T6B.

DOST being reset also unlocks the action step counter, allowing it to go to 1. During the one arithmetic line pass in action step 1 the contents of R1 are duplicated into R0. The action step counter then steps to 2, where the entire operating stack is shifted up. Although the duplication could be accomplished more directly in only one line pass, this method allows the use of logic that is already in the machine for other purposes, resulting in an overall saving in components.

As the stack is shifted up in action step 2, R4 data is shifted into R0. R0 is then cleared when the action step counter returns to 0 and DOST again sets. DOST being set also enables a step to 7 phase, and at the next program line home a jump to 3M6,4 occurs.

Figure 4-69 summarizes the arithmetic action during execution of a Duplicate instruction. Note that as the shift up occurs in action step 2, R4 data is shifted into R0, but is displaced by one column. This is unimportant since R0 is cleared on the next line pass.

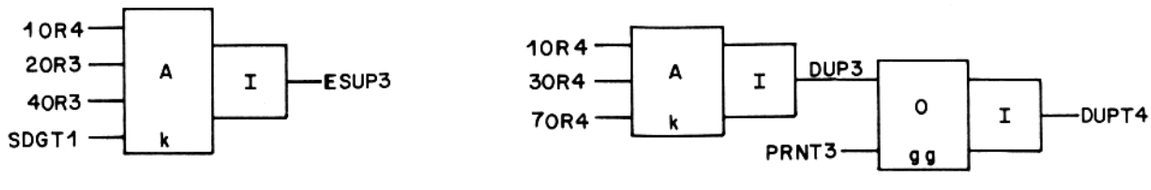
	AS0	End of AS1	End of AS2	End of AS0
R4	0000012345610	0000012345610	0000000000014	0000000000014
R3	0000000000014	0000000000014	0000000657922-	0000000657922-
R2	0000000657922-	0000000657922-	0000000000021	0000000000021
R1	0000000000021	0000000000021	0000000000021	0000000000021
R0	0000000000000	0000000000021	0000123456100	0000000000000

DUPLICATE EXAMPLE

FIGURE 4-69

2. 4 MODE OPERATION

Retrieval of the Duplicate command in 4 mode 3 phase immediately develops three "master" signals, DUPlicate, DUP3, Enable Shift UP, ESUP3, and DUplicate or PrinT, DUPT4. These are shown in Figure 4-70.



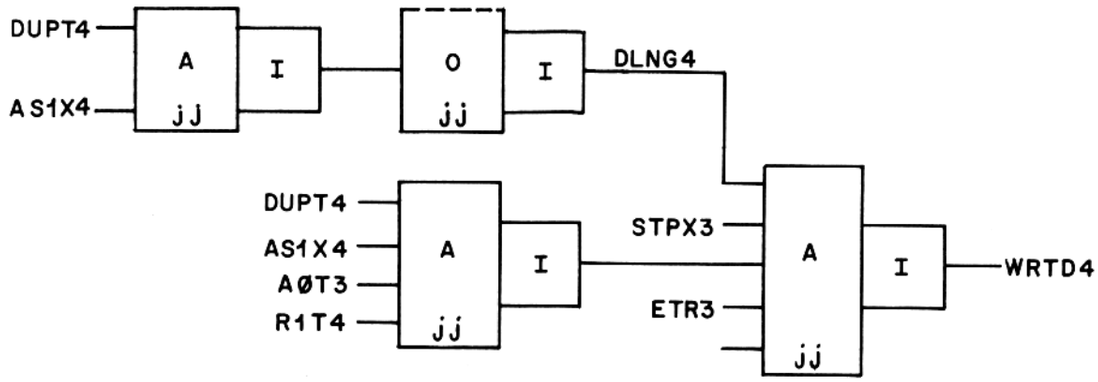
MASTER SIGNALS IN DUPLICATE

FIGURE 4-70

3. 5 MODE OPERATION

DOST resets at the first A1 time in 5 mode 15 phase. This allows the action step counter to step to 1 at the end of the arithmetic line pass. DOST resetting also enables a jump to 6 phase which occurs at the first T6B after the program line timing chain restarts. For a detailed description of program line action at this point, refer to Section 4C, Truncate.

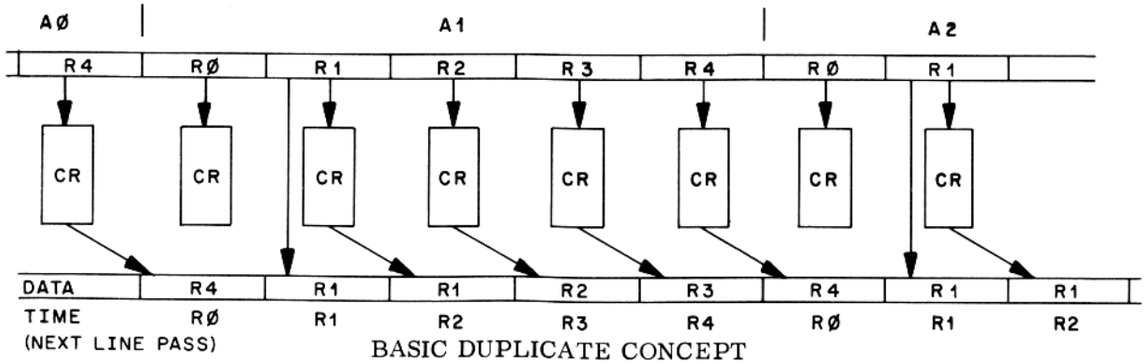
a. ACTION STEP 1. When action step 1 is reached, "data long" goes true (DLNG4 goes false), as shown in Figure 4-71. The sync pulse is written long, as is most other data. Note, however, that during each R1 time, WRTD4 is driven false, which makes "write direct" true. This is also shown in Figure 4-71. Note that write direct is blocked in A0 time so that the sync pulse can be written long.



LINE ROUTING IN ACTION STEP 1

FIGURE 4-71

This procedure is summarized in Figure 4-72. The data path is long (ALCR, CRAL) throughout A0 time, so that A0R4 data (blank) is in the common register at the end of A0 time. The line routing does not change in A1R0 time, so the A0R4 data is written onto the line in A1R0 time, while the A1R0 data is shifted into the common register. At A1R1 time the data path changes to short, or direct (ALAL), and the A1R1 data is written directly onto the line. Note, however, that DLNG4 remains false even though the data path is direct: this allows the A1R1 data to be shifted into the common register in addition to being rewritten. As the A1R1 data is being shifted into the common register, the A1R0 data is shifted out and lost. At A1R2 time the data path returns to long (ALCR, CRAL) and the A1R1 data is written out a second time as the A1R2 data is shifted into the common register. The data path then remains long until the next R1 time (in this case, A2R1), when the process repeats.



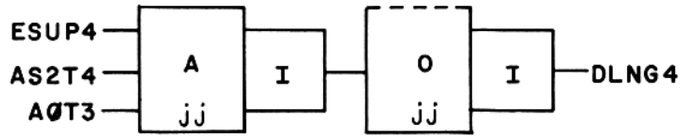
BASIC DUPLICATE CONCEPT

FIGURE 4-72

This is one of the cases when both "write direct" and "data long" are true at the same time. Note that the data path is ALCR throughout the line pass, and that write direct (WRTD4) determines whether the data going back onto the line will come from the common register (CRAL) or from RTMA (ALAL).

Note in the preceding discussion that R1 data was written twice: at R1 time and again at R2 time. R2 data was written at R3 time, R3 data at R4 time, and R4 data at R0 time. However, since the sync pulse was also written long, the data maintains the same relationship to the sync pulse, and its position in the operating stack is unchanged.

b. ACTION STEP 2. At the end of the "duplicate" line pass, the action step counter increments to 2. This drives DLNG4 true again (see Figure 4-71) so that the line pass in action step 2 begins with "write direct" being true. Thus the sync pulse is written direct. At A1 time DLNG4 is again driven false, as shown in Figure 4-73, and the long data path is resumed. The line routing remains ALCR, CRAL for the rest of this line pass.

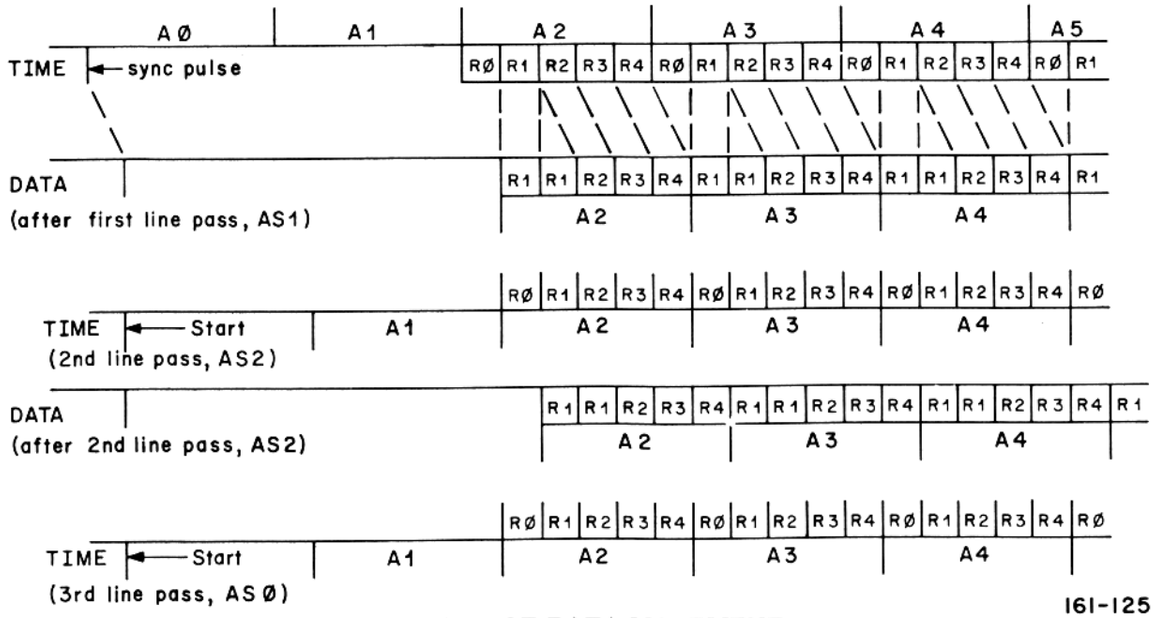


LINE ROUTING IN ACTION STEP 2

FIGURE 4-73

With the sync pulse written direct and the rest of the data written long, the effect of shifting all registers up is accomplished. The original R1 data is now in R1 and R2, R2 data is in R3, R3 data is in R4, and R4 data is in R0 (but displaced one column to the left).

The arithmetic operation in both action steps is shown in Figure 4-74. Note the distinction between "time" and "data" in this illustration. At the end of action step 1 the R1 data has been duplicated into both R0 and R1, and by the end of action step 2 the stack has been shifted up.



SUMMARY OF DATA MOVEMENT

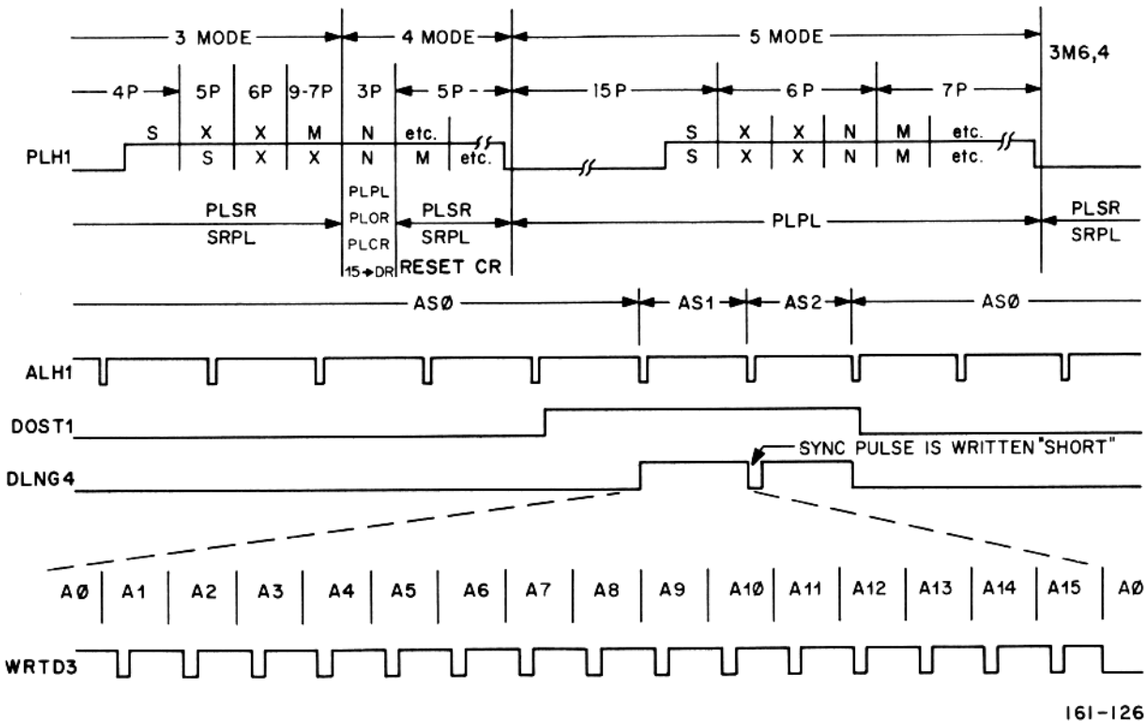
FIGURE 4-74

c. ACTION STEP 0. Upon completion of the second line pass all that remains is to clear the former R4 data out of R0. This is accomplished automatically when DOST again sets after the action step counter returns to 0.

DOST being set also enables a jump to 7 phase, and at the next program line home a jump to 3M6,4 takes place.

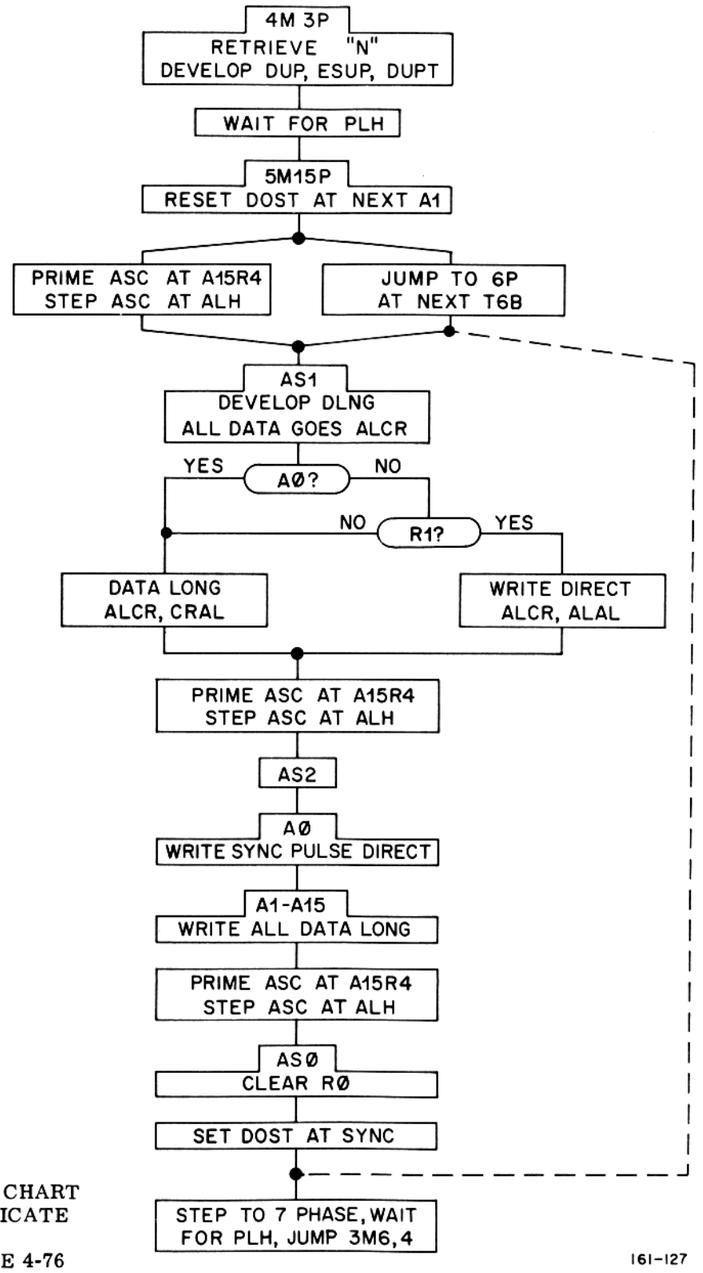
4. TIMING CHART AND FLOW CHART

Execution of the Duplicate instruction is summarized in the timing chart in Figure 4-75 and in the flow chart in Figure 4-76.



TIMING CHART
DUPLICATE

FIGURE 4-75



FLOW CHART DUPLICATE

FIGURE 4-76

161-127

F. CHANGE SIGN

The Change Sign instruction, when executed, converts the sign of R1 from positive to negative, or vice-versa, depending upon the original state. By developing inVert ReTime Arithmetic, VRTA3, at the proper time, the presence or absence of a sign bit in R1 is "inverted", changing the sign of R1.

1. GENERAL OPERATION

When the Change Sign command (a "dash", or lone 7-bit) is retrieved in 4 mode 3 phase, it causes a jump to 5 phase. The logic waits in 5 phase until program line home occurs, and then jumps to 5 mode 15 phase. Next DOST resets, enabling a jump to 6 phase where the logic remains until the Change Sign operation is complete.

With DOST reset, the action step counter goes to 1, where the arithmetic line data is routed "long" through the common register. The Change SiGN signal, CSGN3, is developed during A1R1 time, and this allows the development of VRTA3 during A1R1B0 time. By effectively inverting the state of RTMA during the sign bit time (A1R1B0), the opposite sign is shifted through the common register and written onto the arithmetic line one R-time later.

The sign-changing action is complete after the first line pass (in action step 1), so the logic merely idles through action step 2. After this line pass the action step counter returns to 0 and at the beginning of the next line pass the sync pulse sets DOST. This enables the step to 7 phase, and at the next program line home a jump to 3M6,4 occurs and the search for the next instruction begins.

2. 4 MODE OPERATION

When a Change Sign command is retrieved in 4 mode 3 phase it develops O0AD3 and causes a jump to 5 phase. In 5 phase the marker is written out behind the Change Sign command, and the balance of the program follows.

All of the logic associated with the Change Sign instruction is illustrated in Figure 4-77. As soon as the Change Sign command is retrieved, the "Op Decoder" develops the Terminate Constant Factor signal, TCF3 (which remains true until the next command is retrieved and shifted into the operation register). Nothing further occurs until after program line home is reached, where the jump to 5 mode 15 phase allows DOST to reset.

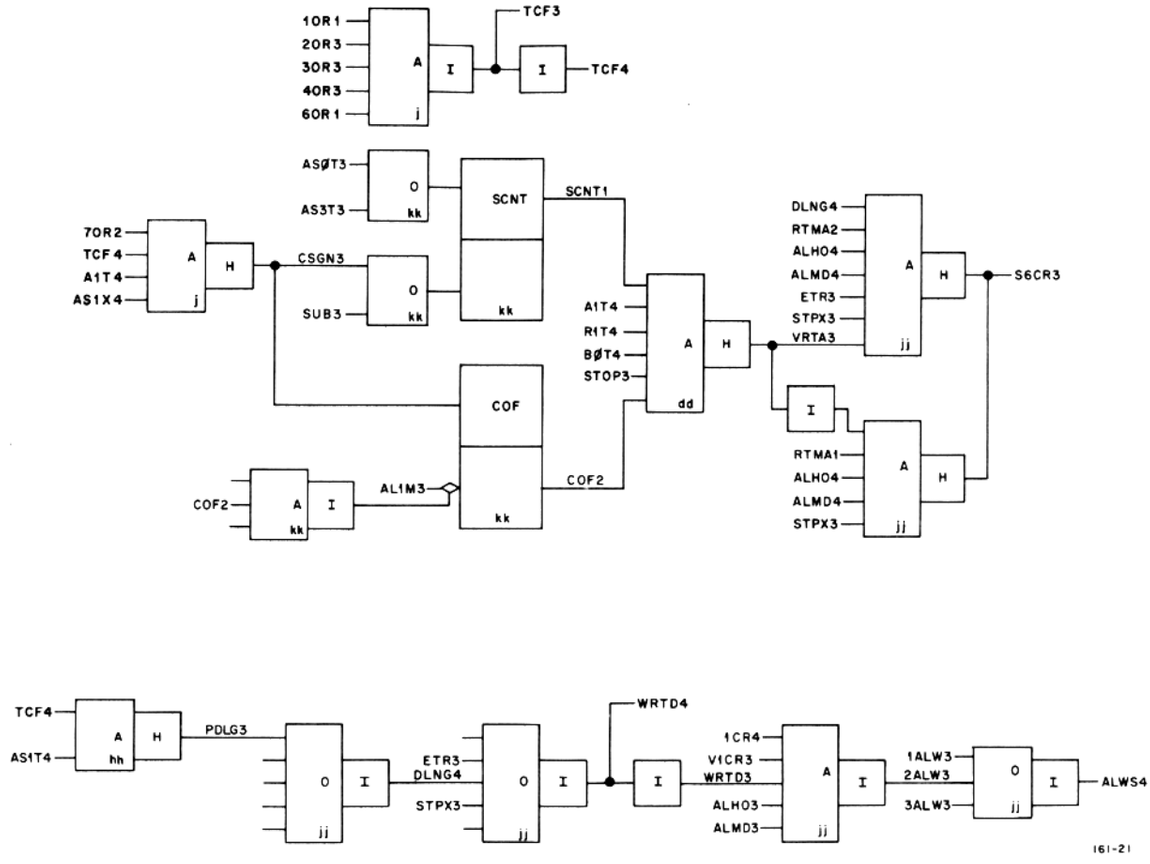
3. 5 MODE OPERATION

After DOST resets, the action step counter steps to 1 at the next program line home, and Pulse Data LonG, PDLG3, becomes true. This drives Data LoNG, DLNG4, false, causing all arithmetic line data to be shifted through the common register before being rewritten onto the delay line.

The Change Sign operation actually takes place during this one line pass in action step 1. The sync pulse is sensed and rewritten during arithmetic column zero (A0 time). During A1 time, CSGN3 is developed, which resets the Sign CoNTrol flip-flop, SCNT, and sets the Carry Over-Flow flip-flop, COF. With SCNT reset and COF set, VRTA3 is produced during A1R1B0 time, the time during which the sign bit of R1 is sensed. This causes the 6CR flip-flop to assume a state opposite to that of RTMA. In other words, if a sign bit is sensed, setting RTMA, 6CR does NOT set, and the sign bit is lost; if no sign bit is sensed and RTMA remains reset, 6CR becomes set. This bit is shifted from 6CR to 4CR at the next bit time, and then to 3CR, 2CR, and 1CR. The bit is then written onto the arithmetic line during the next B0 time (A1R2B0).

CSGN3 returns to false at the end of A1 time. All data continues to pass through the common register before being rewritten onto the line for the remainder of this line pass. At the end of the line pass, two actions occur simultaneously: the COF flip-flop resets and the action step counter goes to 2. In action step 2 PDLG is disabled and writing returns to the direct path. With COF reset, VRTA3 cannot be produced a second time. The system idles through action

step 2 and returns to action step 0 at the end of this line pass. When action step 0 is realized, SCNT becomes set and the next arithmetic sync pulse sets DOST. With DOST set, the step to 7 phase is enabled, providing 6 phase has been realized previously. (Note that Change Sign acts the same as the other "short" arithmetic instructions: the point at which the jump to 6 phase and the subsequent step to 7 phase actually occur is dependent upon the length of program line home.)

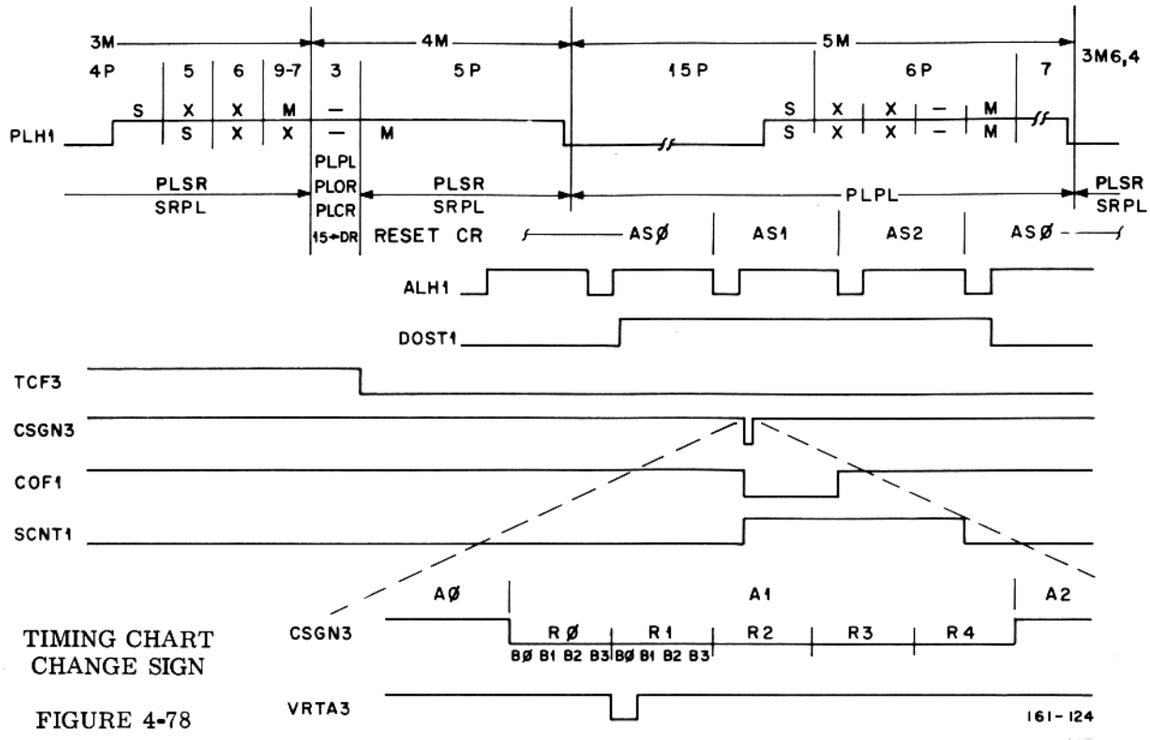


"CHANGE SIGN" LOGIC

FIGURE 4-77

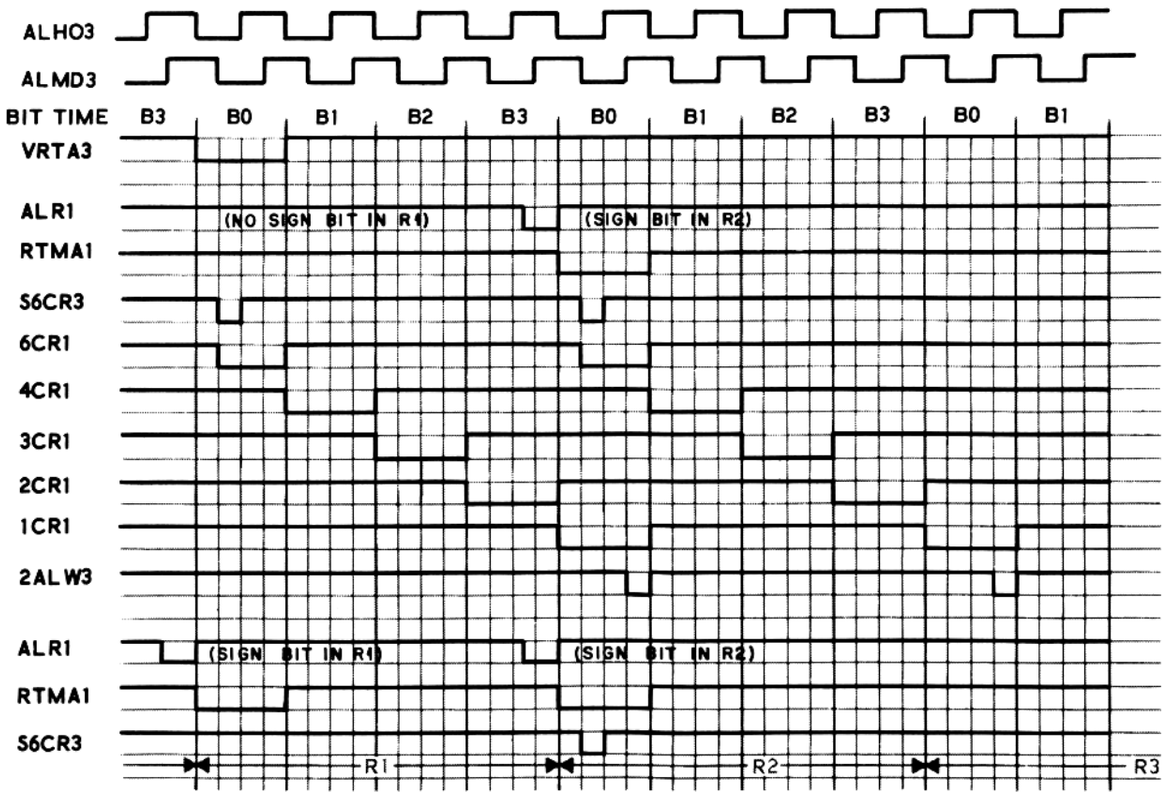
4. ILLUSTRATIONS

The timing chart in Figure 4-78 shows the relationships between the program line, the arithmetic line, and the more important signals. Figure 4-79 relates the arithmetic timing chain to the common register and other important signals during arithmetic column 1, showing first the conversion of a positive number (no sign bit) to a negative, and then a negative to a positive. The flow chart in Figure 4-80 outlines the complete Change Sign operation.



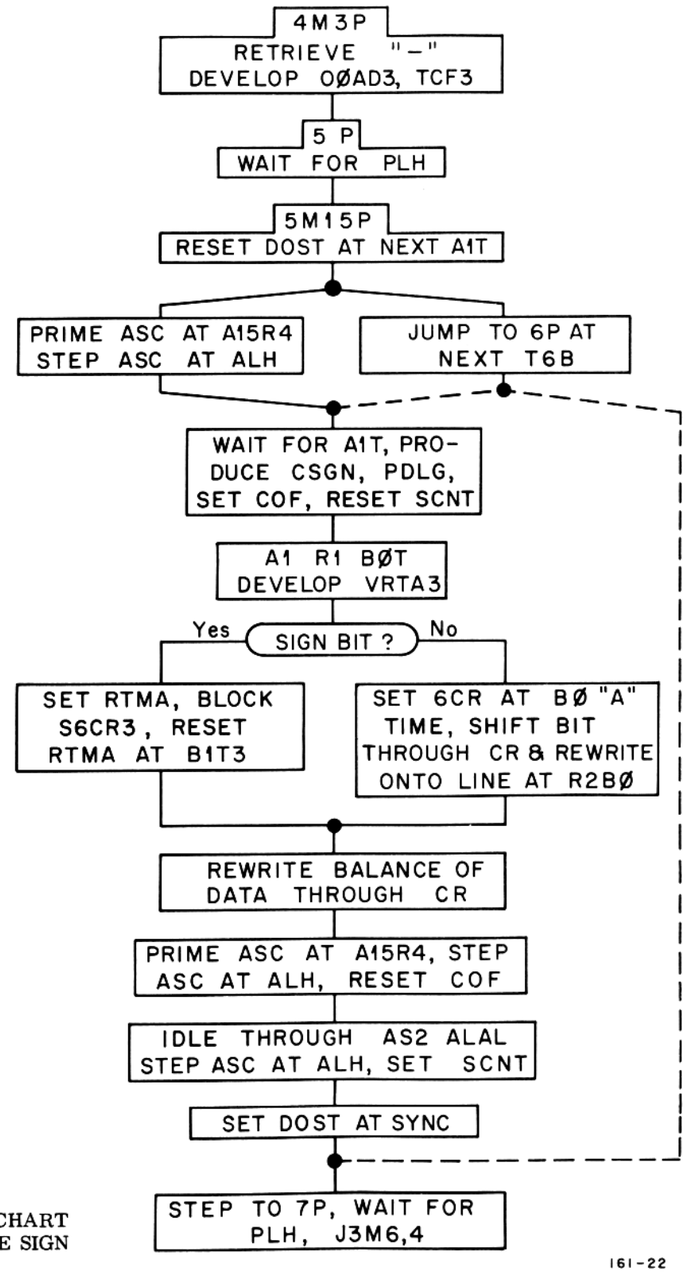
TIMING CHART
CHANGE SIGN

FIGURE 4-78



ARITHMETIC LINE ACTION DURING CHANGE SIGN 161-147

FIGURE 4-79



FLOW CHART
CHANGE SIGN

FIGURE 4-80

G. ENTER

Execution of the Enter instruction allows data to be entered into R1 from the 2205 (reader or keyboard) or from some auxiliary input device. This instruction is rather complex due to the many variables that may occur. For example, a Backspace entry will cause different operation after a digit has been entered than it will if no digit has been entered previously. The same is true of an alphabetic character or a space. Because of these and other "options", this discussion will go completely through the entry operation with a typical Left- and Right-of-Decimal (LRD) pattern being filled both left- and right-of-decimal. After the full operation has been covered, the options will be discussed one-by-one.

In this section, as in Section 4-H, Print, logic signals will be followed only as far as the 2205 interface. Beyond this point, they are covered in Section 8, Electronic Circuits.

1. GENERAL OPERATION

After the Enter command is retrieved, its two address digits, representing the LRD requirements, are stored in the decimal register and the search register. Then, in 5 mode, each digit that is entered via the 2205 is shifted from the common register onto the arithmetic line.

a. 4 MODE OPERATION. When the Enter command (/, 1-5-6 code) is retrieved in 4 mode 3 phase, a jump to 6 phase occurs. Here the first address digit, representing the left-of-decimal requirement, is shifted into the common register. Near the end of 6 phase this same code is set into the decimal register (CRDR). In 7 phase the second address digit, representing the right-of-decimal requirement, is shifted into the search register. Then a step to 8 phase occurs, and at program line home a jump to 15 phase (still 4 mode) occurs. If the writing machine is busy, the logic waits in 15 phase; if the 2205 is not busy, a jump to 5 mode 1 phase occurs.

b. 5 MODE OPERATION. The operation in 5 mode is most easily covered by considering each phase separately, although the action steps are still important. The only phases possible during Enter are phases 1 through 8. Phases 1 through 5 are used once for each digit, whereas phases 6 and 7 are used when the instruction is terminated, and phase 8 is used only when an error occurs. In brief, the purpose of each phase is as follows:

(1) 1 Phase. This phase is almost identical to 1 mode 7 phase: it is used to make the actual entry.

(2) 2 Phase. A ring shift to check the entry for parity is performed in 2 phase. A parity error causes a jump to 8 phase.

(3). 3 Phase. In 3 phase the code just entered is analyzed to determine what course of action is necessary. It is at this point that the many variations possible become apparent. Valid codes that are to be loaded onto the arithmetic line allow a step to 4 phase at this time. All other codes cause a jump to occur to either 1, 6, or 8 phase, depending upon the code and other conditions (SDGT, DPS, etc.).

(4). 4 Phase. Reaching 4 phase denotes the fact that a valid input code has been sensed. Here the phase counter syncs to the arithmetic timing chain so that the code can be transferred from the common register to the arithmetic line.

(5) 5 Phase. The logic remains in 5 phase until R1 has been shifted left. During this arithmetic line pass, the digit just entered is shifted into the Least Significant Digit (LSD) position of R1. At the end of this arithmetic line pass a jump occurs back to 1 phase, where the logic waits for the entry of the next code.

(6) 6 Phase. When the entire entry is completed, 6 phase becomes true and allows the final steps of the arithmetic portion to take place. When the action step counter returns to 0 and DOST sets, a step to 7 phase occurs.

(7) 7 Phase. As before, the logic waits for program line home in 7 phase, and then jumps to 3M6,4.

(8) 8 Phase. This phase is used only after an error (either a parity error or an unwanted backspace). The logic remains in 8 phase until the Reset switch is depressed. When this occurs, the marker is destroyed and a new sequence must be selected.

(9) Action Steps. The functions of the various action steps are, briefly, as follows:

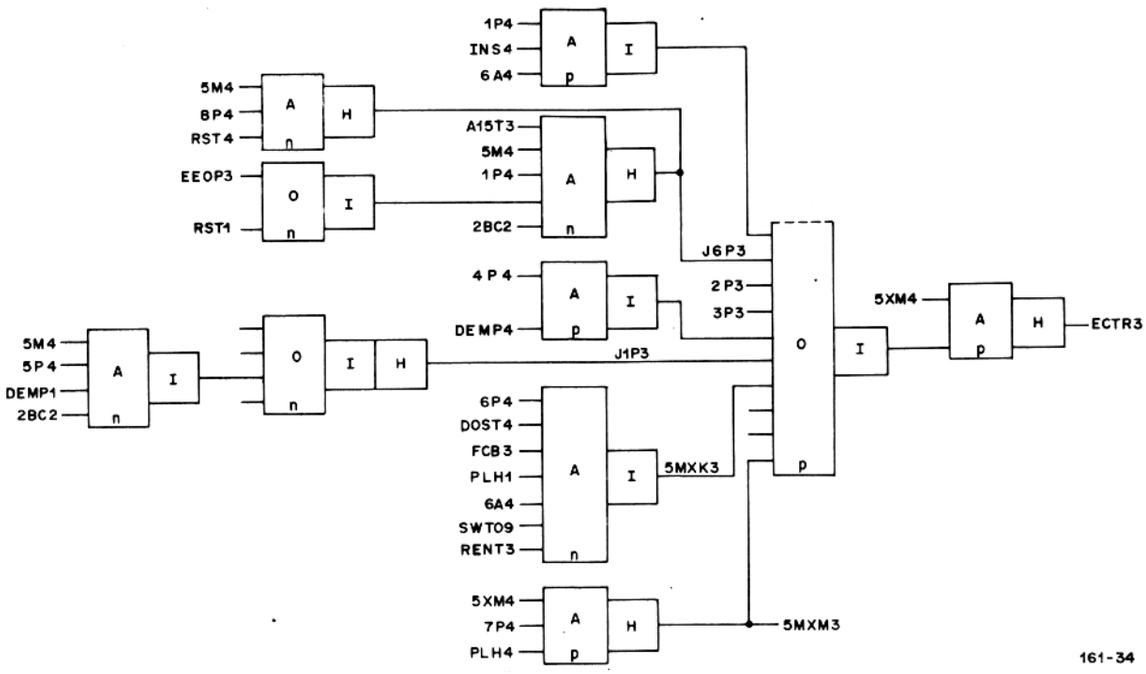
1. No function.
2. Shift up. This is necessary to clear R1 so that the entry can be made into R1.
3. Entry. The actual entry and left shift is accomplished here. Action step 3 remains true while each digit is entered and processed, regardless of the number of times phases 1-5 are used.
4. Change Sign. If the entry is terminated with a dash (minus sign), the sign bit is produced in action step 4.
5. Decimal alignment. This step remains true until the correct number of shifts left have been performed to properly align the entire number just entered around the decimal position.

2. ENABLING THE PHASE COUNTER

The development of ECTR3 during most aspects of the Enter instruction is illustrated in Figure 4-81. In addition to the gates shown, J1P3 and J6P3 can also be developed in other ways, which will be covered shortly.

Note in Figure 4-81 that there are two gates developing ECTR3 in 1 phase. The first is the one used when a code is entered via the 2205, thus developing INS. The second gate is used when either the Reset or Load switch is operated, setting RST, or when the instruction is terminated by filling the LRD pattern.

ECTR3 is "automatic" in phases 2 and 3, but in 4 phase DEMP must be set, signifying that the shift left (in most cases) has begun, before the counter can step. The only way to get out of 5 phase is to develop J1P3, which occurs when DEMP resets, signifying completion of the shift left (or other arithmetic function caused by a decimal or backspace). In 6 phase, ECTR3 is developed when Flexowriter Common Bit, FCB3, goes false and DOST is set. 7 phase requires program line home for development of ECTR3. In 8 phase, operation of the Reset switch enables the development of J6P3, which produces ECTR3.

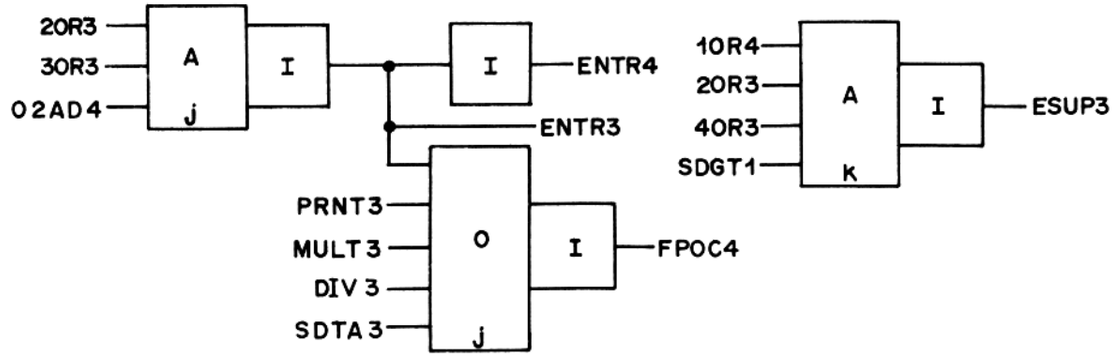


DEVELOPMENT OF ECTR3 IN ENTER

FIGURE 4-81

3. RETRIEVAL OF THE ENTER INSTRUCTION

As soon as the Enter command is shifted into the operation register in 4 mode 3 phase, three "master" signals are developed, as shown in Figure 4-82. ENTeR, ENTR, enables certain signals which are developed ONLY during an Enter instruction. Enable Shift UP, ESUP, allows the operating stack to shift up in action step 2, and Five Phase Op Code, FPOC, allows the action step counter to cycle through all five states.



"MASTER" SIGNALS USED IN ENTRY

FIGURE 4-82

Because the diagonal, or "slash", code is a 2-address code, a jump to 6 phase occurs, where the first address digit, representing the left-of-decimal setting, is shifted into the common register. Just before the step to 7 phase, CRDR strobes this code into the decimal register. The second address digit, representing the right-of-decimal setting, is shifted into the search register in 7 phase. The phase counter then steps to 8 phase, where it remains until program line home when a jump to 15 phase is enabled. BZY3 blocks the jump to 5 mode (refer to the "Retrieve" section) because shortly after 5 mode 1 phase is reached, INput Qualification, INQ, is developed. If the 2205 were still "busy", it would be possible to enter a code into the common register at this time.

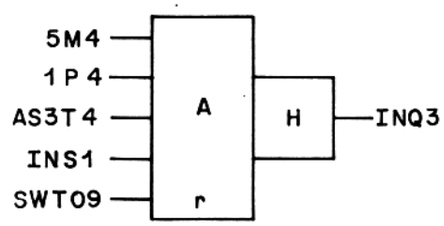
4. PRELIMINARY 5 MODE OPERATIONS

There is only one arithmetic operation that must be performed before the arithmetic logic is ready to accept digits from the interface. This is the shifting up of the operating stack, which provides an "empty" R1 into which the newly entered digits may be shifted.

a. ACTION STEP 1. When 5 mode is first reached, DOST resets, unlocking the action step counter and allowing it to increment to step 1. The arithmetic logic "idles" through one line pass in action step 1, with no function occurring.

b. ACTION STEP 2. The second arithmetic line pass in 5 mode sees ESUP and action step 2 providing a "long" path for all data, but not for the sync pulse. This causes all data to be delayed one register time in relation to the sync pulse, resulting in a shift up. The previously "clear" state of R0 is written into the R1 time slot, and each register is shifted up. R4 data is written into R0 time, but this will be cleared when the instruction is terminated.

c. ACTION STEP 3. Once action step 3 is reached, INput Qualification, INQ3, is developed (Figure 4-83), which makes it possible for 2205 entries to be set into the common register.



DEVELOPMENT OF INQ3 DURING ENTRY

FIGURE 4-83

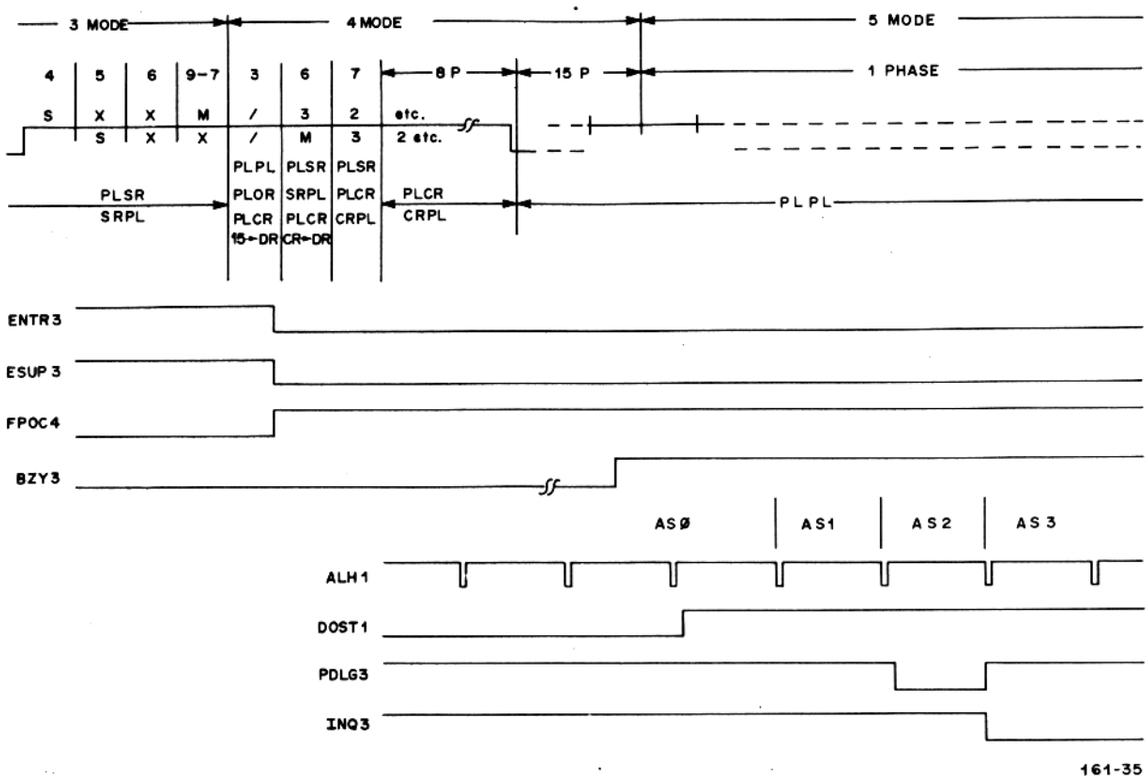
At this point all of the preliminary operations (those necessary before the first digit can be entered) are complete. All of these operations are summarized in the timing chart in Figure 4-84.

5. REPETITIVE OPERATIONS

The operation in action step 3 is dependent upon what codes are entered and the order in which they are entered. As mentioned earlier, this section will first consider the operation occurring when only DIGITS are entered.

Certain factors are also dependent upon whether the digit is being entered to the right or to the left of the decimal. These repetitive operations that are discussed here, however, occur the same regardless of decimal position.

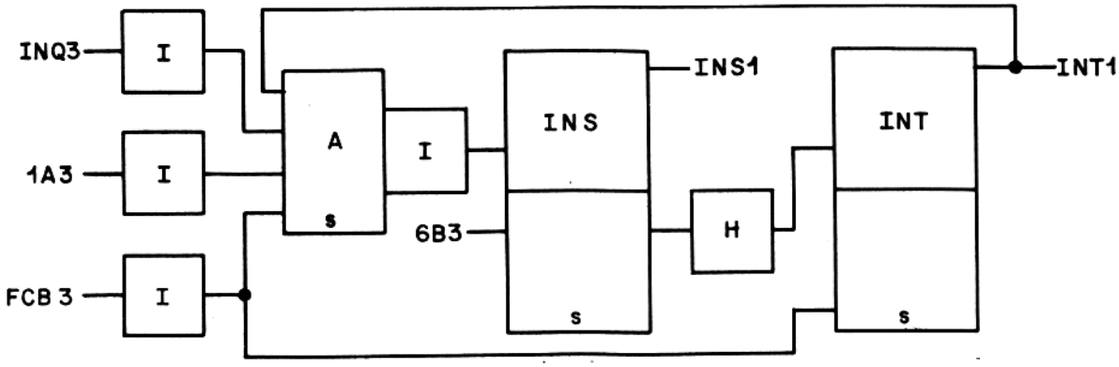
a. 1 PHASE. The main purpose of 1 phase is to allow the entry of a digit into the common register from the writing machine. However, before an entry can actually be made into the processor, INQ must be developed. Refer to Figure 4-83 and note that INQ is not developed until action step 3. It takes between two and three arithmetic line passes in 5 mode before INQ is developed. However, because this time is quite short in relation to the operating speed of the writing machine, INQ3 will always be true before the common bit (FCB3) is produced.



ENTER
PRELIMINARY OPERATIONS

FIGURE 4-84

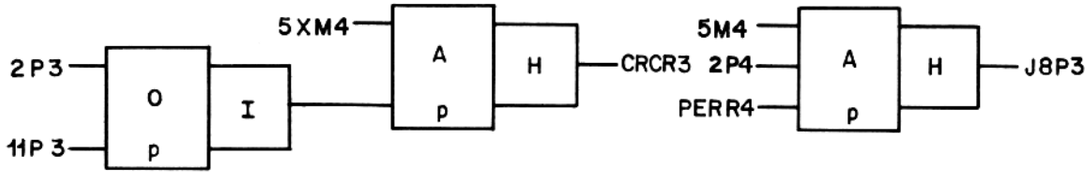
Refer to Figure 4-85 and note that when the common bit is produced by the writing machine, it sets the INS flip-flop. The INS signal developed then causes two operations: the development of ECTR3 for the step to 2 phase, and the development of the input "bit" signals, F1B3, F2B3, F3B3, etc. The development of these latter signals in the interface is explained in Section 8, Electronic Circuits. Each of these signals sets the corresponding common register flip-flop. As an example, assume that the first key struck was the digit 1. This would produce FCB3, INS1, and F1B3, in that order. F1B3 sets 1CR, and INS enables the step to 2 phase. INS1 also drives INQ3 false, (Figure 4-83), preventing a double reader cycle that otherwise could occur under certain conditions.



DEVELOPMENT OF INS

FIGURE 4-85

b. 2 PHASE. Because ECTR3 is "automatic", 2 phase lasts for only one character time. During this time parity is checked in the usual manner. If a parity error is detected, a jump to 8 phase occurs. See Figure 4-86.

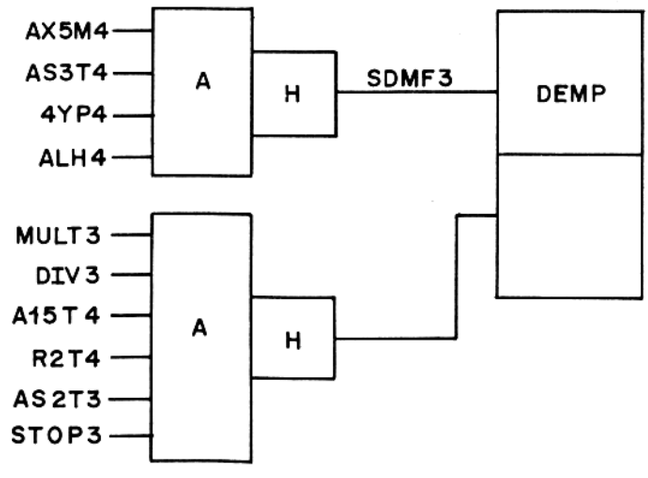


PARITY CHECK DURING ENTRY

FIGURE 4-86

c. 3 PHASE. Nothing actually happens in 3 phase: no line routings are changed, no flip-flops become set or reset, etc. Instead, 3 phase is used to check the character in the common register and determine the next course of action. In other words, 3 phase is used to gate the various jumps that occur when a code other than a digit is entered. Since a digit was entered in the example being used, no jumps are set up, and the phase counter steps to 4 at the next T6B.

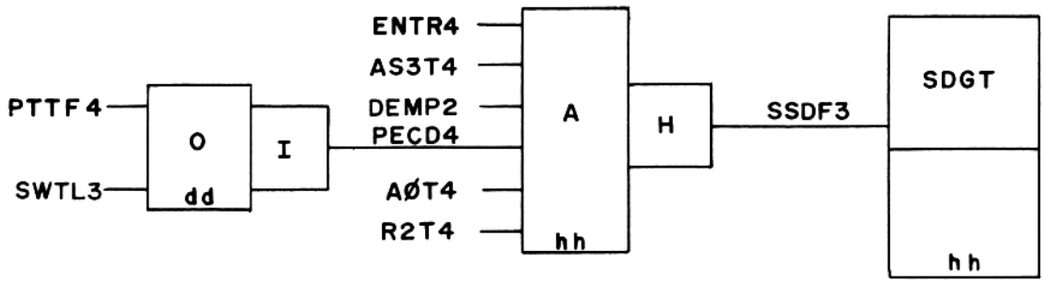
d. 4 PHASE. In 4 phase a shift left of R1 is initiated to shift the new code from the common register into R1. This is accomplished by setting DEMP, as shown in Figure 4-87. 4YP4 is necessary to ensure that there will be at least one more T6B available to step the phase counter to 5 phase after the shift left begins. If this gate were not qualified by 4YP4, it would be possible to set DEMP just before program line home and then be unable to step from 4 phase for lack of a T6B pulse. DEMP would then set at each arithmetic line home, and each arithmetic line pass would produce another shift left until the program line timing chain restarted.



SHIFT LEFT DURING ENTRY
(All on "HH" card)

FIGURE 4-87

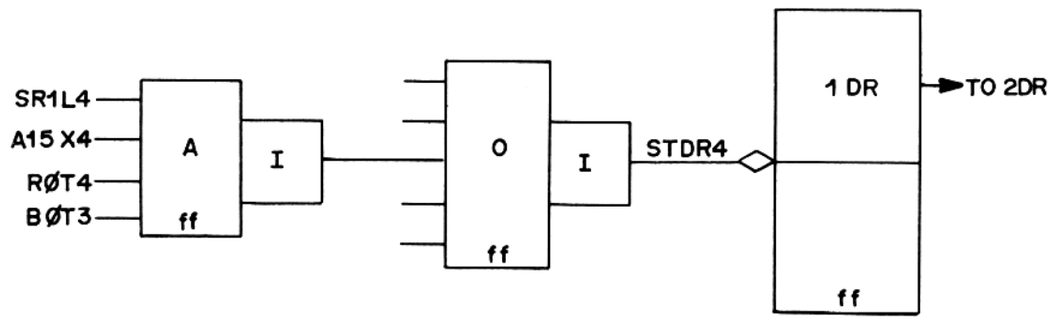
e. 5 PHASE. The Set Significant Digit Flip-flop signal, SSDF3, is developed in 5 phase at A0R2 time, as shown in Figure 4-88. Note that DEMP2 prevents the development of SSDF3 during all other phases. Notice also that SDGT becomes set only after a digit entry: if some code other than a digit had been entered the phase counter would not have gone to 4 phase and DEMP would not have set. (Two exceptions to this, the backspace and the decimal, will be covered shortly.) SSDF3 is developed each time 5 phase is reached, even though SDGT may have been set previously.



SETTING SDGT DURING ENTRY

FIGURE 4-88

5 phase remains true until the shift left has been completed (except as noted for backspace and decimal). Note that because SR1L4 is false, the decimal register is counted down at A15R1 time. This is to keep track of the number of digits entered, and will be covered in detail shortly. See Figure 4-89.



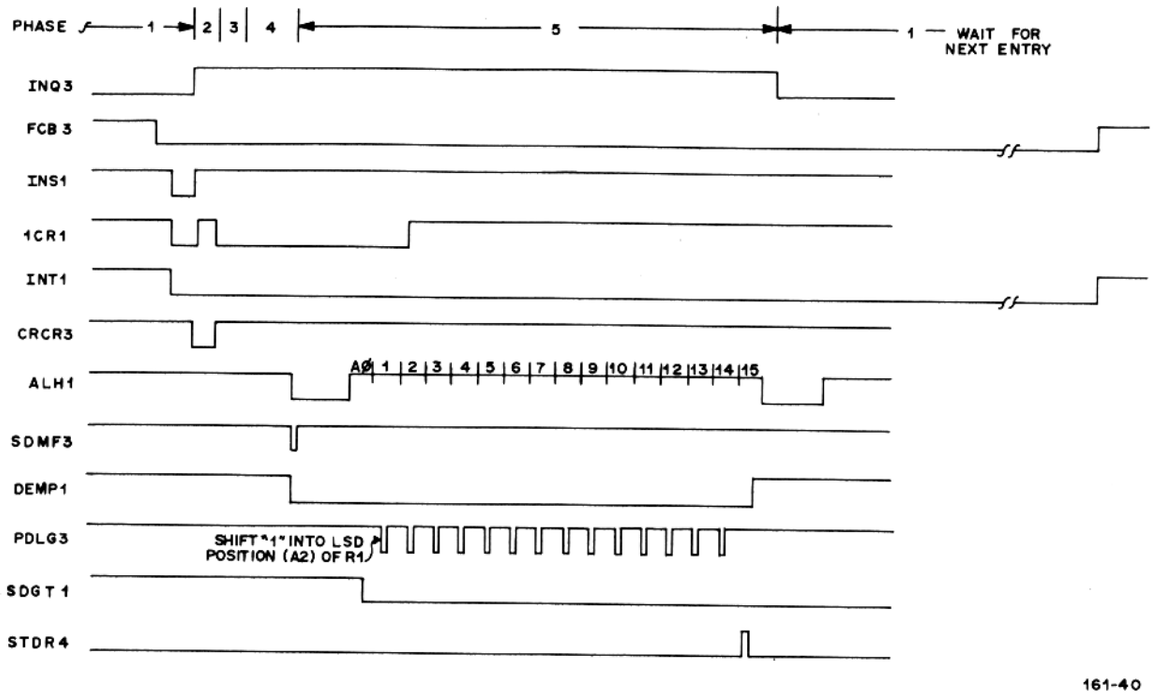
COUNTING DOWN DR DURING ENTRY

FIGURE 4-89

When DEMP resets at A15R2 time, J1P3 is developed (refer to Figures 4-81 and 4-87), and the next T6B causes the phase counter to jump back to 1 phase to wait for the next entry.

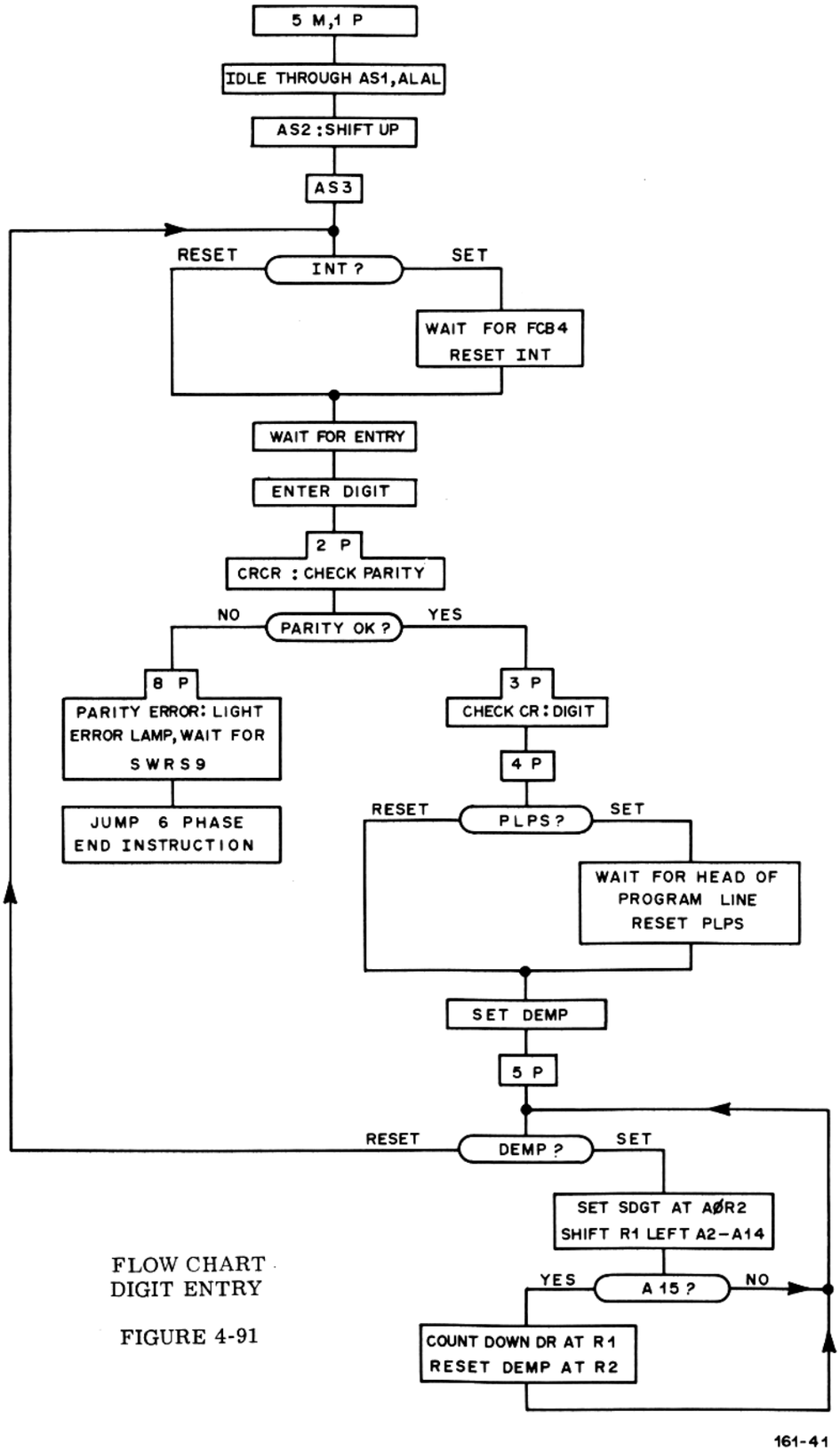
f. SUMMARY. Refer briefly to Figure 4-85 and note that when the jump back to 1 phase takes place, FCB3 is still true from the entry of the previous digit, and INT is still set, preventing development of INQ. FCB3 must first go false, and then another key must be struck (or another code read) to drive FCB3 true again, thus developing INS and allowing the step to 2 phase.

The complete cycle of one digit entry is illustrated in the timing chart in Figure 4-90 and the flow chart in Figure 4-91. All of the events depicted are repeated for each digit that is entered, with the exception of setting SDGT. As R1 is being shifted left in 5 phase, the digit just entered is shifted into the LSD position of R1 and the previously entered digits are shifted ahead to a more significant position.



DIGIT ENTRY

FIGURE 4-90



FLOW CHART
DIGIT ENTRY

FIGURE 4-91

161-41

6. DECIMAL POINT

Just as there can be a wide variation in the characters entered, there can also be different circumstances surrounding the decimal point. The variations will be covered shortly: for now it will only be shown how DPS is set when the left-of-decimal requirements are fulfilled. The setting of the decimal register to 5 to allow for entry of digits to the right of the decimal will also be covered at this time.

After each digit is shifted left onto the arithmetic line, the decimal register is counted down once. When the last digit left-of-decimal is entered and shifted onto the line, the decimal register is counted down to 0. This counting down occurs at A15R1 (refer to Figure 4-89), and shortly thereafter DEMP resets, driving SR1L4 true. Now see Figure 4-92 and note that during the time when DRZ4 and SR1L4 are both false, Set Enable Decimal Point storage, SEDP3, is developed, which sets EDPS. At the end of A15R1B0 time, SROD3 is developed, which sets the decimal register to 5 to allow for entry of digits to the right of the decimal. At the same time, SROD4 enables development of the Decimal ENTry signal, DENT3, which immediately resets DEMP. (This latter action is not necessary at this time: it is needed when a decimal is entered via the 2205. It is mentioned here only to point out the fact that in this instance DEMP is reset before its usual reset occurs at A15R2.)

7. RIGHT-OF-DECIMAL

The operation caused by digits entered to the right of the decimal position is basically the same as for left-of-decimal digits. Each digit that is entered is checked for parity in 2 phase, and analyzed to make sure it is a valid digit in 3 phase. The shift left is initiated in 4 phase, and the digit is shifted from the common register onto the arithmetic line in 5 phase. After each digit is shifted onto the line, the decimal register is counted down once. When enough digits have been entered right-of-decimal to satisfy the LRD requirements, Decimal CoMPare, DCMP3, is developed, and the termination of the Enter instruction is initiated.

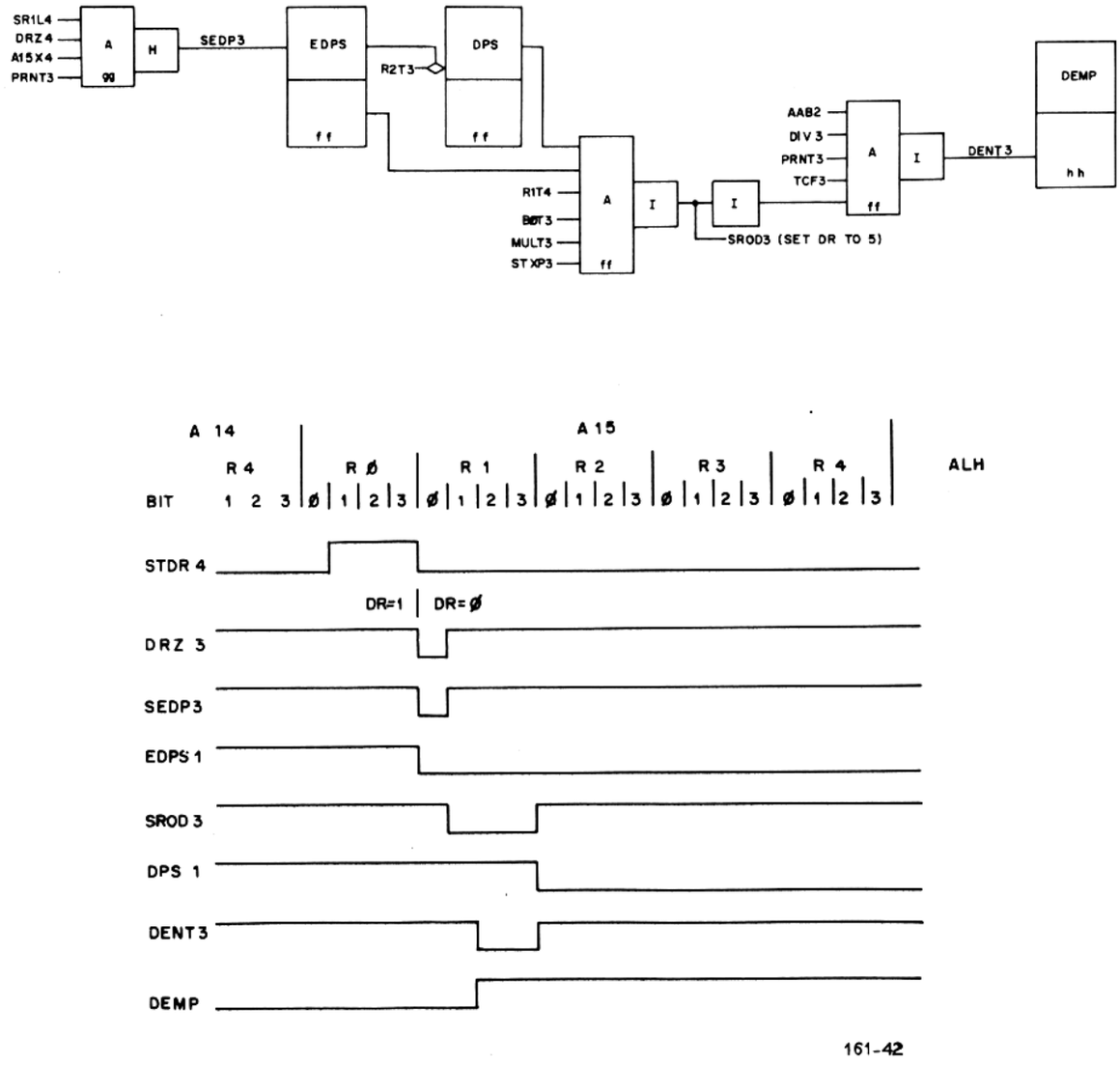
8. TERMINATING ENTER

There are two aspects to the termination of an Enter instruction, the same as there are in the termination of other instructions utilizing the arithmetic line logic. First, the arithmetic logic must be "re-homed", or put back into its "idling" state (ALAL). Then, the program line logic can move on to the next instruction.

When the Enter instruction is to be terminated, the phase counter jumps to 6 phase and waits there for the arithmetic portion to be terminated. The arithmetic factors involved can vary, again dependent upon the many "options". Basically, however, the process involves stepping out of action step 3 after reaching 6 phase, changing the sign, if necessary, in action step 4, and then remaining in action step 5 until the number is aligned around the decimal.

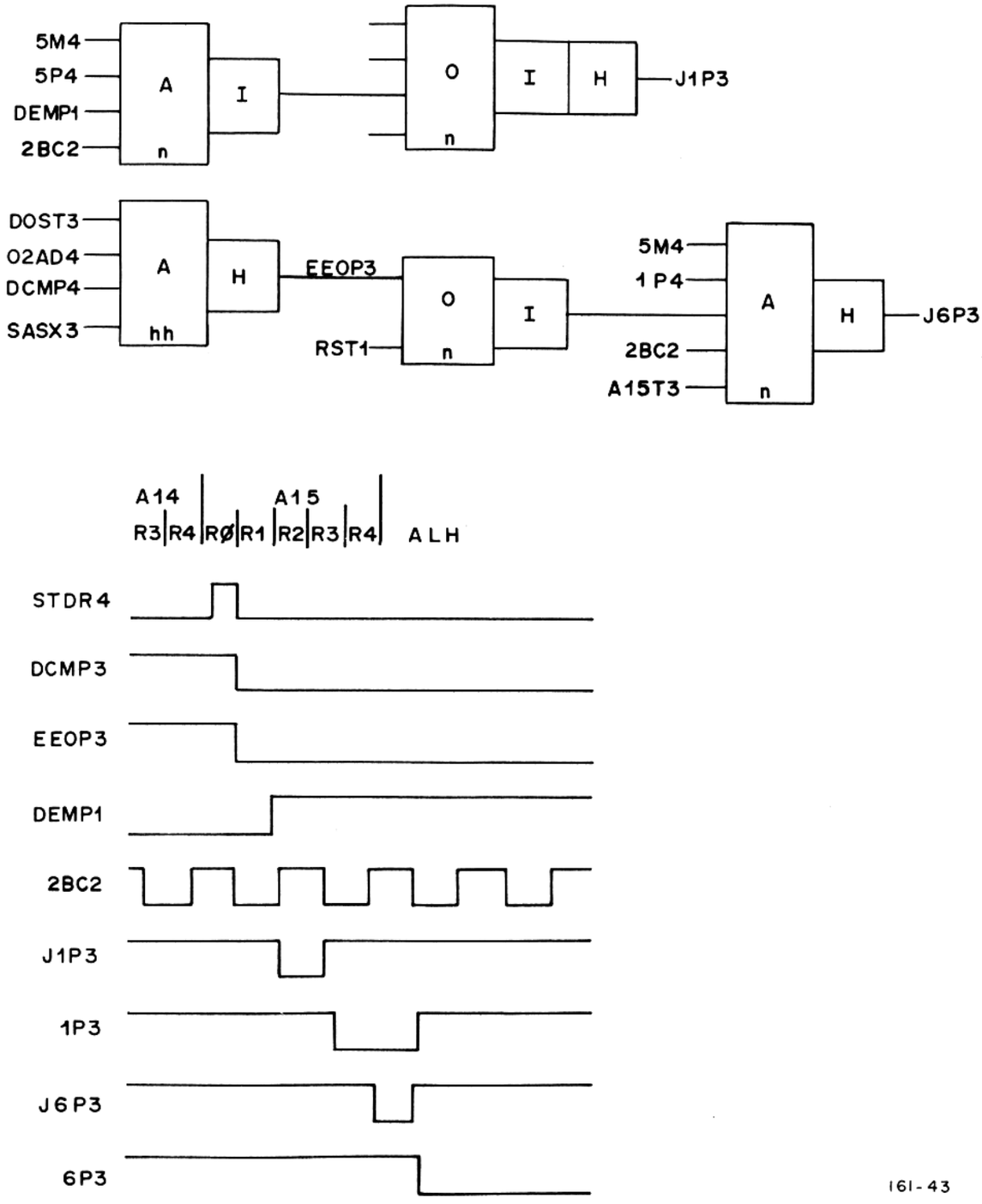
After the number is decimally aligned, DOST sets and allows the phase counter to step to 7 phase. At program line home a jump to 3M6,4 occurs and the search for the next instruction begins.

a. 6 PHASE. When the right-of-decimal setting is satisfied by entry of a sufficient number of digits, Decimal CoMPare, DCMP, is developed. (Refer to Section 8, Electronic Circuits.) After the last digit is shifted left, the decimal register is counted down to a value that, when compared with the right-of-decimal setting in the search register, produces DCMP3. The End of Entry or OutPut signal, EEOP3, is immediately developed as shown in Figure 4-93, and during arithmetic line home a jump to 6 phase is enabled. Note that the phase counter does not step directly from 5 phase to 6 phase: since the only way out of 5 phase is to jump to 1 phase, it first makes the jump to 1 phase and then jumps again from 1 phase to 6 phase. EEOP3 also resets SDGT. 2BC2 and A15T3 prevent electronic "slivers".



FILL LEFT-OF-DECIMAL DURING ENTRY

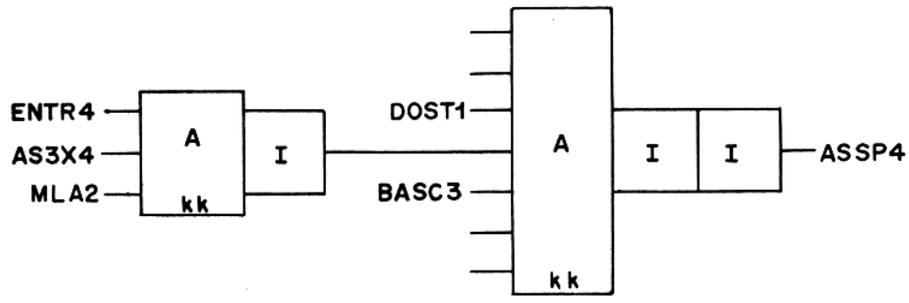
FIGURE 4-92



TERMINATING ENTRY

FIGURE 4-93

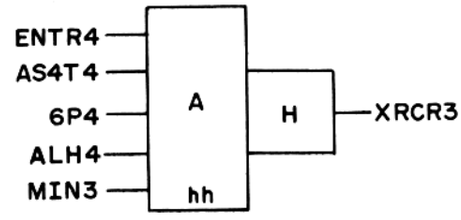
(1) Action Step 4. After 6 phase is reached, one more arithmetic line pass occurs in action step 3, during which MLA resets (to be covered shortly). This allows a step to action step 4, as shown in Figure 4-94. One line pass occurs in step 4, during which the sign of R1 is made negative if the entry was terminated with a minus sign. This will also be covered shortly.



STEPPING FROM AS3 IN ENTER

FIGURE 4-94

If the entry was NOT terminated by the minus sign, the terminating code is reset out of the common register at the beginning of action step 4, as shown in Figure 4-95.



RESET CR IN AS4

FIGURE 4-95

(2) Decimal Alignment. Next the action step counter increments to 5, where the number is decimally aligned. This is accomplished by shifting R1 left and counting down the decimal register once during each line pass, until the decimal register reaches 0.

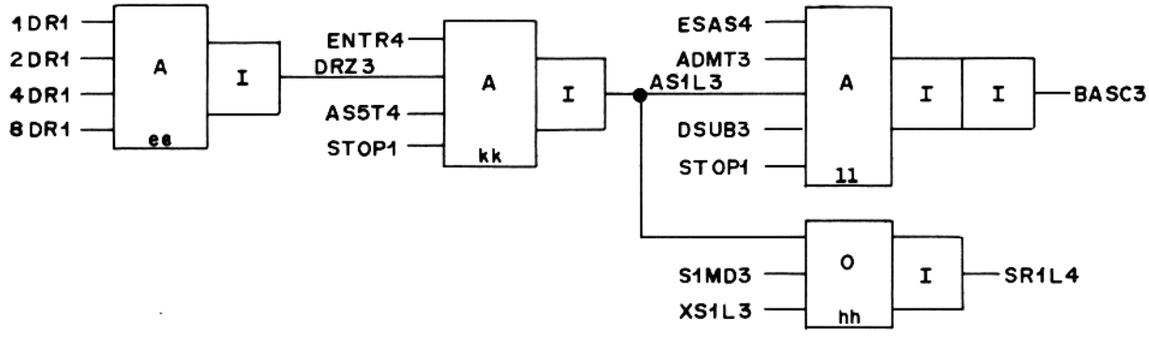
For example, if two digits were entered right-of-decimal, the decimal register would be at 3. The last digit entered is in A2, so after three shifts left the last digit is shifted into A5 and the decimal register is at 0. This example is illustrated in Figure 4-96.

	Arithmetic Column													Decimal
	14	13	12	11	10	9	8	7	6	5	4	3	2	Register
After last entry	0	0	0	8	7	6	5	4	.3	2	1	5	4	3
After first shift left	0	0	8	7	6	5	4	3	.2	1	5	4	0	2
After second shift left	0	8	7	6	5	4	3	2	.1	5	4	0	0	1
After third shift left	8	7	6	5	4	3	2	1	.5	4	0	0	0	0

EXAMPLE OF DECIMAL ALIGNMENT AFTER ENTRY

FIGURE 4-96

The logic associated with the decimal alignment is illustrated in Figure 4-97. Note that Align Shift 1 Left, AS1L3, remains true until the decimal register reaches 0. AS1L3 develops SR1L4 to produce the shift left, and holds BASC3 true so that the action step counter remains in step 5. When the decimal register is finally counted down to 0 at A15R1 time, AS1L3 is driven false, allowing BASC3 to go false at A15R4 time. The action step counter returns to 0 at arithmetic line home, and the next arithmetic sync pulse sets DOST.

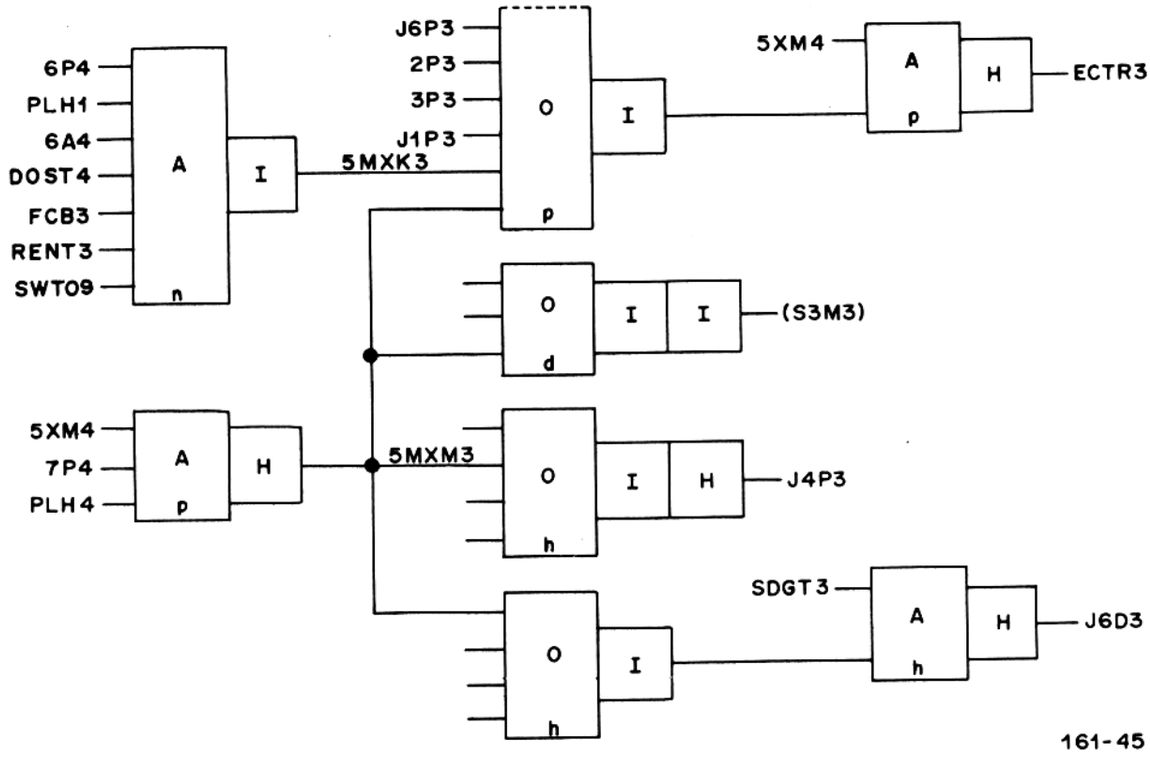


DECIMAL ALIGNMENT AFTER ENTRY

FIGURE 4-97

b. 7 PHASE. At this point all arithmetic operations connected with the Enter instruction are complete. With DOST now set, the phase counter steps to 7. At program line home the usual jump to 3M6,4 occurs and marker search begins, to locate the next instruction.

The logic associated with these last operations is illustrated in Figure 4-98. Note that the logic does not leave 6 phase until after FCB3 returns to false. At this time FCB3 is still true from the entry of the last digit. This qualification is not necessary for the proper operation of the Enter instruction; rather, it is used to prevent any further instructions from occurring until the writing machine is "ready".



FINAL PHASES IN ENTER

FIGURE 4-98

9. VARIATIONS

There are two factors that can cause variations from the normal procedure just described. These are variations in the codes entered and variations in the LRD patterns. The latter are more easily understood after the different codes are covered.

Of the various codes that may be entered, only the backspace is unique, always causing operation different from any other code. Other character and function codes can be grouped to a certain extent, depending mainly upon the state of SDGT. However, the operation caused by entering the decimal is more dependent upon the relationship to the decimal position. The chart in Figure 4-99 briefly summarizes the effect caused by each of the different codes.

CODE ENTERED	SDGT2	SDGT1
Backspace	Increase Left-of-Decimal Setting by 1	Jump to 8 phase Error Lockup
Space	Decrease Left-of-Decimal Setting by 1	
Reset Switch, Tab, or Carriage Return		Jump to 6 phase End of Entry
"Alpha"	No Effect	
Minus		Change Sign
Decimal	Set Right-of-Decimal to 5, Set SDGT (SROD3)	If Right-of-Decimal: If at Decimal Position:No Effect If Left-of-Decimal:SROD3
Digit	Shift R1 Left: Load Digit onto Math Line Set SDGT	

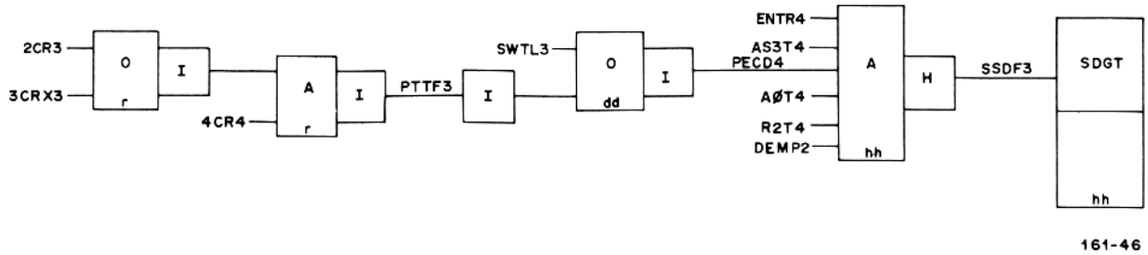
EFFECT OF ENTERING VARIOUS CODES

FIGURE 4-99

a. BACKSPACE. When entered before a significant digit, the backspace increases the left-of-decimal setting by 1. When a backspace is entered after a significant digit, or after the left-of-decimal requirements have been fulfilled, it signifies an error. Thus there are three cases in which the backspace will cause an error: after a significant digit has been entered, after the left-of-decimal positions have been filled with spaces, and at any time during an entry specifying zero left-of-decimal.

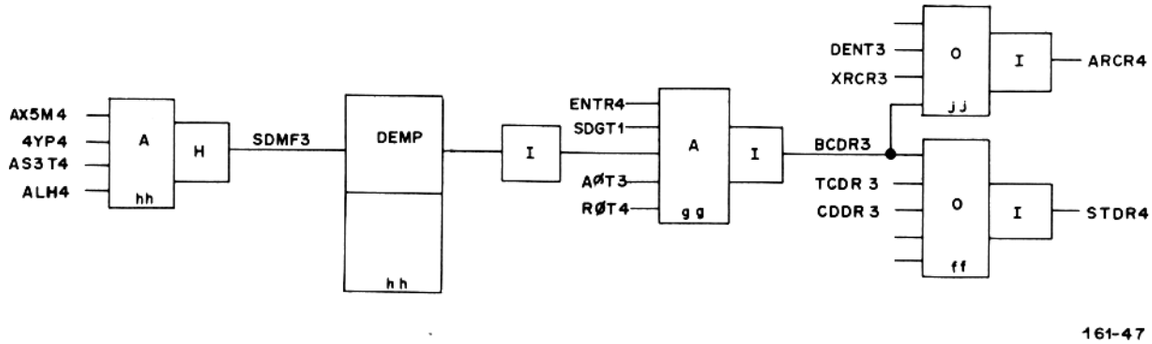
(1) Before SDGT. A backspace entry in 5 mode 1 phase causes the usual operation in phases 1 and 2. If SDGT is not set, the backspace will step the phase counter through phases 3 and 4, setting DEMP, and into phase 5.

(a) Preliminary Logic. See Figure 4-100. The backspace code (2-4-6) in the common register prevents the setting of SDGT in A0R2 time, even though DEMP is set. (PECD4 stands for Permit Entry-Count Digits.) Note that PECD4 will be true, blocking SSDF3, for all codes except digits and spaces: since the space entry does not set DEMP, only digits (including 0) will set SDGT.



BACKSPACE ENTRY, PRELIMINARY LOGIC

FIGURE 4-100



BACKSPACE ENTRY BEFORE SDGT1

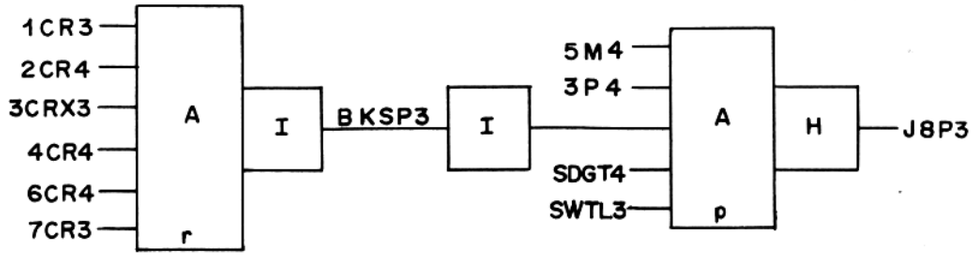
FIGURE 4-101

(b) Counting Down the Decimal Register. Now see Figure 4-101. In 5 phase, Backspace Count-down Decimal Register, BCDR3, is developed at each R0 time, A1 through A15. This drives STDR4 false. When BCDR3 goes false at the end of R0 time, STDR4 goes true and counts the decimal register down once. BCDR3 is developed fifteen times, so at the end of the arithmetic line pass, the decimal register has been counted down fifteen times, leaving it at a setting one higher than it was to start. Note that when BCDR3 is first developed (A1R0 time), ARCR4 goes false to reset the backspace code out of the common register.

A shift left of R1 occurs because DEMP is set, even though R1 is clear at this time. The backspace is the only code that sets DEMP and allows a shift left without setting SDGT.

(c) Possible Operator Error. Subsequent backspace operations cause the left-of-decimal requirements to be increased in the same manner. However, if the left-of-decimal requirement is increased to nine or greater, only the last eight digits entered will be retained in R1. Others will be shifted out of A14 and lost when the final alignment around the decimal is accomplished in action step 5.

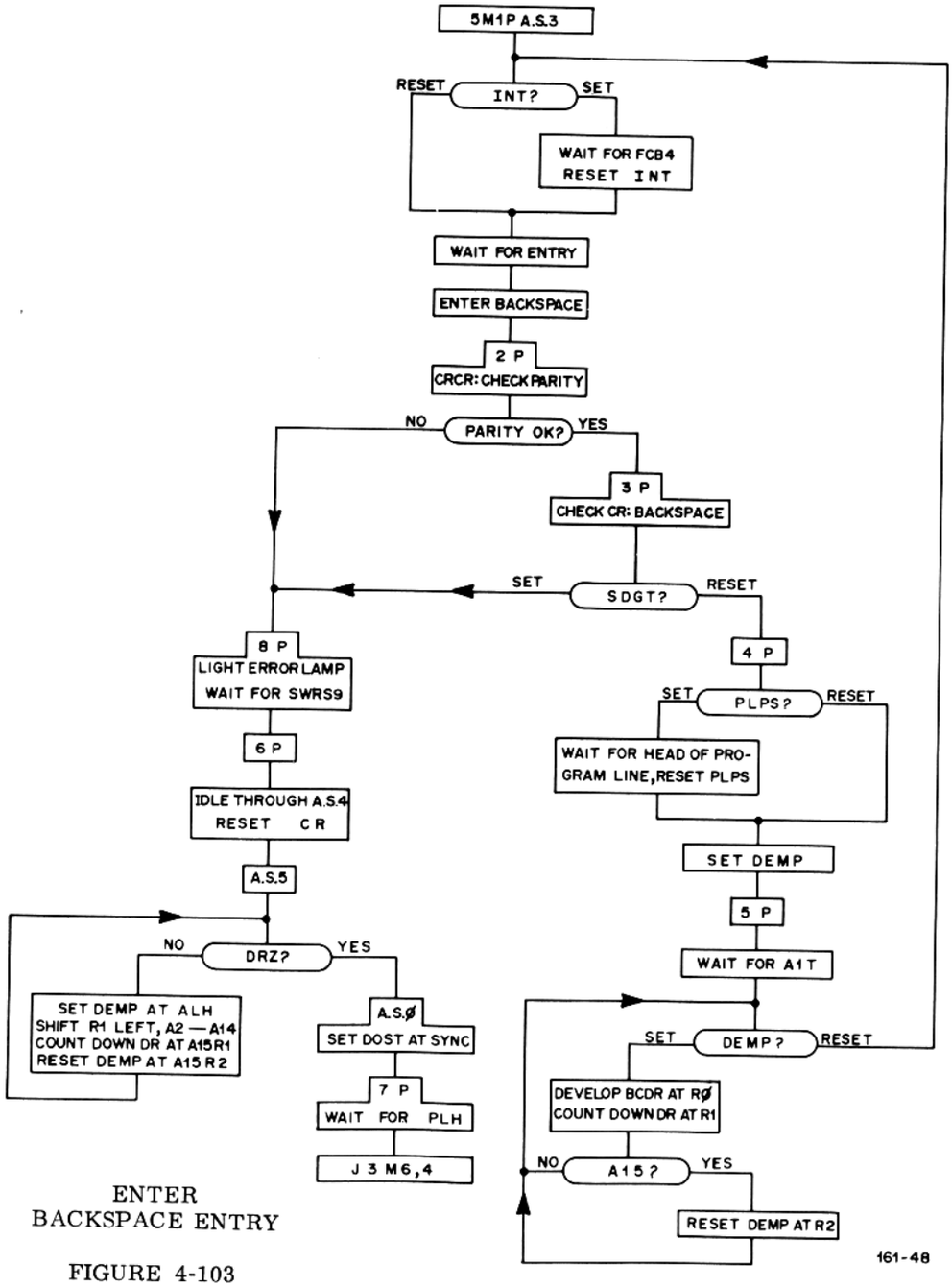
(2) After SDGT. When the backspace is entered after SDGT is set, a jump occurs from 3 phase to 8 phase, as shown in Figure 4-102, and the red error lamp is lit. 8 phase remains true until the Reset switch is operated. Then a jump to 6 phase occurs (refer to Figure 4-81), and the Enter instruction is terminated. Note that when this occurs, the program sequence is also terminated.



J8P3 CAUSED BY BKSP WITH SDGT SET

FIGURE 4-102

(3) Flow Chart. The flow chart in Figure 4-103 illustrates the action caused by entry of a backspace during the Enter instruction.



ENTER
BACKSPACE ENTRY

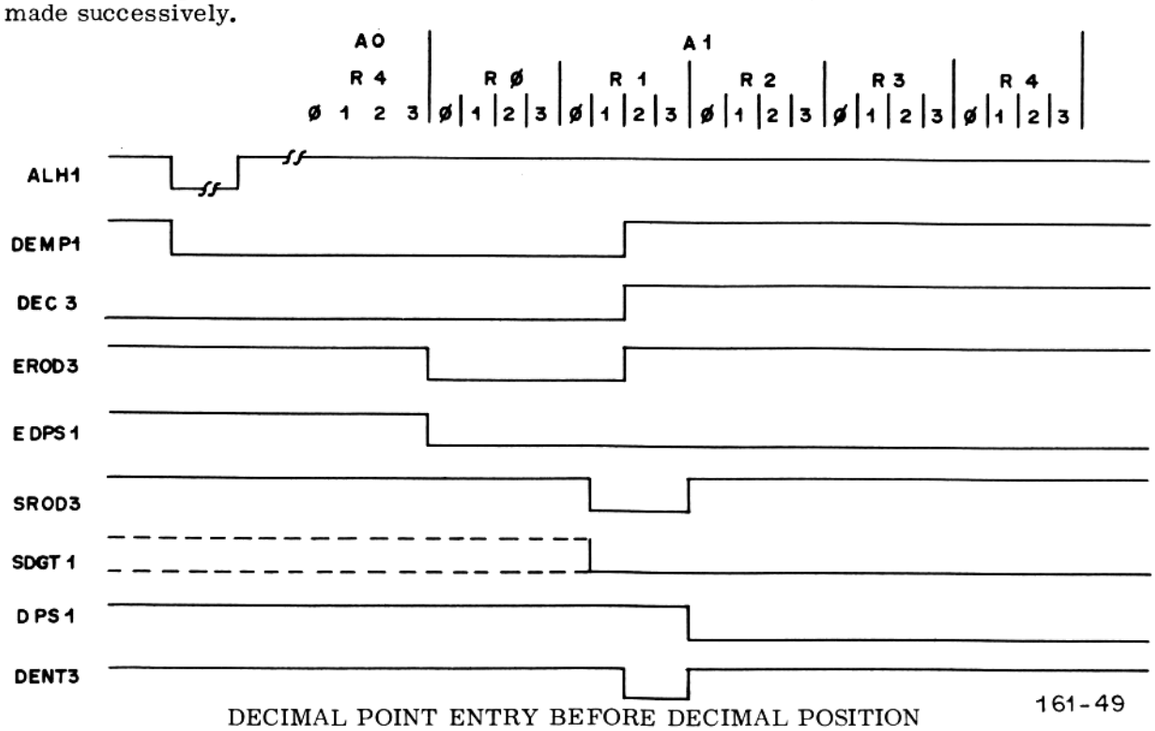
FIGURE 4-103

b. DECIMAL. Entry of a decimal code before the decimal position has been reached automatically satisfies the left-of-decimal requirements, setting the DPS flip-flop and setting a 5 into the decimal register to allow for right-of-decimal entries. It also sets SDGT if it was not previously set. Successive decimal entries have no effect, but if a valid entry has been made right-of-decimal, another decimal entry will terminate the instruction.

(1) Before Decimal Position. When a decimal is entered before the left-of-decimal requirements have been fulfilled, the logic follows the same path as for a digit until 4 phase is reached and DEMP becomes set. Refer to Figure 4-104 (a) and note that at this time the decimal code in the common register allows the Enable Right Of Decimal signal, EROD3, to be developed, which sets EDPS. This occurs as soon as DEMP sets in 4 phase, which is at arithmetic line home. At the next R1, which is A0R1 time, SROD3 is developed, which sets the decimal register to 5. At the same time, SROD4 enables the development of DENT3, which resets DEMP to prevent a shift left of R1. DENT3 also develops ARCR, to reset the decimal code out of the common register. Note that in this case phase 5 is true for only a short time, since DEMP resets in A1, thus enabling the jump back to 1 phase.

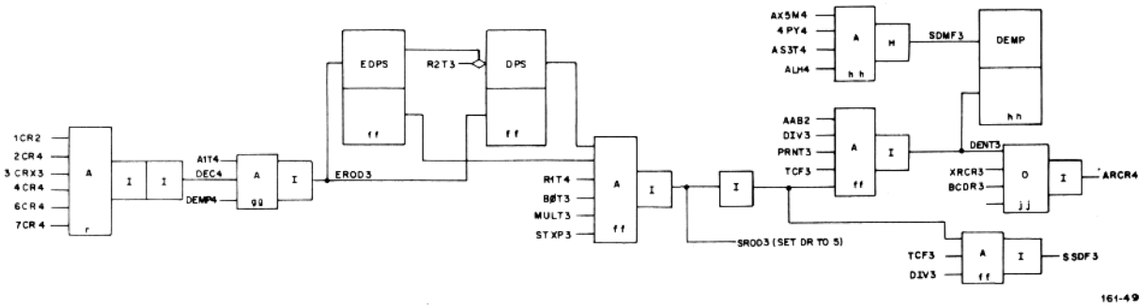
(2) At Decimal Position. If the decimal point is entered at the decimal position, the phase counter makes its usual cycle through the five phases, but no significant operation takes place. Actually, the operation occurs the same as if the decimal position had not yet been reached: the decimal register is set to 5, DEMP is reset, and EDPS, DPS, and SDGT are all set, even though these conditions were all true to start.

Refer again to Figure 4-104 (a) and (b), and note that EROD3 resets DPS in order to allow the development of SROD3, DENT3, and ARCR4. In this case, the sole purpose of the entire operation is to "sluff" the decimal code (reset it out of the common register) without allowing a shift left of R1. This operation would normally occur when the decimal is entered after the left-of-decimal requirements are filled. It would also occur if two or more decimal entries were made successively.



DECIMAL POINT ENTRY BEFORE DECIMAL POSITION
(Timing Chart)

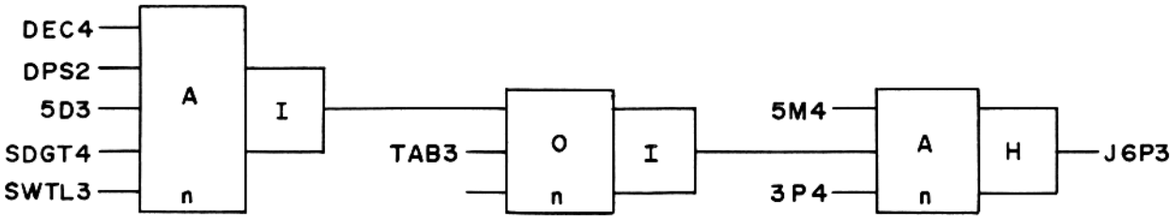
FIGURE 4-104 (a)



DECIMAL POINT ENTRY BEFORE DECIMAL POSITION
(Logic)

FIGURE 4-104 (b)

(3) After Decimal Position. If a decimal is entered after at least one digit has been entered right-of-decimal, the instruction is terminated: since a decimal would not normally be entered right-of-decimal, such an occurrence is considered invalid. Refer to Figure 4-105 and note that a jump to 6 phase is enabled in 5 mode 3 phase when a decimal is in the common register and DPS is set. The 5D3 signal inhibits this gate so that J6P3 will not occur when the decimal is entered at the decimal position.

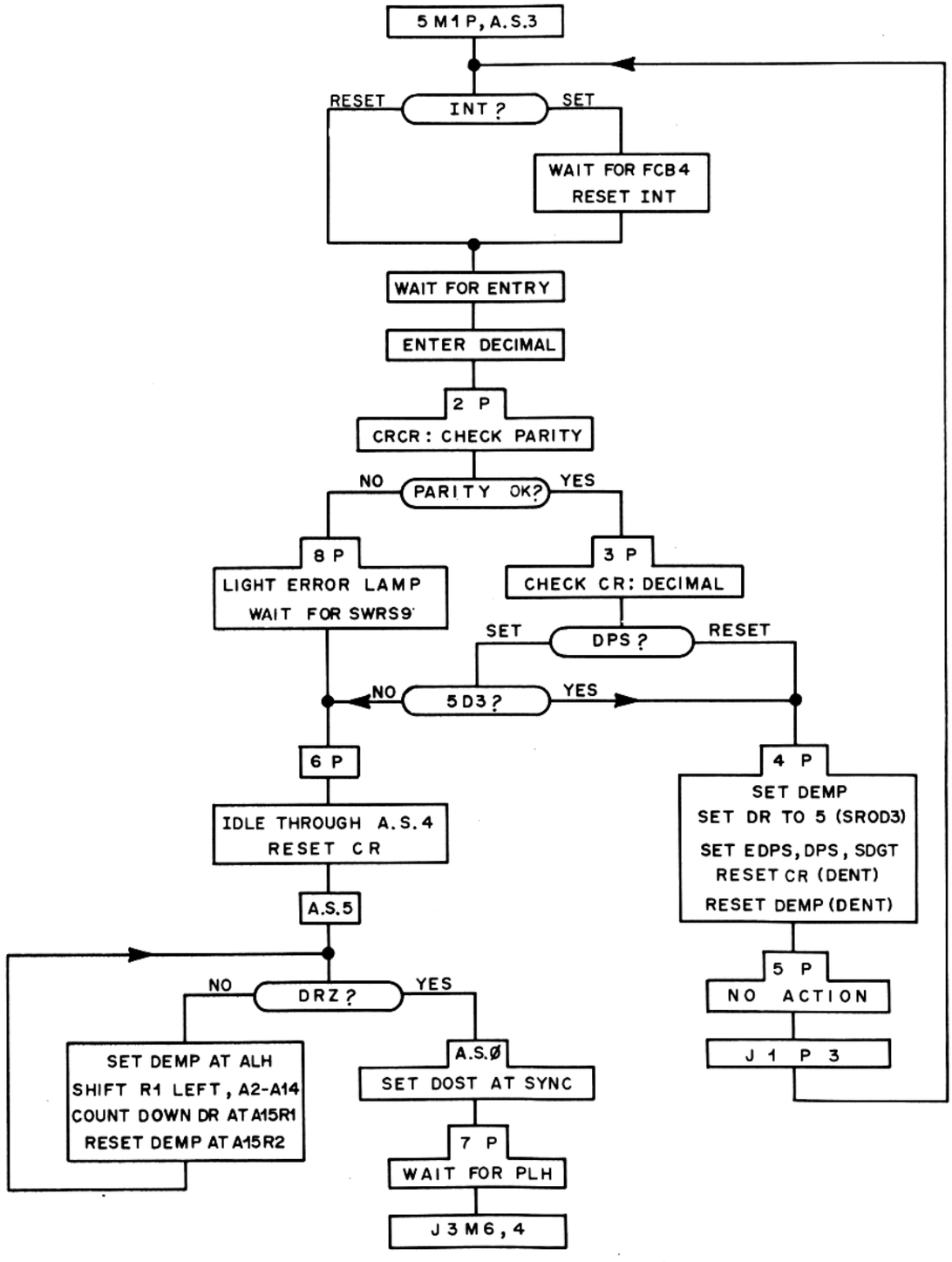


J6P3 CAUSED BY DECIMAL ENTRY RIGHT-OF-DECIMAL

FIGURE 4-105

(4) Flow Chart. The action caused by entry of a decimal code is summarized in the flow chart in Figure 4-106.

c. OTHER CODES. All codes other than digits, backspace, or decimal cause a jump from 3 phase to either 1 phase or 6 phase. The jump to 1 phase can occur only if SDGT is reset. The jump to 6 phase occurs if SDGT is set, or if the code entered is a tab or carriage return.



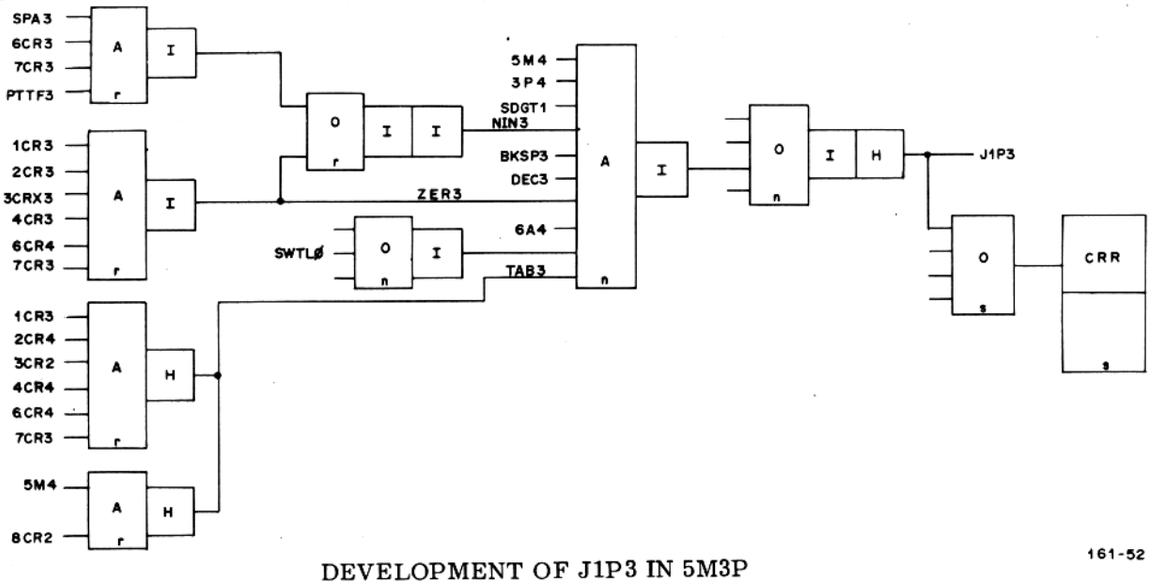
161-51

ENTER
DECIMAL ENTRY

FIGURE 4-106

TP-161SL

(1) SDGT Reset. The logic providing the jump to 1 phase is illustrated in Figure 4-107. If the code entered is NOT a digit (Number INput, NIN3), a backspace, a zero, a decimal, or a tab or carriage return (TAB3), the J1P3 signal will be developed. This means a jump to 1 phase will occur with SDGT reset when either a space, a minus, or an "alpha" code is entered. (In this discussion, "alpha" includes all codes other than carriage return, tab, space, backspace, minus, decimal, and digits, including zero. This includes all "F" codes, case shift codes, symbols, etc.) In other words, all codes other than space, minus, and alpha inhibit the jump to 1 phase.

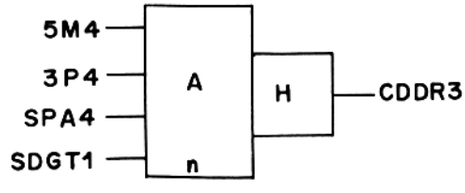


DEVELOPMENT OF J1P3 IN 5M3P

FIGURE 4-107

Notice also in Figure 4-107 that each time a jump to 1 phase occurs, the code causing this jump is reset out of the common register.

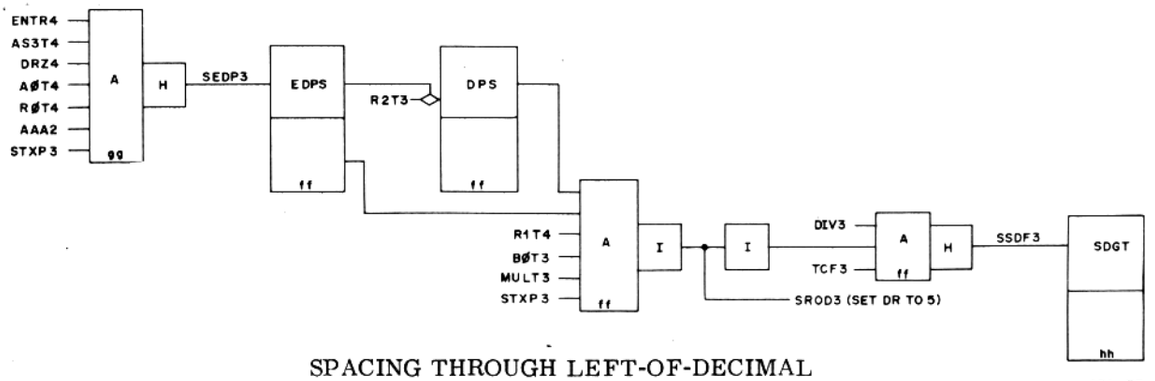
In addition to the jump to 1 phase, a space entry has other aspects peculiar to it alone. Each time a space is entered with SDGT reset, the decimal register is counted down once, as shown in Figure 4-108, to decrease the left-of-decimal requirement. If enough spaces are entered to count the decimal register down to 0, the decimal position has been reached and the decimal



DEVELOPMENT OF CDDR3 BY SPACE IN 5M3P

FIGURE 4-108

register is set to 5. This is accomplished by developing SEDP3 (Figure 4-109), which sets EDPS. This develops SROD3, which sets a 5 into the decimal register and sets SDGT. Any further spaces at this point cause the operation which is discussed next.

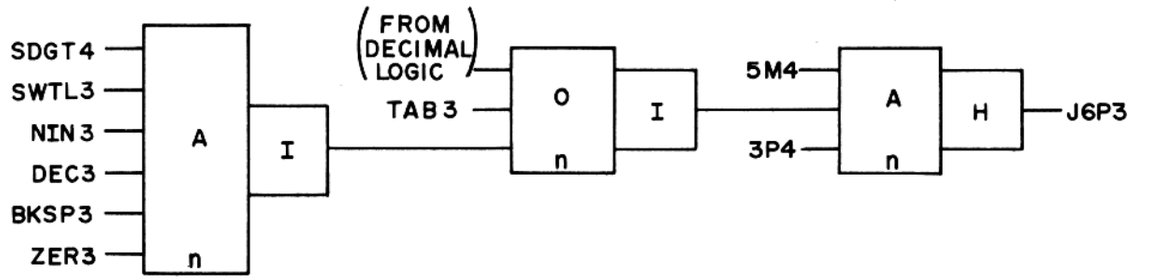


SPACING THROUGH LEFT-OF-DECIMAL

161-53

FIGURE 4-109

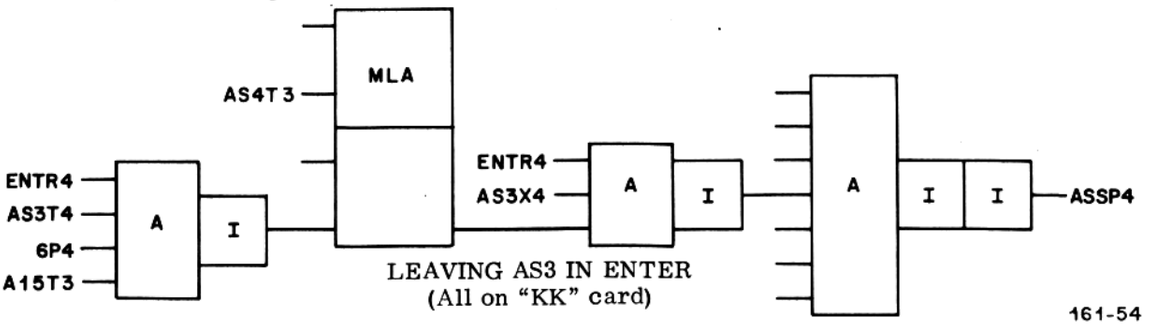
(2) SDGT Set. With SDGT set, the jump to 1 phase is blocked, and the space, minus, or alpha code provides a jump to 6 phase instead. In addition, the decimal code provides a jump to 6 phase if the decimal position has already been passed. (This was illustrated previously in Figure 4-105.) Also, the tab or carriage return provides a jump to 6 phase, regardless of the state of SDGT. The logic involved in these jumps is illustrated in Figure 4-110.



DEVELOPMENT OF J6P3 IN 5M3P

FIGURE 4-110

(3) 6 Phase. Once in 6 phase, all codes cause the same operation, except for the minus, which changes the sign of R1. MLA prevents a "short" pulse on ASSP4 which would occur when 6 phase becomes true just before the end of A15, when BASC3 goes true. Such a short pulse could result in an incomplete shift of the action step counter flip-flops. Refer to Figure 4-111 and note that ASSP4 cannot be developed, for the step to action step 4, until MLA sets. MLA cannot set while BASC3 is true, due to A15T3. Should the timing relationship between the program line (which provides the jump to 6 phase) and the arithmetic line be such that 6 phase is reached during A15 time, one more arithmetic line pass will occur in 6 phase before the action step counter steps to 4.



LEAVING AS3 IN ENTER
(All on "KK" card)

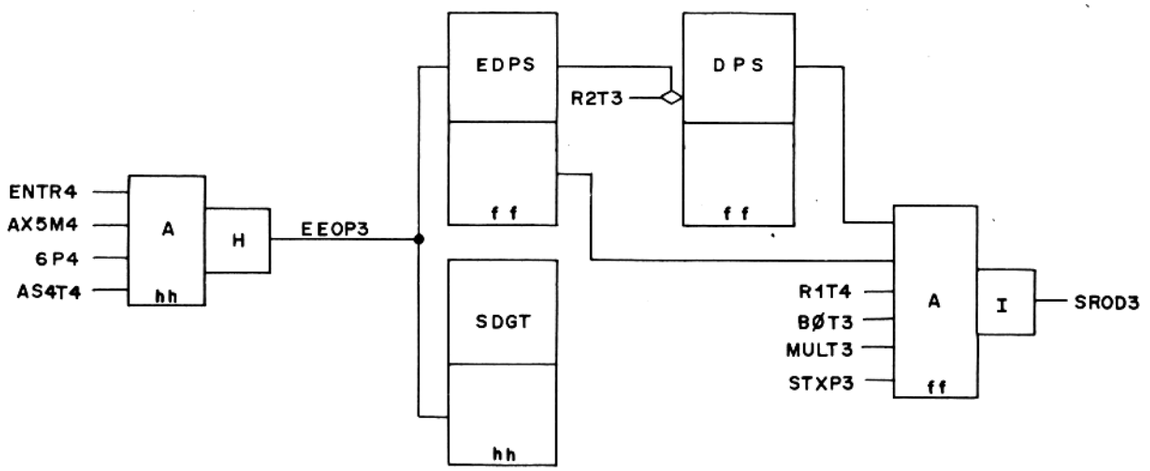
161-54

FIGURE 4-111

Note that the logic in Figure 4-111 is necessary only when the jump to 6 phase occurs from 3 phase: when the jump is from 1 phase (refer to Figure 4-93) the jump is qualified by the arithmetic timing chain.

(a) Prepare for Decimal Alignment. Since it is possible to jump to 6 phase before the decimal position has been reached, all codes cause the development of SROD3 in 6 phase if DPS is not already set. This is necessary to ensure correct decimal alignment for all digits entered. For example, if two digits had been entered under an LRD pattern of 3/2, and then a tab, carriage return, space, etc., followed, the contents of R1 would have to be shifted left five times in order to align the two digits properly around the decimal. Therefore, 6 phase automatically gates a setting of 5 into the decimal register, providing the decimal position has not yet been reached.

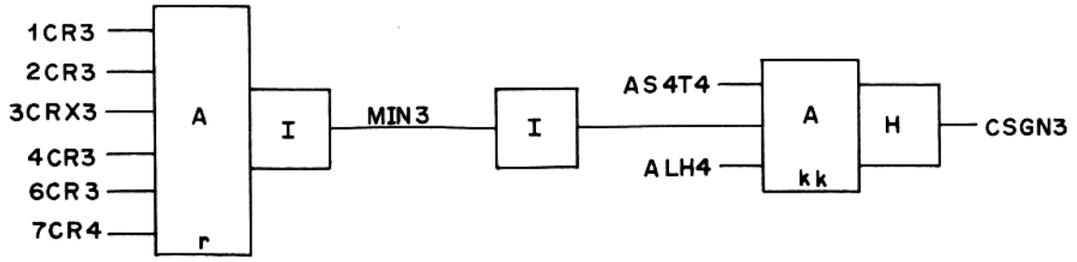
This is accomplished by developing EEOP3 whenever 6 phase is reached. Refer to Figure 4-112. If the decimal position has not previously been reached, EDPS will become set to allow SROD3 to set the decimal register to 5. If the decimal position had been reached previously, EDPS and DPS would already be set, and EEOP3 would have no effect other than to reset SDGT.



SET RIGHT-OF-DECIMAL IN 6 PHASE 161-55

FIGURE 4-112

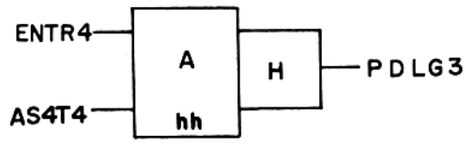
(b) Sign Change from Minus Entry. When the minus is used to terminate the entry, the sign of R1 is made negative in 6 phase by developing VRTA3 at A1R1B0 time. The development of VRTA3 is the same as it was during the Change Sign command, with the exception of the method used to produce CSGN3. In 6 phase the MINus signal, MIN3, produces CSGN3 in action step 4 at arithmetic line home (see Figure 4-113). CSGN3 then sets COF and resets SCNT, producing VRTA3 in A1R1B0. Refer to Section 4-F, Change Sign, for review.



DEVELOPMENT OF CSGN3 DURING ENTER

FIGURE 4-113

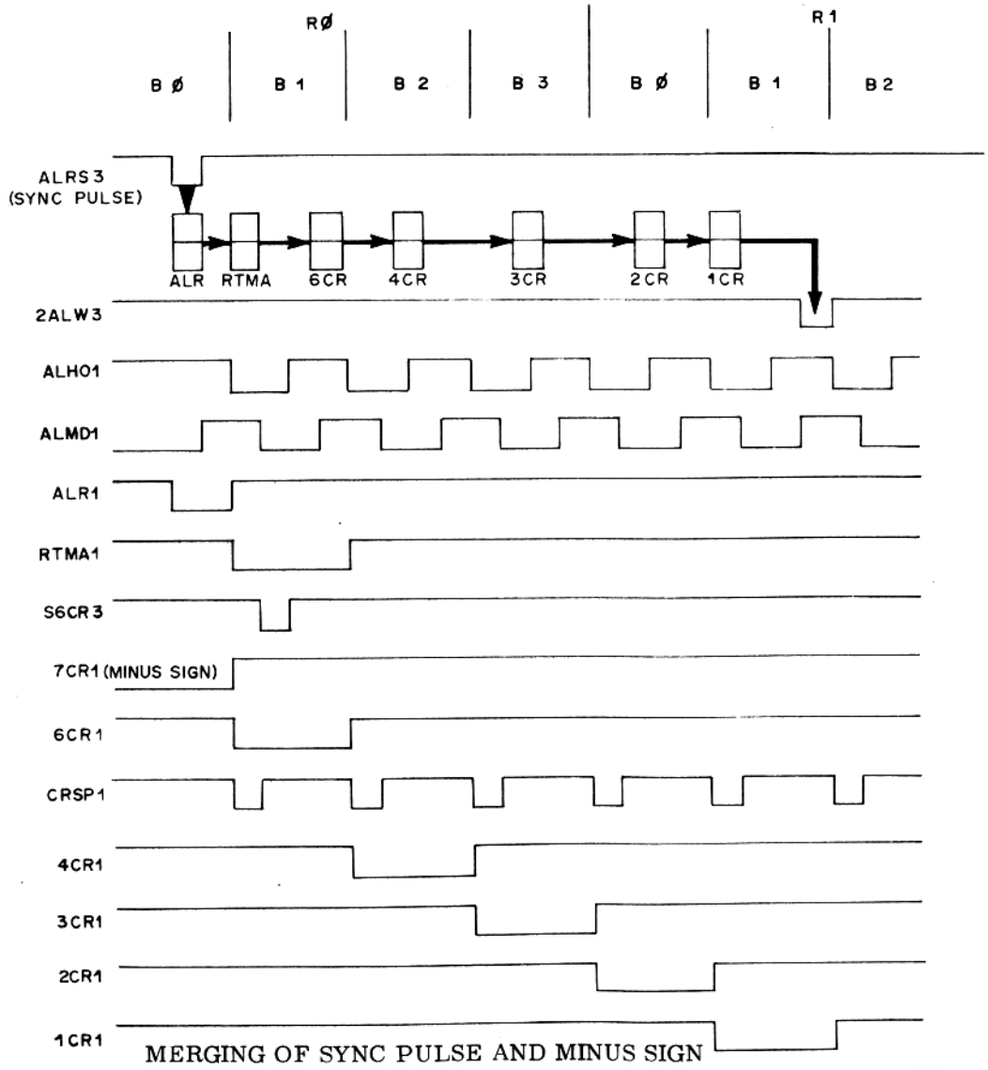
Since the arithmetic data path must be long (ALCR-CRAL) in order to allow VRTA3 to be effective, PDLG3 is developed for the entire linepass in action step 4. See Figure 4-114.



DEVELOPMENT OF PDLG3 IN AS4

FIGURE 4-114

Note that MIN3 is necessary to develop CSGN3 in action step 4 (see Figure 4-113). The common register is normally reset as soon as action step 4 is reached, but in this case, MIN3 inhibits this reset (shown earlier in Figure 4-95) to allow CSGN3 to be developed. The minus sign is not reset out of the common register: instead, since the data path is long in action step 4, it merges with the sync pulse in the common register (6CR), and is rewritten onto the arithmetic delay line. See Figure 4-115. After ALH resets, the 7 bit can safely be shifted out of the common register once the sync pulse is sensed, with no danger of XRCR3 being developed.



MERGING OF SYNC PULSE AND MINUS SIGN

161-57

FIGURE 4-115

d. RESET SWITCH. It was noted previously that operation of the Reset switch sets the XRST flip-flop. If this occurs after a parity error (when LPEL3 is true) or during an entry (when INQ3 is true), the marker is destroyed and the sequence is terminated. When a parity error occurs after an entry in 5 mode, or when a Backspace is entered with SDGT set, the logic goes to 8 phase, develops LPEL3, and waits for operation of the Reset switch. Then the marker is destroyed.

If, however, the Reset switch is operated in 1 phase to terminate the entry (as in the case when the operator has entered a wrong digit and wishes to perform an error correction routine), the logic jumps to 6 phase, as shown previously in Figure 4-81, and the Enter instruction is terminated. With RST set, the marker is searched out and destroyed.

A third case occurs when the Reset switch is operated during some phase other than 1 or 8. When this happens, XRST sets immediately, but RST does not set until the logic follows its normal path to 1 phase. Then the same jump to 6 phase occurs and the instruction is terminated.

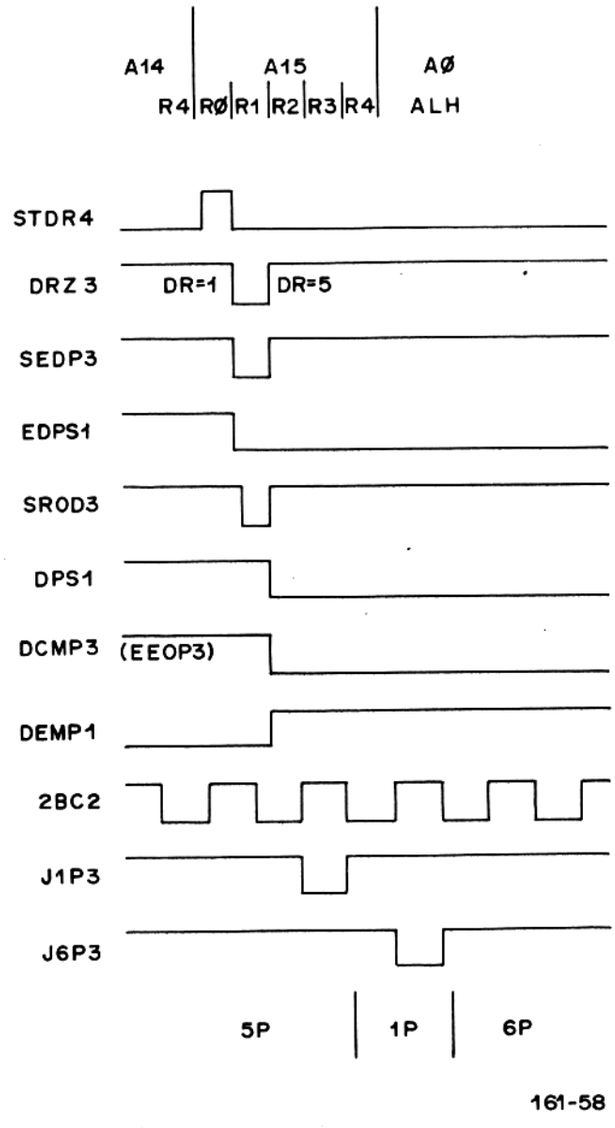
In 6 phase the arithmetic portion of the instruction is terminated as before. In action step 4 EEOP3 is developed, resetting SDGT, and if the decimal position has not yet been reached, SROD3 sets the decimal register to 5. In action step 5 R1 is shifted left until the decimal register reaches 0, and the action step counter then steps to 0. The next arithmetic sync pulse sets DOST, and the next T6B steps the phase counter to 7 phase. At the next program line home a jump to 3M6,4 takes place. Then the marker is located and destroyed, and the logic ends up in 3M0,1, waiting for the entry of a new sequence title.

e. LRD VARIATIONS. There are three variations from the "normal" LRD patterns that require special treatment. These are 0 left-of-decimal, 0 right-of-decimal, and 6 right-of-decimal.

(1) 0 Left-of-Decimal. When an LRD pattern is used that specifies 0 left-of-decimal, the decimal register is set to zero when the instruction is retrieved in 4 mode. Then, as soon as action step 3 is reached in 5 mode, and the timing chain restarts, SEDP3 is developed, which sets EDPS. At the next R1 time SROD3 is developed, which sets the decimal register to 5 and sets SDGT. This action has already been shown in Figure 4-109 as a result of filling the left-of-decimal requirements with spaces. These two conditions are the only ones utilizing the SEDP3 gate shown in Figure 4-109.

(2) 0 Right-of-Decimal. If an LRD pattern is used specifying 0 right-of-decimal and the instruction is not terminated early by one of the methods already discussed, it is completed when the decimal position is reached. There are three methods by which the decimal position may be achieved: namely, filling the left-of-decimal requirements with digits, filling left-of-decimal with spaces, and entering a decimal. Although the timing involved in each of these functions is slightly different, the basic operation is the same.

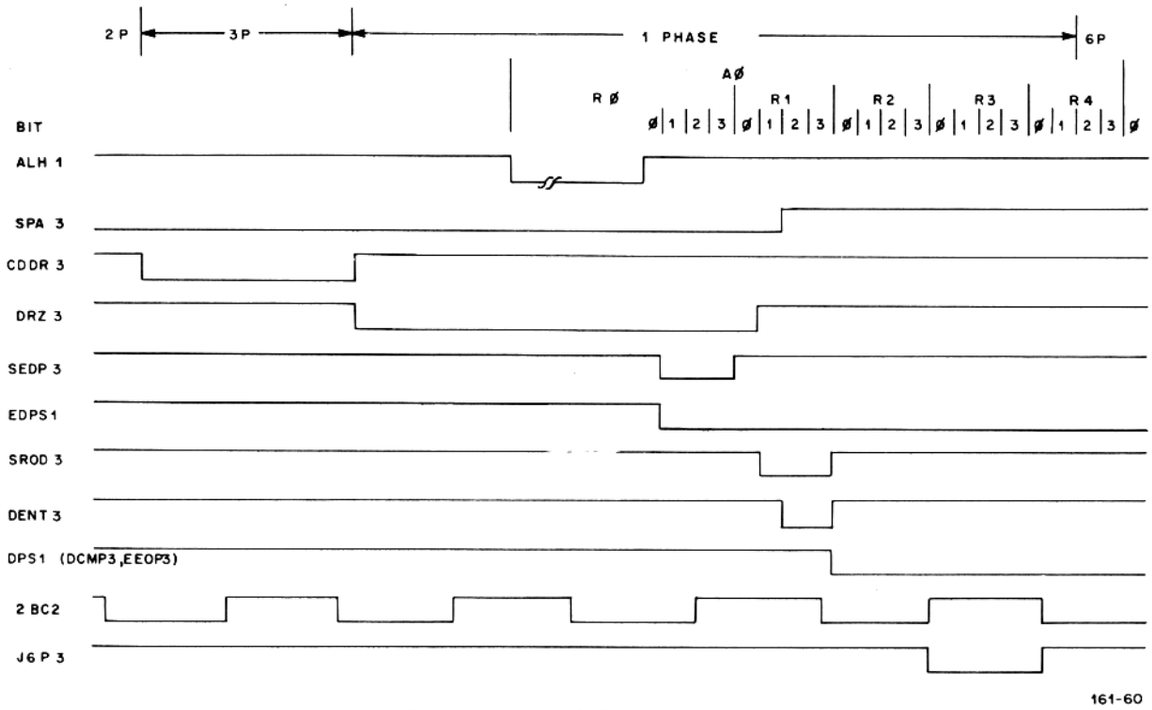
(a) Fill Left-of-Decimal With Digits. (Figure 4-116.) As the last digit left-of-decimal is shifted out of the common register and R1 is shifted left, the decimal register is counted down to 0 in A15 time. In A15R1 time, SROD3 sets the decimal register to 5, and at R2 time DPS sets, developing DCMP3 and EEOP3. At the same time, DEMP resets, enabling a jump to 1 phase. As soon as the logic reaches 1 phase, EEOP3 enables a jump to 6 phase, where the arithmetic portion is terminated.



FILL LEFT-OF-DECIMAL WITH DIGITS
0 RIGHT-OF-DECIMAL

FIGURE 4-116

(b) Fill Left-of-Decimal With Spaces. (Figure 4-117). When the last position left-of-decimal is filled with a space, CDDR3 is developed in 3 phase, and as the jump back to 1 phase occurs, the decimal register is counted down to 0. At A0R0B1 time, SEDP3 is developed, which sets EDPS. At the next R1 time, SROD3 sets the decimal register to 5 and DENT resets the common register (ARCR). At R2 time DPS sets, DCMP3 is developed, and EEOP3 becomes true, enabling the jump to 6 phase.

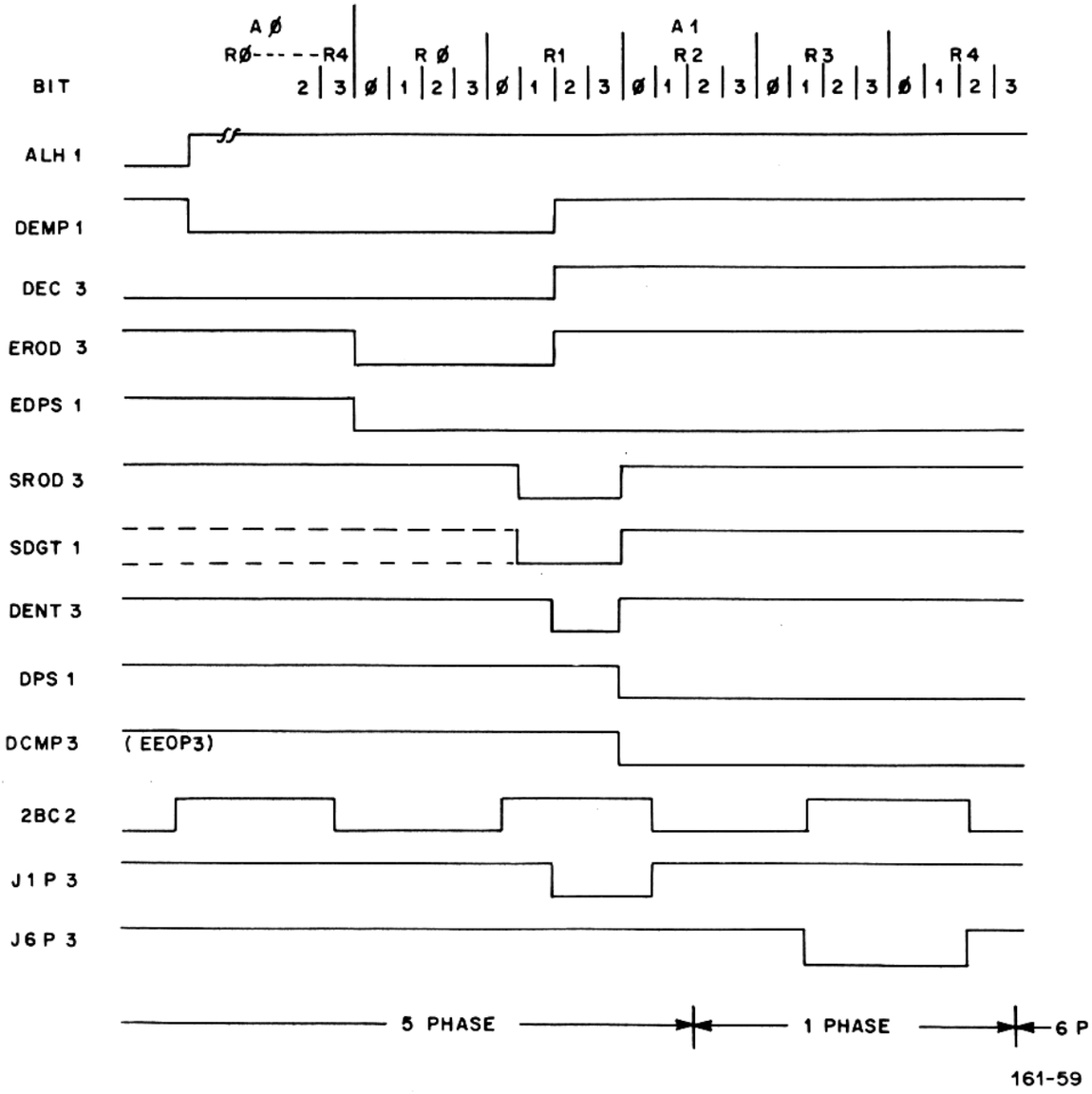


FILL LEFT OF DECIMAL WITH SPACES
0 RIGHT-OF-DECIMAL

FIGURE 4-117

(c) Enter Decimal. (Figure 4-118.) When the decimal is entered, the first A1 time (in 5 phase) develops EROD3, which sets EDPS. At the next R1 time, SROD3 is developed, which produces DENT3 to reset the common register and the DEMP flip-flop. DEMP being reset enables a jump to 1 phase. At R2 time, DPS sets, developing DCMP3 and EEOP3. The latter then enables a jump from 1 phase to 6 phase.

(3) 6 Right-of-Decimal. When the application calls for either a positive or negative entry of 5 to the right, the program must specify a 6-right LRD pattern: if only 5 digits to the right were allowed, the entry would terminate after the fifth right-of-decimal digit were entered. This would not allow for entry of the sign. Instead, the 6-right LRD setting allows the operator to enter either a space or a minus to terminate the entry after entering five digits right-of-decimal. Note, however, that when a right-of-decimal setting of 6 is called for, DCMP3 cannot be developed: no amount of digits entered will terminate the entry. In this case, either a minus or space, tab, etc., MUST be entered in order to terminate the instruction. The logic involved in this operation can be found in the "Decimal Compare" discussion in Section 8.

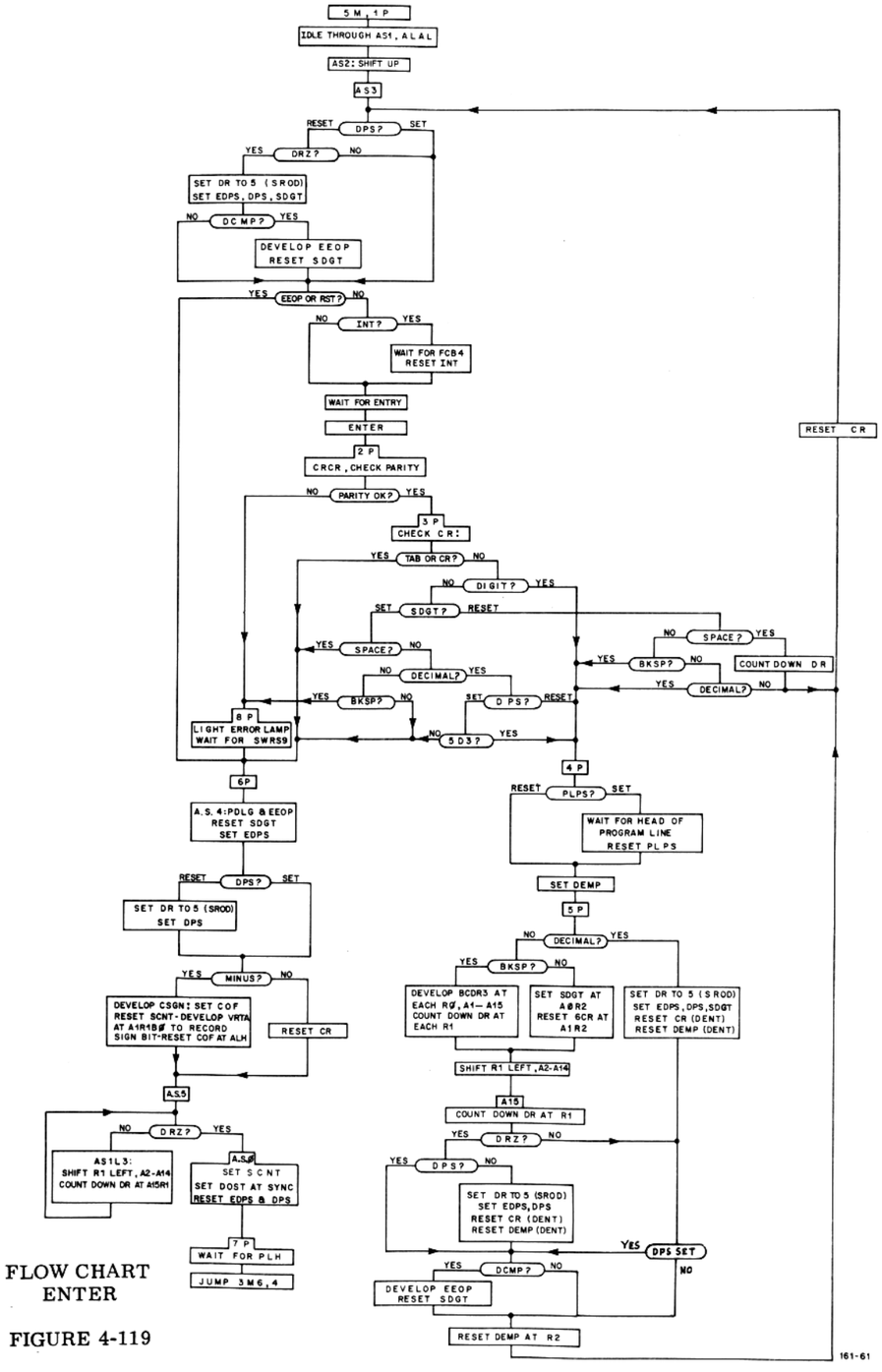


DECIMAL ENTRY
0 RIGHT-OF-DECIMAL

FIGURE 4-118

10. FLOW CHART

The flow chart in Figure 4-119 summarizes the complete Enter operation with all possible variations.



FLOW CHART
ENTER

FIGURE 4-119

H. PRINT

The two print instructions, Decimal Print and Non Decimal Print, are identical in all but a few respects. The two will be treated as one, with a short explanation added to cover the printing of the decimal.

The print instructions are detailed here only to the point of developing the logic signals which are sent to the writing machine interface. A later discussion, in Section 8, Electronic Circuits, will show how these signals are converted to drive the 2205.

1. GENERAL OPERATION

When one of the print commands is retrieved, it causes the two address digits following to be stored in the decimal register and search register. Then, in 5 mode, each digit of R1 is shifted from the arithmetic delay line into the common register, and then to the writing machine.

a. 4 MODE OPERATION. When a "T" or an "X" is retrieved in 4 mode 3 phase, the phase counter jumps to 6 phase, where the marker is written out immediately behind the command, and the first address digit, representing the number of digits to be printed left-of-decimal, is shifted into the common register. Just before the step to 7 phase, this digit is strobed into the decimal register. In 7 phase the second address digit, representing the number of digits to be printed right-of-decimal, is shifted into the search register. Then a step to 8 phase occurs and the balance of the program is circulated PLCR, CRPL until the jump to 5 mode 9 phase, which occurs at program line home.

b. 5 MODE OPERATION. The operation in 5 mode can best be broken down into operation by phases. However, the action steps are somewhat inter-related with the phases, and they must also be covered.

5 mode is entered in 9 phase, and the phase counter waits there until action step 3 is reached. Then the phase counter steps through phases 10, 11, 12, 13, and 14 for each digit that is printed, and returns to 9 phase for the next digit. Action step 3 remains true during this time. After the last digit is printed, 14 phase remains true while the action step counter passes through steps 4 and 5 and returns to 0. Then a jump to 6 phase occurs and at the next program line home a jump to 3M6,4 takes place to begin the search for the next instruction.

(1) 9 Phase. The first time 9 phase is realized in 5 mode, the phase counter waits for the action step counter to reach 3. In action step 1 the contents of R1 are duplicated into R0. In action step 2, R1 is shifted left to bypass any high order digits that are not to be printed. In action step 3 the first digit to be printed is retrieved from the arithmetic delay line. Action step 3 also enables the step to 10 phase.

9 phase is used on subsequent "cycles" to shift each digit to be printed into the common register. Also, the logic waits in 9 phase for the writing machine to complete the typing of the previous digit.

(2). 10 Phase and 11 Phase. These two phases provide odd parity for the code to be printed. The 5CR flip-flop becomes set in 10 phase, and a common register ring shift occurs in 11 phase.

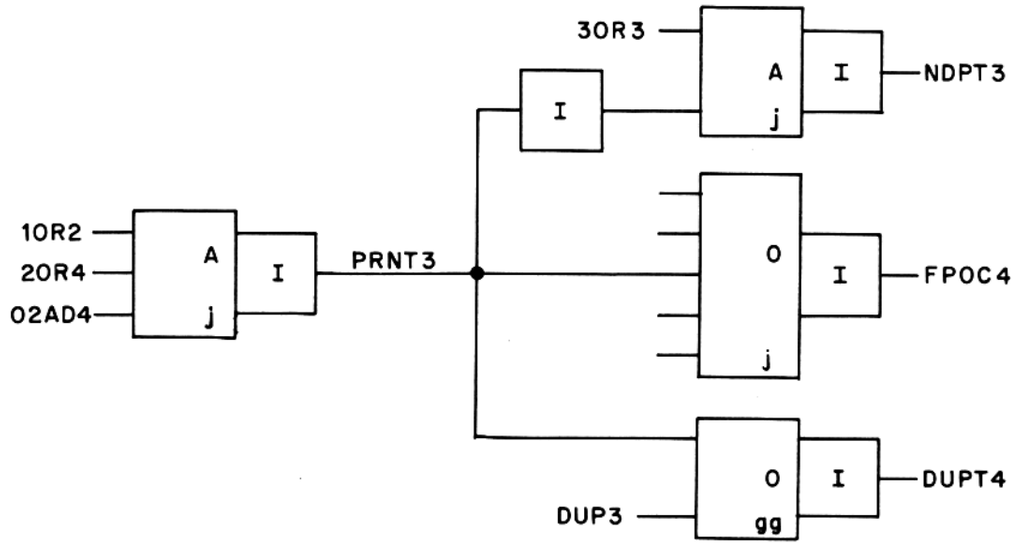
(3) 12 Phase. The actual pulse to drive the writing machine occurs in 12 phase. The logic remains in 12 phase until after the 2205 translator begins to rotate and the busy signal, BZY, is developed.

(4) 13 Phase. The common register becomes reset in 13 phase. The phase counter then syncs onto the arithmetic line for the step to 14 phase.

(5) 14 Phase. The next shift left, for the retrieval of the next digit to be printed, is initiated in 14 phase. A jump to 9 phase then occurs and the entire process repeats. After the last digit is printed, the logic remains in 14 phase until the action step counter returns to 0. Then a jump to 6 phase takes place.

2. RETRIEVAL OF THE PRINT INSTRUCTION

As soon as a "T" or an "X" is shifted into the operation register in 4 mode 3 phase, PRNT3 is developed. This signal drives DUPT4 (DUplicate or PrinT) and FPOC4 (Five Phase Op Code) both false. DUPT4 allows the duplication of the R1 data into R0 in action step 1. FPOC4 is necessary to enable the action step counter to utilize all five steps. In addition, the Non Decimal Print code "T" (1-2-6) does not contain a 3-bit and develops NDPT3. The Decimal Print "X" (1-2-3-5-6) holds NDPT3 false. See Figure 4-120.



"MASTER" SIGNALS IN PRINT

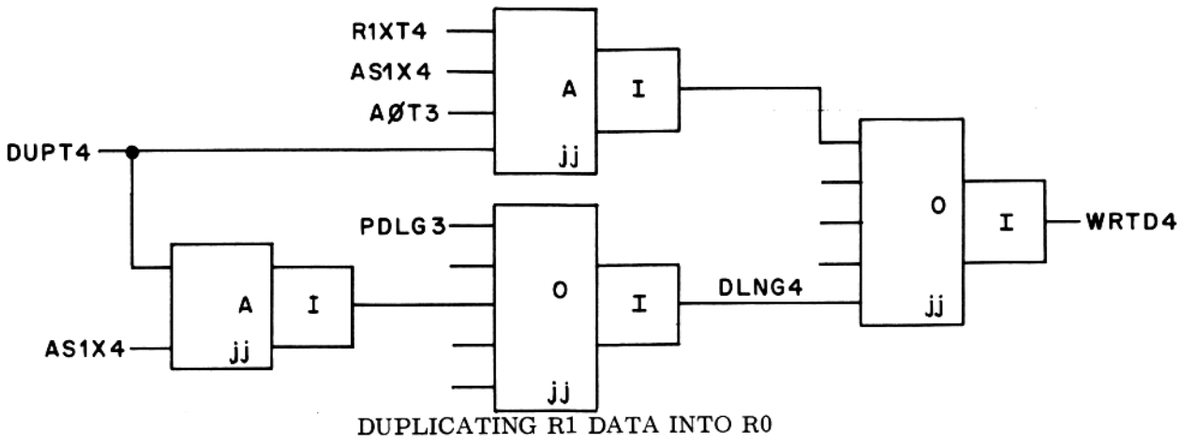
FIGURE 4-120

At the end of 4 mode 3 phase the logic jumps to 6 phase, and then steps to 7 phase and 8 phase, meanwhile retrieving the address digits. Refer to Section 3H, Instruction Retrieval, for review. In 8 phase two requirements must be met before moving on: the writing machine must not be busy and the program line must be at home. In other words, if the 2205 is not busy, the jump to 5 mode 9 phase will occur at the next program line home. If the 2205 is busy, the logic waits for it to finish and then jumps at the next program line home.

3. PRELIMINARY 5 MODE OPERATIONS

Before the first digit of the printout can be sent to the writing machine, it must be retrieved from the arithmetic delay line. Because the digits are not shifted back onto the delay line after they are printed, the first action performed is the duplication of the R1 information into R0. Then, upon completion of the print instruction, the untouched data in R0 is put back into R1.

a. ACTION STEP 1. The duplication of R1 data into R0 during print is accomplished in exactly the same manner as in the Duplicate instruction. DUPT4 is ANDed with AS1X4 to drive DLNG4 false for the entire line pass, but at the same time "write direct" is enabled during R1 time. This causes R1 data to be written during both R1 and R2 time, so that on the next line pass it appears in both R0 and R1 time. The gates are illustrated in Figure 4-121. Refer to Section 4E, Duplicate, for review.



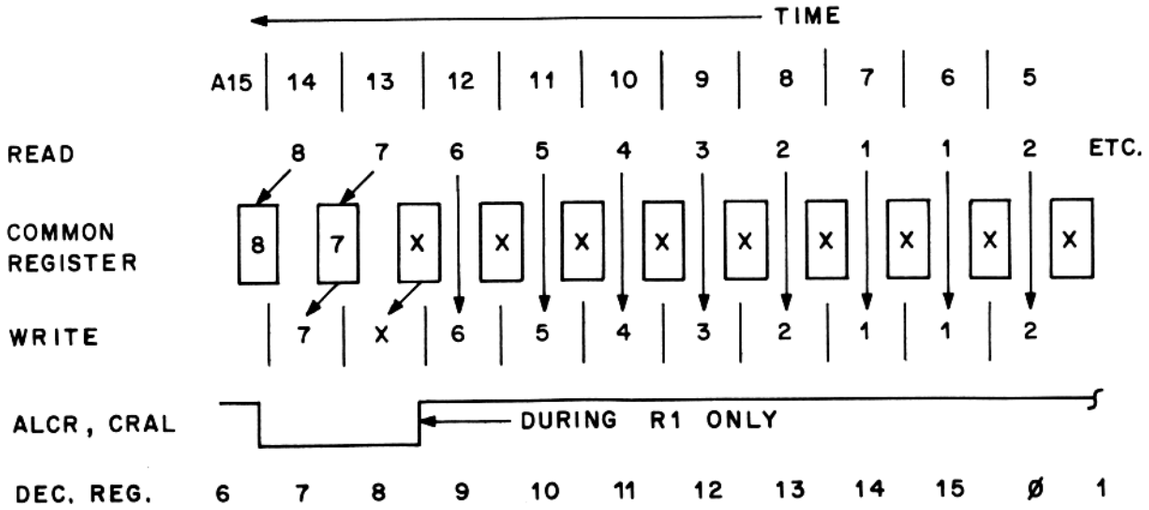
DUPLICATING R1 DATA INTO R0

FIGURE 4-121

b. ACTION STEP 2. At the end of this line pass the action step counter goes to step 2, where the contents of R1 are shifted left until the first digit to be printed is ready to be shifted into the common register. Action step 2 may require from 1 to 9 line passes, depending upon the left-of-decimal setting in the decimal register.

The method used to locate the first digit to be printed is rather unique. To begin, a binary "ten" is inserted into R1, in the column one greater than the left-of-decimal setting. Then R1 is shifted left until this "ten" remains in the common register at the end of the shift left line pass. When this occurs, the step to action step 3 is enabled, where one more line pass shifts the first digit to be printed into the common register.

The binary "ten" is inserted into R1 at the correct position as follows: first, a binary ten is set into the common register. Next, the decimal register is counted down once (from the left-of-decimal setting) for each arithmetic column. Then, when the decimal register is equal to 9, the write amplifier has just finished writing the first digit to be printed. At this instant the line routing changes from ALAL to ALCR, CRAL, or from "short" to "long". The binary "ten" is shifted out of the common register onto the arithmetic line, and the next digit is shifted into the common register. This line routing remains for the balance of the line pass, and this, in effect, shifts the rest of the R1 data to the left one column. An example is illustrated in Figure 4-122. In other words, the digits to the left of those to be printed are shifted left, those to be printed do not move, and the binary ten is inserted into the "hole" remaining.

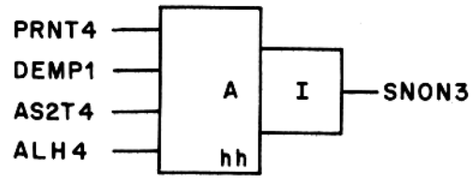


INSERTING BINARY "TEN" INTO R1

FIGURE 4-122

TP-161SL

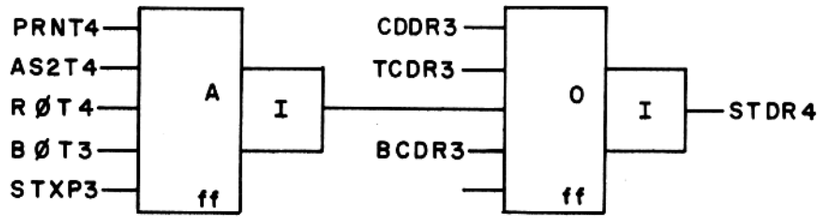
The binary ten is produced as shown in Figure 4-123. The Set NOn Number signal, SNON3, sets 2CR and 4CR.



DEVELOPMENT OF SNON3

FIGURE 4-123

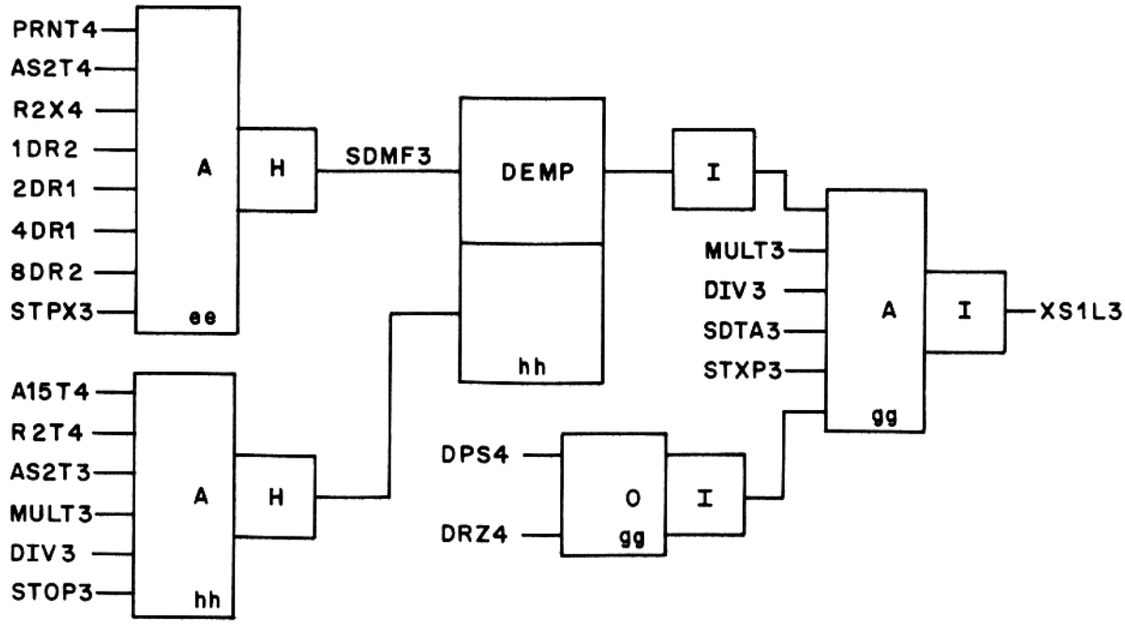
Then STDR4 is driven false at R0B1 time in each column (see Figure 4-124), and when it returns to true at R1 time, it counts down the decimal register once.



STEPPING DR DURING AS2

FIGURE 4-124

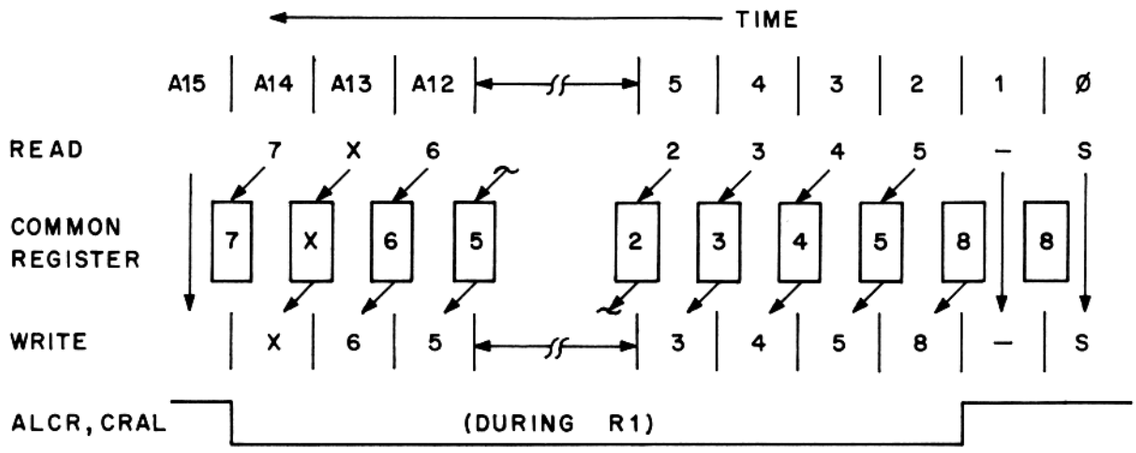
When the decimal register reaches 9, SDMF3 is produced at R2 time (see Figure 4-124), and DEMP sets, driving XS1L3 true. This causes the high order digits of R1 to be shifted left, starting with the digit just to the left of the first one to be printed.



SHIFT LEFT DURING AS2

FIGURE 4-125

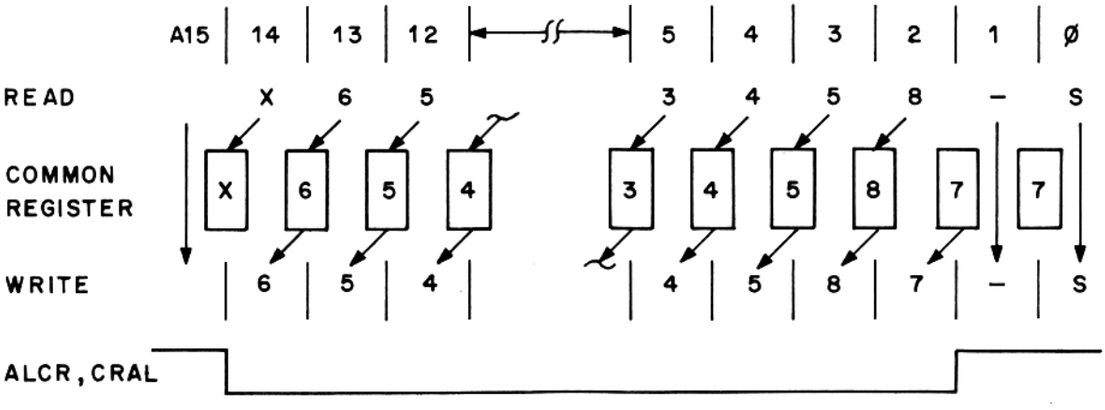
Still referring to Figure 4-125, note that DEMP does not reset in action step 2, so subsequent line passes cause additional shifts left. These additional shifts, however, shift ALL of R1: the code remaining in the common register at the end of the previous line pass is shifted into A2R1, and each digit in R1 is shifted one column to the left. Figure 4-126 illustrates the second line pass in action step 2, utilizing the same example shown previously in Figure 4-122.



SECOND SHIFT LEFT IN AS2

FIGURE 4-126

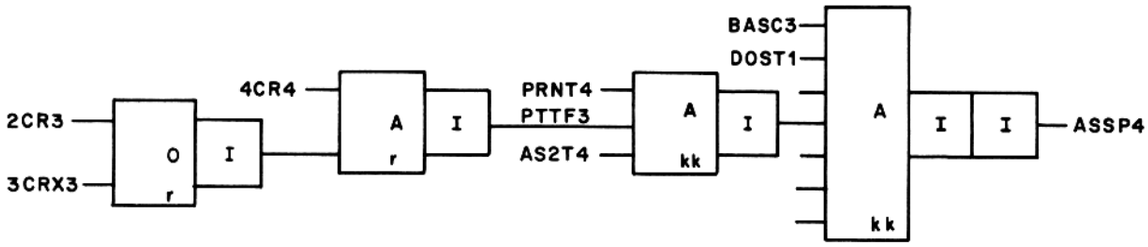
Subsequent line passes continue to shift R1 left until the "non-number" is left in the common register at the end of the line pass. In the example being used here, this occurs after one more line pass. This is illustrated in Figure 4-128.



LAST LINE PASS IN AS2

FIGURE 4-127

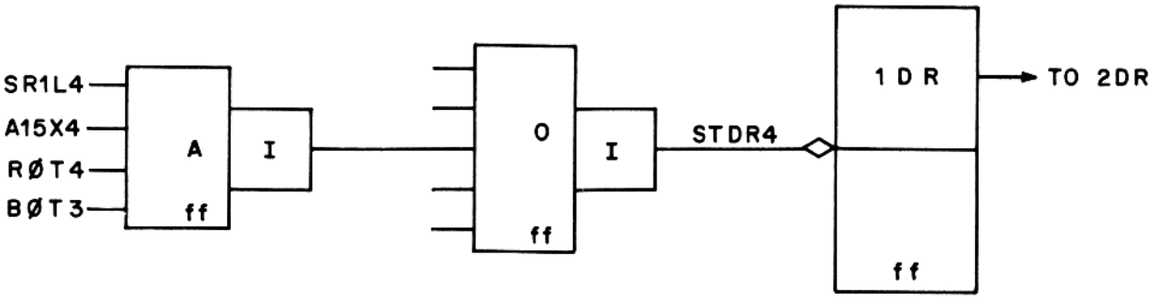
Note that the decimal register continues to be counted down once in each column. This has no purpose now that the non-number has been written out, but because it does no harm, it is not inhibited. Note also that the decimal register is still at the left-of-decimal setting (6 in this example) at each arithmetic line home. Whenever the non-number is in the common register, it develops the Print Ten To Fifteen signal, PTTF3. In action step 2 this signal is necessary to increment the action step counter. See Figure 4-128. Since the action step counter can only be "primed" at A15R4 time, it is necessary to have PTTF3 true at A15R4 time in order to get out of action step 2 into step 3. This occurs when the non-number has been shifted left, out of A14 into the common register. Thus, when the action step counter is allowed to step from 2 to 3, the common register contains a "ten" and the first digit that is to be printed is located in A14R1.



PTTF3 ACTION

FIGURE 4-128

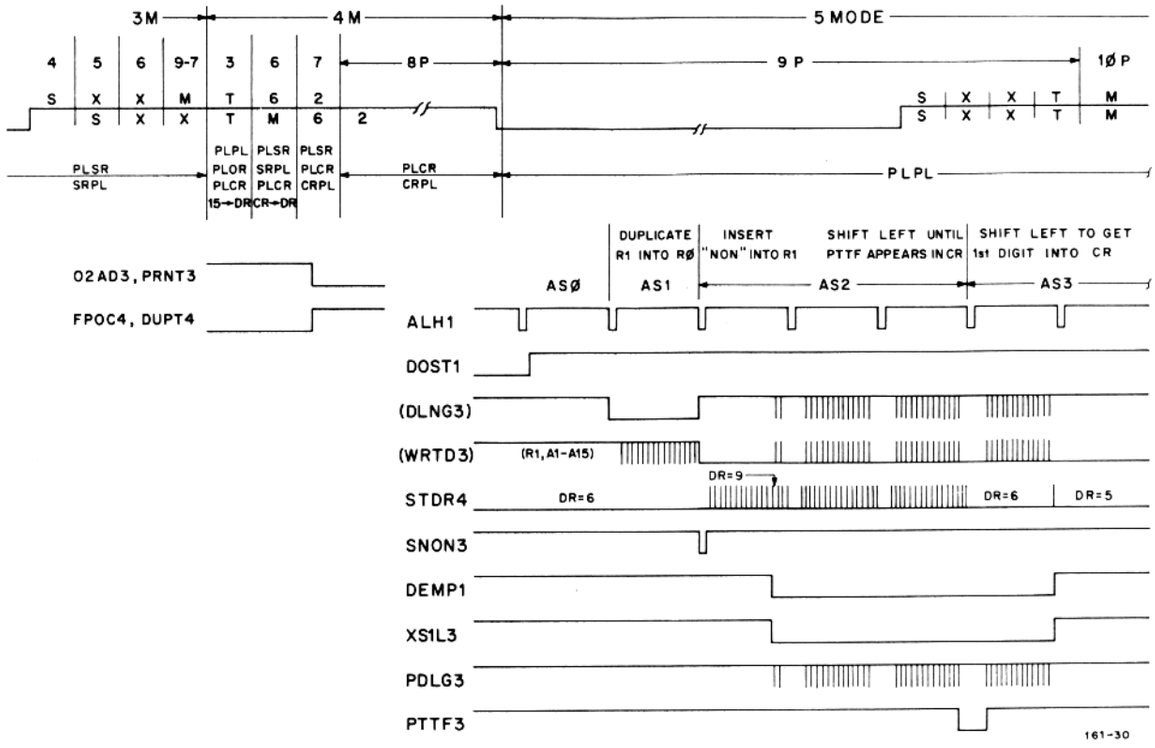
c. ACTION STEP 3. In action step 3 the digit to be printed is retrieved from the arithmetic line and sent to the 2205. The first arithmetic line pass in action step 3 causes a shift left of R1 because DEMP is still set and XS1L3 is still true (refer to Figure 4-125). During this shift left the digit to be printed is shifted out of A14R1 into the common register. At the end of this line pass the decimal register is counted down once (A15R1) and DEMP becomes reset (A15R2). Refer to Figure 4-125 for the reset of DEMP. The gate developing STDR4 is illustrated in Figure 4-129.



COUNTING DOWN DR DURING SHIFT LEFT

FIGURE 4-129

d. SUMMARY. All of the events discussed up to this time are summarized in the timing chart in Figure 4-130. Except for the shift left just covered these items can be considered as occurring only once during the Print instruction.



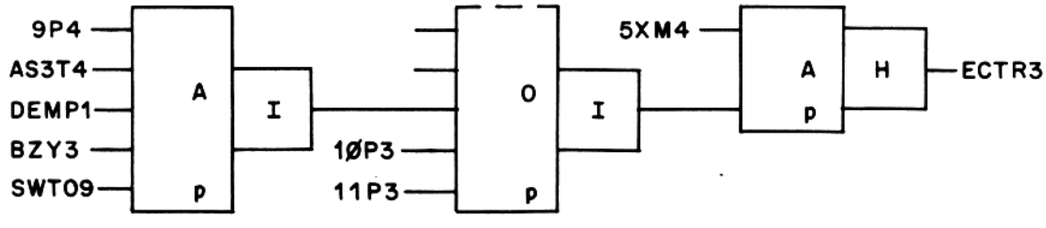
PRINT
PRELIMINARY OPERATIONS

FIGURE 4-130

4. REPETITIVE OPERATIONS. The events which follow are considered repetitive because they occur once for each digit that is printed.

a. 9 Phase. The shift left and the counting down of the decimal register occur for each digit printed, and therefore are repetitive operations. They were covered earlier, however, to show the relationship of the first digit to the preliminary 5 mode operations.

When DEMP becomes reset in action step 3, ECTR3 is developed as shown in Figure 4-131, and the phase counter steps to 10 at the next T6B. Figure 4-131 also shows that the development of ECTR3 in phases 10 and 11 is automatic, requiring only a T6B for the stepping of the phase counter.

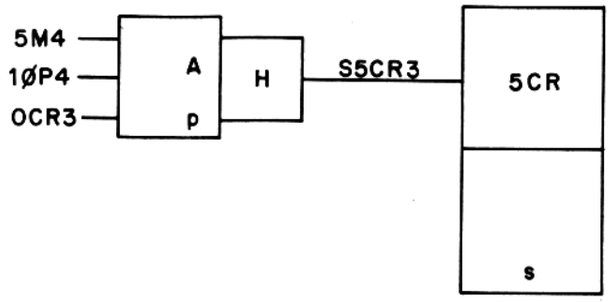


DEVELOPMENT OF ECTR3 IN PHASES 9, 10 AND 11

FIGURE 4-131

TP-161SL

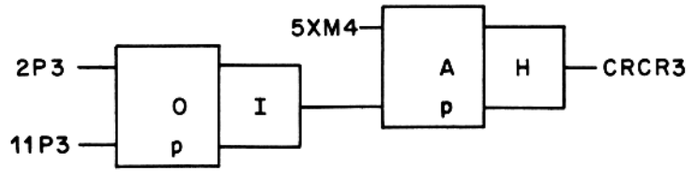
b. 10 PHASE AND 11 PHASE. In 10 phase the 5CR flip-flop becomes set as shown in Figure 4-132. (OCR3 inhibits this during the carriage return instruction to prevent the reset of 8CR.)



DEVELOPMENT OF S5CR3

FIGURE 4-132

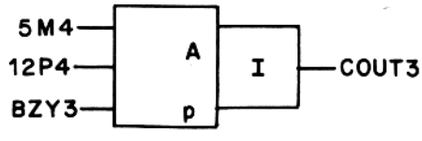
A common register ring shift (CRCR3) occurs during 11 phase, as shown in Figure 4-133. During this ring shift, 5CR becomes reset if the first four CR flip-flops contain an odd number of bits (1 or 3). If an even number of CR flip-flops are set, 5CR remains set to provide odd parity. In other words, if the digit being printed is a 1, 2, 4, 7, or 8, the 5CR flip-flop resets; if the digit is a 0, 3, 5, 6, or 9, 5CR remains set. (The space-to-zero conversion will be covered shortly.) Note that CRCR is one of the few "line routings" used in 5 mode. Remember that the actual program line data always recirculates PLPL during 5 mode.



DEVELOPMENT OF CRCR3 IN 5M11P

FIGURE 4-133

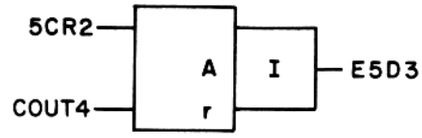
c. 12 PHASE. At the beginning of 12 phase the code to be sent to the writing machine is in the common register. The Common OUTput signal, COUT3, is developed (Figure 4-134), which causes a Silicon Controlled Rectifier (SCR) in the interface to fire and energize the 2205 translator clutch. The circuitry involved from the development of COUT3 onward is discussed in Section 8, Electronic Circuits.



DEVELOPMENT OF COUT3

FIGURE 4-134

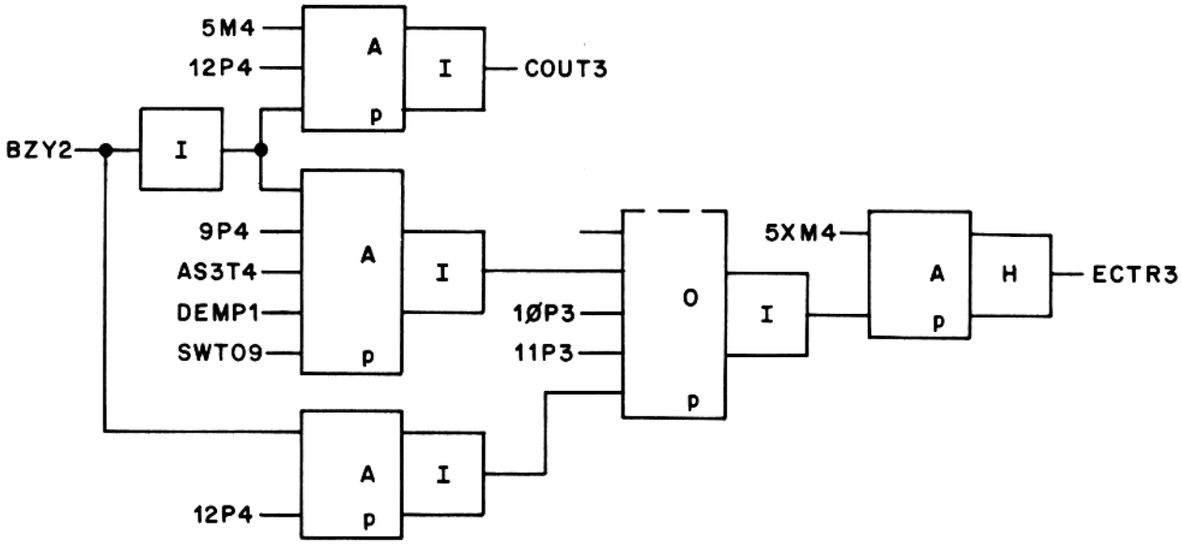
At the same time, COUT4 enables the various translator bit magnets. An example is illustrated in Figure 4-135. If 5CR is set, E5D3 becomes true when COUT3 is developed. E5D3 is then routed to the interface to fire another SCR, which energizes LT5 in the 2205.



DEVELOPMENT OF TRANSLATOR BIT DRIVE SIGNAL

FIGURE 4-135

After the translator clutch is energized and the translator passes 20 degrees of rotation, the Busy flip-flop, BZY, sets, and simultaneously drives COUT3 false (Figure 4-134) and develops ECTR3 for the step to 13 phase. This latter action is illustrated in Figure 4-136.



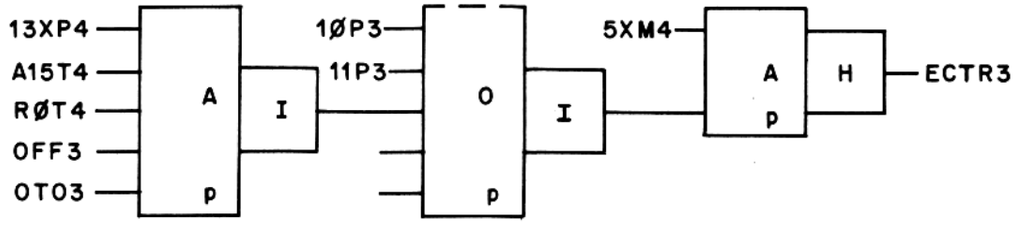
EFFECT OF BZY IN 5M12P

FIGURE 4-136

The development of BZY is somewhat involved, so it is treated separately in Section 6, Flexowriter Commands. Because it involves both the processor and the writing machine, it cannot be stated that BZY is developed at any exact time in relation to the translator cycle.

At this time it is well to note that the FFF flip-flop on the "P" card may set and reset during the course of a Print instruction. Since this operation might be noticed and a question arise, it is mentioned here only to explain that it has no function during print.

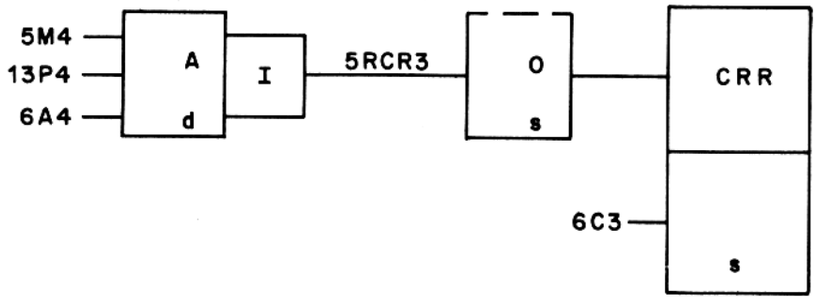
d. 13 PHASE. In 13 phase the phase counter is again synchronized with the arithmetic line. Because the step from 12 phase to 13 phase is gated by the writing machine, it occurs randomly in relation to the arithmetic line. Therefore, the phase counter must be re-synchronized with the arithmetic line in order to retrieve the next digit to be printed. This is accomplished by qualifying ECTR3 with A15R0 time, as shown in Figure 4-137.



DEVELOPMENT OF ECTR3 IN 5M13P

FIGURE 4-137

In addition, the code just printed is reset out of the common register in 13 phase when 5RCR3 is developed to set the CRR flip-flop. See Figure 4-138.

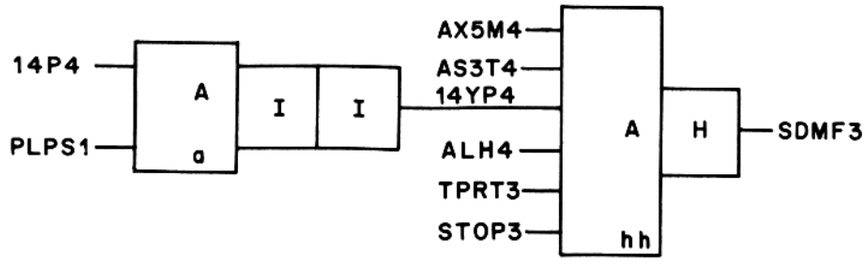


DEVELOPMENT OF 5RCR3

FIGURE 4-138

e. 14 PHASE. In 14 phase a shift left is initiated to retrieve the next digit to be printed. In addition, the jump back to 9 phase is set up. At the completion of the Print instruction this jump is replaced by a jump to 6 phase. This will be covered shortly.

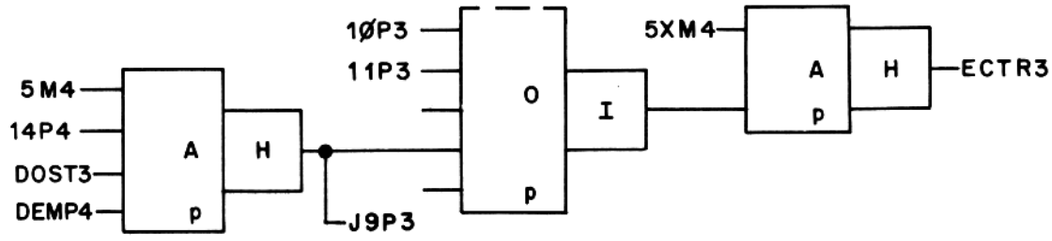
(1) Shift Left. Shortly after 14 phase becomes true, arithmetic line home is reached and DEMP becomes set, as shown in Figure 4-139. DEMP drives XS1L3 true, and during the ensuing line pass R1 is shifted left and the next digit to be printed is shifted out of A14R1 into the common register. Near the end of the line pass (A15R2), DEMP is reset in the usual manner (refer to Figure 4-125).



DEVELOPMENT OF SDMF3 IN 5M14P

FIGURE 4-139

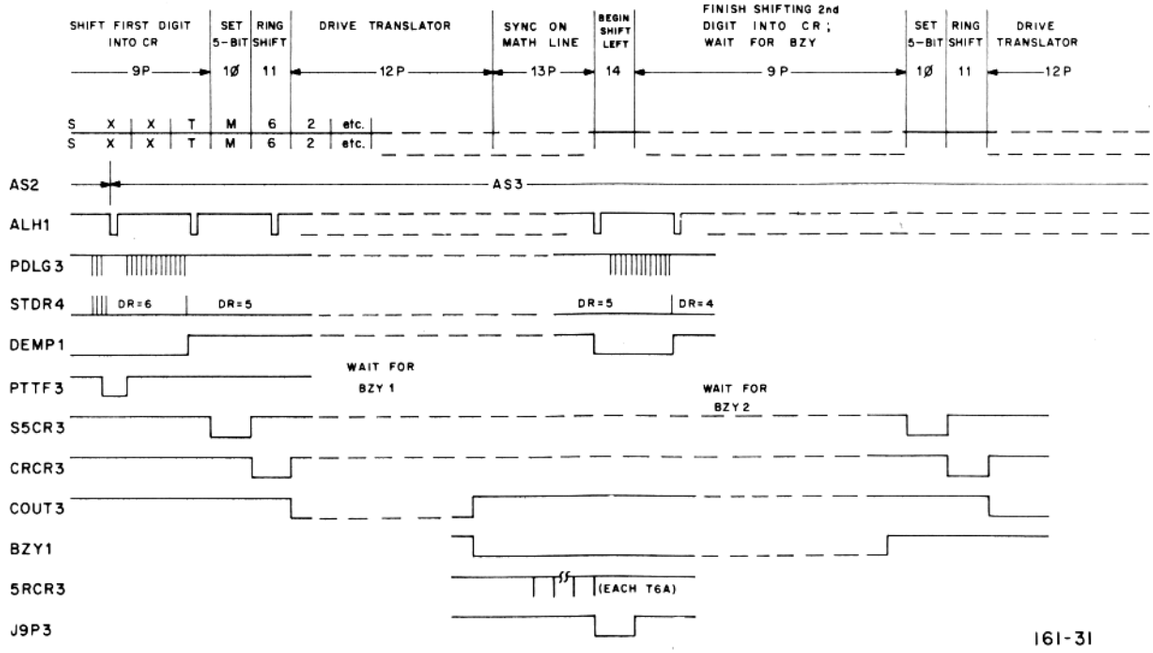
(2) Jump to 9 Phase. The jump to 9 phase is set up as shown in Figure 4-140. Note that this jump occurs before the shift left is completed. 14YP4 qualifies SDMF3 (Figure 4-139) to ensure that there will be at least one more T6B, after the shift left begins, to jump the counter out of 14 phase: if this qualification were not present, it would be possible to set DEMP just before program line home and then be unable to jump out of 14 phase for lack of a T6B pulse. Then DEMP would set at each arithmetic line home, and each arithmetic line pass would produce another shift left.



DEVELOPMENT OF J9P3 IN 5M14P

FIGURE 4-140

f. SUMMARY. The repetitive operations are summarized in the timing chart in Figure 4-141, which begins with the retrieval of the first digit to be printed and carries through with one more digit. Each digit causes the same operations to occur, and these operations are used in printing digits both to the left- and right-of-decimal. The logic involved in printing the decimal will be presented shortly. For now, though, the "repetitive operations" must be understood before going ahead with the rest of the logic.



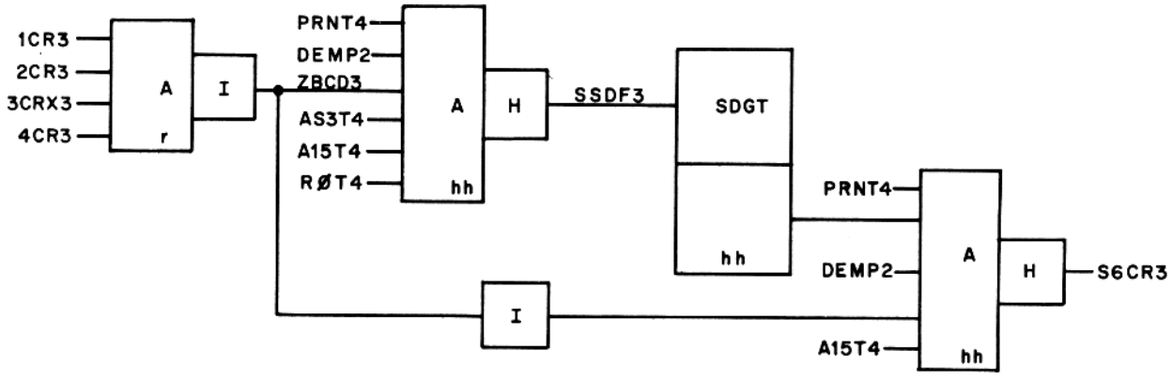
PRINT
REPETITIVE OPERATIONS

FIGURE 4-141

5. ZERO TO SPACE CONVERSION

All insignificant zeros (zeros to the left of both the decimal and the first significant digit) are printed as spaces instead. This conversion is accomplished very easily in phases 10 and 11, where the 5 bit is set and a ring shift follows. If the common register contains a zero (no bits), the 5CR flip-flop does not become reset during the ring shift, and the 5 bit is then sent to the translator (in 12 phase) to type the space.

After a significant digit has been printed, or after the decimal position has been reached, all zeros are printed as zeros; they are no longer converted to spaces. When the first significant digit is retrieved from the arithmetic line, the Set Significant Digit Flip-flop signal, SSDF3, is developed, and the Significant DiGiT flip-flop, SDGT, is set (Figure 4-142). As long as the first four CR flip-flops are reset, the Zero Binary Coded Decimal signal, ZBCD3, will be true to prevent the setting of SDGT. However, when a significant digit is retrieved, at least one of these flip-flops will be set, driving ZBCD3 false. This produces SSDF3, which sets SDGT at A15R0 time, when the shift left is complete. SDGT then remains set until the Print instruction is terminated.



SDGT FACTORS DURING PRINT

FIGURE 4-142

Still referring to Figure 4-142, note that if a zero is later retrieved while SDGT is set, ZBCD4 goes false and S6CR3 is developed at A15 time. This sets the 6CR flip-flop and causes 5CR to become reset during the subsequent ring shift. Thus, a zero is printed instead of a space.

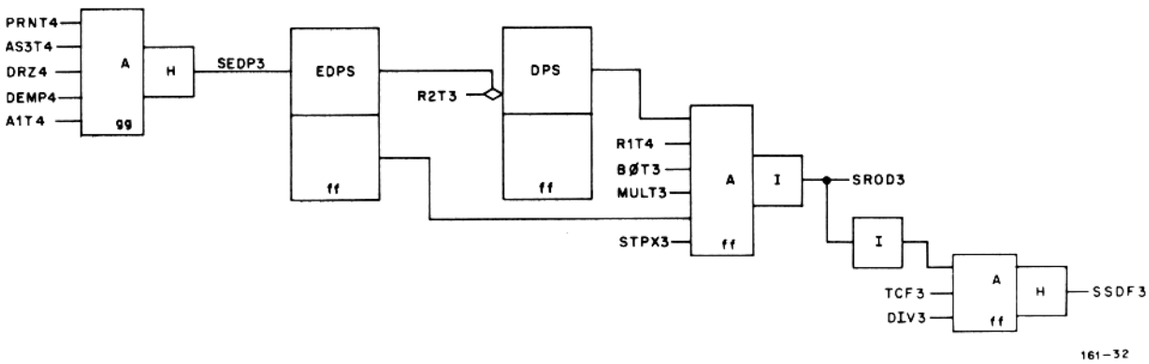
6. DECIMAL CONSIDERATIONS

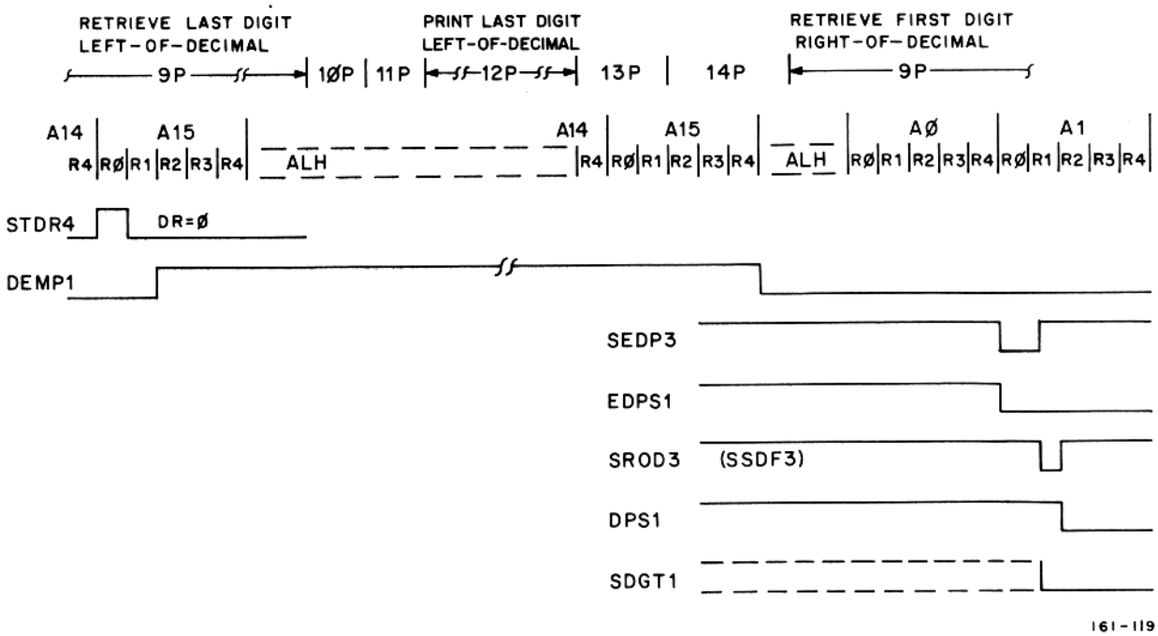
The decimal register is counted down once for each digit printed. When the decimal setting reaches 0, it denotes that the left-of-decimal requirements have been fulfilled. This means that the decimal register must be set to 5 to allow the printing of the right-of-decimal digits. It also means that SDGT must be set if it has not already been set by a significant digit. It means that the decimal point must be printed, if the Decimal Print instruction is being executed. Last, it means that the instruction must be terminated if the right-of-decimal setting is 0. This latter item will be covered shortly, as part of the discussion on terminating the Print instructions.

a. RIGHT OF DECIMAL SET-UP. Once the left-of-decimal digits have been printed, the decimal register must be set up to control the printing of those digits to the right of the decimal. This involves setting the decimal register to 5, and then counting it down once for each digit that is printed. When the proper number of right-of-decimal digits has been printed, Decimal CoMPare (DCMP) is developed, and the Print instruction is terminated.

After the last digit left-of-decimal is retrieved from the arithmetic line, the decimal register is counted down to 0. This occurs in 9 phase. The phase counter follows its usual path through 10, 11, 12, and 13 phases and steps to 14 phase, where the shift left is initiated to retrieve the first digit to the right of the decimal. Just before this shift actually begins, at A1 time, Set Enable Decimal Point storage, SEDP3, is produced to set EDPS. At A1R1B1 time, Set Right Of Decimal SROD3, is developed, which sets 1DR and 4DR, producing the "5" setting in the decimal register. This procedure, illustrated in Figure 4-143, is followed every time the decimal position is reached.

b. SETTING SDGT. If SDGT has not previously been set by a significant digit, it must be set at the decimal position in order to print zeros to the right of decimal, instead of spaces. Figure 4-143 also shows the development of SSDF3, which sets SDGT at this time.





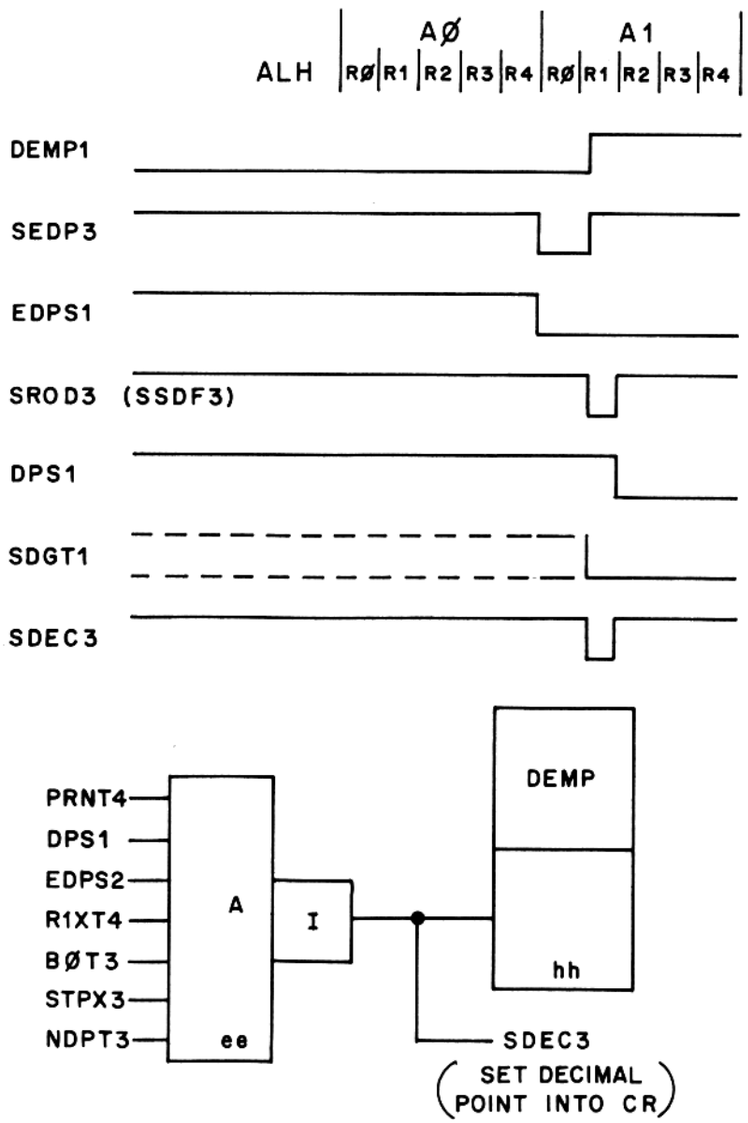
TIMING CHART
SET RIGHT OF DECIMAL

FIGURE 4-143

c. PRINTING THE DECIMAL POINT. If the Decimal Print command (X) is in the operation register at this time, NDPT3 will be false and SDEC3 will be developed as shown in Figure 4-144. In addition to setting the decimal code into the common register, SDEC3 resets DEMP just before the shift left actually begins. In other words, a Non Decimal Print command would allow retrieval of the first digit right-of-decimal at this time; a Decimal Print command blocks this retrieval (by preventing the shift left) and sets the decimal point into the common register instead.

The phase counter then follows its usual path through phases 10-13, wherein the decimal code is sent to the writing machine in the same manner as a digit code. In 14 phase DEMP again sets and R1 is shifted left, shifting the first digit to the right of the decimal into the common register.

This is the only difference between Decimal Print and Non Decimal Print, except for one other item that will be covered under termination of the Print instructions.



DEVELOPMENT OF SDEC3

FIGURE 4-144

d. SUMMARY. There are four main items related to the decimal position during "print". Only one of these ALWAYS occurs when the decimal position is reached. That is the setting of 5 into the decimal register to allow for the printing of digits to the right of the decimal and the subsequent decimal compare.

The other three items occur only under certain conditions. The SDGT flip-flop must become set only if it has not been set previously (although the set signal occurs regardless). The decimal point will be printed only if the Decimal Print instruction is being executed. Last, the instruction must be terminated at the decimal position only if the LRD pattern specified calls for 0 digits to-the-right.

7. PRINTING RIGHT-OF-DECIMAL

The "repetitive operations" in phases 9 through 14 are used for each digit printed right-of-decimal. The printing of these digits is the same as for those left-of-decimal except in the case of the zero: no spaces are ever typed right-of-decimal because the decimal position sets SDGT.

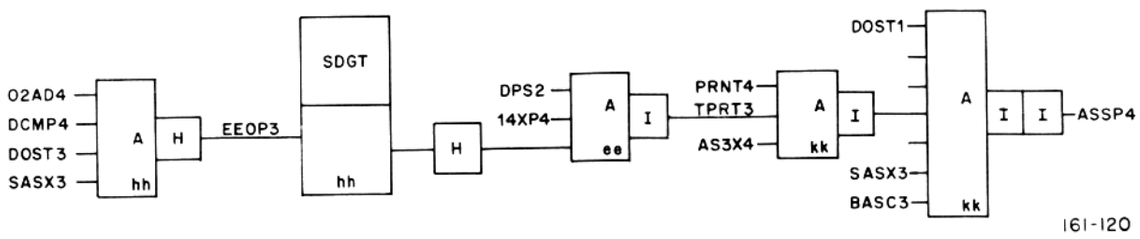
As each digit is retrieved the decimal register is counted down. When the decimal register reaches the correct setting, DCMP3 is produced and the termination procedure begins. (Refer to Section 8, Electronic Circuits, for review of DCMP.)

8. TERMINATING PRINT

The actual sequence of events in terminating a Print instruction varies, depending upon the command and the particular LRD pattern specified. This discussion will first outline a "typical" situation, and then follow up with the "exceptions".

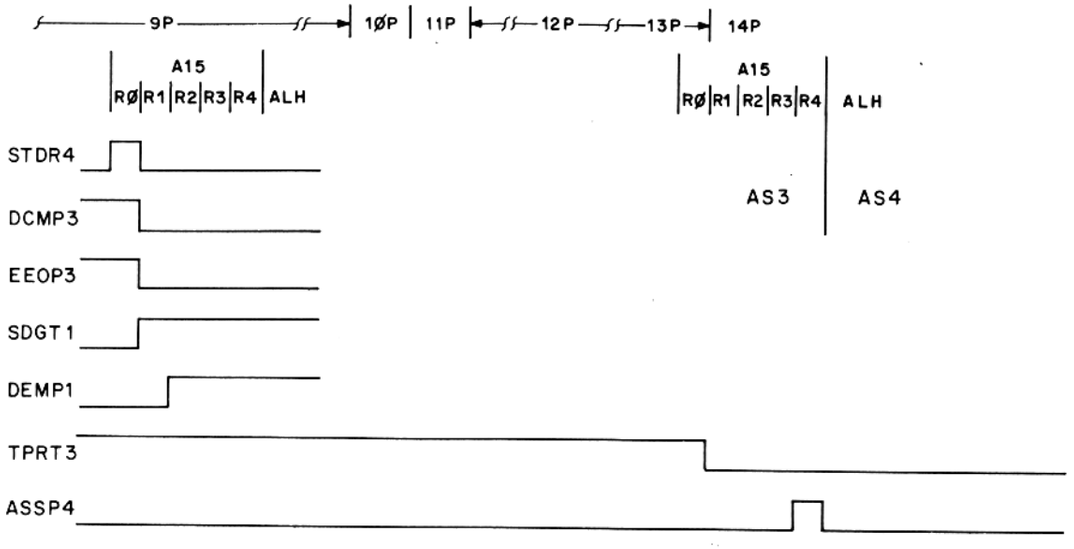
a. STANDARD PROCEDURE. Once the proper number of right-of-decimal digits has been printed, it is still necessary to place the original number back into R1. Since this number was duplicated into R0 at the beginning of the print instruction, this is accomplished by transposing the contents of R0 and R1.

(1) Leaving Action Step 3. As each digit right-of-decimal is retrieved from the arithmetic line, the decimal register is counted down. When the last digit is retrieved, the decimal register is counted down to the setting that allows DCMP3 to be produced. This immediately produces the End of Entry and OutPut signal, EEOP3, which resets SDGT. The phase counter then follows its usual path through phases 10, 11, 12, and 13, while the last digit is sent to the writing machine. When the phase counter steps to 14, however, the Terminate PRinT signal, TPRT3, is developed, which allows the action step counter to leave step 3 at the next arithmetic line home. The termination procedure up to this point is illustrated in Figure 4-145.



LEAVING ACTION STEP 3 IN PRINT

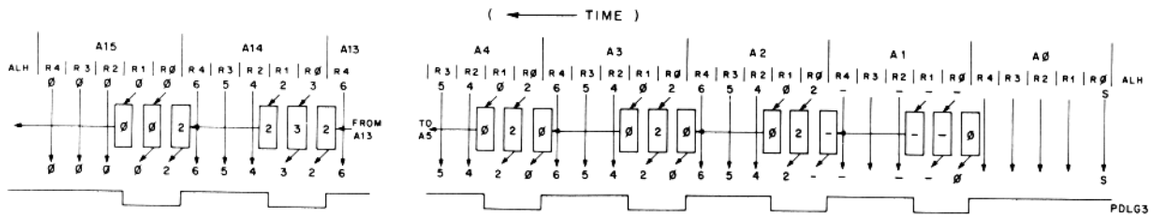
FIGURE 4-145 (a)

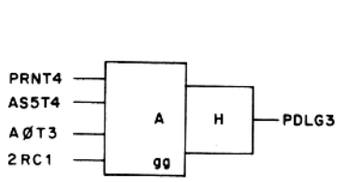


TIMING CHART
LEAVING ACTION STEP 3 IN PRINT

FIGURE 4-145 (b)

(2) Transposing R0 and R1. The procedure previously used to get out of 14 phase was the jump to 9 phase, in which J9P3 also produced ECTR3 (refer to Figure 4-140). This signal cannot be used any longer because DEMP does not become set. Instead, 14 phase remains true while the action step counter idles through action step 4 and increments to step 5. Here the contents of R1 and R0 are inverted: the original number in R1, which was temporarily stored in R0, is put back into R1. The "nonsense" number in R1 is transferred to R0. This is accomplished by simply using the long data path in R0 and R1, and the direct path at all other times. Figure 4-146 shows the gate used to produce PDLG3 at this time, and illustrates a typical example. 2RC1 is false during both R0 and R1, allowing PDLG3 to become true. A0T3 prevents the sync pulse from being recirculated "long". Note that as the "nonsense" number from R1 is being rewritten into R0 it is displaced one arithmetic column. This is of no consequence, however, because it will simply be cleared out of R0 when DOST again sets.





INSTRUCTION: T62 OR X62

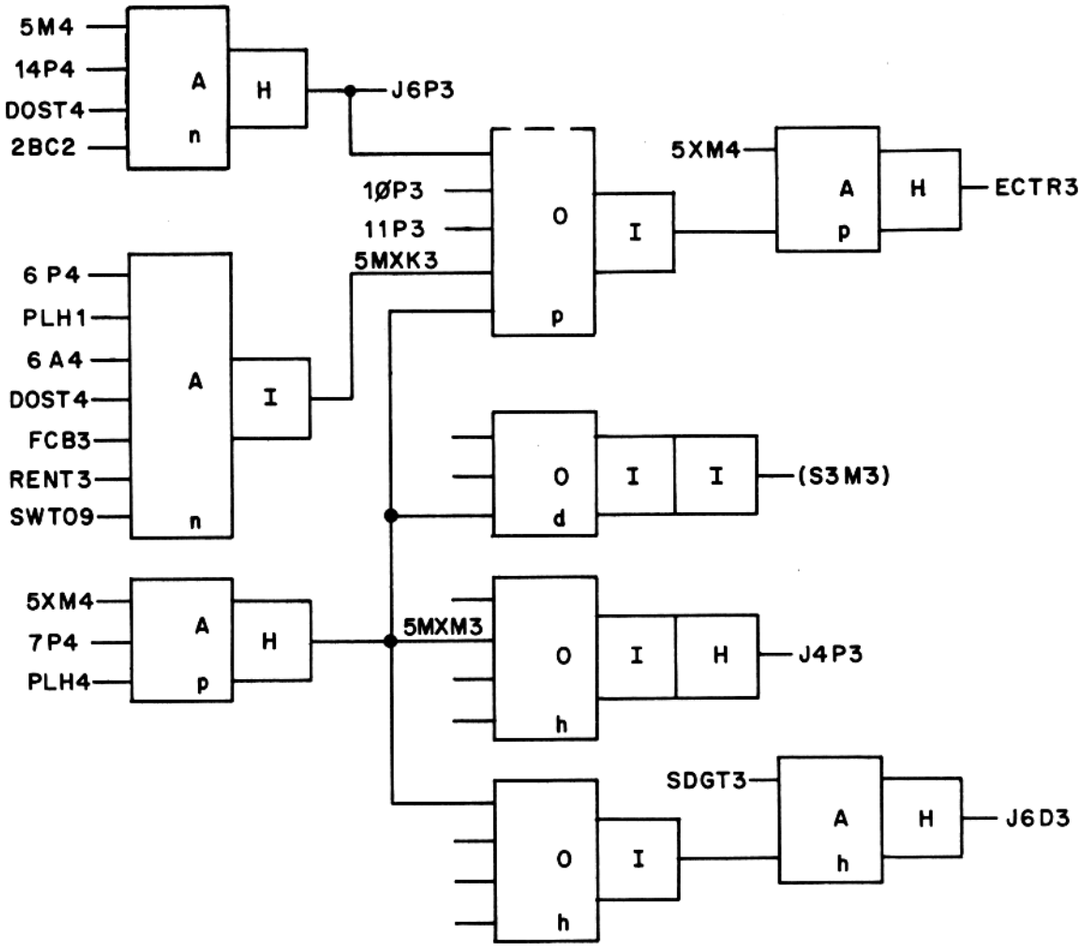
	BEFORE															AFTER													
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	6	6	6	6	6	6	6	6	6	6	6	6	6	-	R4		6	6	6	6	6	6	6	6	6	6	6	6	6	-
	5	5	5	5	5	5	5	5	5	5	5	5	5		R3		5	5	5	5	5	5	5	5	5	5	5	5	5	
	4	4	4	4	4	4	4	4	4	4	4	4	4	-	R2		4	4	4	4	4	4	4	4	4	4	4	4	4	-
	2	2	2	3	3	X								-	R1		3	3	3	3	3	3	3	3	2	2	2	2	2	-
	3	3	3	3	3	3	3	3	2	2	2	2	2	-	RØ	2	2	2	3	3	3	X							-	

161-33

TRANSPOSING R0 AND R1

FIGURE 4-146

(3) Final Phases. After one line pass in action step 5, the action step counter returns to step 0. The next arithmetic sync pulse sets DOST, and this enables a jump to 6 phase, as illustrated in Figure 4-147. The program line logic steps to 7 phase at the next T6B, and then jumps to 3M6,4 at the next program line home, where marker search begins. This is also illustrated in Figure 4-147.

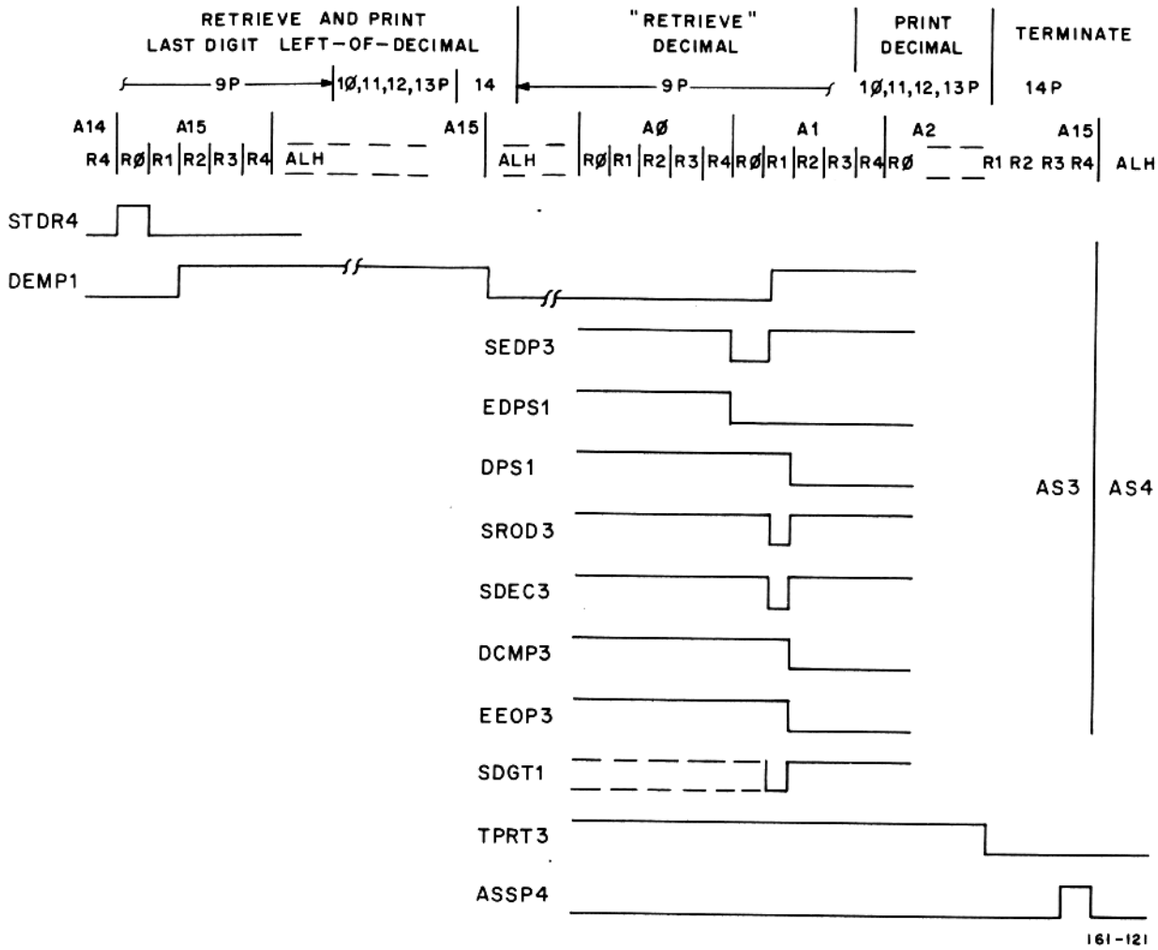


FINAL PHASES IN PRINT

FIGURE 4-147

b. PROCEDURE WITH "0-RIGHT". When the right-of-decimal setting specified is zero, the timing involved in terminating Decimal Print and Non Decimal Print is slightly different. In either case, DCMP3 is produced as soon as SROD3 sets the decimal register to 5 and DPS sets. Note that in this case DCMP3 is developed at a different time (A1R2) than it is normally (A15R1).

(1) Decimal Print. If a Decimal Print instruction specifies "0-to-the-right", the decimal takes the place of the last digit to be printed and the termination of the instruction is much the same as when digits are printed to the right-of-decimal. Refer to Figure 4-148, which illustrates the termination of a Decimal Print instruction with "0 right", and note the similarity between this and the "standard" termination shown in Figure 4-145. The main difference lies in the relationship of the development of DCMP3 to the arithmetic line. Both methods follow basically the same procedure after "retrieving" the last "digit".



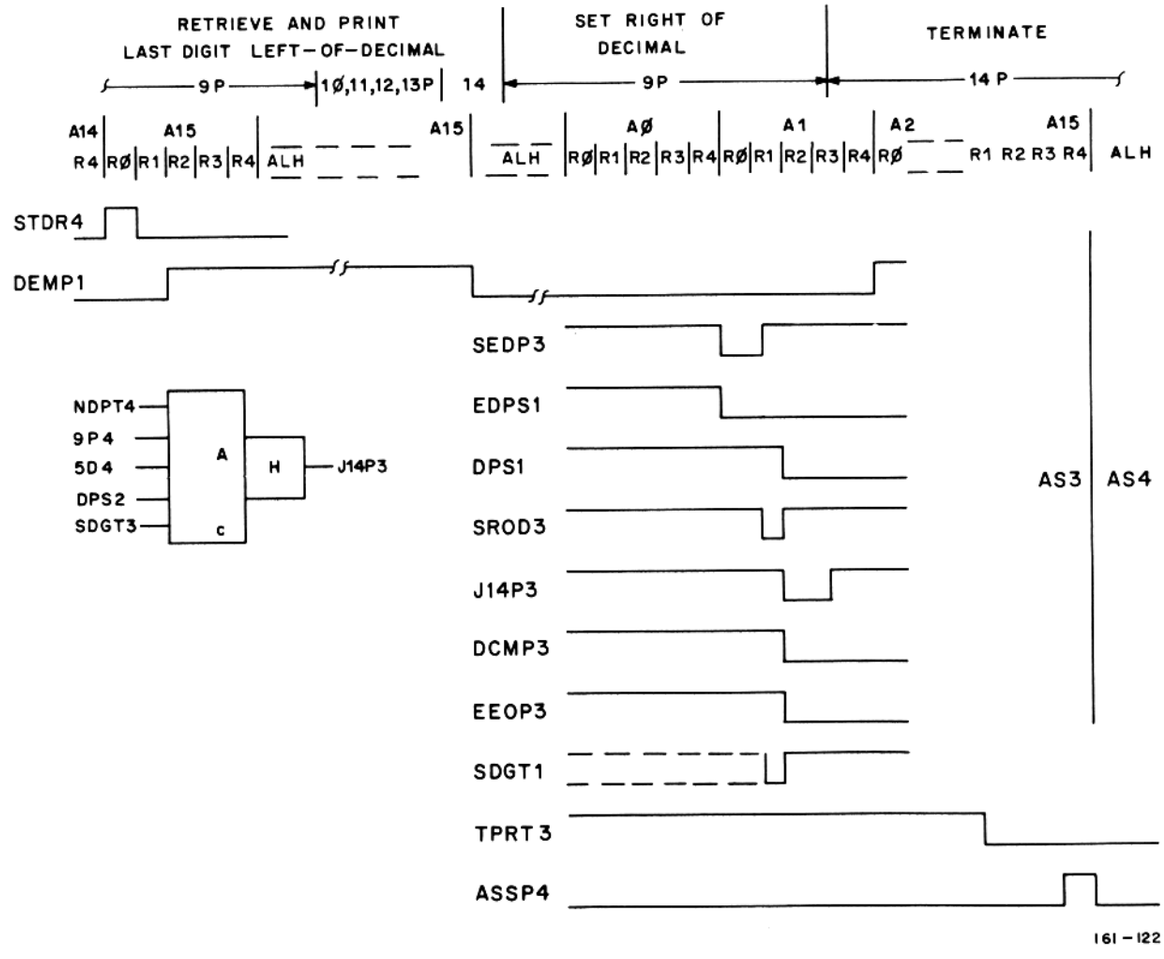
TERMINATING DPR WITH 0 RIGHT OF DECIMAL

FIGURE 4-148

(2) Non Decimal Print. After the decimal register has been set to 5 and DCMP3 has been produced, the phase counter must get to 14 phase before TPRT3 can be developed. During the Decimal Print instruction, the logic uses the stepping through phases 10, 11, 12, and 13 to print the decimal. In Non Decimal Print, however, another translator cycle is not wanted, so a special gate provides a more direct jump to 14 phase. Figure 4-149 illustrates termination of a Non Decimal Print instruction with no digits printed right-of-decimal. Compare this with Figures 4-148 and 4-145. Note that SDGT3 blocks the development of J14P3 if the instruction calls for digits to be printed right-of-decimal.

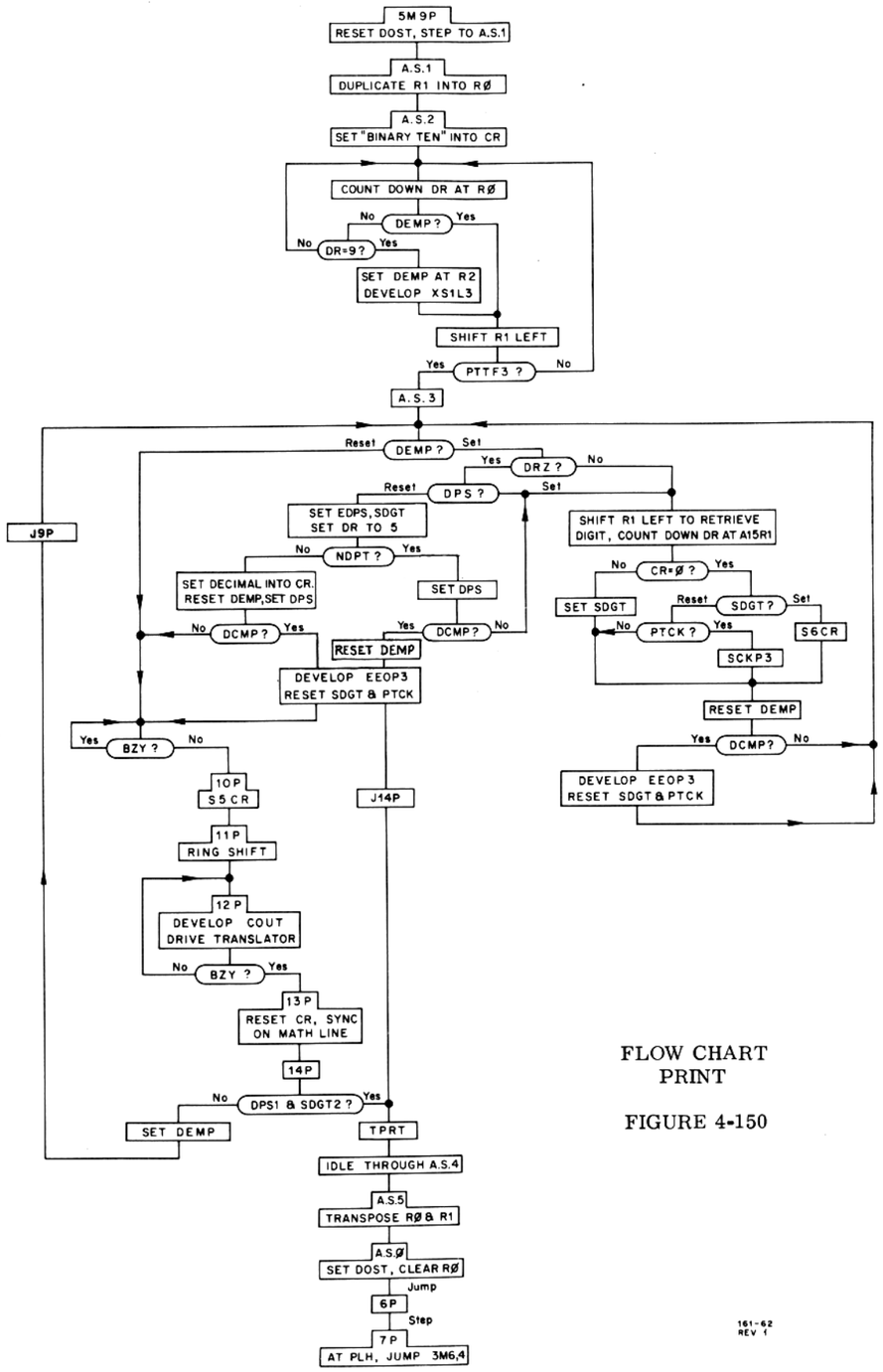
9. FLOW CHART

The Flow Chart covering the operation of both print instructions is presented in Figure 4-150.



TERMINATING NDP WITH 0 RIGHT OF DECIMAL

FIGURE 4-149



FLOW CHART
PRINT

FIGURE 4-150

161-62
REV 1

I. CONSTANTS

Execution of the Constant instruction transfers the address of the Constant command from the program line to R1. Any number of digits up to the maximum LRD pattern may be transferred. If right-of-decimal digits are involved, the decimal must also appear in the address in order to provide the correct decimal alignment.

The Constant instruction is made up of three parts: the Constant command, the constant factor digits, and a code to terminate the instruction. The study of the Constant instruction must, therefore, be divided into three distinct areas. First of these is the retrieval of the Constant command and the setting up of the logic for retrieval of the digits. Second is the actual retrieval of the digits from the program line and the transfer of these digits to the arithmetic line, including retrieval of the decimal. Last is the termination of the instruction, including decimal alignment.

1. GENERAL OPERATION

When the Constant command is retrieved, it is treated the same as any other 0-address command. However, after the jump to 5 mode occurs, the 5 mode logic sets the Significant DiGiT flip-flop, SDGT. Then marker search resumes in order to retrieve the first digit of the constant factor. Each time a digit is retrieved, the marker moves down the line one character position. Each digit is shifted into the common register in 4 mode and transferred to the arithmetic logic in 5 mode. This operation continues until a space or a dash is retrieved. Either of these will reset SDGT in 5 mode, thus allowing regular 4 mode operation to resume.

One digit is taken off the program line for each two line passes: the digit is retrieved during the first line pass and the second line pass takes place while the digit is being shifted onto the arithmetic line.

a. CONSTANT COMMAND. When the Constant command (letter "J", 1-5-7 code) is retrieved in 4 mode 3 phase, it causes a jump to 5 phase, where the marker is written out immediately behind the "J", and the balance of the program is written "long" (through the search register) until program line home is reached. At program line home a jump to 5 mode 15 phase occurs. In 5 mode the "J" in the operation register causes SDGT to set. Then a jump to 3M6,4 occurs and the search for the first constant factor digit begins.

b. CONSTANT FACTOR DIGITS. Retrieval of constant factor digits is one of the two instances requiring the PLCR line routing in 4 mode 3 phase. Each digit is retrieved in 4 mode 3 phase, and because SDGT is set, a jump to 13 phase occurs. In 13 phase the marker is written out behind the digit just retrieved. At program line home a jump to 5 mode 15 phase occurs, and the phase counter makes its usual jump to 6 phase after DOST resets. Then the digit just retrieved is shifted out of the common register onto the arithmetic line. When DOST again sets, the phase counter steps to 7 phase, and at program line home a marker search is initiated in order to locate the next digit.

When the decimal code is retrieved, the ensuing 5 mode operation sets the decimal register to 5 so that the constant factor can be correctly aligned when it is terminated.

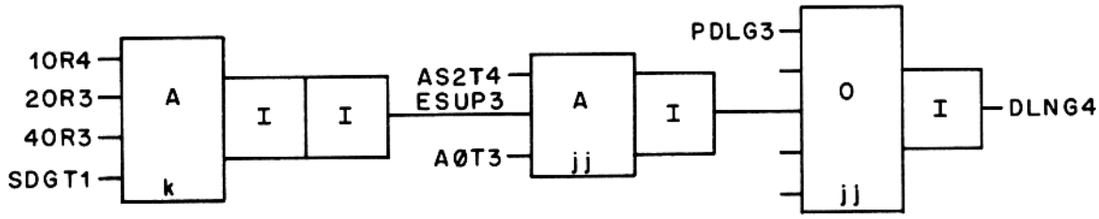
c. TERMINATING CONSTANTS. Either a space (blank) or a dash (minus sign) on the program line is required to terminate the constant factor. When one of these codes is retrieved, it is treated the same as a constant digit in 4 mode. In 5 mode, however, it terminates the constant factor by resetting SDGT, it decimally aligns the new factor, and it records the sign of the factor in A1. When all of these functions are complete, a new marker search is initiated for the next instruction.

2. CONSTANT COMMAND

Constant command retrieval in 4 mode is the same as retrieval of the other "short" arithmetic instructions (DUP, DEL, CSI, etc.). In 4 mode 3 phase the "J" is shifted into the operation register and the common register. Then a jump to 5 phase occurs, where the PLSR, SRPL

line routing sends the marker out immediately behind the "J", and the balance of the program follows. At program line home a jump to 5 mode 15 phase occurs and the phase counter waits for DOST to reset. This occurs at the first A1 time in 5 mode, and the jump to 6 phase is enabled. As in the other arithmetic instructions, the jump to 6 phase cannot occur until after the program line timing chain restarts at the end of program line home. The phase counter waits in 6 phase for the action step counter to complete its cycle and set DOST.

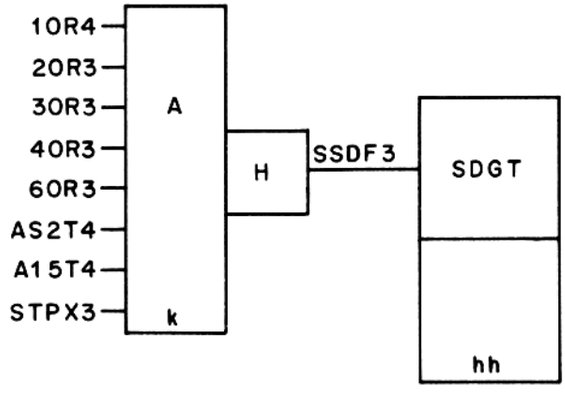
When DOST becomes reset in 5 mode 15 phase, the usual "dummy" arithmetic line pass follows. At arithmetic line home the action step counter goes to 1 and "idles" through the next line pass. When the action step counter increments to step 2, the operating stack is shifted up. Refer to Figure 4-151. The "J" in the operation register produces Enable Shift UP, ESUP3, which is ANDed with action step 2 to produce data long. A0T3 blocks data long so the sync pulse can be written "short". With the sync pulse written short and all other data written long, all registers are shifted up after one line pass. (R4 data is shifted into the R0 position temporarily: R0 will be cleared as soon as DOST is again set.)



SHIFT UP IN AS2

FIGURE 4-151

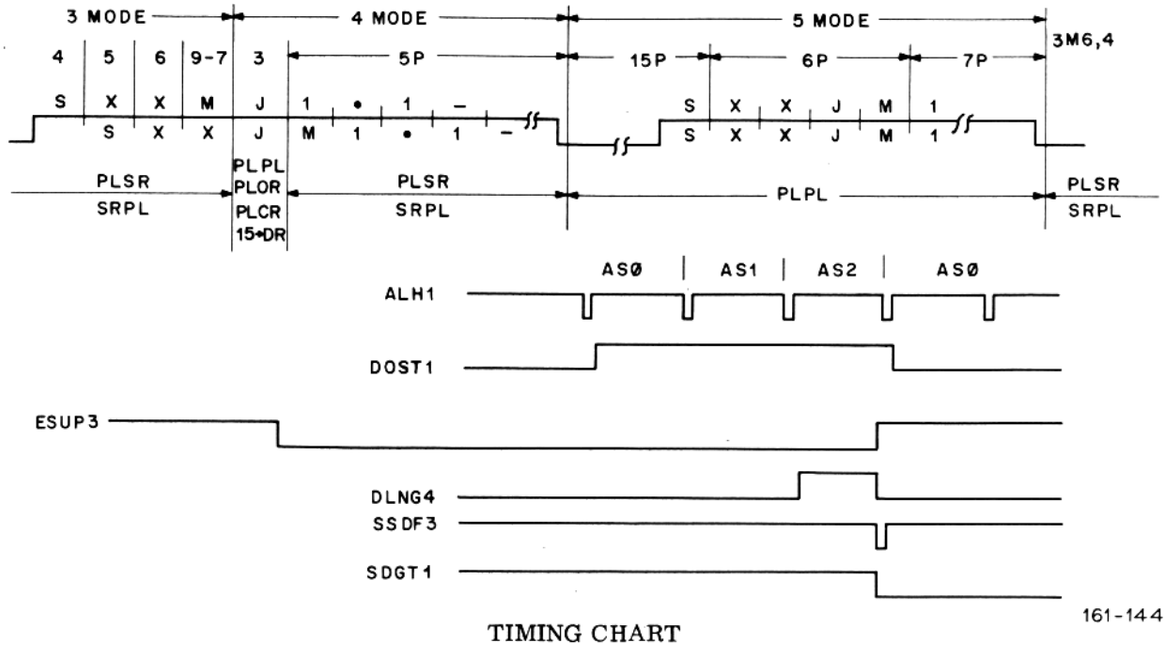
At A15 time in action step 2, the Set Significant Digit Flip-flop signal, SSDF3, is produced as shown in Figure 4-152. SDGT being set controls subsequent 4 mode operation, as will be seen shortly.



DEVELOPMENT OF SSDF3 IN "CONSTANTS"

FIGURE 4-152

At the next arithmetic line home, the action step counter returns to 0 and data long is disabled. The next arithmetic sync pulse sets DOST to enable the phase counter to step to 7 phase, where it remains until program line home. Then a jump to 3 mode 4 phase occurs and the search is on for the first digit of the constant factor. (SDGT being set produces 6D3 without affecting the decimal register. This will be covered shortly.) The retrieval of the "J" is summarized in the timing chart shown in Figure 4-153.



TIMING CHART
RETRIEVAL OF "J"

FIGURE 4-153

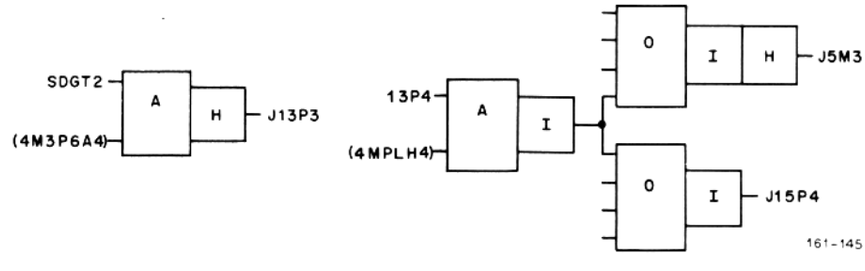
When DOST sets after the action step counter returns to 0, CLR03 clears the previous R4 data out of R0, where it was left as a result of the shift up operation.

Thus, after the Constant command is retrieved and before the first constant factor digit is retrieved, registers R0 and R1 are both clear. The following operation will shift the constant factor into R1 one digit at a time.

3. CONSTANT FACTOR DIGITS

With SDGT set, 4 mode operation is the same, regardless of what code is retrieved. The operation in 5 mode varies depending upon factors such as decimal retrieval, etc.

a. 4 MODE OPERATION. All codes retrieved from the time the "J" has set the SDGT flip-flop until the constant factor is terminated follow the same route in 4 mode: they are retrieved in 3 phase, and SDGT causes a jump to 13 phase. Here the marker is written out behind the code just retrieved, and the balance of the program follows. At program line home, a jump to 5 mode 15 phase occurs. The associated gates are illustrated in Figure 4-154.



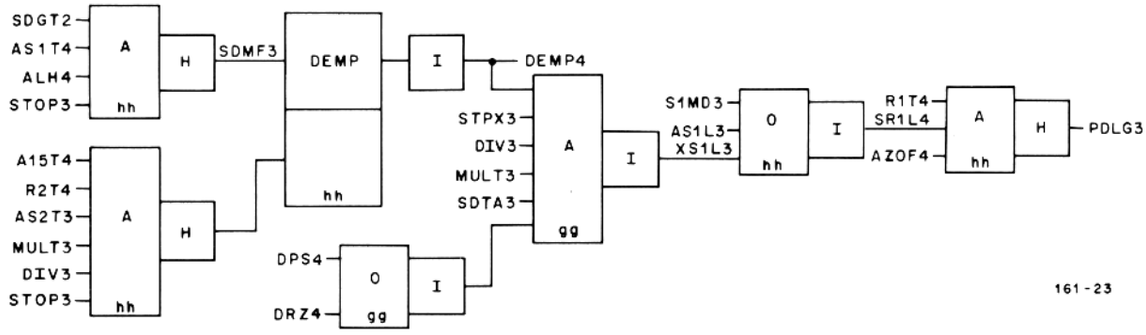
GATES USED IN 4 MODE WITH SDGT SET
(All on "H" card)

FIGURE 4-154

Each code that is shifted into the operation register can develop different "Op Decoder" outputs. These are ignored in 4 mode, however, because SDGT is set. ALL codes follow the same pattern, regardless of the outputs that are developed.

b. DIGITS LEFT-OF-DECIMAL. Each digit of the constant, when it reaches 5 mode, resets DOST and allows the action step counter to make one "short" cycle. In the action step 1 that follows, the code in the common register (the code just retrieved) is shifted onto the arithmetic line into the A2R1 position, and any digits retrieved previously are shifted left.

Refer to Figure 4-155. At the arithmetic line home following the reset of DOST, the action step counter steps to 1, and because SDGT is set, this causes the Set DeMp Flip-flop signal, SDMF3, to become true. This sets the Divide, Enter, Multiply and Print flip-flop, DEMP. As DEMP4 becomes false, it allows eXtra Shift R1 Left, XS1L3, to become true, which drives SR1L4 false. SR1L4 is ANDed with AZOF3 and R1T4 to develop PDLG3 during each R1 time, A2 through A14. This causes the first digit of the constant factor to be shifted out of the common register into A2R1.



SHIFT LEFT UPON ENTRY

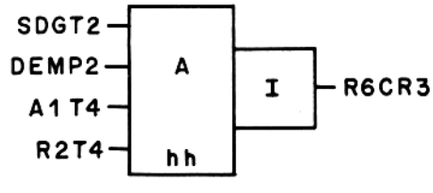
FIGURE 4-155

At the end of the line pass, when the shift left is complete, a signal is developed to reset DEMP at A15R2 time. With DEMP reset, PDLG3 returns to false. The action step counter steps to 2 at arithmetic line home, and the next line pass is completed with no further action occurring. At the end of the second line pass, the action step counter returns to zero, and the next sync pulse sets DOST, enabling the step to 7 phase. The phase counter remains in 7 phase until program line home, when a jump to 3 mode 4 phase occurs and the search begins for the next digit of the constant factor.

Even though it is not necessary at this time, the decimal register is counted down once at the completion of the shift left. This will be explained later. Again, because SDGT is set, 6D3 remains true.

As subsequent digits left-of-decimal are retrieved, they are shifted into the A2R1 time slot on the arithmetic line, and the previous digits are shifted left. For example, if the digits 1, 2, 3 were retrieved in that order, the "1" would be in A4R1, the "2" would be in A3R1, and the "3" would be in A2R1.

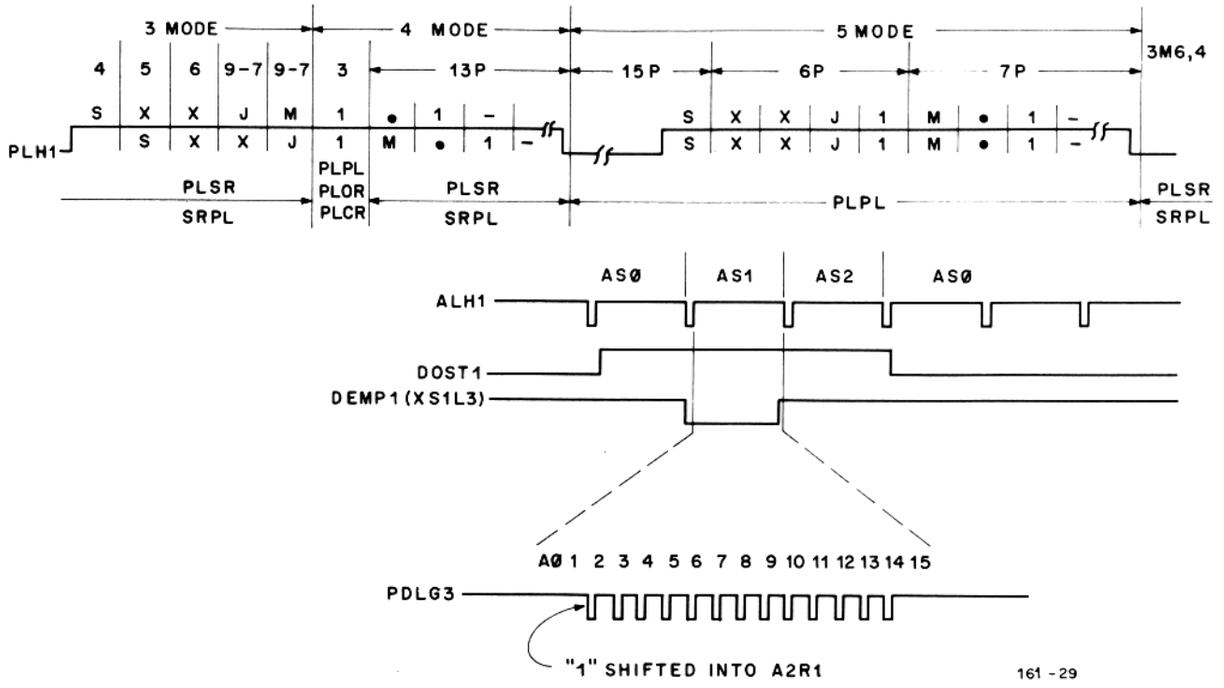
If a zero (6 bit) is retrieved from the program line, it is converted to an arithmetic line zero (blank) by resetting 6CR before the shift left even begins. This is illustrated in Figure 4-156.



DEVELOPMENT OF R6CR3

FIGURE 4-156

The timing chart in Figure 4-157 illustrates the retrieval and subsequent loading onto the arithmetic line of a single constant factor digit. Operation with subsequent digits is identical.



RETRIEVAL OF 1st CONSTANT FACTOR DIGIT

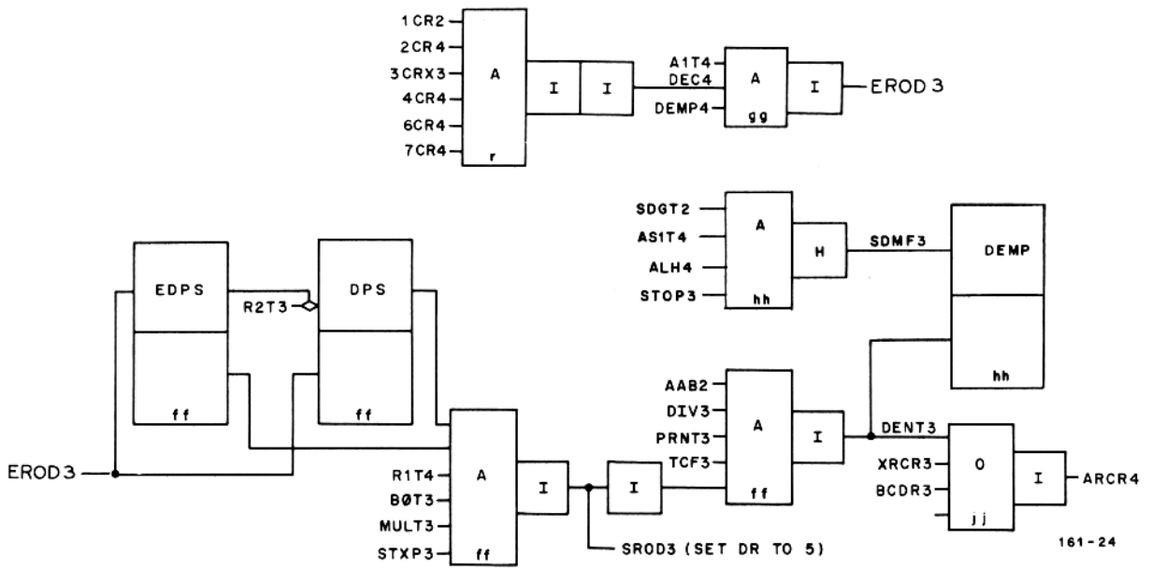
FIGURE 4-157

c. DECIMAL POINT. The retrieval of the decimal point in 4 mode is the same as the retrieval of constant factor digits. When the logic gets to 5 mode, however, the decimal code causes conditions to be set up for the eventual decimal alignment of the constant factor.

Refer to Figure 4-158. When the decimal code is shifted into the common register in 4 mode 3 phase, DEC4 immediately goes false. When the logic jumps to 5 mode, SDMF3 is produced as for other constant factor digits, and DEMP sets (at arithmetic line home). At the next A1 time, Enable Right Of Decimal, EROD3, is produced, which sets the Enable Decimal Point Storage flip-flop, EDPS, and ensures that the Decimal Point Storage flip-flop, DPS, is reset. At the end of B0 time, Set Right Of Decimal, SROD3, is produced, which immediately sets the decimal register to 5. At bit 2 time AAB2 goes false and Decimal ENTry, DENT3, is developed, which resets DEMP to prevent a shift left, and drives Arithmetic Reset Common Register, ARCR4, false. This latter action resets the decimal code out of the common register.

At the beginning of R2 time, DPS sets, thus preventing any further pulses on SROD3 and DENT3.

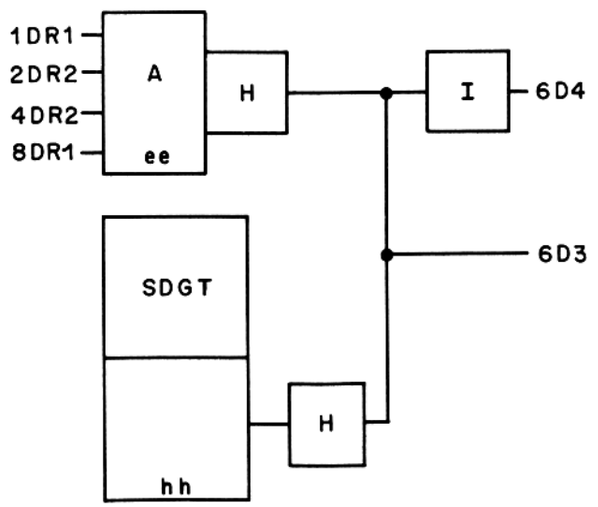
The arithmetic line idles through the balance of action step 1 and through action step 2. It then returns to step 0 and the following sync pulse sets DOST, allowing the phase counter to step to 7 phase. At program line home a jump to 3 mode 4 phase occurs and the search for the first digit right-of-decimal begins.



DECIMAL ENTRY LOGIC

FIGURE 4-158

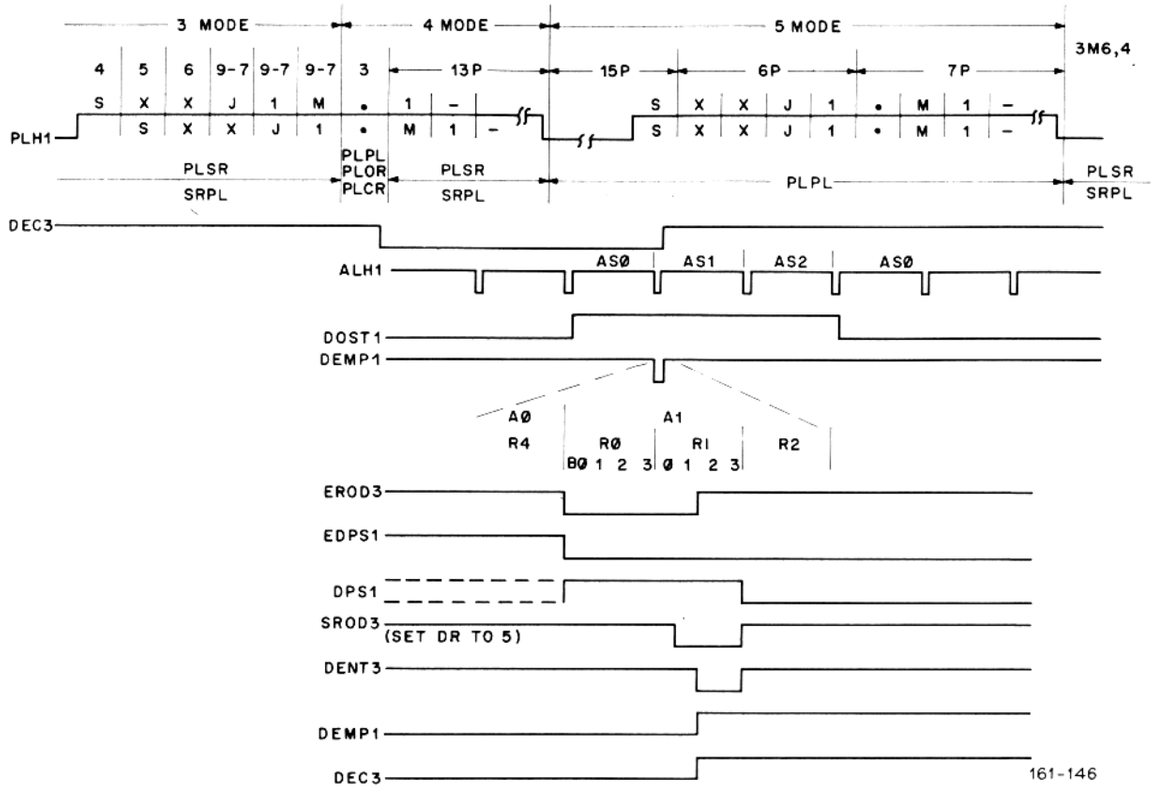
At this point the decimal register is set to 5, but 6D3 is required in order to carry out the marker search. In this case, 6D3 is provided by inverting SDGT2, as shown in Figure 4-159. This makes 6D3 available to the program line logic whenever SDGT is set, and still allows the decimal register to be used for arithmetic purposes. Although 6D3 becomes true every time SDGT sets, it is necessary to produce it in this manner only during Constants: it has no function during the other instructions that use SDGT.



"DUMMYING IN" 6D3

FIGURE 4-159

The timing chart in Figure 4-160 outlines the retrieval of the decimal code, showing the time relationships of the more important signals.



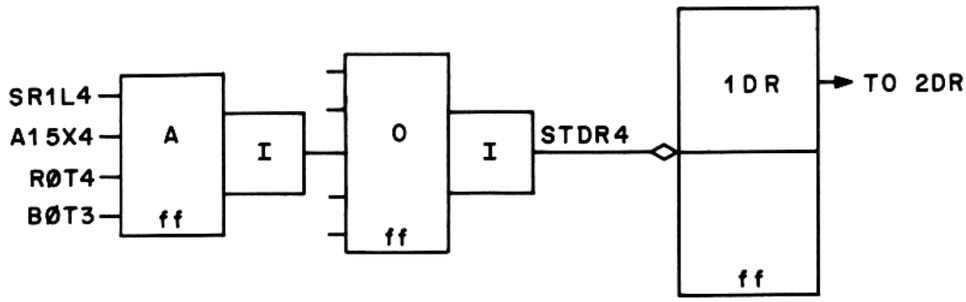
RETRIEVAL OF DECIMAL DURING CONSTANTS

FIGURE 4-160

d. DIGITS RIGHT-OF-DECIMAL. Retrieval of the right-of-decimal constant factor digits is the same as the retrieval of the other digits. The operation in 5 mode is also the same, except that the shift left of R1 is disabled after five digits have been retrieved right-of-decimal.

The number of right-of-decimal digits retrieved is counted in the decimal register. This register is set to 5 when the decimal code is retrieved, and each digit following the decimal causes the decimal register to be counted down once. When the decimal register reaches 0, further shifts left are prevented.

Figure 4-161 illustrates the development of the STep Decimal Register signal, STDR4. (As noted earlier, the decimal register is also counted down with each digit left-of-decimal, even though there is no reason for it at the time.) Since DPS was set by retrieval of the decimal code, XS1L3 will be blocked if the decimal register is counted down to 0, and further shifts left will thus be prevented. (The blocking action of DRZ4 and DPS4 was shown previously in Figure 4-155.) Therefore, if more than five digits right-of-decimal are retrieved, only the first five will be accepted.



COUNTING DOWN DR DURING CONSTANTS

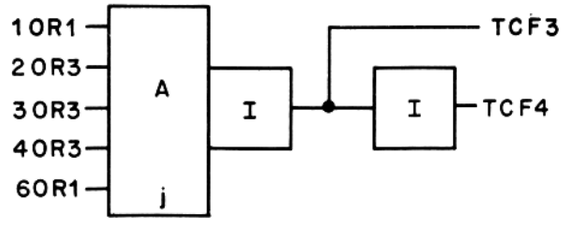
FIGURE 4-161

4. TERMINATING THE CONSTANT FACTOR

It is necessary to have either a space (blank) or a dash (minus sign) loaded on the program line in order to reset SDGT and terminate a constant factor. Theoretically, lack of a terminating code would cause the logic to retrieve each code, in turn, until a space or a dash was located. However, since all of the "Op Decoder" outputs are not disabled by SDGT, unusual combinations are in fact produced which usually cause the system to fall into an endless search. Therefore, a space or a dash is a MUST for terminating a constant factor.

Terminating a constant factor also involves decimal alignment, the operation of which varies depending upon whether or not a decimal code has been retrieved previously. In addition, a dash used to terminate the factor causes a sign change.

a. RETRIEVAL OF THE SPACE OR DASH. The 4 mode retrieval of the space or dash is the same as retrieval of digits and the decimal, as long as SDGT is set. As soon as one of these codes is shifted into the operation register, the Terminate Constant Factor signal, TCF, is produced, as shown in Figure 4-162.

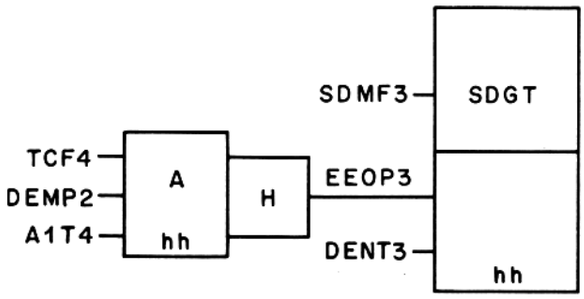


DEVELOPMENT OF TCF

FIGURE 4-162

b. ACTION STEP 1. SDGT becomes reset in action step 1, effectively terminating the constant in respect to subsequent codes retrieved.

In 5 mode DOST becomes reset at the first A1 time, enabling the jump to 6 phase (which occurs after the program line timing chain restarts). At the next arithmetic line home, the action step counter goes to 1 and DEMP again sets as before. In the ensuing A1 time, the End of Entry or OutPut signal, EEOP3, is produced, and SDGT resets. See Figure 4-163. At the next A2 time, DEMP resets. See Figure 4-167.



DEVELOPMENT OF EEOP3 DURING CONSTANTS

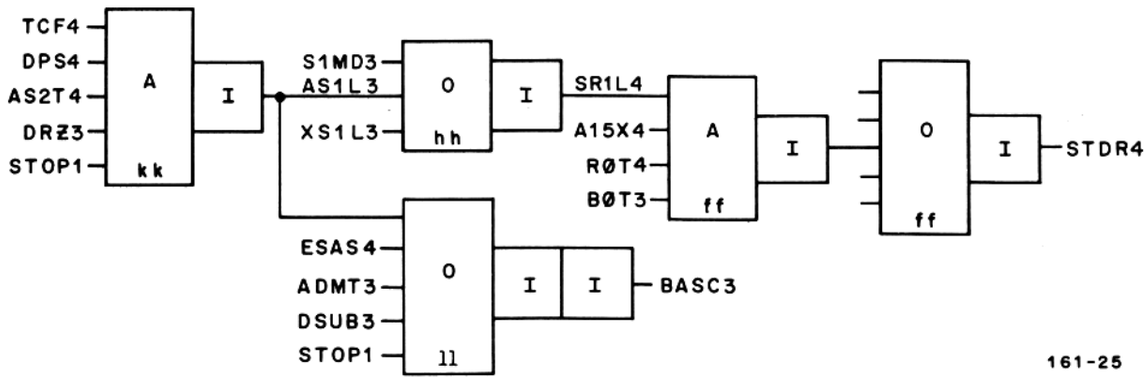
FIGURE 4-163

TCF holds PDLG3 true throughout action step 1. This is necessary in case the dash was used as the terminating code. In this case, CSGN3 is produced in A1 time, and VRTA3 is developed in A1R1B0 to set a sign bit into the common register. This procedure is identical to that followed in performing the Change Sign instruction: refer to Section 4F for review.

The system idles through the balance of action step 1, and at the end of the line pass the counter steps to 2.

c. ACTION STEP 2. The action step counter remains in step 2 until the constant factor has been properly aligned around the decimal. This may require from one to five arithmetic line passes.

Because DEMP is reset in action step 1, XS1L3 remains false, preventing any further shifts left from this source. In action step 2, however, Align Shift 1 Left, AS1L3, is produced as shown in Figure 4-164. AS1L3 drives SR1L4 false, producing PDLG3 at each R1 time. This shifts R1 to the left one character position for each line pass in action step 2. AS1L3 holds BASC3 true so the action step counter remains at 2 until the decimal register is counted down to 0, driving AS1L3 false.

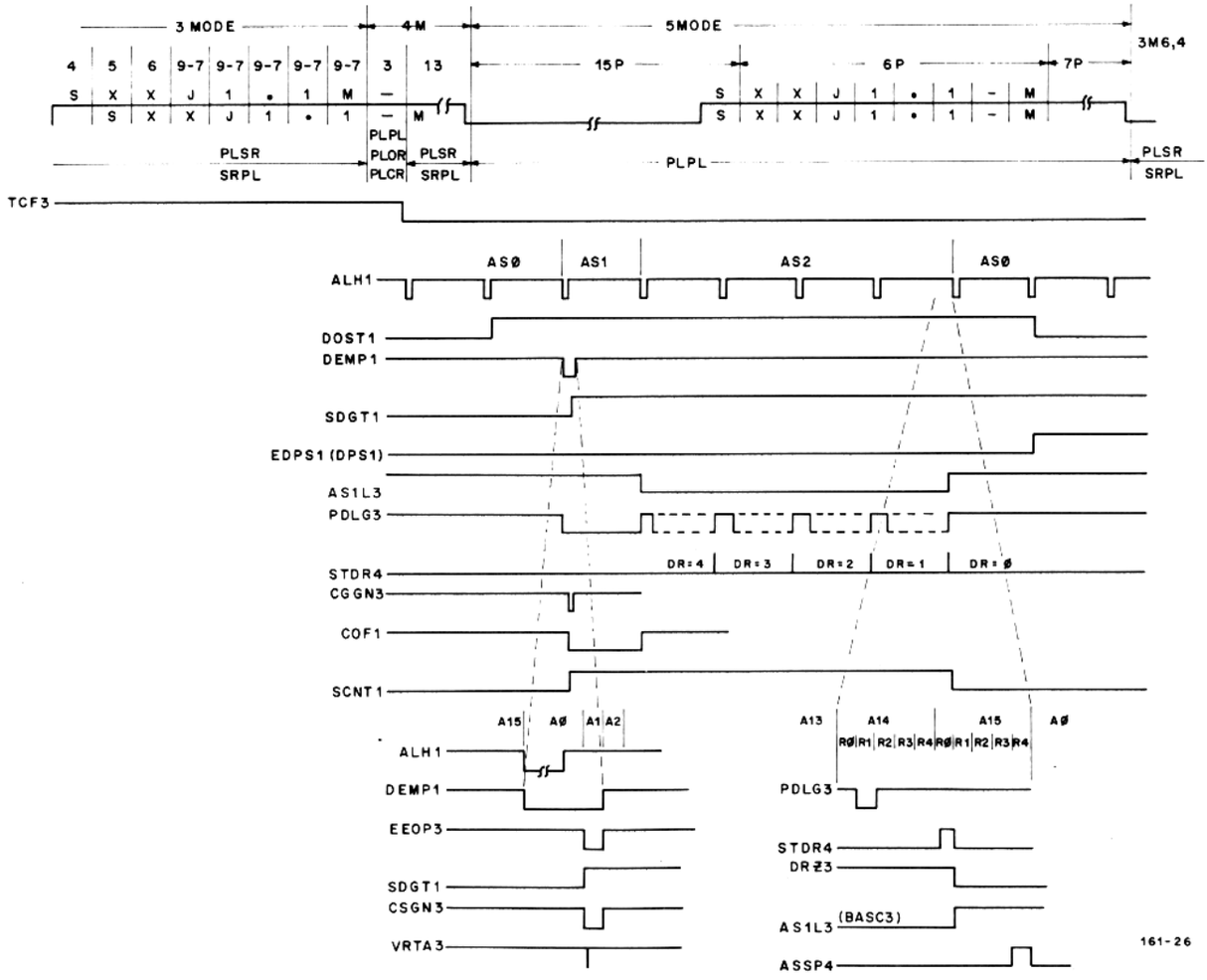


DECIMAL ALIGNMENT DURING TCF

FIGURE 4-164

This has the effect of aligning the factor correctly around the decimal. For example, if one digit had been retrieved right-of-decimal and the next code terminated the constant, four more shifts left would be required for proper decimal alignment. The one digit already retrieved would have counted the decimal register down from 5 to 4. Each of the line passes in action step 2 counts the decimal register down once more, resulting in the DRZ condition after four line passes, or four shifts left.

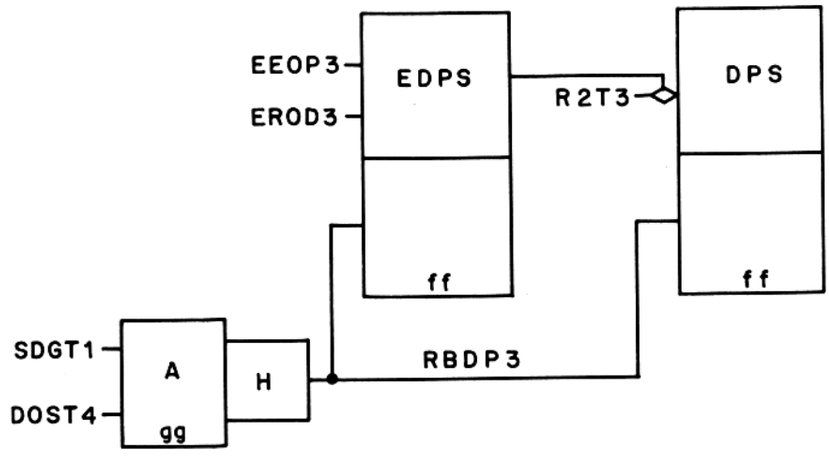
The preceding example is illustrated in the timing chart in Figure 4-165. This chart also includes the more important signals involved in changing the sign when a dash is used as the terminating code.



TIMING CHART
TERMINATE CONSTANT FACTOR

FIGURE 4-165

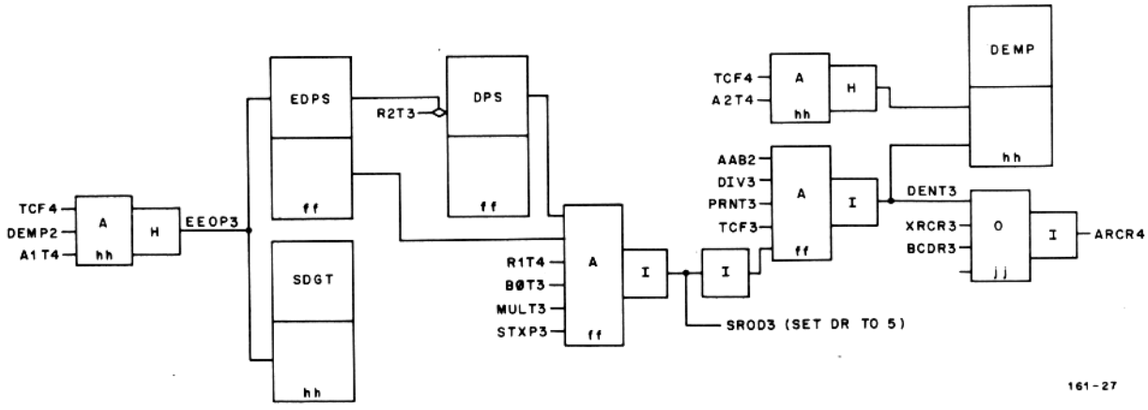
When the decimal register is counted to 0 and the action step counter returns to 0, the next sync pulse sets DOST. At this point, Reset Both Decimal Point flip-flops, RBDP3, is developed (see Figure 4-166), which resets both EDPS and DPS. This is done in order to leave them reset for correct operation of subsequent instructions.



DEVELOPMENT OF RBDP3

FIGURE 4-166

d. TCF WITHOUT DECIMAL. Termination of a Constant that does not contain a decimal results in five shifts left to align the factor as a whole number. In this case, the terminating code also performs the decimal function of setting the decimal register to 5. EEOP3 is produced by TCF and DEMP the same as described previously. This time, however, EEOP3 sets EDPS, and at A1R1, SROD3 is developed to set the decimal register to 5. Then the action step counter goes on to step 2, where five line passes take place and the factor is shifted five columns to the left. See Figure 4-167.



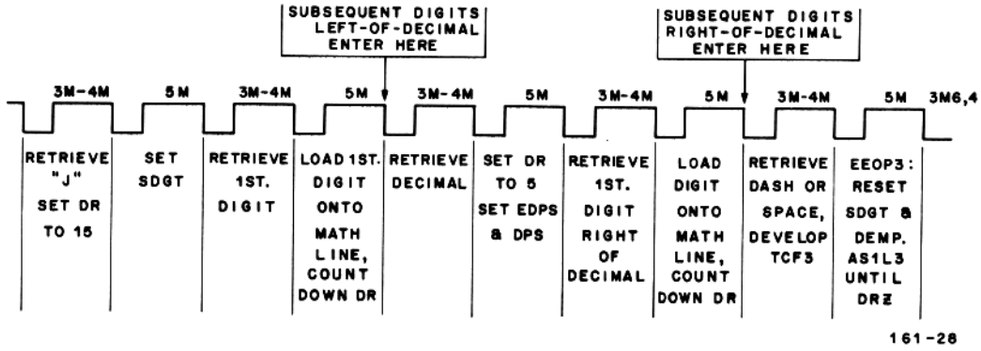
TCF WITHOUT DECIMAL

FIGURE 4-167

e. CONSTANT "0". If the constant is terminated before ANY digits are retrieved, a zero is shifted into R1. In this case, R1 will be completely clear upon termination of the constant. Note that the terminating code still causes five shifts left, even though this function has no purpose at this time.

5. SUMMARY

Figure 4-168 summarizes the entire Constant operation by listing the more important occurrences during each of the program line passes.



SUMMARY OF CONSTANT OPERATION IN RELATION TO PROGRAM LINE

FIGURE 4-168

J. BASIC MATHEMATICAL CONCEPTS

The 5610 is capable of performing four computational operations: addition, subtraction, multiplication, and division. The multiply and divide functions are performed when the Multiply and Divide instructions are retrieved. However, due to the occurrence of signed numbers, an Add instruction may actually cause a subtract operation to occur, and vice-versa.

Add, Subtract, Multiply, and Divide are all 0-address commands: whenever one of these commands is retrieved, the address is implied to be R1 and R2. Note, however, that the first step in each of these commands is a shift down, and that the actual computation occurs between R0 and R1.

An add function is always completed in only one arithmetic line pass. However, the logic leading up to this line pass requires more time. In other words, an Add is similar to a Truncate or a Change sign instruction in that the actual arithmetic operation occurs in one arithmetic line pass, but two PROGRAM line passes are required for the complete operation, from retrieval of one command to retrieval of the next.

A subtract also takes one line pass, unless it results in an overdraft condition. In this case, another line pass is necessary to complement the overdraft and change the sign of the answer. (A second line pass also occurs after an add overflow, but this line pass has no effect on the answer.)

Multiplication and division both take many line passes, since they are actually repetitive addition and subtraction, respectively. The details of arithmetic line operation in adding and subtracting will be covered first. Then, by the time multiplication and division are studied, only the logic that ties these individual additions or subtractions together will be discussed: knowledge of what occurs during these individual add/subtract line passes will be assumed.

1. BINARY ARITHMETIC

In order to understand and service the arithmetic logic in the 5610, the service technician must possess knowledge of certain fundamental binary mathematic principles and processes. This section contains basic information on (a) binary addition and subtraction, and (b) the Binary Coded Decimal system.

a. BINARY ADDITION. Binary addition is similar to decimal addition in that when a column is "full" it returns to zero and a carry is made to the next higher order column. Figure 4-169 illustrates two decimal and two binary addition problems. In Figure 4-169 (a) there is no carry because neither column limit has been reached. In Figure 4-169 (b) the column limits are reached, so a 1 is carried to the next column to the left.

(a) No carry

Decimal	Binary
2	0
+3	+1
5	1

(b) Carry

Decimal	Binary
5	1
+5	+1
(1) 0	(1) 0

FIGURE 4-169

The mechanics of binary addition are most easily explained by merely stating the rules to be followed and giving examples of each. These rules can be applied to any combinations of binary numbers to arrive at the correct sum. See Figure 4-170.

1.	0 + 0 = 0 with no carry.	1000
2.	0 + 1 = 1 with no carry.	0110
3.	1 + 0 = 1 with no carry.	1110
		1001
4.	1 + 1 = 0 with a carry of 1.	1001
		(1) 0010
5.	0 + 0 + a carry = 1 with no carry.	0101
6.	0 + 1 + a carry = 0 with a carry of 1.	0011
7.	1 + 0 + a carry = 0 with a carry of 1.	1000
		1011
8.	1 + 1 + a carry = 1 with a carry of 1.	0011
		1110

RULES FOR BINARY ADDITION

FIGURE 4-170

The same rules can also be used to add three or more binary numbers, but the carries become so involved that it is usually wiser to add only two binary numbers at a time. For example, the last rule can be stated in a number of ways:

1 + 1 + 1 =
(1 + 1) + 1 =
(0 + a carry) + 1 = 1 + a carry of 1

In this case, two of the three binary numbers are first added, and then the third number is added to this sum.

Note also that since a carry is always equal to 1, rules 5, 6, and 7 could be combined into rules 2, 3, and 4.

b. BINARY SUBTRACTION. Binary subtraction is quite similar to decimal subtraction, just as binary addition is similar to decimal addition. The first three rules of binary subtraction are self-evident. They are as follows:

1.	0 - 0 = 0 with no borrow.	0110
2.	1 - 1 = 0 with no borrow.	-0010
3.	1 - 0 = 1 with no borrow.	0100

Notice that these rules are also true for decimal subtraction. If a list of rules for decimal subtraction were to be written, it would start with these three rules, and then continue with rules governing the use of the digits 2 through 9.

The fourth rule of binary subtraction is sometimes a little more difficult to understand. It was noted in addition that whenever a column is "full" it is necessary to perform a carry to the next column. In subtraction, therefore, an "empty" column requires a "borrow" from the next higher order column.

Consider the decimal problem shown in Figure 4-171. Since 1 is subtracted from zero, the right hand column is "empty", and requires a borrow from the left column. Since each 1 in a column represents ten in the column to its right, ten is borrowed from the left column and added to the right column, as shown in Figure 4-171 (b).

(a)
30
- 1
29

(b)
2 10
-0 1
2 9

FIGURE 4-171

Figure 4-172 is another example illustrating the same idea: ten is borrowed from one column and added to the next-right column to allow the subtraction to take place.

(a)
```
 1752
- 835
  917
```

(b)
```
 0 17 4 12
-  8 3  5
 0  9 1  7
```

FIGURE 4-172

In binary subtraction the same rule is followed, except that because the value of each higher order digit increases by a power of two in the binary system, two is borrowed instead of ten. This is illustrated in Figure 4-173.

	Before Borrow	After Borrow
(a)	1 0	0 2
	-0 1	-0 1
	? ?	0 1
(b)	0 1 0 0	0 0 2 0
	-0 0 1 0	-0 0 1 0
	? ? ? ?	0 0 1 0

FIGURE 4-173

With the above in mind, the fourth rule of binary subtraction may be stated:

4. 0 - 1 = 1 with a borrow of 1.

However, there is a little more involved in this rule than first meets the eye. Consider the decimal problem shown in Figure 4-174 (a). A ten must be borrowed to allow the subtraction in the LSD column, but since the middle column is zero, the borrow must be made from the MSD column. This is shown in Figure 4-174 (b). Since this will still not allow the subtraction in the LSD column, 10 must be borrowed from the middle column and added to the LSD column. This is shown in Figure 4-174 (c).

(a)
```
 200
-  5
 195
```

(b)
```
 1 10 0
-     5
```

(c)
```
 1 9 10
-0 0  5
 1 9  5
```

FIGURE 4-174

Figure 4-175 illustrates a similar problem. Note that all columns originally containing zero and from which a borrow was made contain a nine in the final answer (10 - a borrow = 9).

```
 200000
-    50
 199950
```

FIGURE 4-175

Now consider the binary problem shown in Figure 4-176 (a). In order to complete the subtraction in the LSD column a borrow is necessary. However, the borrow cannot be made from the next column to the left because it is already zero. Since it is impossible to borrow from zero, it is necessary to keep going to the left until a 1 is found from which to borrow. This 1 represents two in the column to its right, and this step is shown in Figure 4-176 (b). A 1 in any column is doubled in value when it is moved to the next column to the right.

(a)
```
 0 1 0 0
-0 0 0 1
```

(b)
```
 0 0 2 0
-0 0 0 1
```

(c)
```
 0 0 1 2
-0 0 0 1
 0 0 1 1
```

FIGURE 4-176

This still does not enable the subtraction in the LSD column, but it does allow the LSD column to borrow from the next-left column, as shown in Figure 4-176 (c). The subtraction can now be accomplished and the fifth rule of binary subtraction can be stated:

5. A borrow can only be made from a digit position containing a 1, which then becomes zero.

Figures 4-177 (a & b) illustrate another binary subtraction problem in which zeros must be bypassed in the search for a digit which is able to yield the borrow. Figure 4-177 (c) illustrates a similar decimal system example. Notice that the result is the same as in decimal subtraction: every zero skipped in the search for the borrow becomes 1 less than a "full" column. This last statement can be re-worded to give the sixth rule for binary subtraction:

6. All zeros skipped in the search for the borrow are replaced by 1's.

```
(a)  10000        (b)  01112        (c)  2000
    -00001            -00001            -  4
    -----             -----             ----
     01111             01111            1996
```

FIGURE 4-177

2. BINARY CODED DECIMAL SYSTEM

The 5610 arithmetic logic uses the Binary Coded Decimal (BCD) system in all arithmetic functions. The BCD system can be defined as a system for working with decimal numbers in which each individual decimal digit is coded into, or represented by, its binary equivalent. In other words, the 5610 performs its mathematical functions by using decimal numbers, but as each decimal digit is operated upon, it is in its binary form. As in decimal system mathematics, when two numbers are manipulated, they are operated upon only one digit at a time. All work on one digit must be completed before work on the next digit is begun.

Figure 4-178 helps to illustrate the above. In Figure 4-178 (a) an addition problem is performed using the decimal system. In Figure 4-178 (b) the same problem is performed using regular binary addition. Finally, Figure 4-178 (c) illustrates the same problem using BCD. Each decimal digit is converted into its binary equivalent and the addition takes place one digit at a time.

```
(a)  312        (b)  100111000        (c)  0011   0001   0010
    +211            +011010011            +0010  0001   0001
    ----            ----------            -----  ----   ----
     523            100000101 1           0101   0010   0011
```

FIGURE 4-178

To summarize, it can be said that the BCD system uses decimal numbers and binary digits.

a. "COLUMNS" AND "BITS" CONCEPT. The BCD digits are arranged in "columns", the same as decimal numbers. Just as in the decimal system, there can be only one digit in each column. The BCD system must also be capable of counting up to ten in each column, so each column is divided into four "bits", which represent the first four states in binary counting (1-2-4-8). Figure 4-179 illustrates a decimal number and its corresponding BCD equivalent. The bits and columns are identified. Bit 0 has a decimal value of 1, bit 1 has a decimal value of 2, bit 2 has a decimal value of 4, and bit 3 has a decimal value of 8. Column 1 has a decimal value of 1, column 2 has a decimal value of 10, column 3 has a decimal value of 100, and column 4 has a decimal value of 1000. This concept of bits and columns is very important to the understanding of the BCD system.

```
                    4                   3                   2                   1      Columns
Decimal                 7                   3                   4                   6
Binary      0  1  1  1     0  0  1  1     0  1  0  0     0  1  1  0
            B3 B2 B1 B0    B3 B2 B1 B0    B3 B2 B1 B0    B3 B2 B1 B0      Bits
```

FIGURE 4-179

In the discussion that follows, the term "digit column" will occasionally be used to further differentiate between bits and columns. A "digit column" contains four bits in the BCD system.

b. CORRECTION FACTORS. Just as in the decimal system, a BCD column cannot have a value greater than nine. However, since a BCD column has four bits, it is capable of retaining counts up to 15, its 16 possible states representing the decimal numbers zero through 15. Since only ten of these 16 states are used (0 through 9), special correction factors must be applied at times.

(1) BCD Addition. The 5610 arithmetic logic checks the result of each binary addition. If the result is greater than a decimal 9, it automatically adds 6 to the answer as a correction factor. Figure 4-180 illustrates two examples of this type of correction. Notice that two things happen as a result of this correction: the answer is expressed in its correct binary form, and the carry is made to the next higher order column. The 1 in parentheses (1) denotes the carry.

	Decimal	BCD		Decimal	BCD	
(a)	5	0101		8	1000	
	+5	+0101		+7	+0111	
	10	1010		15	1111	
		+0110	correction		+0110	correction
		(1) 0000			(1) 0101	

FIGURE 4-180

The same type of correction is necessary in still another situation. Figure 4-181 illustrates an addition problem in which the binary sum in the column under consideration is not >9 (greater than 9), but still requires a correction in order to produce the proper BCD character in this column. In this case, however, the binary addition has already produced the signal which will cause the carry to the next column. The 5610 senses this carry and again makes the correction by adding 6. It should be noted that even though the binary sum in the column being operated upon is not >9, the carry itself denotes a sum >9.

(a)	9	(b)	1001	
	+9		+1001	
	18		(1) 0010	
			+0110	correction
			1000	

FIGURE 4-181

Still referring to Figure 4-181, note that a carry takes place from bit 0 to bit 1. No correction is necessary here. However, when a carry is made to the next higher order digit column, it denotes a binary sum greater than 15, and since 15 is >9, a correction is necessary. It can be stated then, that carries between digit columns require corrections, while carries between bits do not.

In all of the examples of Figures 4-180 and 4-181 the column-to-column carries and the corrections go together. Whenever the binary answer is equal to some decimal number from 10 through 19, a correction and a carry to the next higher order column are both necessary. Note that this carry is also needed in decimal addition.

The reason for adding 6 can easily be explained: it was noted in decimal addition and in regular binary addition that when a column becomes "full" it is returned to zero and a carry is made to the next higher order column. In the BCD system the four bits are capable of retaining any count up to 15. Therefore, if a 1 is added to a 9 and a carry is made, 6 more must be added to the column in order to return it to zero.

(2) BCD Subtraction. In BCD addition, a correction and a carry from one column to the next is produced whenever the sum in the first column is >9. In BCD subtraction, whenever a digit is subtracted from a smaller digit, it is necessary to borrow from the next higher order digit in order to raise the minuend to a value sufficient to allow the subtraction to take place. When this borrow is made from the next column, a correction must also take place.

The correction in BCD subtraction is very similar to that in addition, except that the 6 is subtracted instead of being added. As in addition, a borrow between bits requires no correction, but a borrow between columns does. 16 counts are required to completely "fill up" a BCD column and return it to zero, so a borrow from one column actually adds 16 to the next-right column. Since a borrow of 10 is all that is needed, 6 more must be subtracted in order to arrive at the correct BCD equivalent.

The mechanics of the 5610 arithmetic logic cause it to operate somewhat differently in a subtract correction than it does in an add correction. The >9 condition in a column after subtraction occurs is of no consequence, because it will always be accompanied by a borrow. (However, a borrow does not necessarily dictate a >9 condition.) Instead, the only condition needed to initiate a correction is the fact that a borrow was necessary to allow the subtraction to take place.

The important thing to remember concerning the 5610 is that when one digit is subtracted from a smaller digit, a decimal 10 is automatically added to the minuend in order to arrive at the correct answer in the column being operated upon. While working in this one column, the source of the extra 10 is of no consequence. However, a 1 must later be subtracted from the next higher order column when it is operated upon.

Figure 4-182 illustrates two BCD subtraction problems in which the subtrahend is greater than the minuend in one column. In Figure 4-182 (a) the intermediate answer is >9; in Figure 4-182 (b) it is not. In both cases the 5610 would sense that a borrow was necessary to allow the subtraction, and thus perform the correction.

(a)	13	0001	(1) 0011	(b)	11	0001	(1) 0001
	- 4	-(0000)	0100		- 9	-0000	1001
	(0)9	0000	1111		2	0000	1000
			-0110 correction				-0110 correction
		0000	1001			0000	0010

FIGURE 4-182

Still another situation should receive some explanation. Refer to the problems shown in Figure 4-183. Since the 5610 operates on only one digit at a time, it does not know whether the 5 is being subtracted from 4, or from 14, or 24, or 34, or 104, etc. It therefore automatically borrows from the next column to the left in order to produce what would be the correct answer if another digit (other than zero) were to appear in some column to the left. The logic then searches through the remaining columns, looking for a digit >0. If one exists, the answer is developed as shown in Figure 4-183 (a). If no other digits exist, however, the answer produced is an overdraft. (In the example of Figure 4-183 (b) the entire answer would be 99999999.00000.) Whenever an answer of this type (an "overdraft") is produced, the 5610 automatically performs an additional routine which complements this answer and changes the sign. In the above example the final answer would be -1. This final conversion will be explained in more detail later on.

(a)	104	0001	(1) 0000	(1) 0100	(b)	4	(1) 0000	(1) 0000	(1) 0100
	- 5	-(0000)	(0000)	0101		- 5	-0000	0000	0101
	99	0000	1111	1111		- 1	1111	1111	1111
			0110	0110 corr.			-0110	0110	0110 corr.
		0000	1001	1001			1001	1001	1001

FIGURE 4-183

K. ADD OR SUBTRACT

The Add and Subtract instructions are very similar in their operation. They are so similar, in fact, that it is possible for an Add instruction to cause a subtraction, and vice-versa. This occurs when the signs of R1 and R2 are different. Because these two instructions are so similar, their general operation will be covered as one: the machine functions that differ between the add and subtract operations will then be discussed individually.

1. GENERAL OPERATION

An Add or Subtract command is retrieved in 4 mode 3 phase. Being 0-address commands, they cause a jump to 5 phase, and at program line home a jump to 5 mode 15 phase occurs. 5 mode resets DOST, enabling a jump to 6 phase, which occurs at the first T6B after the sync pulse. Then, when the problem is completed, a step to 7 phase occurs. The logic waits in 7 phase until program line home, where a jump to 3M6,4 occurs.

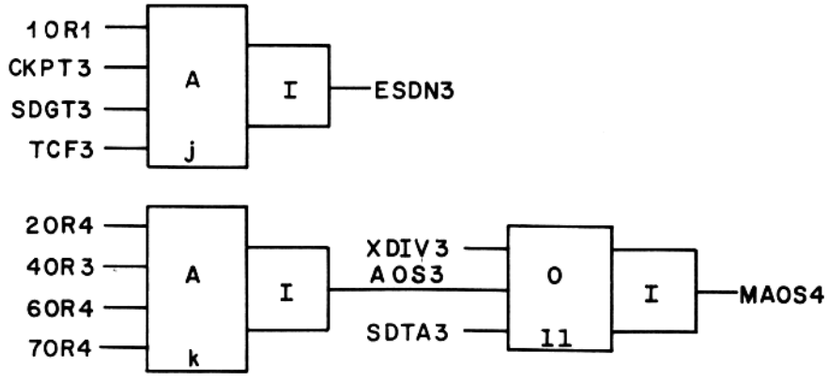
When DOST resets it unlocks the action step counter, and the next arithmetic line home allows a step to action step 1, where the operating stack is shifted down. At the end of this line pass, a step to action step 2 occurs, where the actual add or subtract takes place. Just before the computation occurs, the signs in R0 and R1 are compared. If they are the same, the function named in the command occurs: if they are different, the opposite function occurs. Then the number formerly in R1 and now in R0 is added to (or subtracted from) the number formerly in R2 and now in R1. The sum (or difference) is in R1 at the completion of the line pass. The number in R0 is rewritten unchanged. (This is of no value in add or subtract, but it is necessary during the additions and subtractions that occur during multiplication and division, as will be seen later.)

During this second line pass, the line routing is ALCR, CRAL, so that the data sensed in R0 time is rewritten during R1 time. This data remains R0 data, however, because the sync pulse was also written "long". Likewise, the sum (or difference) is written out during R2 time, but is found in R1 during subsequent line passes. The same is true of R2, R3, and R4: the data is written one R-time late (R3, R4, and R0, respectively), but because the sync pulse was written long, the same time relationship exists between the data and the sync pulse.

After one line pass in action step 2, the counter returns to 0 and the following sync pulse sets DOST. This enables the step to 7 phase.

2. RETRIEVAL OF THE "AOS" INSTRUCTION

An Add command or a Subtract command is retrieved in 4 mode the same as any other 0-address command. As soon as it is shifted into the operation register in 3 phase, the Add Or Subtract signal, AOS3, and Enable Shift DowN, ESDN3, are produced, as shown in Figure 4-184. AOS3 also develops Machine Add Or Subtract, MAOS4.



MASTER SIGNALS IN AOS

FIGURE 4-184

TP-161SL

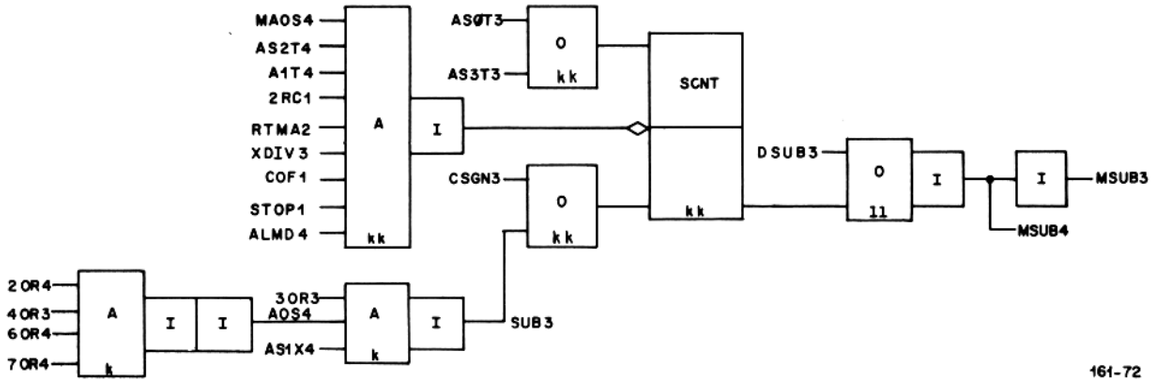
From 3 phase the logic jumps to 5 phase, and at program line home jumps to 5 mode 15 phase.

3. PRELIMINARY 5 MODE OPERATION

Only three arithmetic line passes are necessary in 5 mode: the first primes the action step counter, the second provides the shift down of the operating stack, and the third performs the actual computation. (If the computation results in a carry or a borrow at the completion of the line pass - that is, an overflow condition - one more line pass occurs. This will be covered later.)

a. SHIFT DOWN. In 5 mode the first A1 time resets DOST, and at the following arithmetic line home the action step counter increments to step 1. During the following line pass the operating stack is shifted down in exactly the same way as in the Delete instruction: the Storage and Arithmetic Shift Down signal, SASD4, is ANDed with ESDN4 to produce data long, DLNG, which provides the ALCR line routing for the entire line pass. At the same time these two signals are ANDed with A0T3 and R0T3 to enable write direct at all times other than A0 and R0. Refer to Section 4D, Delete, for review.

If a Subtract command has been retrieved, the 3OR flip-flop is reset, and in action step 1 the SUB3 signal is produced, which resets the Sign CoNTrol flip-flop, SCNT, as shown in Figure 4-185. Note that SCNT has been set by AS0T3, so if an Add command had been retrieved, SCNT would be set at this time.



SCNT OPERATION

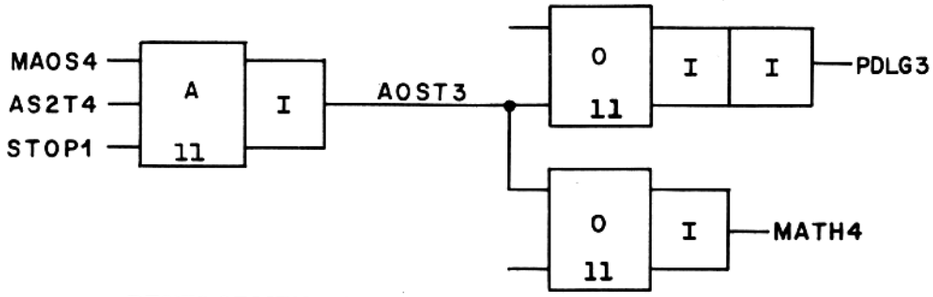
FIGURE 4-185

b. COMPARING THE SIGNS. After the stack has shifted down, the action step counter steps to 2, where the first occurrence is the comparing of the signs. Still referring to Figure 4-185, note that any bit (RTMA) appearing in R0 or R1 (2RC1) during column A1 will toggle SCNT. Thus, if the signs are different, SCNT will be toggled once, driving MSUB3 true if an Add had been retrieved, or driving it false if a Subtract had been retrieved. In the former case, Machine SUBtract, MSUB3, causes a subtract function to occur. This is termed "machine subtract" to differentiate from a Subtract instruction. In the latter case, with MSUB3 false, a "machine add" occurs, even though a Subtract command had been retrieved. Thus, if the signs in R0 and R1 are different, an Add command causes a machine subtract, and a Subtract causes a machine add.

If both signs are positive, SCNT is not toggled; if both signs are negative, SCNT is toggled twice. In either case SCNT ends up in its original state and the instruction does not change: an Add command causes a machine add and a Subtract command causes a machine subtract.

During machine add the sum will take the sign of R1. During machine subtract, the difference will take the sign of R1, unless an overflow occurs (one number subtracted from a smaller number): in this case the sign will be changed to that opposite the sign of R1. This will be explained later.

c. COMPUTATION. The actual computation occurs during one line pass in action step 2. During this line pass all data is written long, including the sync pulse. This arithmetic line routing is necessary for the computation, and is developed as shown in Figure 4-186. MAOS4 is ANDed with action step 2 to produce Add Or Subtract Time, AOST3. This produces both PDLG3, to provide the long data path, and MATH4, which is used in the computation.

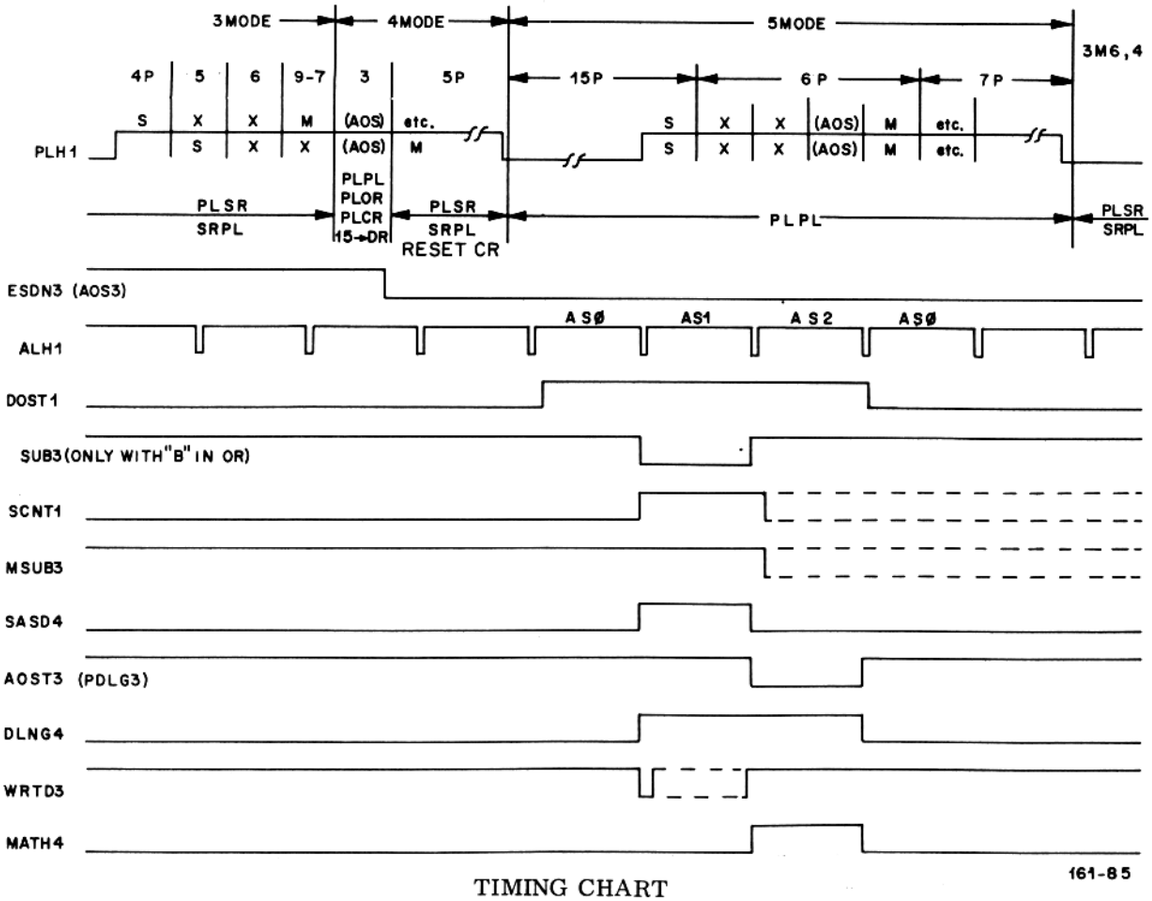


DEVELOPMENT OF PDLG3 AND MATH4 DURING AOS

FIGURE 4-186

4. SUMMARY

The timing chart in Figure 4-187 summarizes the overall operation of an Add or Subtract instruction. The actual details of the computation will be covered next.



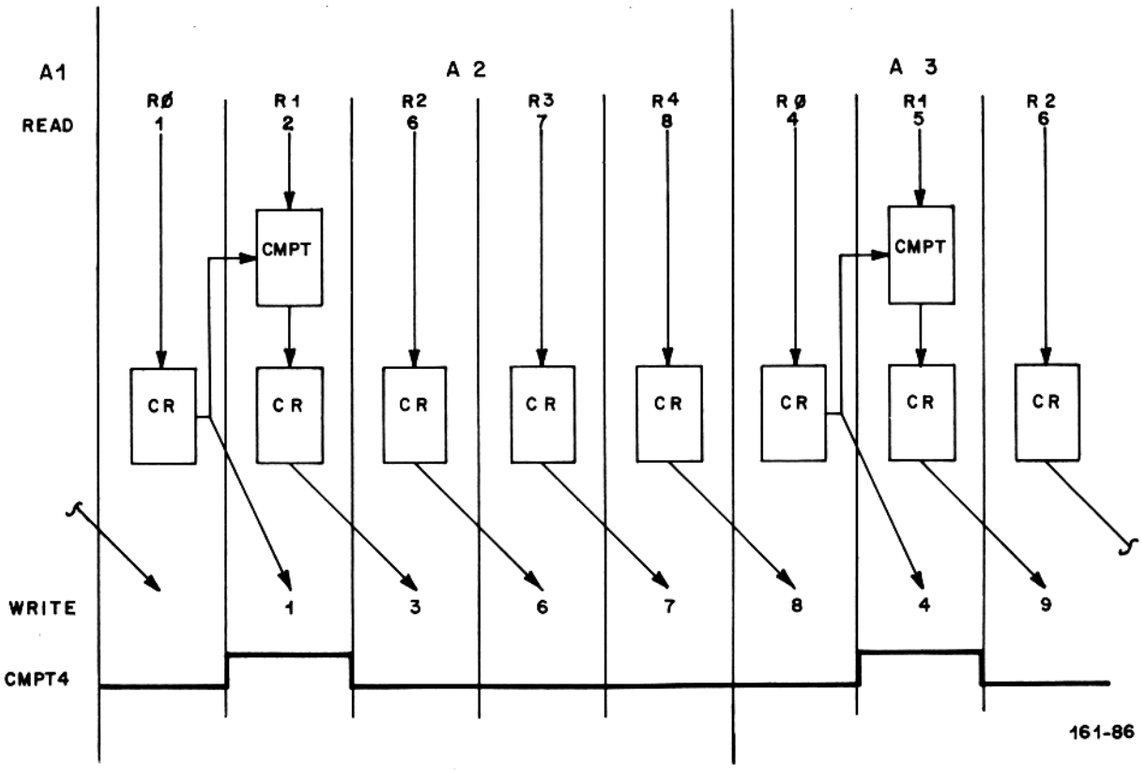
TIMING CHART
ADD OR SUBTRACT - ENTIRE INSTRUCTION

FIGURE 4-187

5. IMPORTANT CONCEPTS

There are certain basic concepts pertaining to the AOS operation that must be understood before going further. Although some of these concepts may appear to bear no resemblance to one another at first, their relationship will soon become apparent.

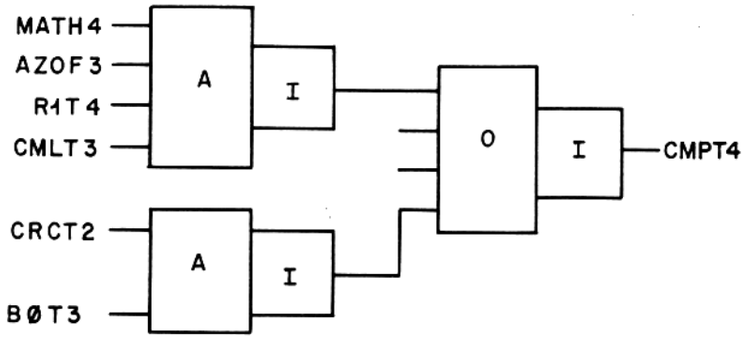
a. BASIC TIMING. In action step 2 the line routing is ALCR, CRAL. The data in R0 (the former R1 data) is shifted into the common register during R0 time. During R1 time this data is written back onto the delay line unchanged. The data in R1 (the former R2 data), being sensed simultaneously, is compared with the data in the common register as the R0 data is shifted out onto the line. This sum or difference is then shifted out onto the line in R2 time, while the data in R2 is shifted into the common register. (Any necessary "correction" is made during R2 time just before the answer is written out.) In R3 time the R2 data is written out while the R3 data is shifted into the common register, and in R4 time the R3 data is written out while the R4 data is stored in the common register. In R0 time the process starts again: R0 data is shifted into the common register (while the R4 data is written out), and at R1 time the computation occurs. Remember that no data is displaced by all this shifting because the sync pulse was also shifted through the common register before being rewritten. This basic operation is illustrated in Figure 4-188.



AOS BASIC TIMING

FIGURE 4-188

b. "COMPUTE" TIME. All computations are performed in R1 time, A2 through A14, when the CoMPuTe signal, CMPT, is developed. See Figure 4-189. Whenever a correction is necessary, the CoRreCTion flip-flop, CRCT, is set in R2 time, developing CMPT in R2 during B1, B2, and B3. This will be more fully explained when the correction operation is studied.



DEVELOPMENT OF CMPT4
(All gates on "LL" card)

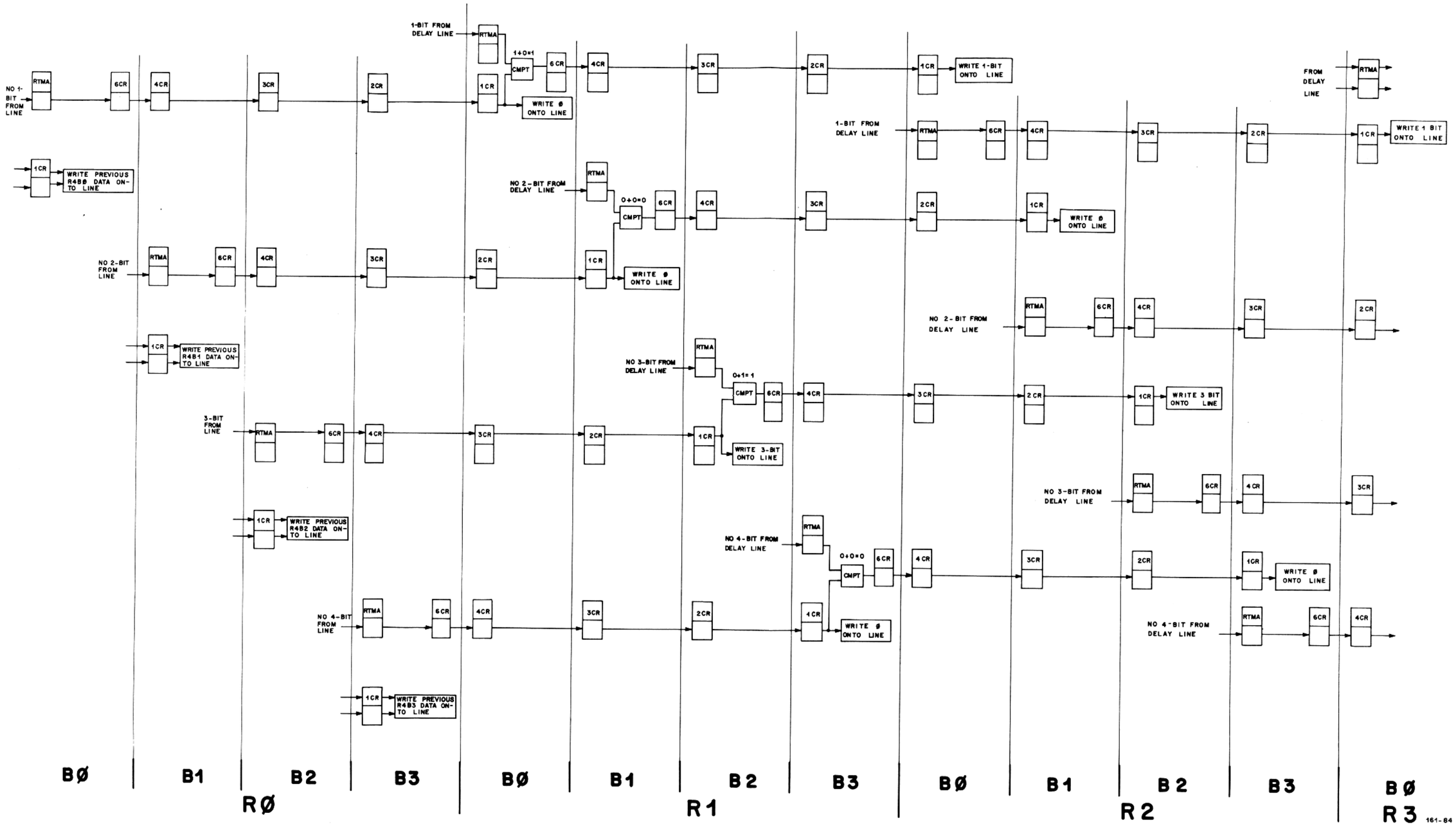
FIGURE 4-189

c. COMPUTATION BY "BITS". Since data is recorded on the delay line one bit at a time, the computation in R1 time must follow suit. It would be impossible to compare (add or subtract) all four bits of R1 data and the contents of all four CR flip-flops at one time. Instead, the contents of the 1CR flip-flop are added to (or subtracted from) the contents of RTMA, and the resultant sum (or difference) is set into 6CR. One register time later this "answer" has been shifted through the common register into 1CR, from whence it is written onto the delay line.

For example, assume that a 4 (3-bit) in some column of R0 is to be added to a 1 in the same column of R1. Refer to Figure 4-190 and note that during R0 time the four is shifted into the common register. At R1B0 time a bit is sensed and RTMA sets. Immediately RTMA and 1CR are "added", and the 1 that results is set into 6CR. At R1B1 time this 1 is shifted into 4CR, at B2 it moves to 3CR, at B3 to 2CR, and at the next B0 time, which is R2B0, it is shifted into 1CR, from whence it is written onto the line. While this 1 is being shifted through the common register, the four is being written back onto the line. The other three "bits" in R1 are also added to the remaining three bits in the common register during R1 time: during B2 time the bit in 1CR (representing the four) is added to the zero in RTMA, producing a logic 1 to set 6CR. At R1B3 time this bit shifts into 4CR, and four bit times later (R2B2) it is in 1CR, from whence it is written onto the line. Thus, in R2 time a bit is written out in B0 time and again in B2 time, representing the total (5) of R0 and R1.

Note that in R0, R2, R3, and R4 times the output of the delay line is copied directly into the common register, but in R1 time it is the result of the computation. Therefore the line routing in action step 2 is always ALCR, CRAL, EXCEPT as modified by the computation.

d. CARRY/BORROW. In addition to the data in 1CR and RTMA, a third source of data enters into the computation during each bit time of R1: the Carry/Borrow flip-flop, CB. Either a carry (in machine add) or a borrow (in machine subtract) occurring in the previous column or bit will set CB.



BASIC ADD OPERATION, SAMPLE PROBLEM
FIGURE 4-190

TP-161SL

e. EIGHT POSSIBLE CONDITIONS. With three sources of data, it is possible to have eight different conditions during any bit time in R1. These possibilities are shown in the chart in Figure 4-191. Note that the sum or difference in each case is the same, regardless of whether a machine add or a machine subtract is occurring: the only difference is in the setting and resetting of the Carry/Borrow flip-flop.

R1 DATA (RTMA, or "Y")	0	1	0	1	0	1	0	1
R0 DATA (1CR, or "X")	0	0	1	1	0	0	1	1
CARRY/BORROW (CB)	0	0	0	0	1	1	1	1
SUM/DIFFERENCE (6CR)	0	1	1*	0*	1*	0*	0	1
CARRY PRODUCED?	No	No	No	Yes	No	Yes	Yes	Yes
BORROW REQUIRED?	No	No	Yes	No	Yes	No	Yes	Yes

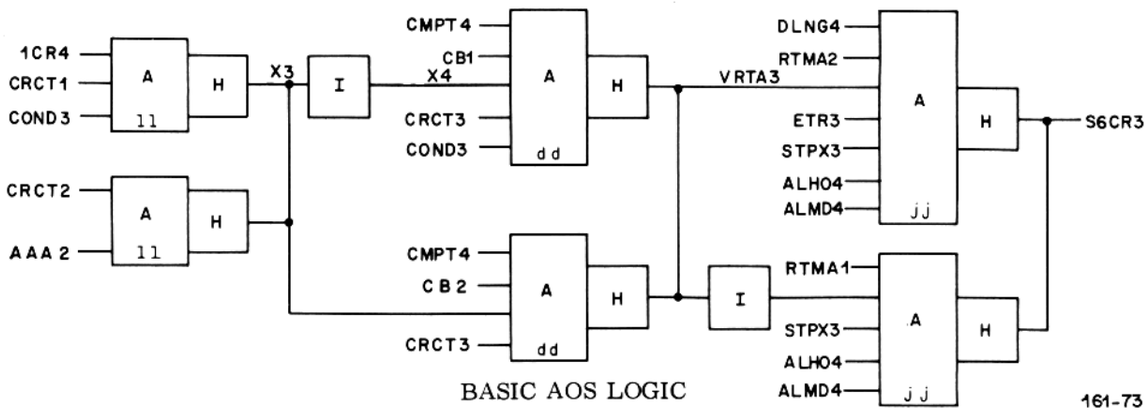
EIGHT POSSIBLE CONDITIONS

FIGURE 4-191

f. "X-Y" VARIABLES. The contents of R0 and R1 are represented in the computation by "X" and "Y" variables. RTMA and 1CR are not used directly because machine subtract makes it necessary for them to be used differently under special circumstances (during "correction"). Instead, the X3 and Y3 signals are used. In regular machine addition or subtraction (not during a correction), 1CR (R0 data) is represented by X, and RTMA (R1 data) is represented by Y.

g. COMPUTATION PROCESS. Since the data path throughout the line pass is ALCR, CRAL, the contents of RTMA are set into 6CR, and at the next bit time the 6CR state is shifted into 4CR. (Note that since there are no signals available to set 7CR, each time the Common Register Shift Pulse, CRSP, occurs, 6CR becomes reset.) Refer again to Figure 4-191 and note that in four of the cases (*) the state of 6CR is opposite to that of RTMA. Therefore it must be said that ALCR is not necessarily true during the compute time. Instead, 6CR assumes a state determined by all three data sources. Still referring to Figure 4-191, note that if X and CB are the same, 6CR assumes the same state as RTMA, but if X and CB are opposite, 6CR assumes the state opposite to that of RTMA.

The gates providing the basic computation are pictured in Figure 4-192. Compare the operation of these gates to each of the eight possible states noted in Figure 4-191.



BASIC AOS LOGIC

FIGURE 4-192

As noted previously, the sum or difference, which is written out in R2 time, is dependent upon the state set into 6CR during each of the bit times in R1. Figure 4-192 illustrates the two gates used to set 6CR in the example of Figure 4-190. The top gate was used when RTMA was set, containing the 3 bit from R0.

In using the second gate it is necessary to develop inVert ReTime Arithmetic, VRTA3. This can be accomplished by developing the X variable, mentioned earlier. Thus, if 1CR is set, X3 becomes true and enables the development of VRTA3. With VRTA3 true, 6CR assumes the state opposite to that of RTMA.

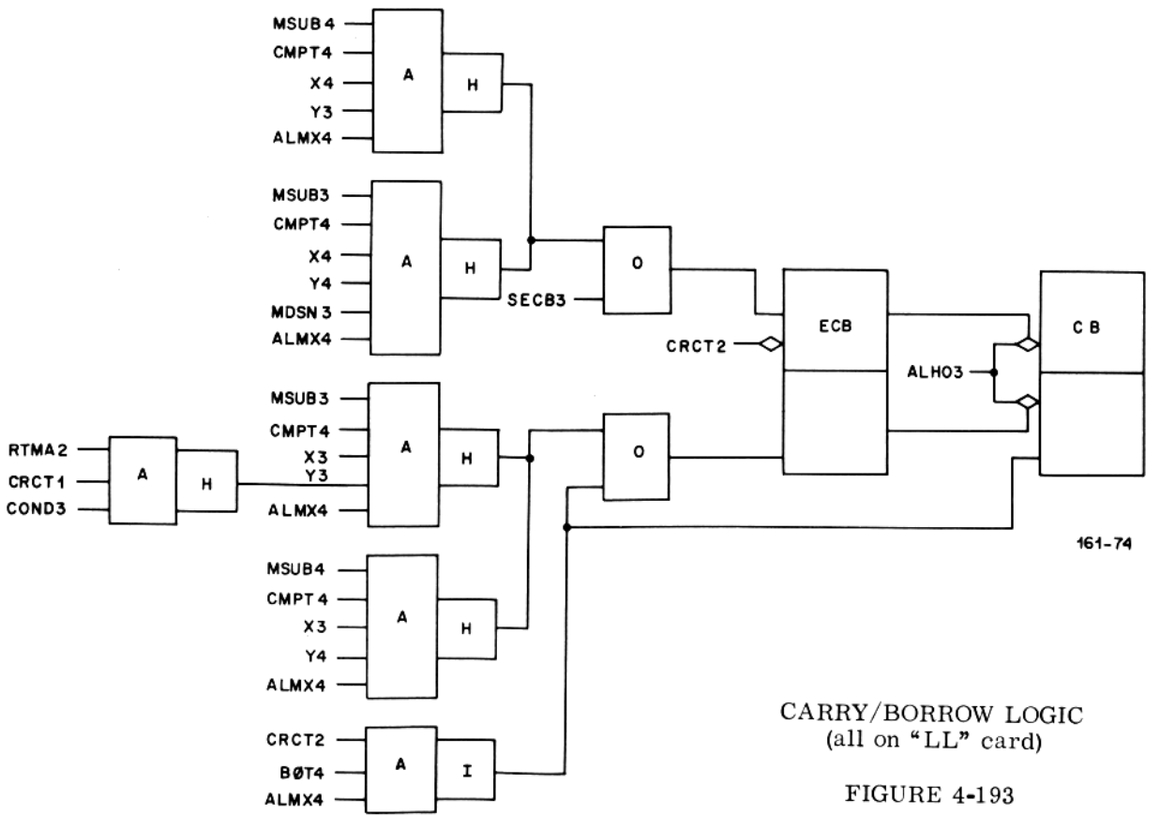
Still referring to Figure 4-192, note that VRTA3 can also be developed when 1CR is reset, by having CB set. Thus, there are two conditions under which VRTA3 is developed and the condition of RTMA is "inverted" as it is set into 6CR: when 1CR is set and CB is reset, and when 1CR is reset and CB is set. Compare this to the chart in Figure 4-191. The rule presented earlier can be restated as follows: when 1CR and CB are in the same logic state, 6CR assumes the same state as RTMA; when 1CR and CB are in the opposite logic state, 6CR assumes the state opposite to that of RTMA.

6. CARRY/BORROW LOGIC

So far the "X" variable has been the only one explained. In fact, the "Y" variable is not needed in determining the state to be set into 6CR. It is, however, necessary in developing the carry-to or borrow-from the next bit. Since this operation is the only point of difference between machine add and machine subtract, the two operations will be discussed separately at this time.

a. MACHINE ADD. Figure 4-193 illustrates the logic developing the inputs to CB. During machine add the Enable Carry/Borrow flip-flop, ECB, becomes set when both "X" and "Y" are true (both X4 and Y4 are false), denoting the presence of bits in both 1CR and RTMA. This "bit" is then shifted from ECB to CB at the next bit time (when ALHO3 goes true), and here it is "added" to the contents of RTMA and 1CR in this next bit time.

For example, if a 1 in R0 is added to a 1 in R1, RTMA and 1CR will both be set. Since 1 + 1 = 0 + a carry, VRTA3 is produced in B0 time to "invert" the set condition of RTMA and prevent 6CR from setting. Since X and Y are both true, ECB sets, and CB sets at the next bit time. Since no other bits are present, VRTA3 is again produced, and the reset condition of RTMA causes 6CR to become set. Thus, the two 1-bits are converted to a single 2-bit.



CARRY/BORROW LOGIC
(all on "LL" card)

FIGURE 4-193

Once ECB becomes set, it does not reset until an absence of bits appears in both 1CR and RTMA simultaneously (or until a correction occurs). This is necessary to allow for the correct progression of the carry through the course of the problem. For example, refer to Figure 4-194 and note that in B0 time both 1CR and RTMA are set, causing ECB to set and producing a carry to B1 time. In B1 time CB sets and enters into the add (S6CR) process. ECB is not affected, so it remains set. In B2 time, CB remains set because ECB is still set, and ECB again is not affected. In B3 time, ECB finally resets due to the absence of bits in both 1CR and RTMA. CB remains set from the previous bit time, but because ECB resets in B3 time, CB will reset at the next bit time, which happens to be B0 of the next register.

	B0	B1	B2	B3
1CR	1	1	0	0
RTMA	1	0	1	0
Sum	0	0	0	1
ECB	Sets	Remains Set	Remains Set	Resets
CB	----	Sets	Remains Set	Remains Set (Resets in next Column)
	1+1=0+ a carry	1+0+ a carry= 0+ a carry	0+1+ a carry= 0+ a carry	0+0+ a carry =1

EXAMPLE OF A CARRY

FIGURE 4-194

The carry rules for machine add can be summarized as follows: if R0 and R1 both contain bits (1CR and RTMA both set, Y and X both true), ECB will set and CB will set at the next bit time; if neither R0 nor R1 contain a bit (1CR and RTMA both reset, Y and X both false), ECB will reset. If either R0 OR R1 contains a bit (but not both), ECB does not change its state.

b. MACHINE SUBTRACT. During machine subtract, ECB becomes set only when a "1" is subtracted from a "0", or when X is true and Y is false. It resets only when the opposite is true: when 0 is subtracted from 1, or when Y is true and X is false. If X and Y are both in the same logic state, ECB is not affected it remains set or reset, as the case may be. In all cases, CB assumes the state of ECB at the next bit time. The gates are shown in Figure 4-193.

An example is illustrated in Figure 4-195. In B0 time a 1 is subtracted from a 0. This requires a borrow from B1 time, so ECB sets immediately and CB sets as the step to B1 occurs. ECB is not affected in B1 time because X and Y are in the same logic state. In B2 time, however, a 0 is subtracted from a 1, causing ECB to reset. The fact that CB is set, denoting "one" has already been borrowed from this column, simply subtracts one more from the "answer", changing the 1 to a 0. CB resets at the step to B3 time, and during B3, since X and Y are in the same logic state, ECB remains reset.

The borrow rules for machine subtract can thus be restated as follows: a 1 from a 0 sets ECB; a 0 from a 1 resets ECB. A 1 from a 1 or a 0 from a 0 does not affect ECB.

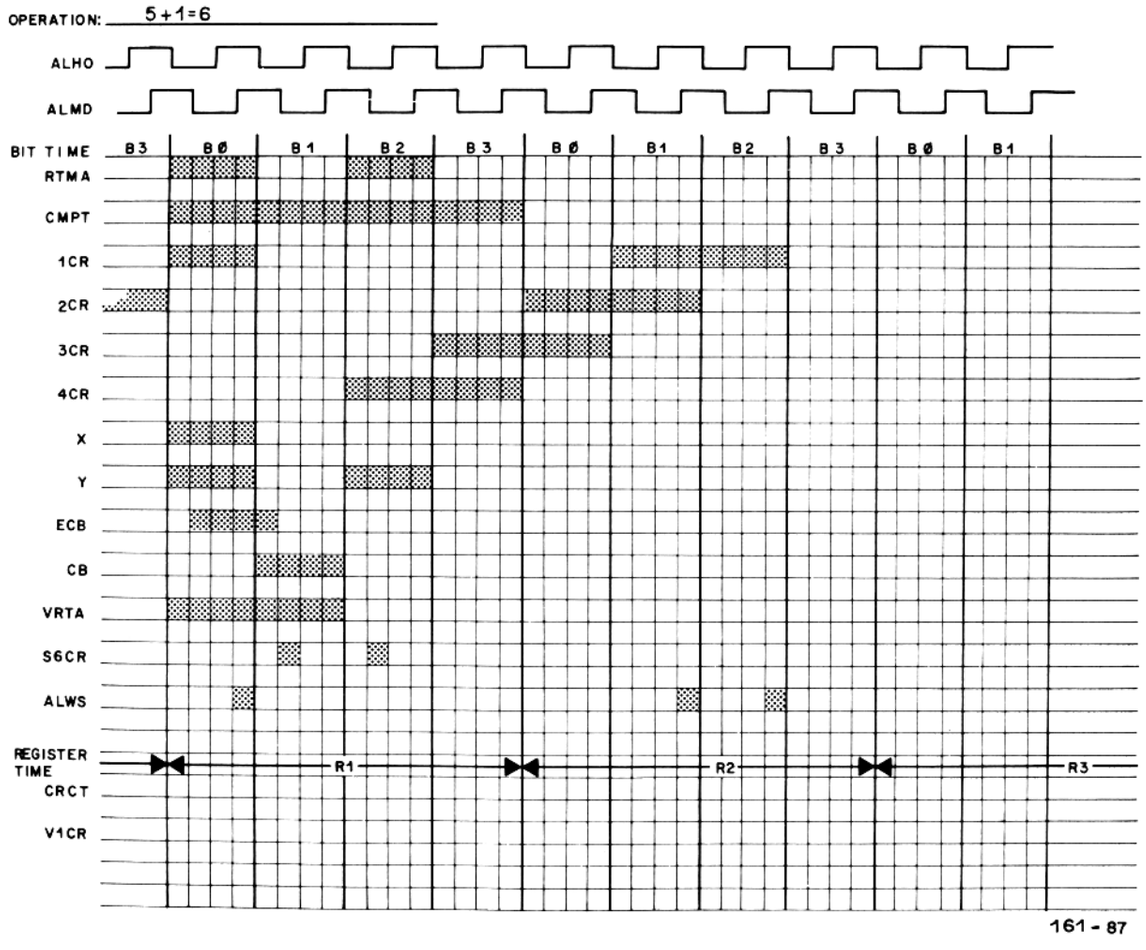
	B0	B1	B2	B3
RTMA (Y)	0	1	1	0
1CR (X)	1	1	0	0
DIFFERENCE (6CR)	1	1	0	0
ECB	Sets	Remains Set	Resets	Remains Reset
CB	----	Sets	Remains Set	Resets
	0-1=1+a borrow	1-1+ a borrow = 1+ a borrow	1-0+ a borrow=0	0-0=0

EXAMPLE OF A BORROW

FIGURE 4-195

7. SUMMARY OF TIMING

Observation of the timing signals applied to the various gates discloses the fact that the following signals change state ONLY at fractional bit time "SH": RTMA, all common register flip-flops, X3, Y3, CB, and VRTA. In addition, ECB changes only at fractional bit time "A", and S6CR3 occurs only during "A". ALWS, of course, occurs only during "C" time.



TIMING CHART
MACHINE ADD SAMPLE

FIGURE 4-196

a. MACHINE ADD. A sample machine add problem is illustrated in Figure 4-196. Note each of the following items:

R1 TIME: CMPT is true throughout R1 time.

B0 "SH": RTMA is set, denoting a 1-bit in R1.

1CR is set, denoting a 1-bit R0 time. 2CR, 3CR, and 4CR are reset, denoting the absence of any other bits in R0.

Y3 true and CB reset develop VRTA3.

"A": X3 and Y3 are both true, causing ECB to set at "A" time.

VRTA3 inverts the set state of RTMA, as applied to 6CR, preventing the setting of 6CR.

"C": The 1-bit in 1CR is written onto the line.

B1 "SH": RTMA and 1CR are reset, denoting the absence of 2-bits in R0 and R1.

CB sets because ECB was set previously.

"A": X3 and Y3 are both false, causing ECB to reset at "A" time.

VRTA3 inverts the reset state of RTMA, setting 6CR.

"C": With 1CR reset no pulse is written onto the line.

B2 "SH": RTMA is set, denoting a 3-bit in R1 and causing Y3 to become true.

1CR is reset, denoting no 3-bit in R0 and causing X3 to remain false.

The previously set condition of 6CR is shifted into 4CR.

6CR resets.

CB resets because ECB was reset previously.

Since X3 is false and CB is reset, VRTA3 is not developed.

"A": ECB is not affected because X3 and Y3 are in opposite logic states.

With VRTA3 false, the set condition of RTMA is transferred to 6CR.

"C": With 1CR reset, no pulse is written onto the line.

B3 "SH": 1CR and RTMA are both reset, denoting the absence of 4-bits in both R0 and R1: X3 and Y3 are both false.

The previously set state of 6CR shifts into 4CR.

6CR resets.

The previously set state of 4CR shifts into 3CR.

Since X and CB are in the same logic state, VRTA3 is not developed.

"A": With VRTA3 false, the reset state of RTMA prevents a set of 6CR.

With X and Y both false, ECB receives a reset pulse, even though it is already reset.

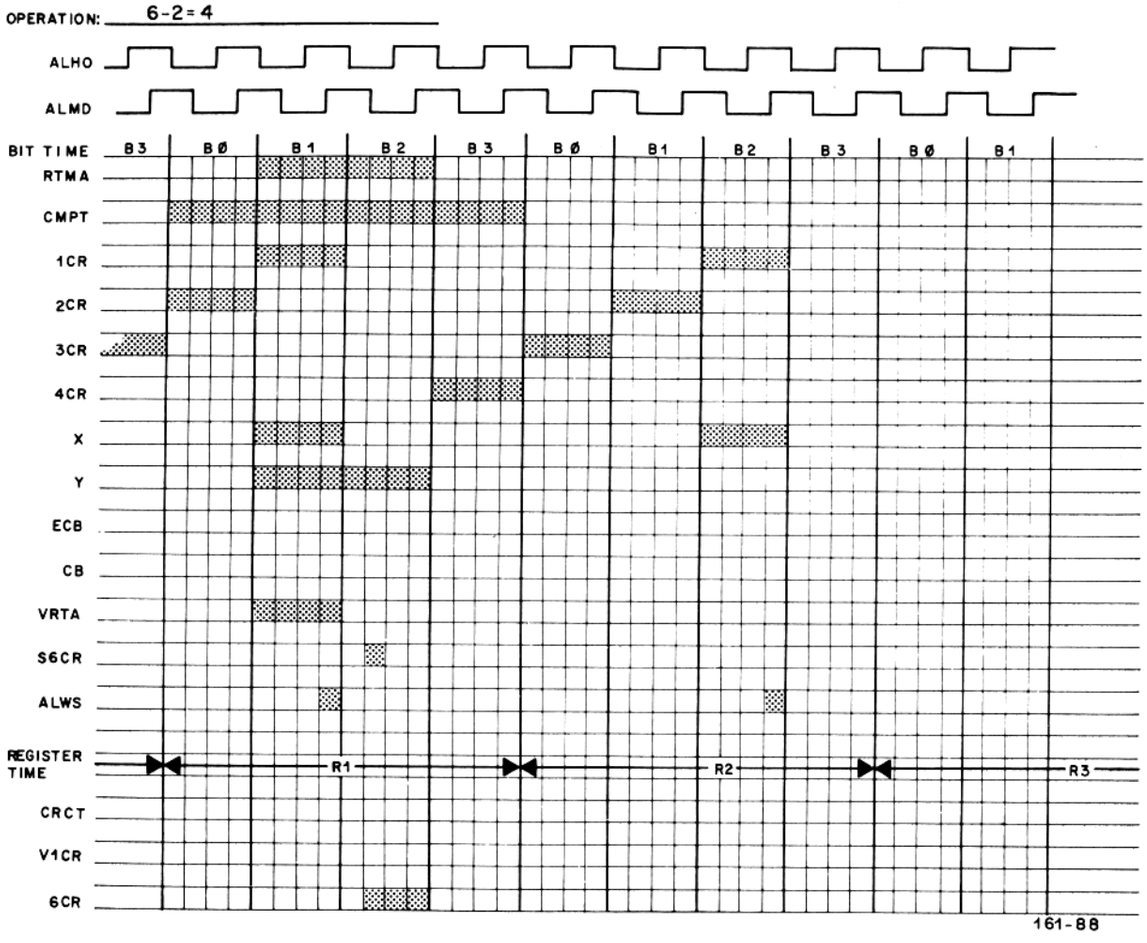
"C": With 1CR reset, no pulse is written onto the line.

R2 TIME: CMPT3 returns to false.

Any necessary correction is performed (to be covered shortly).

The answer is shifted through the common register and is written onto the line as it passes through 1CR.

The data in R2 is shifted into the common register.



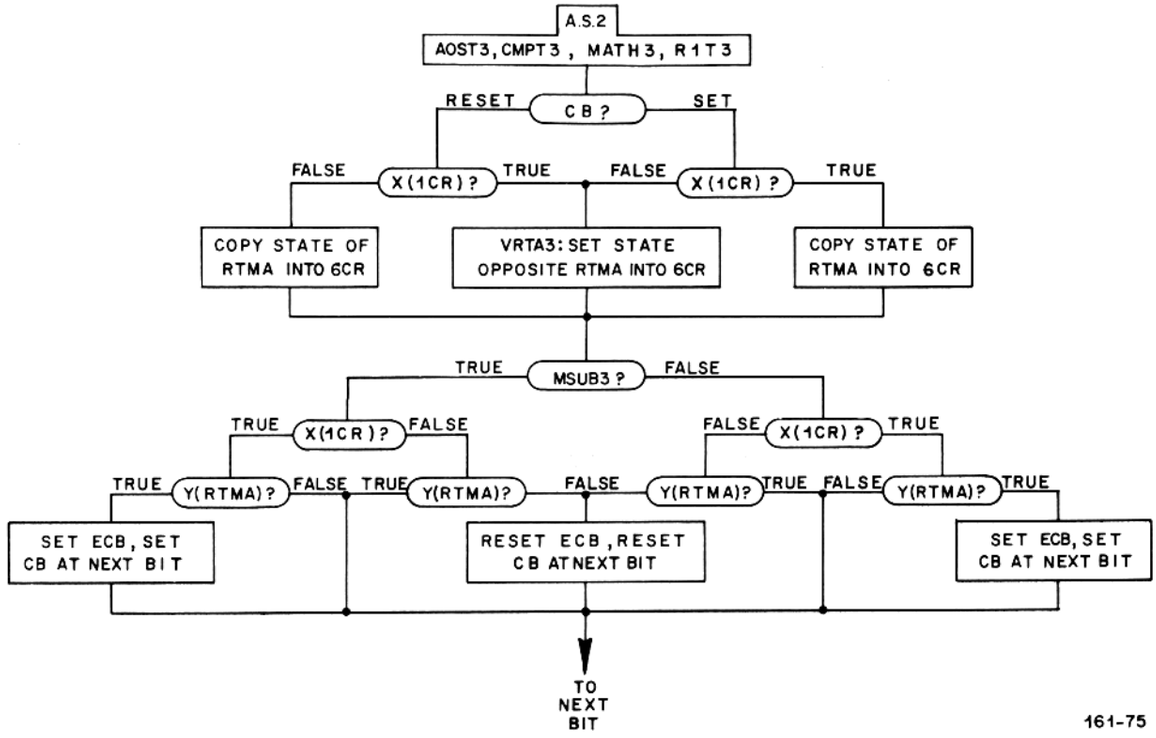
TIMING CHART
MACHINE SUBTRACT SAMPLE

FIGURE 4-197

b. MACHINE SUBTRACT. A sample machine subtract problem is illustrated in Figure 4-197. Note that the timing of the various signals is exactly the same as during machine add: S6CR3 occurs only during "A" time, ALWS occurs only in "C" time, ECB changes only at "A" time, and all other signals change at "SH" time. Note that 6CR sets when S6CR is developed, and resets at "SH" time, when the common register is shifted.

8. FLOW CHART

The flow chart in Figure 4-198 summarizes the add-or-subtract logic operation in R1 time.



FLOW CHART
AOS LOGIC

FIGURE 4-198

9. CORRECTION

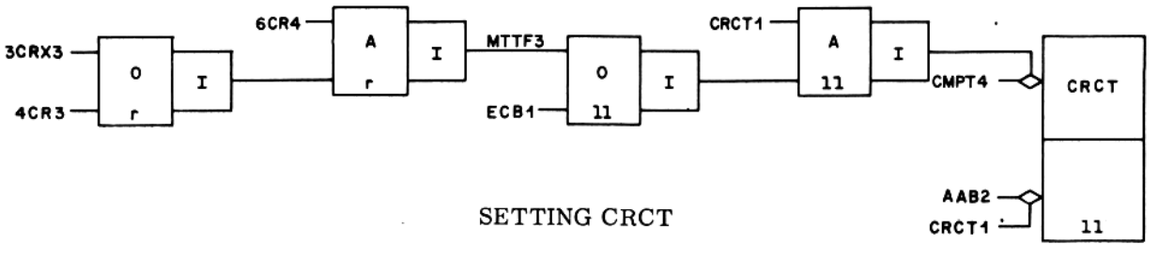
The number in the common register at the end of R1 time is the "uncorrected" sum or difference. When no correction is necessary, this number is merely shifted out onto the delay line unchanged. When a correction is necessary, however, the effect of the 1CR flip-flop on the Arithmetic Line Write Signal, ALWS4, is "inverted" when necessary to add or subtract 6 to or from the uncorrected sum. To accomplish this, X3 is automatically developed in B1 and B2 time, and inVert 1CR, V1CR3, is developed whenever X3 and CB are in the opposite logic states, except during B0 time.

CMPT is developed during B1, B2, and B3 times, but not during B0 time (see Figure 4-189.) It is not necessary during B0 time because the binary 1 bit is never affected by adding or subtracting 6.

During machine add, the fact that a correction occurs indicates that the uncorrected sum developed is >9. During machine subtract, a correction indicates that a borrow was necessary to allow the subtraction. In either case, ECB and CB are set after the correction, to allow the carry to (or borrow from) the next column.

a. CONDITIONS REQUIRING CORRECTION. All corrections occur during R2 time. If the addition in R1 produces an answer >9, or if the subtraction in R1 requires a borrow, a correction is necessary.

Refer to Figure 4-199 and note that either ECB1 or the Math Ten To Fifteen signal, MTTF3, will enable the set gate to the CoRreCTion flip-flop, CRCT. If this set gate is enabled at the end of R1 time, when CMPT4 returns to true, CRCT will set, allowing a correction to occur during R2 time of that column.



SETTING CRCT

FIGURE 4-199

Notice that the number in the common register at the end of R1 time is still displaced by one shift: 6CR being set at the end of R1 denotes a binary 8, and this bit is shifted into 4CR as the step to R2 occurs. Therefore, the combination of 6CR being set along with either 4CR or 3CR covers all possible sums between (and including) ten and fifteen.

It is possible for a correction to be necessary without MTTF being true. This occurs in problems having column sums (or differences) equal to 16, 17, 18, or 19. In these four cases MTTF3 remains false, but ECB becomes set (or remains set) during B3 time of R1.

(1) Machine Add. An example can be found in the case of an 8 in R0 being added to an 8 in R1. The sum of 16 appears in the common register as zero, but the fact that ECB becomes set in R1B3 enables the set gate of CRCT, which sets as the timing chain steps to R2. The ensuing correction procedure then adds 6 to the zero in the common register, and the corrected sum (6) is written onto the line during R2 time. (Note that the 6 is not actually added to the number while it is in the common register, but instead this number is modified as it is shifted onto the delay line.)

(2) Machine Subtract. The correction process in machine subtract is similar, except that the 6 is subtracted from the uncorrected number instead of being added. Since a total of 16 has been borrowed from the next higher column, 6 more must be subtracted in order to equate the borrow to only 10.

At first glance the conditions requiring correction appear to be the same as during add: either MTTF or ECB will cause a correction to occur. Note, however, that MTTF is really not necessary during subtract, because any uncorrected "difference" that is greater than 9 will already have made a borrow and ECB will be set. For example, if 8 is subtracted from 4, the uncorrected difference in the common register at the end of R1 time is 12 (1100) and a borrow has already been made (CB becomes set in R1B3). This is illustrated in Figure 4-200. Review briefly the other uncorrected "differences" that are possible and note that in all cases ECB has been set by the end of R1 time.

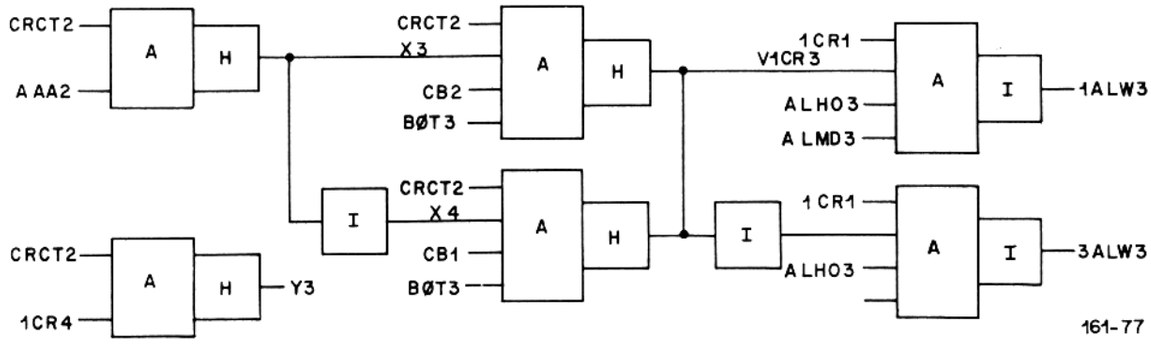
		B0	B1	B2	B3	
	Y	0	0	1	0	
	X	0	0	0	1	
uncorrected difference		0	0	1	1	(1)
CB sets here					↑	

EXAMPLE OF A DIFFERENCE GREATER THAN 9

FIGURE 4-200

b. LOGIC. During a correction, the X variable, which represents the 6, is made true during B1 and B2 time. This was shown previously in Figure 4-192 and is shown again in Figure 4-201. The Y variable is represented by 1CR data. Note that this is a change from R1 time, when 1CR represented the X variable. If the machine add function were the only one requiring a correction, this change would not be needed. Machine subtract, however, requires 1CR data to be the subtrahend during R1 time and the minuend during a correction. Thus, the varying development of X and Y allows the actual computing logic to remain the same during both R1 time and during the correction.

The basic logic used during correction is illustrated in Figure 4-201. Note that X3 is always true during B1 and B2, when AAA2 is false. (It is not dependent upon the numbers involved.) This is what adds the 6. Y3 is not used for the actual computation: it is necessary only in determining the state to be assumed by ECB. 1CR is used directly, however, in determining the "answer" that is written onto the line.



CORRECTION LOGIC
(All gates on "LL" card)

FIGURE 4-201

Still referring to Figure 4-201, note that if V1CR3 is false, the state of 1CR is copied onto the delay line; if V1CR3 is true, the state opposite to that of 1CR is "written" out. In addition, note that V1CR3 is true when X and CB are in the opposite logic states.

The preceding is further borne out in the chart in Figure 4-202. This chart lists each of the eight combinations possible in a single bit time during a correction. Note that the condition of RTMA is of no consequence: X3, the automatic 6, takes the place of RTMA in providing eight possibilities.

	B3		B1	B2	B3		B2	
Y3 (1CR)	0	1	0	1	0	1	0	1
X3 (Add 6)	0	0	1	1	0	0	1	1
CARRY/BORROW (CB)	0	0	0	0	1	1	1	1
SUM/DIFFERENCE (ALWS)	0	1	1*	0*	1*	0*	0	1
CARRY PRODUCED?	No	No	No	Yes	No	Yes	Yes	Yes
BORROW REQUIRED?	No	No	Yes	No	Yes	No	Yes	Yes

8 POSSIBLE CONDITIONS DURING CORRECTION

FIGURE 4-202

Compare the chart in Figure 4-202 to the chart shown previously in Figure 4-191. Note that the entire chart is identical, except for the fact that during correction X represents the automatic 6 and the 1CR data becomes the Y variable. Where VRTA3 was developed in R1 time, V1CR3 is now developed in the correction (*). Comparison of these charts further exemplifies the similarity between the logic in R1 time and the correction: the difference is in the transfer of 1CR data to the Y variable, the automatic 6, and the development of V1CR and ALWS instead of VRTA and S6CR. The difference between machine add and machine subtract during correction (as related to ECB and CB) is the same as during R1 time.

c. TIMING. A few items relating to timing during correction have already been mentioned, such as CMPT being developed only in B1, B2, and B3 time, and X3 being true only during B1 and B2 time. Still referring to the chart in Figure 4-202, note that certain of the other conditions are possible only during given bit times:

(1) B0 Time. CRCT sets at the beginning of B0 time. At fractional bit time "A" ECB and CB receive a reset pulse, regardless of their present state. This was illustrated earlier in Figure 4-193. This is to safeguard against any carry or borrow entering into the computation in B1 time, when CMPT4 again goes false. (Note that in adding or subtracting 6, there can never be a carry or a borrow produced in B0 time.) As noted earlier, V1CR3 is inhibited by B0T3, so in B0 time the state of 1CR is always copied onto the delay line.

(2) B1 Time. Starting in B1 time, when X3 becomes true, it is possible to produce a carry or a borrow. The same gates (Figure 4-193) are used to set ECB that are used during R1 time. During machine add, if there is a bit in 1CR during B1 time, ECB will set, providing a carry to B2 time. During machine subtract ECB will set if 1CR is reset during B1 time, denoting a borrow was necessary. As noted earlier, CMPT also goes true (CMPT4 is driven false) at the beginning of B1 time.

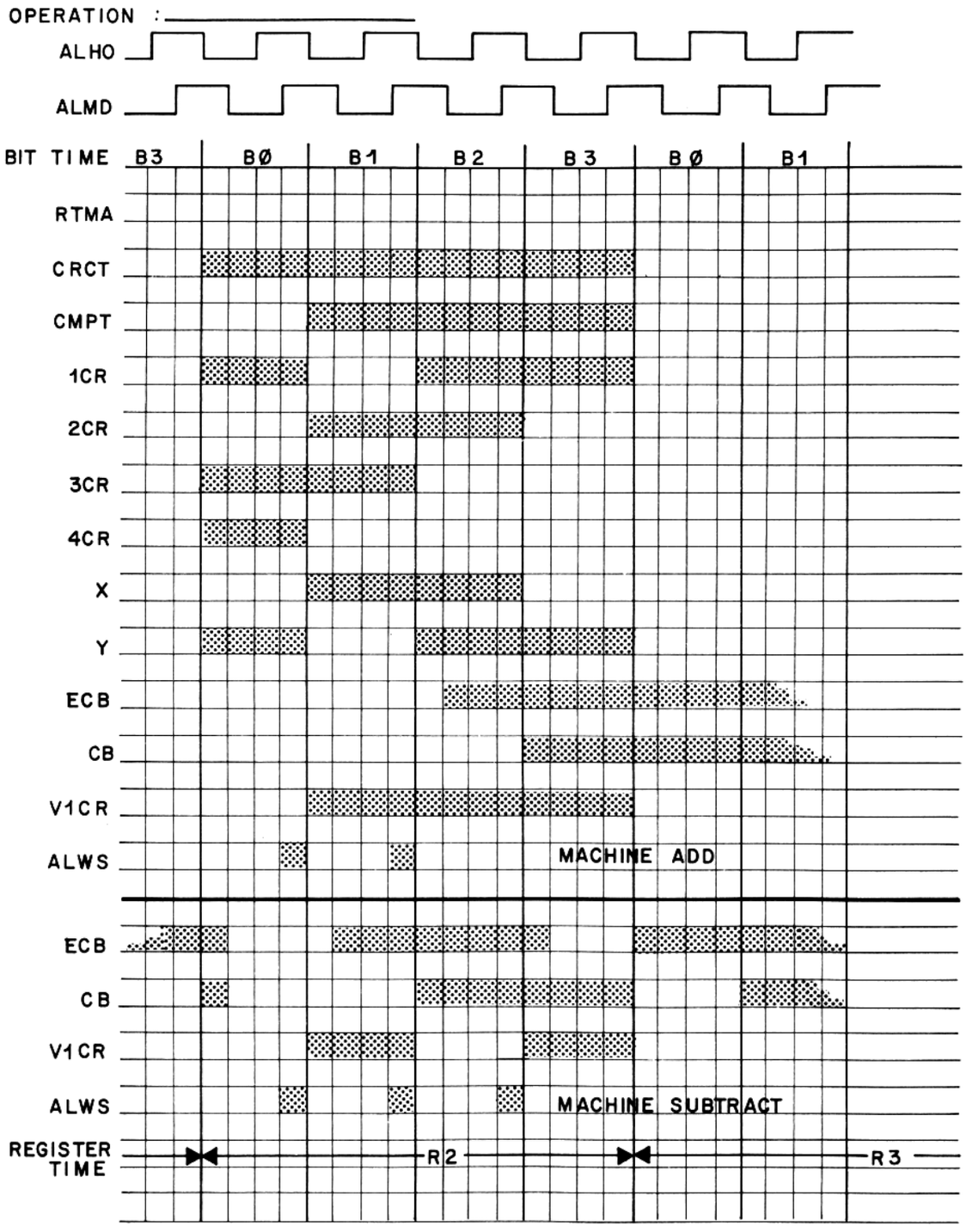
Because ECB is always reset in B0 time, CB will always be reset in B1 time. Since X3 is always true in B1 time, these two conditions always produce V1CR3 in B1 time. Therefore, it can be said that whenever a correction occurs, the state of the 2 bit in the uncorrected sum is always "inverted" when it is "written" onto the line.

(3) B2 Time. A carry or a borrow may also be produced in B2 time because X3 is still true. Note, however, that if ECB was set in B1 time, it cannot be reset in B2 time for the same reason - X3 is true. V1CR3 may or may not be true in B2 time, depending upon the state of CB.

(4) B3 Time. Since X3 is false in B3 time, ECB may reset: by the same token, it cannot set in B3 time. The condition of CB in B3 time is important in determining the state of V1CR3, but the state of ECB is of no importance at all, as will be explained next.

(5) Carry to the Next Column. Whenever a correction occurs, it tells one of two things: in machine add, it denotes that the addition in one arithmetic column resulted in that column's becoming "full" and carrying over into the next. In machine subtract, it denotes that a borrow from the next column was required in order to complete the subtraction in the first arithmetic column. This carry or borrow is transported to the next column by the CB flip-flop, the same as bit-to-bit carries and borrows are moved. Refer briefly to Figure 4-199 and note that CRCT resets at the end of R2 time, when AAB2 goes true. As soon as CRCT resets, it sets ECB (if it is not already set), as shown previously in Figure 4-193. This is the only time that ECB sets at "SH" time instead of "A" time. CB then sets at the next bit time (R3B1), and remains set through the remainder of the column time and into R1 time of the next column, where it enters into the computation. ECB and CB cannot be reset before R1 time of the next column because CMPT4 is no longer false. Note that it is possible for ECB to have been set during R2 time by the correction process: in this case, CB will set earlier than R3B1.

(6) Sample Problems. Two sample corrections, one for machine add and one for machine subtract, are illustrated in the timing chart in Figure 4-203. In both cases, the computation in R1 has produced an uncorrected answer of 13, which is found in the common register at the beginning of R2 time. The upper part of the chart shows those signals that are the same for both machine add and machine subtract: the lower part shows ECB, CB, V1CR, and ALWS for machine add first, and machine subtract last.



TIMING CHART
SAMPLE CORRECTIONS

FIGURE 4-203

TP-161SL

(a) Preliminary Conditions. CRCT sets at the beginning of R2 time. In machine add, MTTF3 would be true during R1B3 time: in machine subtract, ECB would be set in R1B3. 1CR, 3CR, and 4CR are set, denoting the uncorrected answer of 13.

(b) B0 Time. CMPT is false, so in both machine add and machine subtract the bit in 1CR is copied onto the delay line, shown as ALWS. ECB and CB receive a reset at B0 "A" time, even though they were already reset in the machine add example. The short time that CB is set in the machine subtract example is of no consequence because CMPT is always false during B0 time, and V1CR3 is inhibited by B0T3.

(c) B1 Time. In B1 time the 1 bit has shifted out of 1CR, and because there is no 2 bit, 1CR resets. At the same time X3 is driven true due to AAA2 action. Since ECB was reset in B0 time, CB cannot set in B1 time, and with X3 true, V1CR3 is developed. Since 1CR is reset, a bit is written onto the line. 1CR being set also makes the Y variable false, causing ECB to become set in the machine subtract example, although it is not affected in machine add.

(d) B2 Time. Because both X3 and Y3 are true in B2 time, ECB sets in the machine add example and remains set in machine subtract. In the latter case, ECB was set previously, so CB sets, preventing the development of V1CR and allowing the bit in 1CR to be written out. In machine add, since CB is reset, V1CR is developed, which "inverts" the bit in 1CR: no bit is written out.

(e) B3 Time. In B3 time X3 again goes false, and since CB is set in both cases, V1CR is true and the bit in CR is again "inverted" and no pulse is written out. The only difference between machine add and machine subtract in B3 time is that ECB resets during the latter because Y is true and X is false. Because of this, CB resets at the next bit time (R3B0). This is of no consequence, however, because ECB receives another set pulse at R3B0, as CRCT resets, and CB then sets at the following bit time (R3B1). When CRCT resets, CMPT is driven false (CMPT4 goes true), preventing any further variations in the normal data flow until the next R1 time: V1CR3 is blocked by CRCT2 and VRTA3 is blocked by CMPT4. Note that X3 and Y3 can still be developed, but without CMPT, they have no effect.

Still referring to Figure 4-203, note that the state of RTMA has no effect on the computation in R2 time: it is merely sensing the R2 data from the line and shifting it into the common register, to be written out in R3 time.

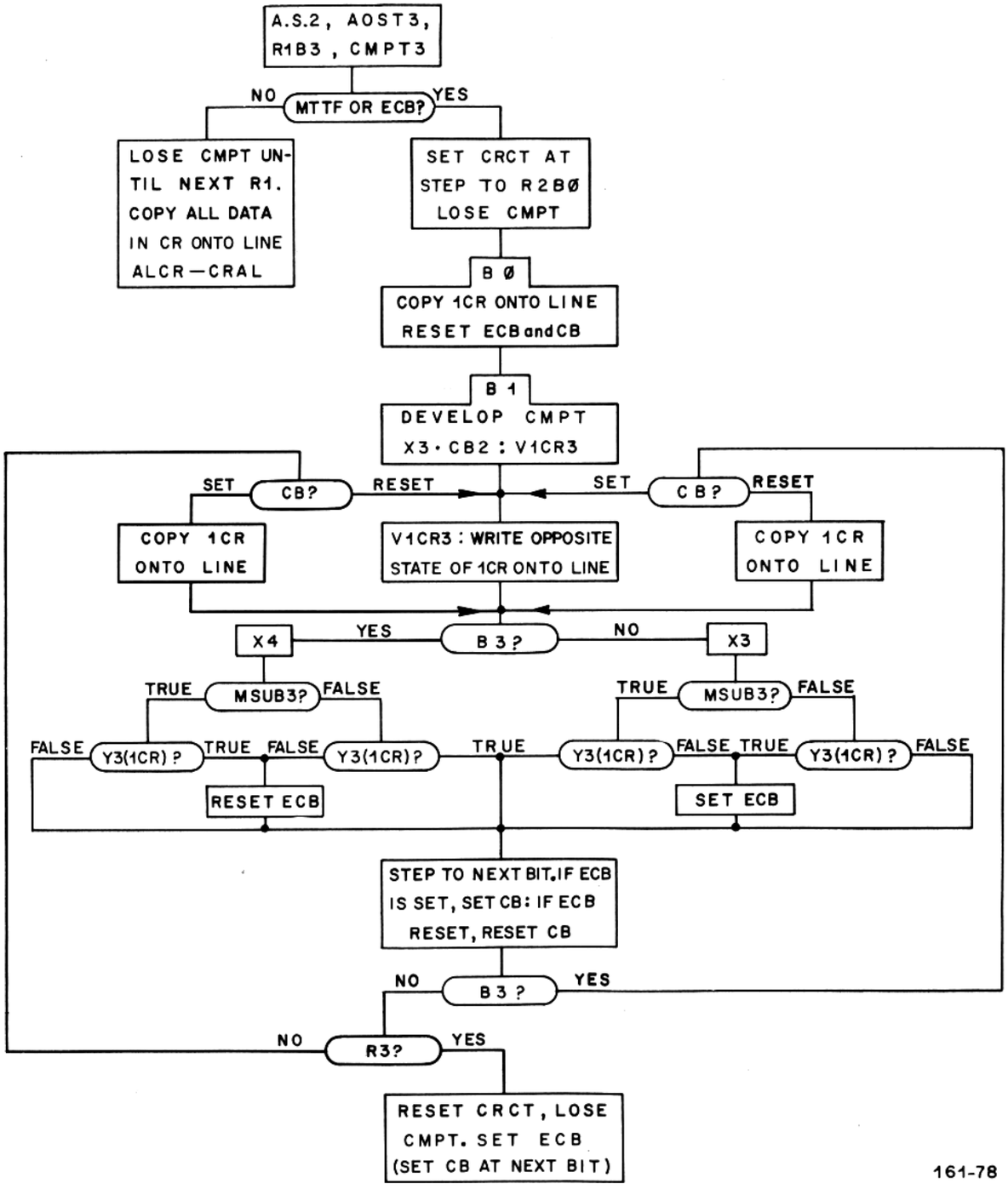
d. FLOW CHART. The flow chart in Figure 4-204 summarizes the entire correction process.

10. OVERFLOW

The add or subtract process is usually completed in one line pass in action step 2, after which the action step counter returns to step 0, and the next arithmetic sync pulse sets DOST. Following this the phase counter steps to 7 phase (at the next T6B), and the next program line home causes a jump to 3M6,4 to locate the next instruction.

If, however, the computation in arithmetic column 14 results in a correction, ECB and CB will be set at the end of the line pass. When this occurs, one more line pass must be made in order to reset both flip-flops. During machine add this line pass has no other function: during machine subtract it is used to complement an overdraft and change the sign.

When a correction occurs in column 14 during machine add, it indicates that the addition of the two MSD's has produced a carry. When this occurs, there is no outward sign to tell the 5610 operator that an overflow has occurred: this last carry is lost and the number remaining in R1 is incorrect in this respect.



161-78

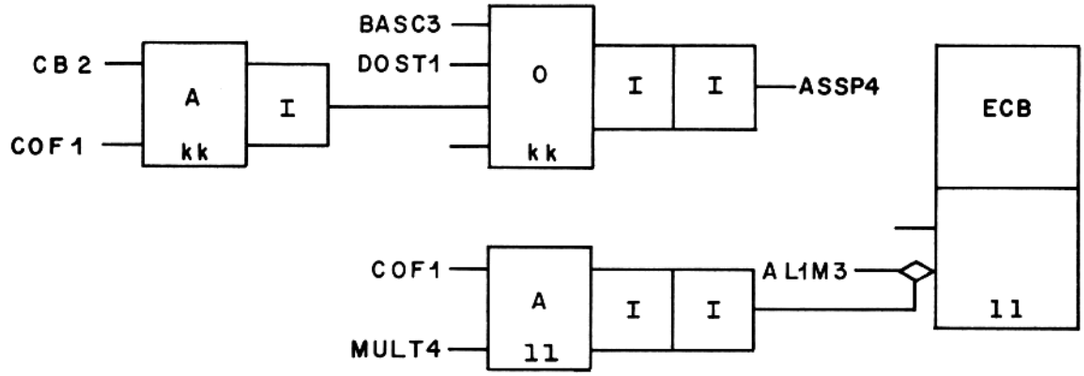
FLOW CHART
AOS CORRECTION

FIGURE 4-204

TP-161SL

During machine subtract, when one digit is subtracted from a smaller digit in any one column, the logic automatically borrows from the digit in the next column so that the subtraction can take place. Then, when the next column is operated upon, this borrow causes "one" more to be subtracted. If a borrow is necessary to allow the subtraction in column 14, it means that the entire number in R0 is larger than the number in R1, and CB remains set at the end of the line pass. The "difference" in R1 at this time is termed an "overdraft", and this number must be complemented and its sign changed in order to make it a negative number. (It is also possible for an overdraft to have a negative sign, in which case its complement receives a positive sign.)

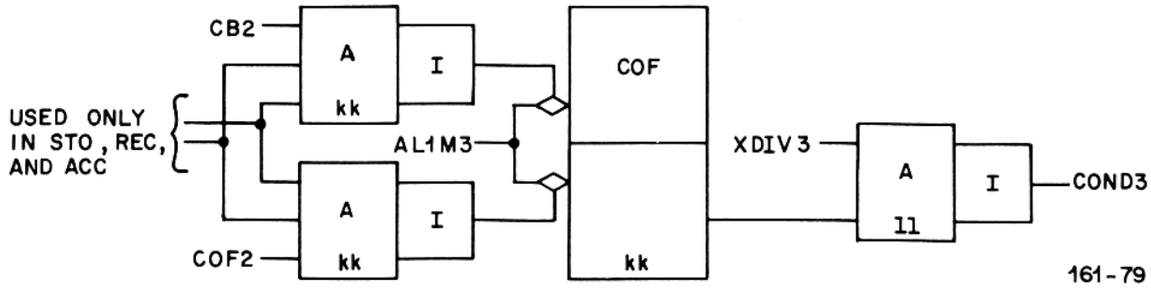
a. GENERAL OPERATION. Refer to Figure 4-205 and note that with CB set in A15 time, ASSP4 is blocked and the action step counter remains in step 2. At arithmetic line home, ECB resets (also shown in Figure 4-205), and the next time ALHO3 goes true (when the timing chain re-starts), CB resets.



OVERFLOW LOGIC

FIGURE 4-205

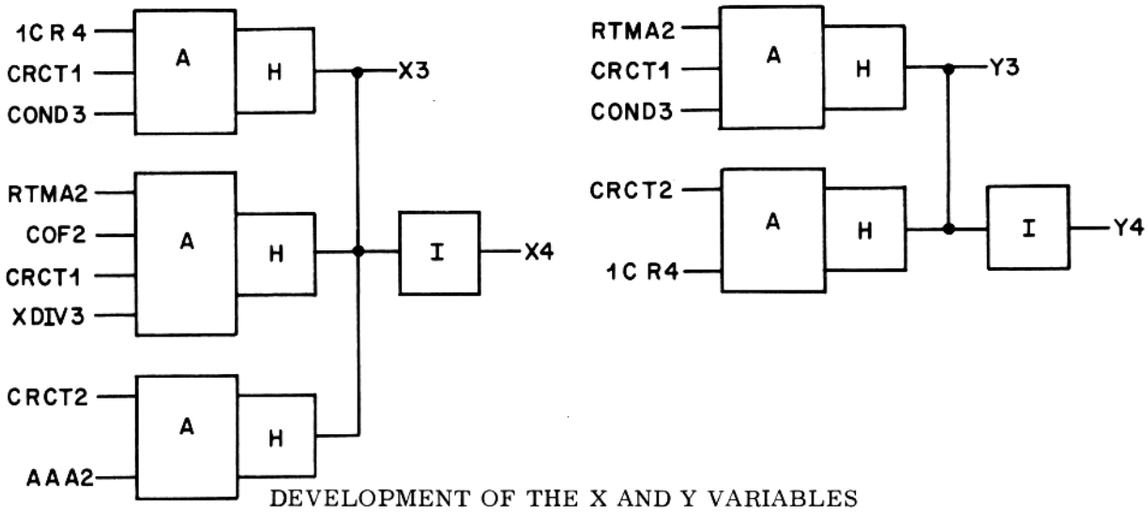
Note in Figure 4-206, however, that the same arithmetic line home that resets ECB also sets the Carry OverFlow flip-flop, COF. This flip-flop is used in the re-complement/change sign line pass in machine subtract: in machine add, its only function is to prevent the adding of R0 to R1 a second time. It does this by developing the Carry Overflow Not Divide signal, COND3.



COF LOGIC

FIGURE 4-206

As in "regular" machine add and subtract, the development of the X and Y variables is the same for either machine add or machine subtract during the overflow line pass. All of the gates developing X and Y are brought together in Figure 4-207. Note that Y3 can become true only during a correction. X3 will still add 6 during a correction, and also become true whenever RTMA sets during the time COF is set. This is needed for machine subtract.



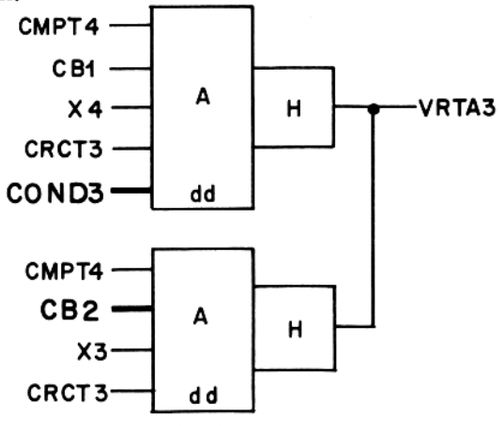
DEVELOPMENT OF THE X AND Y VARIABLES
(All gates on "LL" card)

FIGURE 4-207

b. MACHINE ADD. The second line pass that results from an add overflow has no effect on the answer produced. It is necessary during machine subtract and during the addition that occurs in Multiply: it is allowed to occur in add because it does no harm.

During the second line pass, CMPT is developed the same as before, during R1 time in each column, A2 through A14. However, because Y3 is inhibited by COND3 (Figure 4-207), ECB cannot set and there can be no carries. Since ECB cannot set, CB cannot set, so CB2 remains true, effectively blocking one of the gates producing VRTA3, as shown in Figure 4-208. COND3 inhibits VRTA3 from the other gate. Thus, with VRTA and CB both blocked throughout the line pass, all data sensed by RTMA is duplicated into 6CR: the normal ALCR, CRAL line routing is followed, and ALL data is rewritten unchanged.

At the end of the second line pass, COF being set allows the action step counter to return to 0 (see Figure 4-205), and COF resets simultaneously (Figure 4-206). This is followed by termination of the add instruction.



BLOCKING VRTA DURING OVERFLOW
(Both gates on "DD" card)

FIGURE 4-208

c. MACHINE SUBTRACT. As noted previously, the only difference between machine add and machine subtract is in the handling of the carries and borrows. Speaking from the "logic" standpoint, the difference lies in the setting and resetting of ECB. During an add overflow, ECB remains reset and no data is changed. During a subtract overflow the fact that ECB can set is of prime importance: the one gate that sets ECB during machine subtract provides the key to the difference between the two operations.

(1) General Operation. With CB set at the end of the first subtract line pass, the action step counter is prevented from stepping, the same as during machine add, and one more line pass occurs. It is during this second line pass that the sign of the "difference" is changed, and the 10's complement of the difference appears in R1. The principle involved in this change sign/complement process is illustrated in Figure 4-209. In actual practice, the sign is changed BEFORE the complement is taken: the sign does not affect the complementing process.

FIRST LINE PASS	SECOND LINE PASS	
Subtract	Complement	Change Sign
4	10	
-8	-6	
6	4	-4

CHANGE SIGN/COMPLEMENT PROCESS

FIGURE 4-209

It is interesting to note that in complementing a number, the correct "answer" is achieved by taking the 10's complement of the LSD and the 9's complement of all other digits. An example is illustrated in Figure 4-210. In order to complement the LSD "3", it must be subtracted from 10. Since there is a zero in the same column, above the 3, a borrow must be made from the next higher column in order to allow the subtraction. When this first borrow is made, it changes all previous zeros to nines, as shown in Figure 4-210 (b). Subtraction in subsequent columns in Figure 4-210 (b) requires no additional borrowing, but it should be remembered that the 5610 operates on only one column at a time: the original borrow, which converts the zero to a nine, must be transferred from each column to the next, via CB, during the complement line pass.

```
(a)  (1) 0 0 0 0 0        (b)  9 9 9 9 (10)
         9 9 9 4 3                9 9 9 4  3
         0 0 0 5 7                0 0 0 5  7

                                                  ← Time
(c)  (1) 0 0 0 0 0 0 0 0.0 0 0 0 0
         9 9 9 9 9 7 4 9.5 8 0 0 0
         0 0 0 0 0 2 5 0.4 2 0 0 0 (-)
```

SAMPLE COMPLEMENTS

FIGURE 4-210

An additional example, which more closely represents actual 5610 operation, is illustrated in Figure 4-210 (c). Note that the first action to occur is the sign change in arithmetic column 1. No data changes in A2, A3, or A4, because only zeros are present. In A5 time, however, the 10's complement of 8 is written into the R1 time slot, and in each succeeding column, A6 through A14, the 9's complement is written out.

(2) Complement Rules. Since the four-digit BCD columns are capable of retaining counts up to 15, the 10's and 9's complements cannot be computed directly. Instead, the 16's and 15's complements are taken, respectively, and the correction process of subtracting 6 is utilized to convert these answers to the 10's and 9's complements.

The rules for complementing are as follows:

1. 9's complement: invert all data, then subtract 6.
2. 10's complement: write all zeros and the very first bit (the least significant bit) as they appear; invert all subsequent data, and then subtract 6.

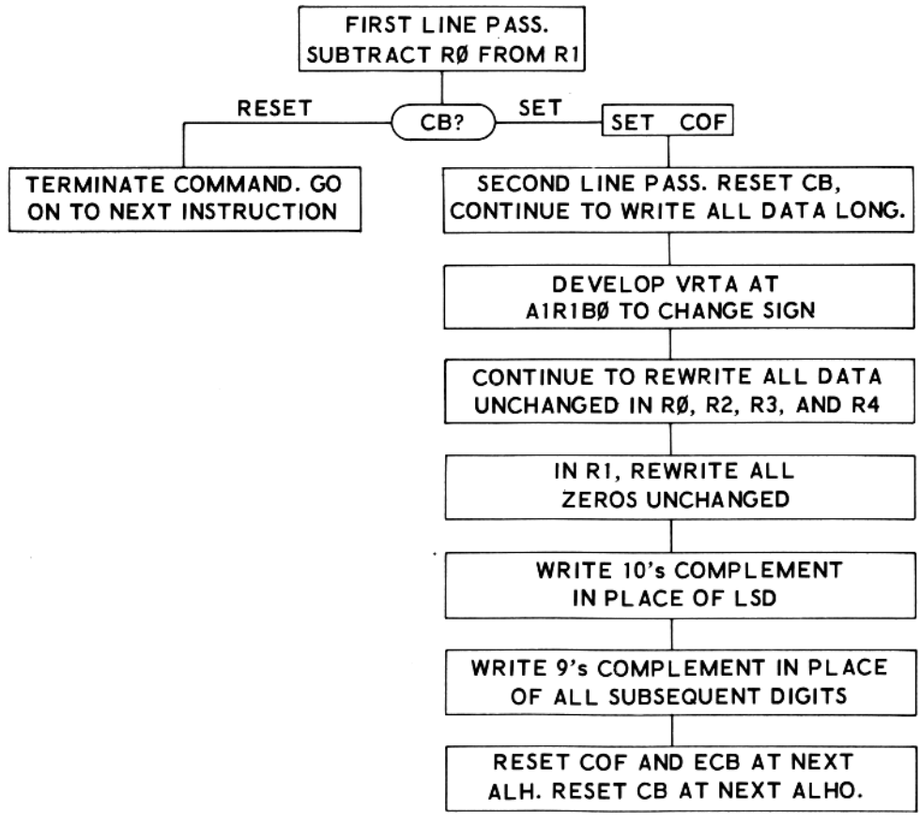
Figure 4-211 shows examples of the preceding rules. These rules can be used to work out the complement of any BCD digit, 0 through 9.

9's Complement DECIMAL	B C D 8 4 2 1		10's Complement DECIMAL	B C D 8 4 2 1
6	0 1 1 0	Original Number	6	0 1 1 0
	1 0 0 1	Uncorrected Complement		1 0 1 0
	-0 1 1 0	Correction		-0 1 1 0
3	0 0 1 1	Complement	4	0 1 0 0
2	0 0 1 0	Original Number	2	0 0 1 0
	1 1 0 1	Uncorrected Complement		1 1 1 0
	-0 1 1 0	Correction		-0 1 1 0
7	0 1 1 1	Complement	8	1 0 0 0

APPLICATION OF COMPLEMENT RULES

FIGURE 4-211

(3) Concept Flow Chart. The flow chart in Figure 4-212 summarizes the basic operation of the change sign/complement process.

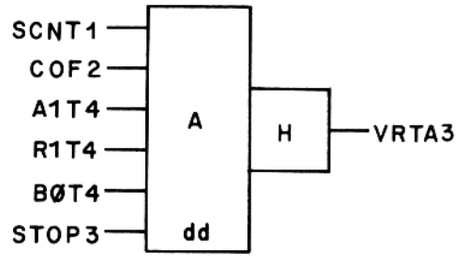


BASIC FLOW CHART
CHANGE SIGN/COMPLEMENT

FIGURE 4-212

(4) Logic. The Carry OverFlow flip-flop logic was shown previously in Figure 4-206. COF sets at arithmetic line home after the first line pass (when CB is set) and it resets after the second line pass. The ECB flip-flop resets at arithmetic line home, at the same time that COF sets. This was shown in Figure 4-205. The next time ALHO3 goes true (A0R0B1), CB resets. CB then remains reset until after ECB again sets: this occurs when the first data bit is sensed in R1 time, as will be explained shortly.

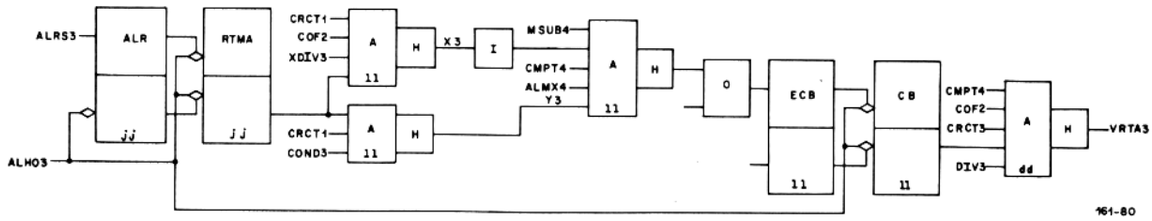
(a) Change Sign Logic. The very same gate is used to change the sign as is used during the change sign command. See Figure 4-213. SCNT is reset (this allowed the original machine subtract to occur: see Figure 4-185), and COF is set, allowing VRTA3 to be developed at A1R1B0 time. Note that during add overflow, SCNT1 prevents the development of VRTA3.



INVERTING THE SIGN BIT

FIGURE 4-213

(b) 10's Complement. In order to follow the development of the 10's complement, certain factors must be understood. Refer to Figure 4-214. First, Y3 is held false throughout the line pass (except during correction) by COND3. Second, 1CR has no effect on X3, for the same reason. Last, X3 is developed each time a bit is sensed coming off the line, setting RTMA (except during a correction). This latter fact is of value only in sensing the first data bit off the line in R1, for when this occurs, the combination of X without Y sets ECB. CB then sets at the next bit time (ALHO3), and with CB and COF both set, VRTA3 is produced for the remainder of R1 time.



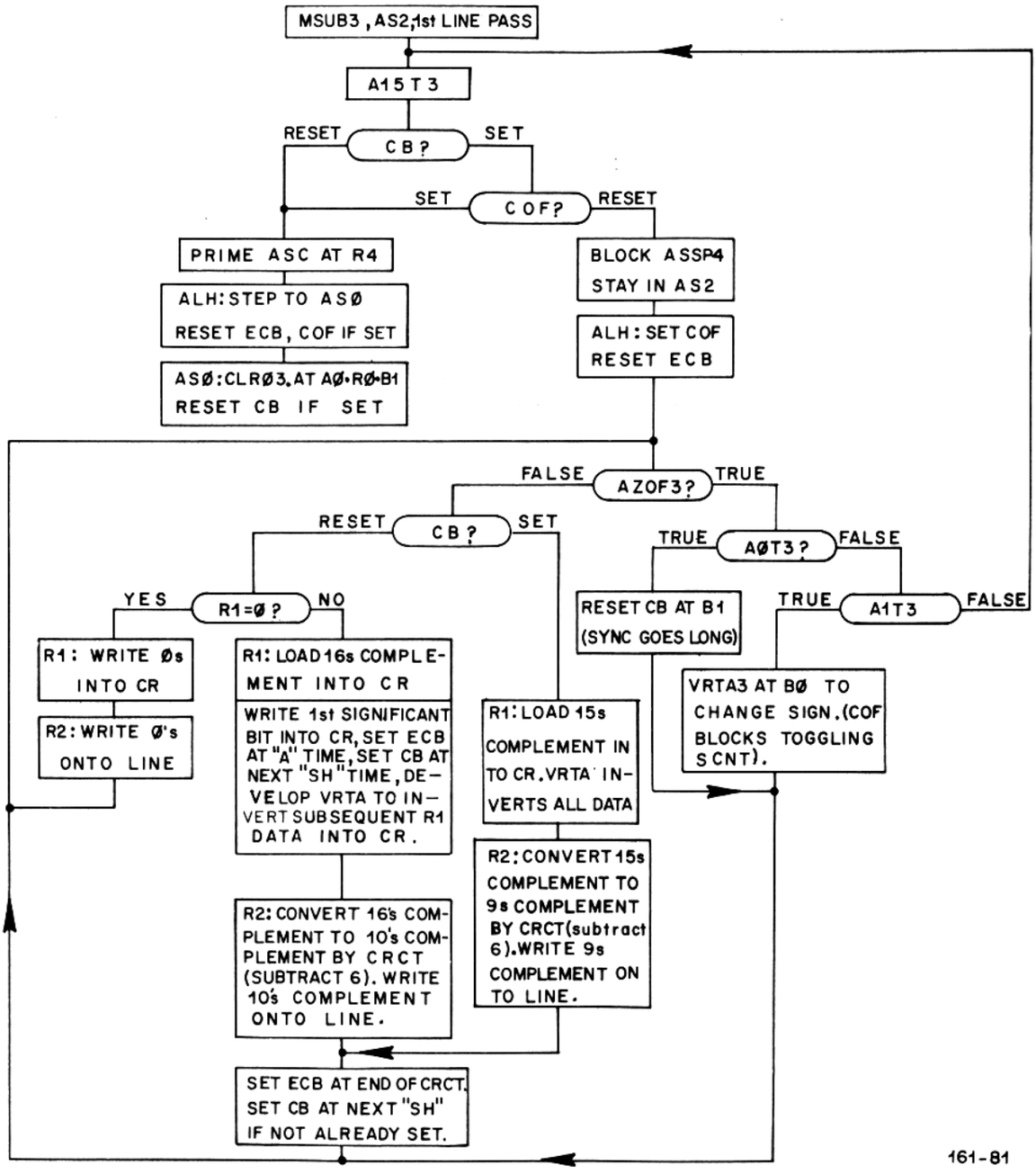
10's COMPLEMENT LOGIC

FIGURE 4-214

(c) 9's Complement. Notice that once ECB sets (at the first data bit in R1), it cannot be reset in R1 time because Y3 is held false, so a correction occurs, subtracting 6. ECB and CB become set by the correction, so each time CMPT4 goes false in R1 time, VRTA3 is produced (as in Figure 4-214) and all data is inverted. In other words, CB is set throughout R1 time and after the R2 correction: it may be reset during the correction, but it always becomes set at the end of the correction. The only time ECB and CB can reset during the change sign/complement line pass is during the correction in R2.

(5) Terminating The Complement Process. At the end of the second line pass COF and CB are both set, allowing the action step counter to return to step 0. At the same time, COF and ECB reset, and CB resets at the next ALHO3. The sync pulse resets DOST, allowing the phase counter to step to 7 phase. The first arithmetic line pass in action step 0 also clears all old data out of R0.

d. FLOW CHART. A flow chart depicting the complete change sign/complement line pass is illustrated in Figure 4-215.



FLOW CHART
SUBTRACT OVERFLOW

FIGURE 4-215

11. SUPPLEMENTARY INSTRUCTIONAL AIDS

Most of the rules and facts pertaining to add and subtract are covered in the following charts, forms, etc., which may be used in studying the computing process.

a. AOS TRUTH TABLE (FIGURE 4-216). This truth table is simply another way of summarizing the facts presented in Figure 4-191, the "eight possible conditions". It shows that when X and CB are in opposite logic states, VRTA is produced; when X and Y are the same during machine add, ECB is set, and so on. This table can be used to check the accuracy of the timing chart blank after it is filled out.

POSSIBLE CONDITIONS					ADD	SUBTRACT		
X(1CR)	Y(RTMA)	CB	VRTA	SUM	CARRY	ECB		BORROW
0	0	0	F	0	No	Resets	–	No
1	0	0	T	1	No	–	Resets	No
0	1	0	F	1	No	–	Sets	Yes
1	1	0	T	0	Yes	Sets	–	No
0	0	1	T	1	No	Resets		Yes
1	0	1	F	0	Yes	+	Resets	No
0	1	1	T	0	Yes	+	Sets	Yes
1	1	1	F	1	Yes	Sets	+	Yes

– Denotes ECB remains Reset + Denotes ECB remains Set

AOS TRUTH TABLE

FIGURE 4-216

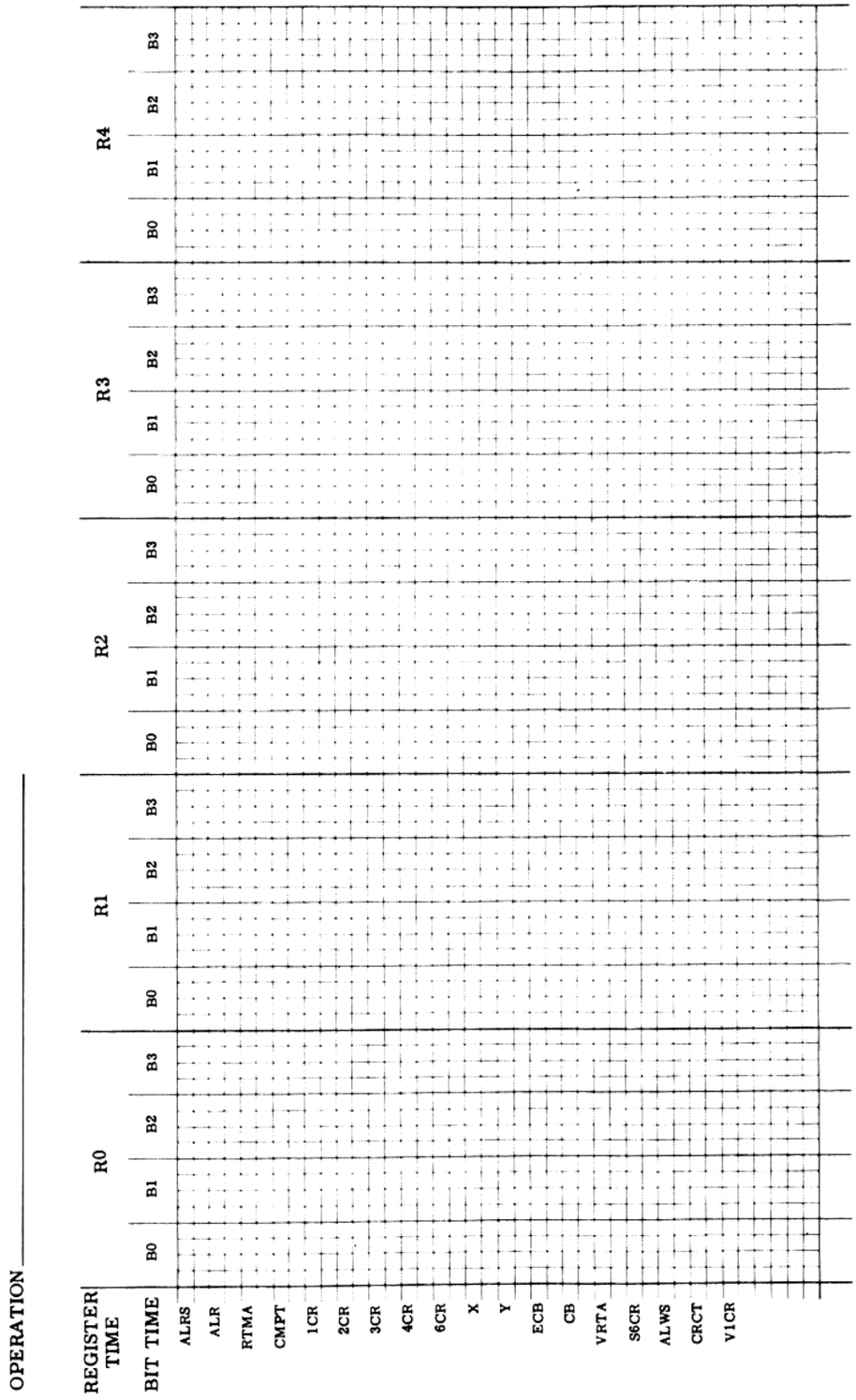
b. MATH TIMING CHART BLANK (FIGURE 4-217). This blank is used for working add or subtract problems. It is helpful mainly in learning the way in which the 5610 computes, rather than in troubleshooting. The numbers (polarity indicators, or logic state indicators) have been left off signal names so that only "true" signals need be drawn. There is room for one additional signal to be drawn at the bottom. This could be 6CR, MTTF, or any other signal that might be chosen to see its relationship to the signals listed. Filled-out samples of this blank were illustrated previously in Figures 4-196 & 4-197. In practice this blank can be filled out with either a soft black pencil, a felt or bamboo marking pen, or colored pencil.

c. AOS CHECKLIST (Figure 4-218). The Add or Subtract Checklist is used with the timing chart blank. By accurately filling out the timing chart blank in conjunction with each step on the checklist, one can develop a chart of the computing process for any single arithmetic column.

d. RULES FOR AOS (FIGURE 4-219). The chart listing the Rules for Addition and Subtraction can also be used in conjunction with the timing chart blank. It gives some clue as to the development of each signal listed on the timing chart blank: what the signal represents, what is required to make the signal true, etc.

e. CORRECTION TRUTH TABLES (FIGURES 4-220 & 4-221). There are two truth tables for correction; one for machine add and one for machine subtract. These tables would become rather complicated if they were to be combined. The machine add truth table lists the ten possible sums that are > 9, the condition (MTTF or ECB) that causes the correction, and the state of ECB, CB, and V1CR for each bit time in R2. The machine subtract table lists the true decimal answer and the uncorrected decimal answer for each of the possible "differences" requiring a borrow from the next column. Note that ECB is set in R1B3 in all cases, regardless of the state of MTTF. These tables can also be used to test the accuracy of any timing charts prepared.

f. AOS FLOW CHART (FIGURE 4-222). A flow chart covering the overall operation of the Add and Subtract instructions is illustrated in Figure 4-222. This flow chart ties together the different sections of add and subtract that have been charted earlier: refer to the other flow charts for details not covered on this chart.



MATH TIMING CHART BLANK
FIGURE 4-217

1. Do problem on paper (use binary 1 & 0) before doing timing chart.

2. The problem is done bit-by-bit: do not extend a pulse past the bit time you are checking unless such is noted.

3. Do RTMA for the entire chart. This is determined by what is coming off the line (your choice of the problem).

4. For R1 time:
   a. R0 data will be rewritten unchanged.
   b. Draw CMPT waveform for this R-time.
   c. Do 1CR, 2CR, 3CR, and 4CR for this bit time.
   d. Determine X and Y.
   e. Is CB set? (Was ECB set previously?)
   f. Does ECB set or reset?
   g. Do VRTA waveform.
   h. Do S6CR waveform.
   i. Fill out ALWS (check 1CR).
   j. Increment the bit counter at ALHO and return to step "c".

5. For R2 time:
   a. Refer to R1B3 time and check for MTTF or ECB.
   b. Is CRCT set? If CRCT1, go on to "c"; if CRCT2, return to 4b.
   c. Do CMPT waveform for entire R2 time.
   d. Determine X for the entire R2 time. (Add binary six, 0110.)
   e. Do 1CR, 2CR, 3CR, and 4CR for this bit time.
   f. Determine Y.
   g. Is CB set? (Was ECB set previously?)
   h. Does ECB set or reset? (Note reset circuit in B0 time.)
   i. Do V1CR waveform.
   j. Accomplish ALWS waveform.
   k. Do S6CR waveform.
   l. Increment the bit counter at ALHO and return to step "e".

AOS CHECKLIST

FIGURE 4-218

	COMPUTE	CORRECTION	OVERFLOW OR COMPLEMENT
	R1 TIME**	R2 TIME***	R1 TIME
RTMA	R1 Data	R2 Data	R1 Data
CMPT	B0-B3	B1-B3	B0-B3
1CR	R0 Data	Uncorrected Sum	R0 Data
"X"	1CR	"6"	RTMA
"Y"	RTMA	1CR	False
ECB ("A" time)	(ADD) Sets if X3 and Y3 Resets if X4 and Y4 Does not change if X3 and Y4 or X4 and Y3	(SUBTRACT) Sets if X3 and Y4 Resets if X4 and Y3 Does not change if X3 and Y3 or X4 and Y4	
CB	Sets if ECB is set Resets if ECB is reset		
VRTA	X3 and CB2 X4 and CB1	– – – – – –	CB1
S6CR ("A" time)	RTMA1 and VRTA4 RTMA2 and VRTA3	RTMA (R2 Data)	RTMA1 and VRTA4 RTMA2 and VRTA3
ALWS ("A" time)	1CR (R0 Data)	1CR1 and V1CR4 1CR2 and V1CR3	1CR (R0 Data)
CRCT	– – – – – –	B0-B3	– – – – – –
V1CR	– – – – – –	X3 and CB2 X4 and CB1 (Never in B0) (Always in B1)	– – – – – –

* ECB & CB reset if CRCT1 at B0A in R2 and R3.
** CMPT is also true in R2 during correction, and in R3 during mult overflow if CRCT1.
*** CRCT can also be true in R3 during mult overflow.

RULES FOR AOS

FIGURE 4-219

POSSIBLE SUMS					R1B3		R2BØ			R2B1			R2B2			R2B3			CORRECTED SUM			
DEC	B3	B2	B1	BØ	ECB	MTTF	CB	ECB	V1CR	CB	ECB	V1CR	CB	ECB	V1CR	CB	ECB	V1CR	B3	B2	B1	BØ
10	1	0	1	0	0	1	0	0	0	0	1	1	1	1	0	1	1	1	0	0	0	0
11	1	0	1	1							1		1		0				0	0	0	1
12	1	1	0	0							0		0		1				0	0	1	0
13	1	1	0	1							0		0		1				0	0	1	1
14	1	1	1	0							1		1		0				0	1	0	0
15	1	1	1	1	0	1					1		1	1	0	1	1	1	0	1	0	1
16	0	0	0	0	1	0					0		0	0	1	0	0	0	0	1	1	0
17	0	0	0	1							0		0	0	1	0		0	0	1	1	1
18	0	0	1	0							1		1	1	0	1		1	1	0	0	0
19	0	0	1	1	1	0	0	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1

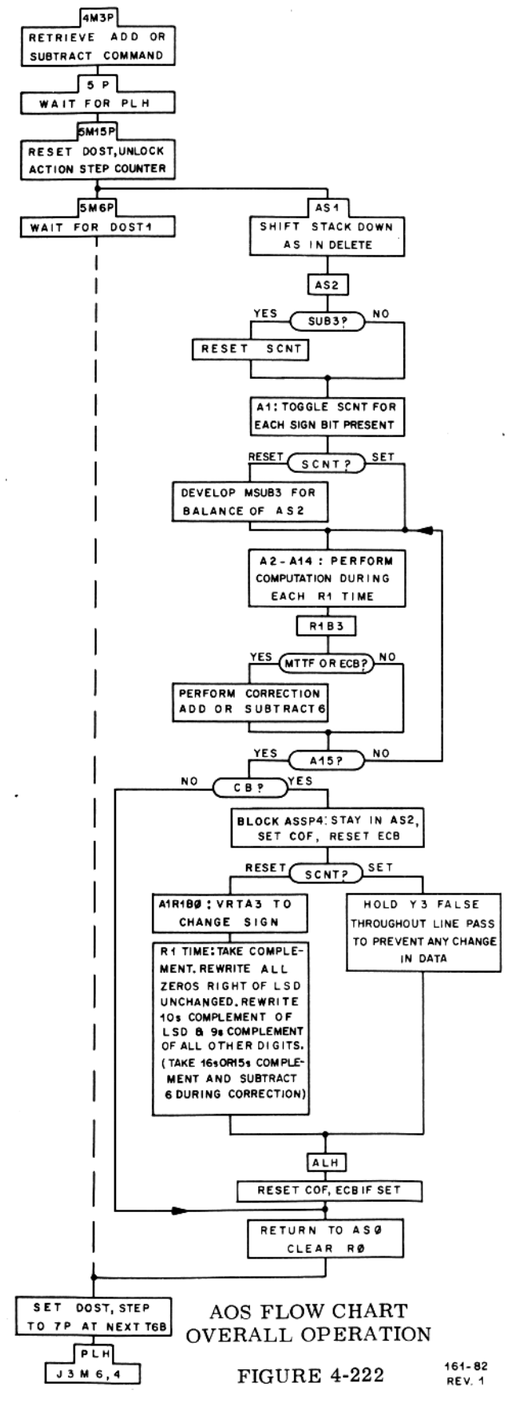
TRUTH TABLE FOR CORRECTION
(MACHINE ADD)

FIGURE 4-220

POSSIBLE SUMS					R1B3		R2BØ			R2B1			R2B2			R2B3			CORRECTED SUM			
DEC	B3	B2	B1	BØ	ECB	MTTF	CB	ECB	V1CR	CB	ECB	V1CR	CB	ECB	V1CR	CB	ECB	V1CR	B3	B2	B1	BØ
(-10) 6	0	1	1	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
(-9) 7	0	1	1	1							0		0	0	1	0		0	0	0	0	1
(-8) 8	1	0	0	0							1		1	1	0	1		1	0	0	1	0
(-7) 9	1	0	0	1		0					1		1		0				0	0	1	1
(-6)10	1	0	1	0		1					0		0		1				0	1	0	0
(-5)11	1	0	1	1							0		0		1				0	1	0	1
(-4)12	1	1	0	0							1		1		0				0	1	1	0
(-3)13	1	1	0	1							1		1	1	0	1		1	0	1	1	1
(-2)14	1	1	1	0							0		0	0	1	0		0	1	0	0	0
(-1)15	1	1	1	1	1	1	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1

TRUTH TABLE FOR CORRECTION
(MACHINE SUBTRACT)

FIGURE 4-221



AOS FLOW CHART
OVERALL OPERATION

FIGURE 4-222

161-82
REV. 1

L. MULTIPLY OR DIVIDE

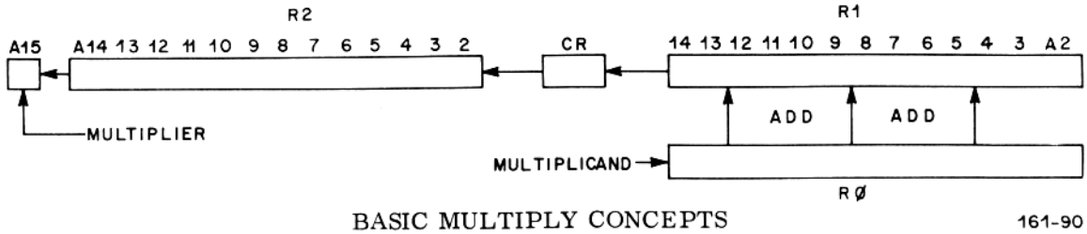
There are many similarities between multiply and divide, just as there are between add and subtract. For this reason it is easier to cover the two instructions together, splitting up the discussion only when the operations vary significantly. Unlike the add and subtract functions, multiply and divide require more than just the first two action steps in order to be completed: they are "Five Phase Op Codes", requiring all five action steps. Also, many of the action steps are repeated: steps 1 and 5 are performed only once, but steps 2, 3, and 4 are used repeatedly. Unlike add or subtract, the Multiply and Divide instructions ALWAYS cause their respective machine functions: the signs are used only to determine the sign of the answer.

1. GENERAL OPERATION

The retrieval and basic operation of the Multiply and Divide instructions is very similar to that of the other arithmetic instructions. The command is retrieved in 4 mode and causes a jump to 5 mode at program line home. Once 5 mode is reached, DOST resets and the action step counter unlocks, allowing the arithmetic action to take place. When this is finished, the action step counter returns to 0, DOST sets, and the next T6B allows a step to 7 phase. At program line home a jump to 3M6,4 occurs.

Once the action step counter unlocks in 5 mode and allows a step to action step 1, the signs of R1 and R2 are added, the sum appearing in R2. This becomes the sign of the answer. (ECB is blocked so nothing greater than a BCD 1 can appear in R2.) Then the contents of R0 and R1 are transposed: since R0 had been clear, R1 now becomes clear. This action is identical in both multiply and divide.

a. MULTIPLY. Refer to Figure 4-223. After R0 and R1 are transposed, R2 is shifted left and its MSD is shifted into A15. R0 is then added to R1 the number of times specified by this multiplier digit in A15R2: after each add line pass, the number in A15R2 is counted down once. These repeat adds continue until A15R2 is counted down to zero. Next R1 is shifted left, effectively multiplying the intermediate product in R1 by 10. Then R2 is shifted left again to move the next multiplier digit into A15. Another series of repeat adds occurs, followed by another shift of R1. Notice that each time R1 is shifted left its MSD remains in the common register at the end of the line pass. Since R2 is shifted left on the very next line pass, this digit is shifted into A2R2. Thus R2 is used as an "extension" of R1, allowing the product to be extended into R2 as it is developing in R1. Notice also that it is possible to have a carry developed in A14R1 during the repeat adds: when this occurs the carry is transported to A2R2. Thus R2 functions as a part of R1. (Since the entire multiplier is shifted left prior to the first shift of R1, there is no chance of mixing the two numbers.)

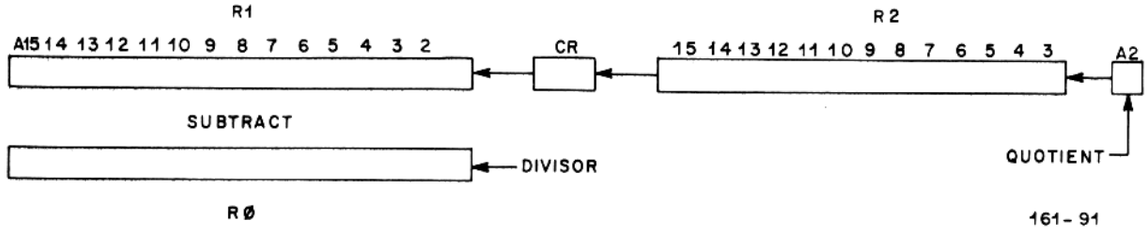


BASIC MULTIPLY CONCEPTS 161-90

FIGURE 4-223

After 13 pairs of shifts occur, one for each arithmetic column, the multiplication is complete and the product is fully developed. It is, however, decimally incorrect and must be aligned. When multiplying two numbers having 5 digits right-of-decimal, the product uses 10 digits to the right. In the 5610, since the capacity is 5 right, the 5 farthest right digits must be ignored. Once the product is fully developed, it would only be necessary to shift right 5 times to accomplish the correct decimal alignment and to "sluff" the 5 farthest-right digits. However, because the delay line data cannot be shifted right, 8 more left shifts occur, resulting in the product's being decimally aligned in R2. The final step, then, is to shift the stack down so that the product appears in R1.

b. DIVIDE. Refer to Figure 4-224. Just as multiply is a series of repeat additions, divide is a series of repeated subtractions. R2 is also used as an extension of R1 in divide, but it "extends" R1 on the opposite side, in the LSD direction.

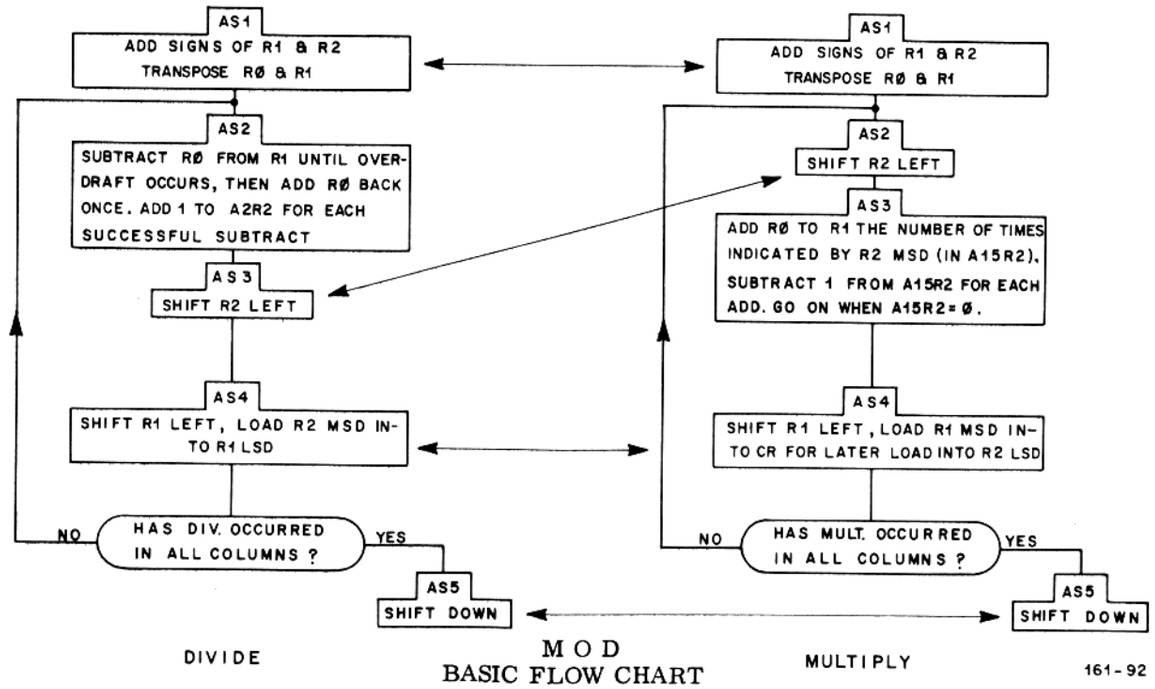


BASIC DIVIDE CONCEPTS

FIGURE 4-224

After R0 and R1 are transposed, R2 is shifted left, followed by a shift left of R1. This shifts the dividend out of R2 and into R1, one digit at a time. After each digit is shifted into R1, a test subtraction is made. If this test results in an overdraft, R0 is added back into R1 to rescind the overdraft and another pair of shifts follows. This process repeats until enough R2 digits are shifted into R1 to allow R0 to be subtracted successfully (with no overdraft). For each successful subtract, the R2 LSD (A2R2) is counted up once. After all possible successful subtracts are performed, another overdraft occurs and R0 is added again to rescind the overdraft. Then another pair of shifts occurs and the next digit of R2 is shifted into R1. This shift of R2 also shifts the quotient, making room for its further development. Further repeat subtracts are followed by more shifts, until the entire operation is complete. The last step, as in multiply, is to shift the stack down.

One additional item used in divide is the logic that checks for the presence of a number in R1 before the divide actually begins. If R1 is clear, a jump to 8 phase occurs and the red "error" lamp is illuminated. When this occurs, the Reset switch must be depressed, with the resultant loss of the program sequence. Thus it is impossible to divide by zero.



M O D
BASIC FLOW CHART

FIGURE 4-225

c. COMMON FACTORS. Refer now to Figure 4-225 and note the factors that are common to both functions. Some of these have already been mentioned, such as the adding of the signs and the transposition of R0 and R1 in action step 1, and the final shift down in action step 5. The shift left of R2 in action step 2 during multiply is the same as that in action step 3 during divide. The shift left of R1 in action step 4 is identical in either case, although the reasons are slightly different: during multiply, R1 is shifted left in order to transfer the R1 MSD into the R2 LSD position; during divide, R1 is shifted left in order to transfer the R2 MSD into the R1 LSD position.

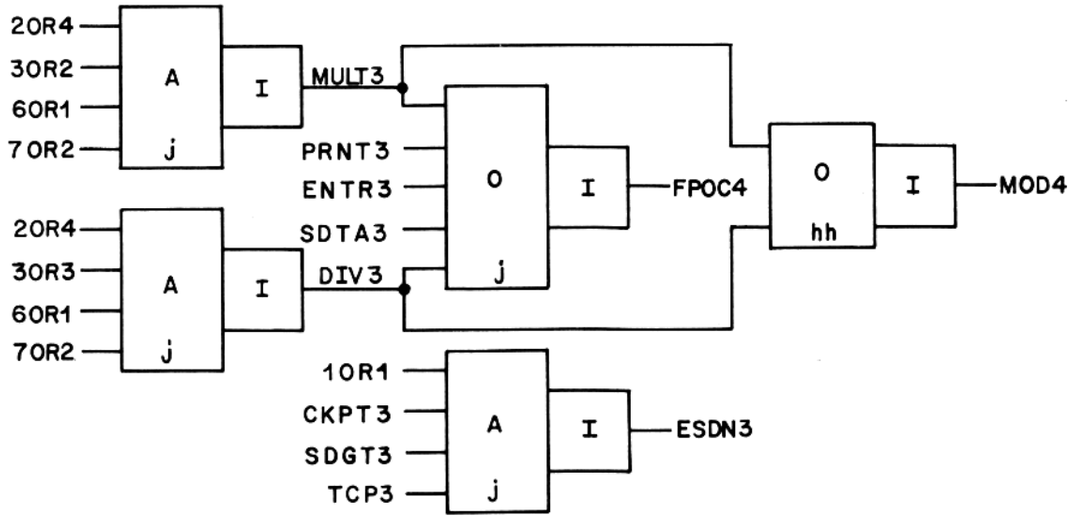
In both multiply and divide the decimal register is used to keep track of the number of columns that have been operated upon: after each shift left of R1 the decimal register is counted down.

There are only four processes used in multiply and divide that have not been used in other arithmetic instructions. Three of them, (1) the transposition of R0 and R1, (2) the shifting left of R2, and (3) the counting up or counting down of a number existing on the delay line, are mentioned in the flow charts in Figure 4-225. Once these three processes have been covered, multiply and divide are simply matters of rearranging these processes, along with others already covered. The fourth process, noted previously, is that which blocks division when R1 is clear.

2. INSTRUCTION RETRIEVAL

The Multiply and Divide commands are retrieved in 4 mode 3 phase, and subsequent 4 mode operation is the same as for any other 0-address command: a jump to 5 phase follows retrieval, and at program line home a jump to 5 mode 15 phase occurs. Remember that the decimal register is set to 15 in 4 mode 3 phase: the importance of this step will be seen shortly.

As soon as the Multiply or Divide command is fully shifted into the operation register, either MULT3 or DIV3 is developed, driving Multiply Or Divide, MOD4, false. See Figure 4-226. Enable Shift DowN, ESDN3, is developed, and Five Phase Op Code signal, FPOC4, is also driven false, which later allows the action step counter to utilize all 5 steps instead of just the first two. Bear in mind that these master signals remain until the function is completed and the next instruction is retrieved.



MASTER SIGNALS IN MULT OR DIVIDE

FIGURE 4-226

TP-161SL

3. ACTION STEP COUNTER OPERATION

During multiply and divide there are four instances in which the action step counter does not automatically go from one step to the next. The action step counter returns to step 2 each time it leaves step 4 until the Terminate Multiply Or Divide signal, TMOD3, is developed. TMOD3 allows the counter to go to step 5, where the final shift down occurs.

The only time BASC3 is blocked and the action step counter does not step is when ADMT3, ADd in MulTiply, or DSUB3, Divide-SUBtract, is true. The former becomes true in action step 3 to allow for the correct number of repeat adds. The latter is developed in action step 2 to enable the repeat subtracts.

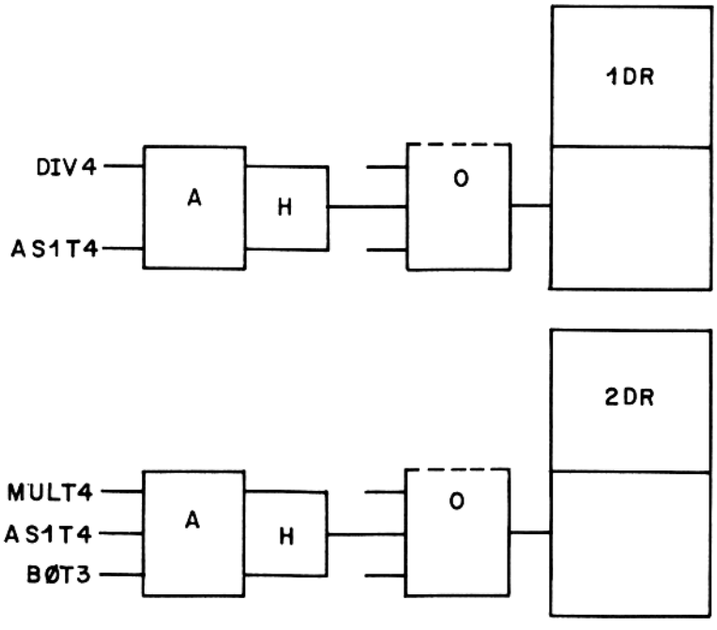
All of the preceding conditions are explained in more detail in Section 4B, Action Step Counter.

The only other situation requiring the blocking of ASSP4, which prevents the action step counter from stepping, is when an overflow condition occurs in either machine add or machine subtract. This occurs in divide when the repeat subtracts have produced an overdraft, and in multiply when the repeat add process produces a carry in A14. In either case, CB becomes set and blocks ASSP4 in the same way as it is blocked during a regular machine add or machine subtract overflow. This is described in Section 4K, Add or Subtract.

4. PRELIMINARY 5 MODE OPERATION

The first action step is the same in multiply and divide except for its effect on the decimal register. In multiply the decimal register is set to 13, and in divide it is set to 14. The addition of the signs and the transposing of R0 and R1 are identical in both cases.

a. SETTING THE DECIMAL REGISTER. As noted earlier, the decimal register is automatically set to 15 in 4 mode 3 phase. During action step 1 in multiply 2DR becomes reset: in divide 1DR is reset. This leaves the decimal setting at 13 and 14, respectively, so that the proper number of columns can be operated upon. It will be seen shortly that due to the way in which the logic is arranged, divide always begins with a "dummy" subtract: this is the reason for having the decimal register set one higher. See Figure 4-227.

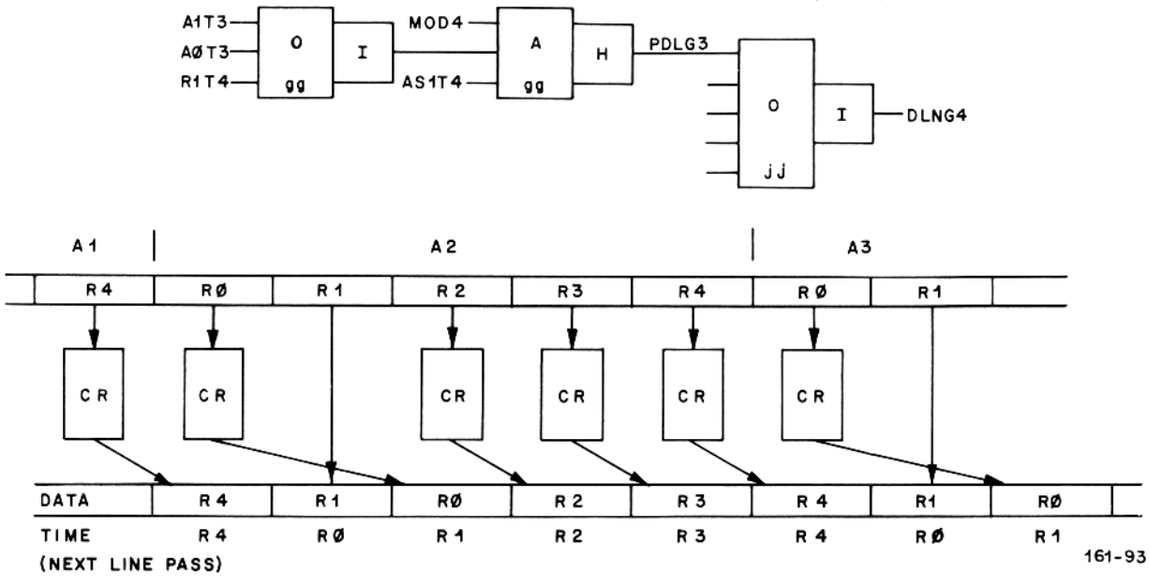


SETTING DR IN AS1 DURING MOD
(All on "FF" card)

FIGURE 4-227

TP-161SL

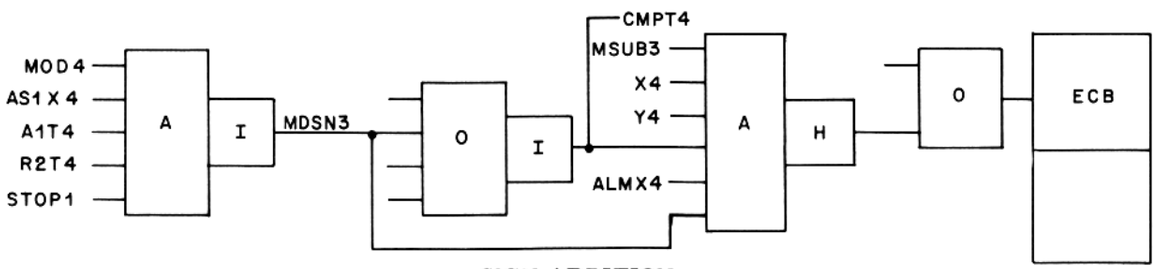
b. TRANSPOSING R0 AND R1. The line routing in action step 1 is ALCR, CRAL at all times except during R1 in columns 2 through 14: Figure 4-228 shows the development of PDLG3 at these times. With the sync pulse written long and R1 data written direct, the net effect is a shift down of R1 only. R0 data (blank) is shifted into the common register at R0 time. It cannot be written out during R1 time because PDLG3 goes false (writing becomes direct), so it is written out during R2.



TRANSPOSING R0 AND R1

FIGURE 4-228

c. ADDING THE SIGNS. The addition of the signs is accomplished by developing CoMPuTe, CMPT, in A1R2 time. The sum of A1R1 and A1R2 is set into 6CR during A1R2 time, and is written onto the delay line during A1R3. (Since the data path is long, this remains R2 data.) CMPT is produced by first developing the Multiply or Divide SigN addition signal, MDSN3, as shown in Figure 4-229. MDSN3 also prevents any carry from occurring, by inhibiting the setting of ECB. The presence of a sign bit in EITHER R1 or R2 results in the writing of a sign bit into the A1R2B0 time slot: if BOTH R1 and R2 contain sign bits, no sign is rewritten into R2B0, and the carry to B1 is lost. Thus no digit greater than 1 can be written into R2. Notice how this procedure coincides with the rules for signs in multiply and divide: like signs produce a positive answer (product or quotient); unlike signs produce a negative answer.



SIGN ADDITION
(All on "LL" card)

FIGURE 4-229

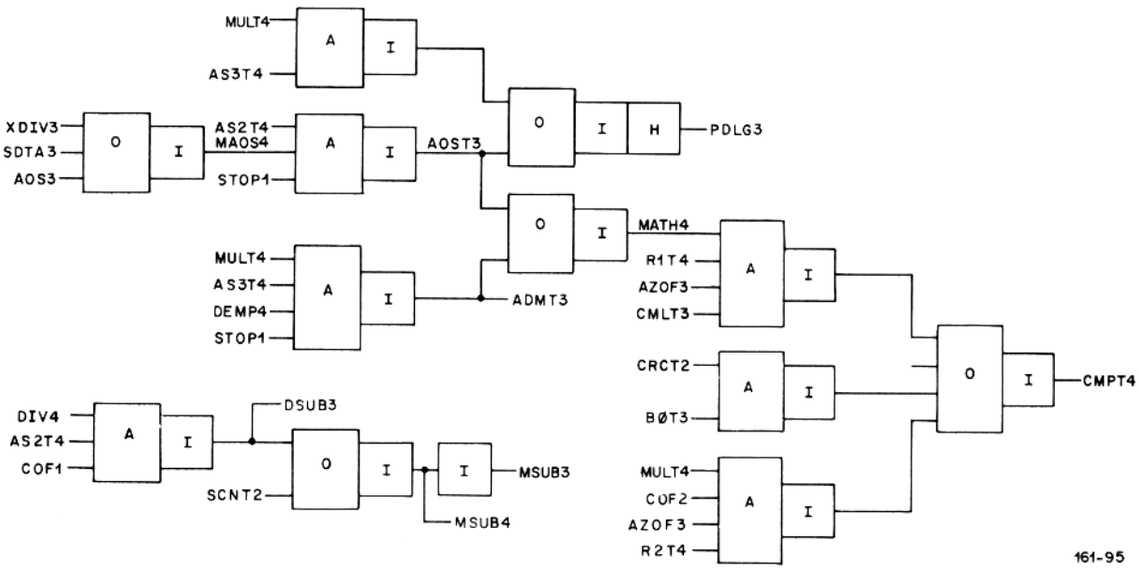
Note that the addition of the signs actually occurs before the transposition of R0 and R1: it is discussed last here because it was necessary to first establish the ALCR, CRAL line routing.

5. THE MULTIPLY/DIVIDE "LOOP"

In both multiply and divide, action steps 2, 3, and 4 are repeated once for each step of the decimal register. (The decimal register is counted down in action step 4 at the end of the R1 shift left.) Action step 2 in multiply, action step 3 in divide, and action step 4 in both functions all require a single arithmetic line pass during each cycle of the "loop". Action step 3 in multiply performs the repeat adds, and may require anywhere from 1 to 19 line passes. Action step 2 in divide always requires at least two line passes: even if the machine subtract produces an overdraft on the very first line pass, one more line pass is required to add R0 back into R1 to rescind the overdraft. With a possibility of nine successful subtracts, up to 11 line passes are possible in action step 2 during divide.

The repeat adds and repeat subtracts are the same as the machine add and machine subtract "cycles" that occur in add or subtract. The technician must be familiar with the basic machine add and subtract processes in order to fully understand the remainder of this section.

Refer to Figure 4-230 and trace the development of MATH4, which develops CMPT4 in action step 3 during multiply and action step 2 during divide. Remember that development of CMPT automatically causes a machine add unless MSUB3 is also developed, in which case a machine subtract occurs instead. Notice also that either Add Or Subtract Time, AOST3, or MULT4 and AS3T4 will develop PDLG3 to provide the "long" line routing (ALCR, CRAL) necessary for the computation.

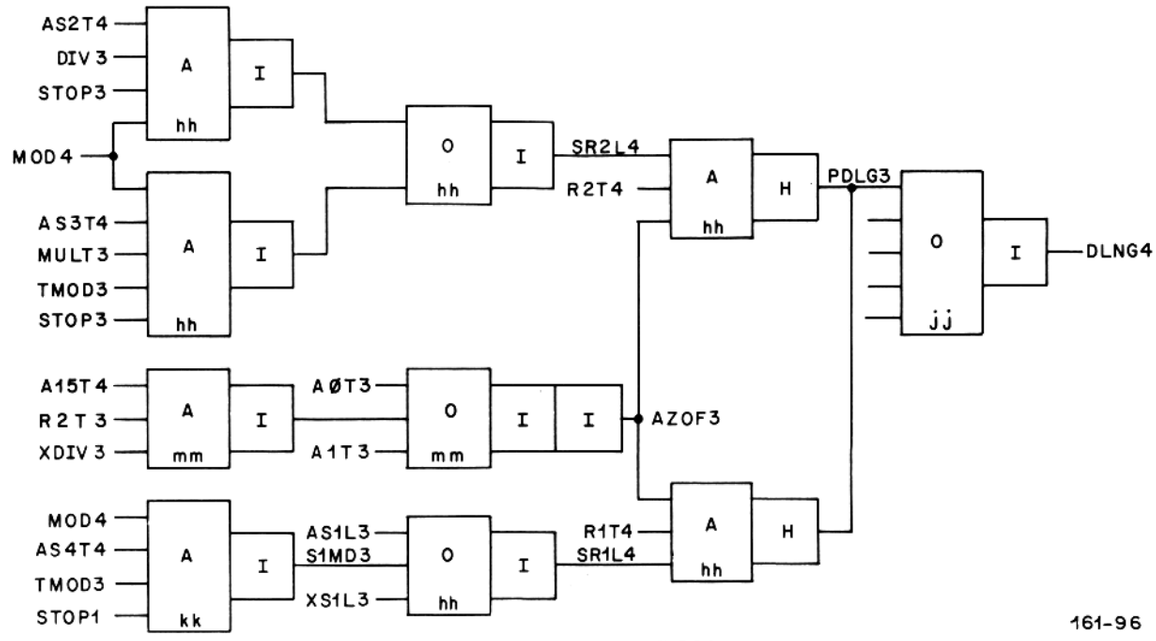


BASIC COMPUTATION LOGIC IN MOD
(All on "LL" card)

FIGURE 4-230

a. SHIFT LEFT. Of all the left shifts in multiply and divide, only the left shift of R1 during multiply is the same as the left shifts discussed earlier under Enter (Section 4G) and Print (Section 4H). All other shifts (i.e. the shift of R1 during divide and the shift of R2 in both multiply and divide) are similar, but data is also shifted through A15.

All of the gates associated with left shifting during multiply and divide are illustrated in Figure 4-231. Note that a shift of R2 occurs during action step 2 in multiply and during action step 3 in divide. R1 is shifted left during action step 4 in both functions, when the Shift R1 in Multiply and Divide signal, S1MD3, is developed.



SHIFT LEFT DURING MOD

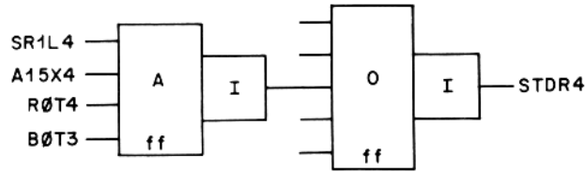
FIGURE 4-231

Still referring to Figure 4-231, trace the development of AZOF3, and note that it always inhibits PDLG3 during A0 and A1 of the shift left line pass. During A15 time, AZOF3 is false during R2 time, so it does not prevent the shift left of R2. Note also that AZOF3 is false throughout A15 time during divide, allowing R1 to be shifted left also. Therefore, the following statements can be made about shifting during multiply and divide:

(1) R2 is shifted left, A2-A15, during action step 2 in multiply and during action step 3 in divide.
(2) R1 is shifted left during action step 4 in both functions. However, in multiply R1 is shifted during A2 through A14; in divide it is shifted during A2 through A15.

It will be seen shortly that it is necessary to shift into A15R2 during multiply, because this is where the multiplier digit is counted down. A15R2 is not necessary during divide, but since the shift does no harm, it is not inhibited. In division it is also necessary to shift into A15R1 and to develop CMPT in A15R1: this is needed to allow division by numbers having 8 digits left-of-decimal.

Each time R1 is shifted left, the decimal register is counted down the same as it is during Enter and Print. See Figure 4-232.



COUNT DOWN DR AFTER SR1L

FIGURE 4-232

TP-161SL

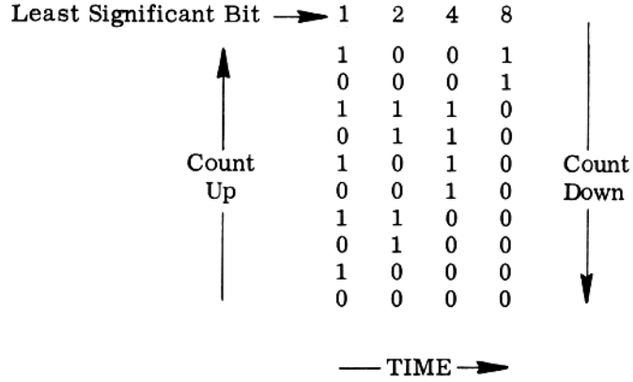
b. COUNT UP/COUNT DOWN. During multiply, each repeat add in action step 3 causes A15R2 to be counted down once: when A15R2 reaches zero, the repeat adds stop. During divide, each successful subtract in action step 2 causes A2R2 to be counted up once: this is the quotient developing in R2.

Because the repeat adds are halted when A15R2 reaches zero, A15R2 is never counted down below zero. Likewise, there can never be more than 9 successful subtracts in any one column, so A2R2 never contains a value greater than 9.

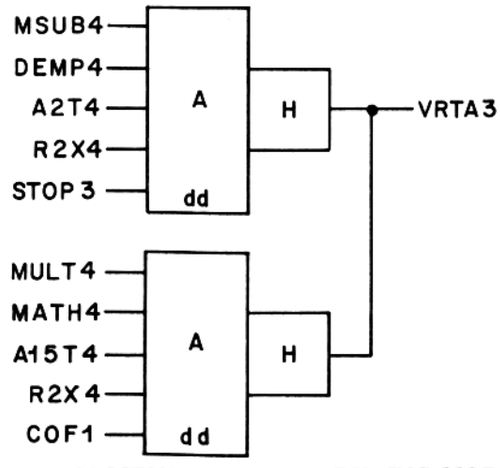
The rules for counting up or counting down in the 5610 are as follows:

(1) To count up, invert all data up to and including the first logic 0. Write all subsequent data unchanged.
(2) To count down, invert all data up to and including the first logic 1. Write all subsequent data unchanged.

It is important to remember that these rules apply only when starting with the least significant binary bit in A15R2 (count down in multiply) or A2R2 (count up in divide). Refer to the list which follows and note how the preceding rules apply to any two adjacent BCD numbers.



In counting up and counting down, VRTA3 is used to invert the data. The gates used to develop VRTA3 are shown in Figure 4-233. During multiply, VRTA3 is developed if DEMP is set at the beginning of A15R2 time, and it remains true until DEMP resets. (DEMP controls MATH.) This will be covered shortly. During divide, VRTA3 is developed by DEMP being set in A2R2, and again it remains true until DEMP becomes reset. It will be shown shortly how DEMP becomes reset by the first 1-bit in A15R2 during multiply and by the first absence of a 1-bit in A2R2 during divide.



DEVELOPMENT OF VRTA3 DURING MOD

FIGURE 4-233

It is important to remember that VRTA3 affects ONLY the setting and resetting of 6CR. It changes the data as it is sensed, and not as it is written. Remember that the data path is long, a necessity if VRTA3 is to accomplish its purpose. The development of PDLG3 was shown earlier in Figure 4-230.

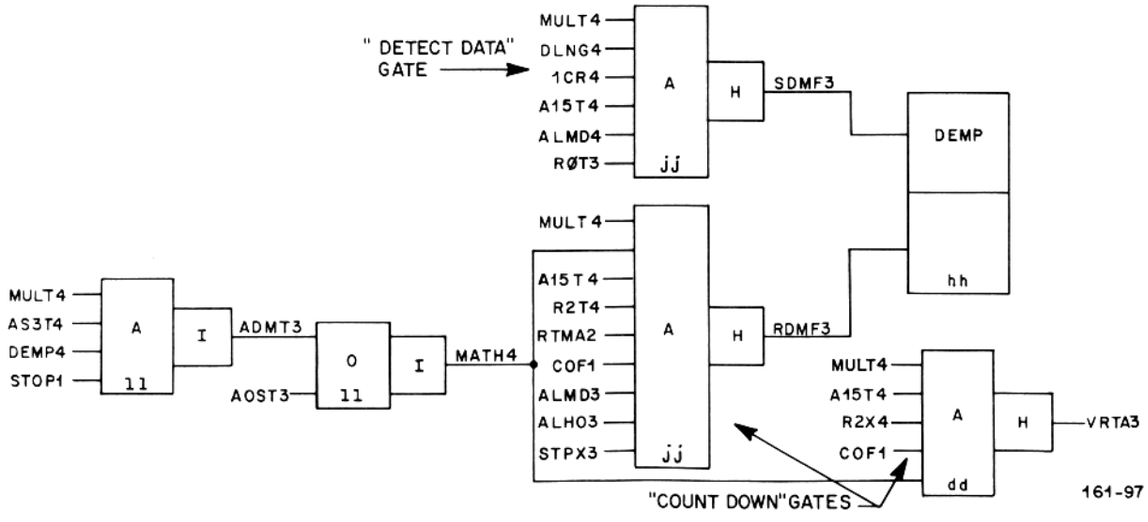
c. MULTIPLY. The first action to occur in action step 2 during multiply is a shift left of R2, as has already been covered. At the end of this shift left, A15 is checked: if A15 contains a digit, DEMP sets, allowing CMPT to be developed for the next line pass. Then the action step counter steps to 3.

(1) The Multiply Loop. In most multiply problems, R2 is not "full", and only zeros appear in A15R2 for the first few cycles of the multiply "loop". If this is the case, DEMP remains reset and CMPT is not developed, which results in a "dummy" line pass in action step 3. Next the action step counter steps to 4, where R1 is shifted left and the decimal register is counted down. Then a jump occurs back to action step 2, where R2 is again shifted left and A15 is checked again.

This procedure repeats, one "loop" being made for each setting of the decimal register, until a digit other than zero is shifted into A15R2 (in action step 2) and DEMP sets. When this occurs, CMPT is developed and R0 is added to R1. At the end of the add line pass, A15R2 is counted down once, and DEMP is reset in the process. However, if A15R2 has not yet been counted down to zero, DEMP becomes set again as the new number in A15R2 is written out. With DEMP set, ADMT3 blocks the development of BASC3 and the action step counter remains in step 3. Another add takes place during the next line pass, and A15R2 is counted down once more. This process repeats until A15R2 is counted down to zero, at which time DEMP does NOT become set, ADMT3 becomes false, and the action step counter steps to 4.

(2) DEMP Action. Note that in the preceding discussion, DEMP has a twofold purpose: it controls the development of CMPT and ADMT, thus providing the repeat adds, and it controls VRTA3 by resetting after the first data bit in A15R2, allowing the "count down" to occur. In other words, it checks for the presence of data, and when it finds data, it causes the repeat adds to occur.

The logic associated with DEMP in multiply is shown in Figure 4-234. SDMF3 can occur in either action step 2 or 3. In action step 2, DEMP can set in A15R2 when the former A14 data is being written out as a result of the shift left of R2. In this case, DLNG4 is false in A15 only during R2. In action step 3, DEMP becomes set in A15R3 time, after the number in A15R2 has been counted down (not yet to zero) and is being rewritten. Note that in action step 3, DLNG4 is false throughout A15 time, and not just during R2. Note also that even though the number that was just counted down is rewritten during R3, it remains R2 data because the sync pulse was also written "long". Last, note that in both cases 1CR must become set in order to set DEMP: if A15R2 = 0, DEMP does not set.



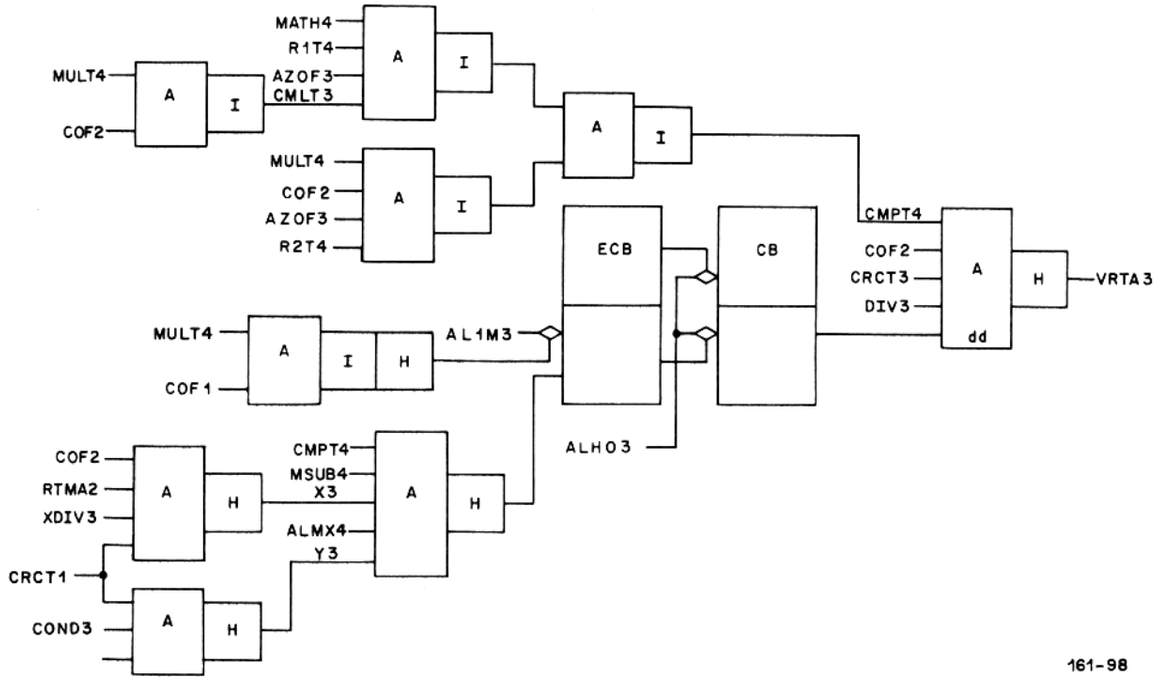
DEMP LOGIC IN MULTIPLY
FIGURE 4-234

TP-161SL

The first time that DEMP sets in any one multiply "loop" is at the end of action step 2, when a digit (other than zero) is shifted into A15R2 from A14R2. DEMP always resets in the "count down" process in action step 3, and it sets at the end of each line pass in action step 3 until A15R2 is counted down to zero.

Still referring to Figure 4-234, note that DEMP being set in action step 3 allows ADMT and MATH to be developed, thus allowing the additions to occur. At the end of each add line pass, VRTA3 goes true in A15R2 until DEMP resets and drives MATH4 true. RDMF3 resets DEMP when RTMA sets, denoting the presence of a data bit. Thus all data is inverted (as it is sensed) until after the first data bit.

(3) Add Overflow. It is possible for a carry to occur in A14 during any of the repeat adds (except the very first). Should this occur, the carry is added to A2R2, which is actually the extension of R1. This is known as the "hairy carry". When CB sets as a result of a correction in A14, it holds the action step counter at 3. At arithmetic line home, COF sets, driving Carry in MuLTiply, CMLT3, true. This prevents CMPT from being developed in the normal manner (R1), but allows it to be developed in R2 time instead. See Figure 4-235.



ADD CARRY IN MULTIPLY
(All on "LL" card unless otherwise noted)

FIGURE 4-235

Notice that ECB does not reset automatically at arithmetic line home, as it does in other arithmetic functions. Instead, ECB remains set until both X3 and Y3 become false during CMPT. During A2R2, when CMPT is developed, VRTA3 is produced as long as CB is set. Thus all data in A2R2 is inverted until CB resets. ECB resets at the first "no-bit" condition sensed in A2R2, and CB resets at the following bit.

This carry process is similar to the count-up process in that all data is inverted up to and including the first logic 0. In this process, however, it is also possible to have another carry produced in A2R2, making a correction necessary. A correction is not possible in the count-up process because CMPT is not developed. If a 9 is present in A2R2 and a carry from A14R1 is added to it, the resulting 10 requires that a correction be made. CB sets as a result of the correction, and this causes a carry to be made to A3. The next time CMPT is developed (A3R2), all data is again inverted until CB resets. It is possible, after a number of multiply loops have been made, to have a series of 9's in the lower order positions of R2. If a carry should occur in A14R1 under these circumstances, all of the 9's would be converted to 10's, and the carry would stop only when it was added to some digit < 9. Once CB becomes reset and no carry is produced, no further data is changed.

(4) Decimal Register. After 13 cycles of the multiply loop, all 13 digits originally in R2 have been shifted through A15 and each has caused its respective number of machine adds. The product is fully developed, but not yet decimally aligned.

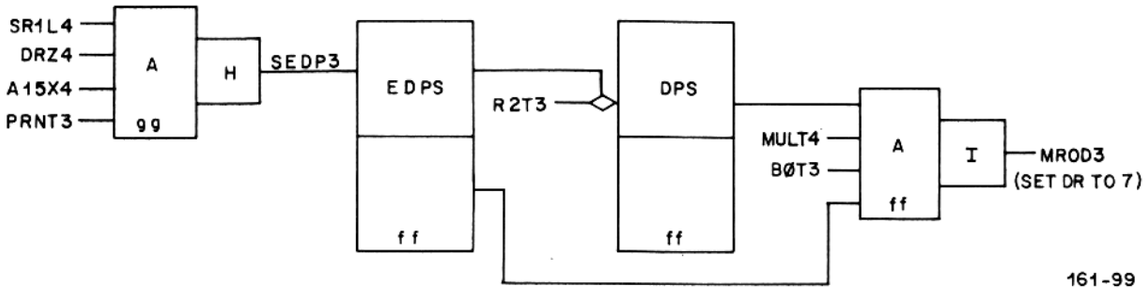
After the last repeat add occurs, it is necessary to shift the product left 8 times in order to align it correctly around the decimal in R2. Since R1 and R2 are being used as one long register, both registers must be shifted, and the shifts can be thought of as occurring in pairs. After being counted down to zero, the decimal register is set to 7 to keep track of these last 8 shifts. An example, illustrated in Figure 4-236, shows 321.9 being multiplied by .00001. Note that the farthest-right digit of the answer (9) remains in R1 at the end: it will later be shifted down into R0 and then cleared.

R2		R1		DR
A15 A2	A14 A2			
1 0 0 0 0 0 0 0 0 0.0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0.0 0 0 0 0 0 0 0 0 0 0 0 3 2 1.9 0 0 0 0 0		R0	1
0 0 0 0 0 0 0 0 0 0.0 0 0 0 0 0	0 0 0 0 0 0 3 2 1.9 0 0 0 0 0			1
0 0 0 0 0 0 0 0 0 0.0 0 0 0 0 0	0 0 0 0 0 3 2 1 9.0 0 0 0 0 0			0-7
0 0 0 0 0 0 0 0 0 0.0 0 0 0 0 0	0 0 0 0 3 2 1 9 0.0 0 0 0 0 0			6
0 0 0 0 0 0 0 0 0 0.0 0 0 0 0 0	0 0 0 3 2 1 9 0 0.0 0 0 0 0 0			5
0 0 0 0 0 0 0 0 0 0.0 0 0 0 0 0	0 0 3 2 1 9 0 0 0.0 0 0 0 0 0			4
0 0 0 0 0 0 0 0 0 0.0 0 0 0 0 0	0 3 2 1 9 0 0 0 0.0 0 0 0 0 0			3
0 0 0 0 0 0 0 0 0 0.0 0 0 0 0 3	3 2 1 9 0 0 0 0 0.0 0 0 0 0 0			2
0 0 0 0 0 0 0 0 0 0.0 0 0 0 3 2	2 1 9 0 0 0 0 0 0.0 0 0 0 0 0			1
0 0 0 0 0 0 0 0 0 0.0 0 0 3 2 1	1 9 0 0 0 0 0 0 0.0 0 0 0 0 0			0

LAST 8 SHIFTS

FIGURE 4-236

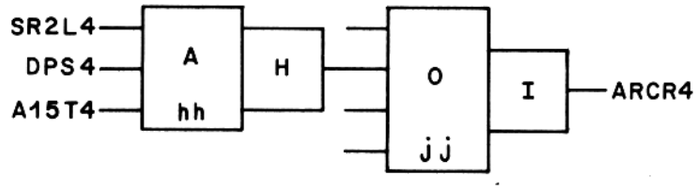
The last 8 shifts begin as soon as the last repeat add is complete: R1 is shifted left in action step 4 and R2 is shifted in action step 2. Note, however, that the decimal register is counted down between the two shifts. After the shift of R1 which follows the last repeat add, the decimal register is counted down to zero, and EDPS sets, as shown in Figure 4-237. This allows Multiply Right Of Decimal, MROD3, to be developed, which sets 1DR, 2DR, and 4DR. Note that the timing involved in this operation is identical to that used in Enter and Print: the decimal register is counted down (to zero) at A15R1, SEDP3 immediately sets EDPS, which produces MROD3 and sets the decimal register to 7, and DPS sets at A15R2, disabling MROD3.



DEVELOPMENT OF MROD3

FIGURE 4-237

The last 8 shifts each require a loop through action steps 2, 3, and 4. These cycles are the same as the earlier ones, except that DPS being set holds the common register reset during A15. See Figure 4-238. This prevents the higher order digits of the answer (product) from being shifted into A15, so DEMP cannot set and no repeat adds can take place.



DEVELOPMENT OF ARCR4

FIGURE 4-238

Note that it is possible for data to be present in the higher order columns of R2 at this time if the answer being developed is quite large: it is possible for the product to overflow as much as 8 columns. The data in these 8 columns is lost as R2 is shifted left and the common register is held reset during A15.

Note also that the overflow situation in multiply is the same as that during add: there is no outward sign to tell the 5610 operator that an overflow has occurred.

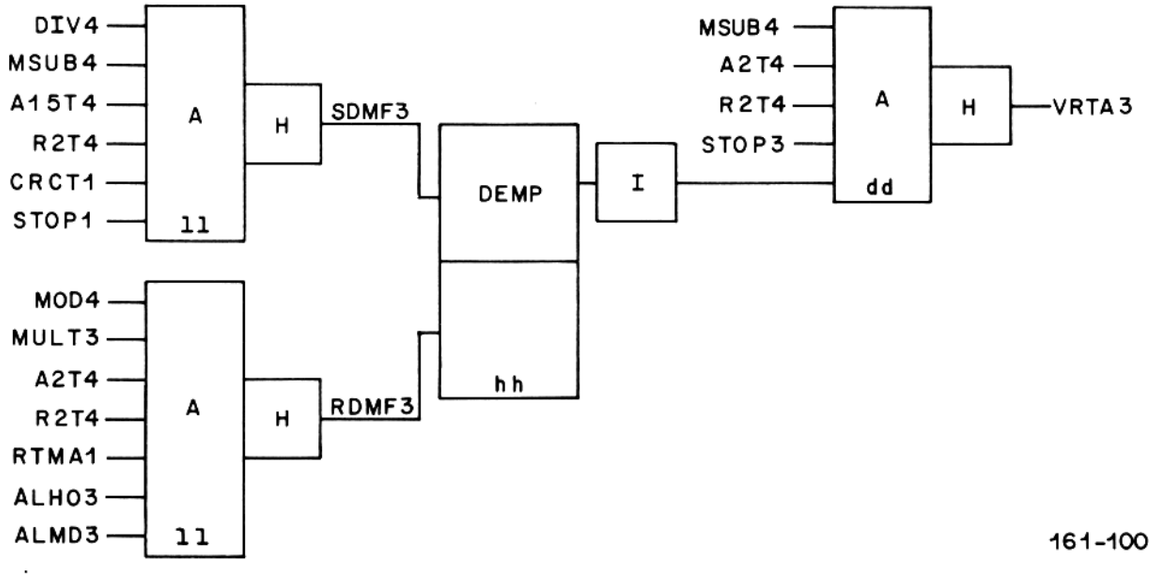
d. DIVIDE. As soon as action step 2 is reached in divide, R0 is subtracted from R1. The development of Machine Add Or Subtract (MAOS), Add Or Subtract Time (AOST), MATH, CMPT, and PDLG was shown earlier in Figure 4-230. Since R1 is clear, this first subtract always results in an overdraft, and CB is set at the end of the line pass. This locks the action step counter in step 2 for an additional line pass. COF sets at arithmetic line home. The second line pass does not add 1 to A2R2, as is done in multiply, because ECB resets at arithmetic line home, just as COF becomes set. The recomplement process which would normally occur during this second line pass in a regular machine subtract is prevented at this time by DIV3 and XDIV3 both being true. The sign is not changed because SCNT is still set. COND3 is false, allowing the X and Y variables to function as in a normal machine add or subtract. COF1 blocks SDUB and MSUB, and since CMPT is still valid, a normal machine add occurs during the next line pass. This machine add rescinds the overdraft and restores R1 to its "clear" condition. At the end of this line pass the action step counter steps to 3, and COF resets.

(1) The Divide Loop. In action step 3, R2 is shifted left, as previously described, and in step 4, R1 is shifted left and the decimal register is counted down. Then the action step counter returns to step 2 and another subtract occurs. This second subtraction also results in an overdraft because R1 is still clear: during the first pair of shifts nothing is shifted into R1. The first time through action step 3 the MSD of R2 is shifted into A15. Since A15 was clear to start, the common register is clear at the end of this shift, and nothing is shifted into R1 the first two times through action step 4.

The second line pass in action step 2 again adds R0 to R1 to cancel the overdraft. Then another pair of shifts occur in action steps 3 and 4. During these shifts the digit originally in the MSD position of R2 is shifted into the LSD position of R1, and the decimal register is counted down again. Then the action step counter jumps back to step 2 for the third subtraction.

Although not very likely, it is possible for this third subtraction to be "successful": that is, NOT cause an overdraft. This would be the case if the divisor in R0 had a digit (other than 0) ONLY in the position 5 right-of-decimal, and if a larger digit had been shifted into R1 during the last shift left. In most divide problems, however, several more "cycles" through action steps 2, 3, and 4 occur before a successful subtract can be made.

(2) DEMP Action. Each time the divide loop makes a cycle, the number in R1 increases by a power of ten, and eventually the subtraction that occurs in action step 2 does NOT result in an overdraft. In this case there is no borrow necessary in A15R1, and the CoRreCTion flip-flop, CRCT, does not set in A15R2 time. This allows DEMP to set, as shown in Figure 4-239.



DEMP ACTION IN DIVIDE

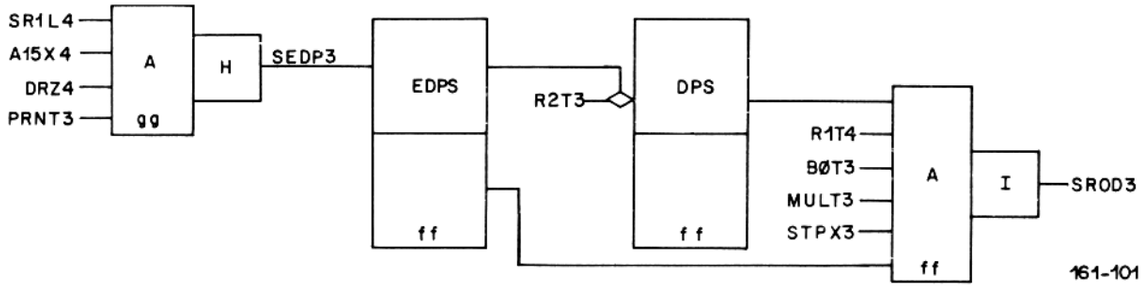
FIGURE 4-239

Because no overdraft occurs, COF remains reset and DSUB3 remains true. This blocks BASC3 and allows the action step counter to remain in step 2. Another line pass in action step 2, with DSUB3 and MSUB3 both true, causes another machine subtract.

DEMP being set also causes 1 to be added to A2R2. This occurs during the next line pass, as the machine subtract is occurring. After this first successful subtract, A2R2 is clear, so DEMP simply allows VRTA3 to be developed in A2R2B0, to change the logic 0 to a logic 1. This same logic 0, by virtue of its NOT setting RTMA, resets DEMP. This is also shown in Figure 4-239.

This procedure repeats, 1 being added to A2R2 for each line pass, until an overdraft occurs. Note that A2R2 is counted up and DEMP is reset during the same line pass that produces the overdraft. DEMP does not set at the end of this line pass because CRCT sets in A15R2 to perform the correction made necessary by the borrow that occurs in A15R1. One more line pass in action step 2 follows, to add R0 back into R1. A2R2 is not affected during this last line pass because DEMP remains reset.

(3) Decimal Register Operation. The divide loop repeats, once for each step of the decimal register, until the decimal register is counted down to zero. When this occurs, SEDP3 is developed, the same as in multiply, and EDPS sets. This produces SROD3, which sets the decimal register to 5, the same as in Enter, Print, and Constants. DPS sets at A15R2 to to disable SROD3. See Figure 4-240 and note how SROD3 is blocked during muitiply.

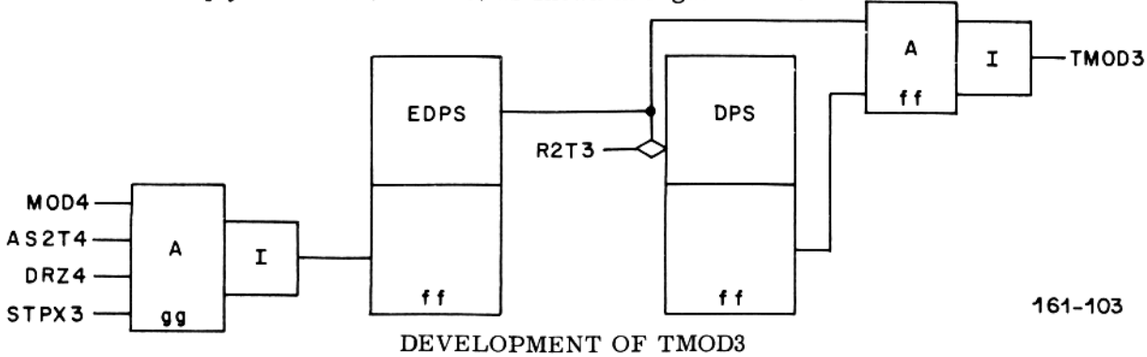


DEVELOPMENT OF SROD3 IN DIVIDE

FIGURE 4-240

Since the decimal register is counted down to zero (and set to 5) in action step 4, the action step counter jumps back to step 2 and another test subtraction occurs. The operation is exactly the same as it was before EDPS and DPS became set: the repeat subtracts and the counting up of A2R2 continue the same as before. Contrast this to the last 8 "cycles" through the multiply loop, in which the repeat adds are disabled and only the shifting occurs. During multiply, 8 shifts are needed to shift the answer out of R1 into R2. In divide, 5 shifts are required to shift the whole number answer from A2 to A7. Also, during divide the division process continues during these shifts in order to develop the decimal portion of the answer. Note also that it is possible for the repeat subtracts to occur and for A2R2 to be counted up even after the very last shift has occurred. This is necessary in developing the fifth answer digit right-of-decimal.

6. TERMINATE MULTIPLY OR DIVIDE. When the decimal register is finally counted down to zero the second time, MROD3 and SROD3 are not developed because DPS is already set. The action step counter returns to step 2 for the last time, where Reset Only Enable Decimal point storage, ROED3, is developed. This signal resets EDPS and allows the development of Terminate Multiply Or Divide, TMOD3, as shown in Figure 4-241.



DEVELOPMENT OF TMOD3

FIGURE 4-241

TMOD3 blocks SR1L in both multiply and divide. It also blocks SR2L in divide (refer to Figure 4-231). During multiply the last shift of R2 is necessary to align the product in R2. TMOD3 does not block either the repeat adds or the repeat subtracts, but no repeat adds can occur anyway, because A15 is held reset during the shift of R2 by DPS being set. In divide, however, the subtract operation can still take place and A2R2 may still be counted up at this time.

TMOD3 prevents the shift left of R1 during action step 4, so the arithmetic logic simply idles through this step. TMOD3 also blocks the reset of the 3AS flip-flop in the action step counter, so at the end of action step 4 the counter goes on to step 5. In action step 5 the Storage and Arithmetic Shift Down signal, SASD4, is developed (KK card), and the operating stack is shifted down the same as in a Delete operation. At the end of one line pass in action step 5, the action step counter returns to 0, the next sync pulse sets DOST, and the phase counter steps to 7 phase. At the next program line home a jump to 3M6,4 occurs and the search for the next instruction begins.

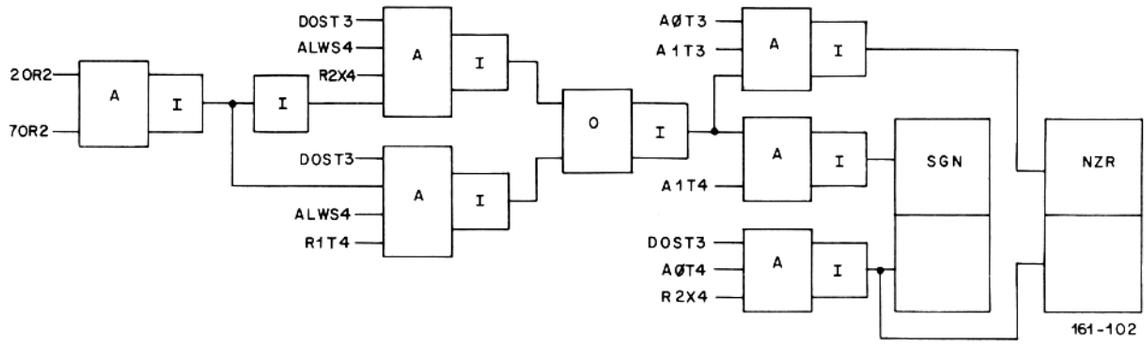
7. PREVENT DIVIDING BY ZERO

Before each divide function begins, R1 is checked to make sure it is not clear. If R1 contains 0, the divide process is blocked and a jump to 8 phase occurs, where the red error lamp is turned on. To get out of 8 phase the Reset switch must be depressed. This results in destruction of the marker and loss of the program sequence.

The logic involved in checking R1 is closely tied in with the development of RWNG3, which is used in conditional branch. For this reason the operation of the SiGN flip-flop, SGN, and the development of RWNG3 will also be covered at this time.

a. CHECKING R1. It was stated earlier that R1 is checked before each divide problem. Actually, R1 is checked at the END of every arithmetic function, or at least after every function that might alter the contents or the sign of R1. If R1 contains any data at all in columns 2 through 14, the Non-Zero Register flip-flop, NZR, sets. It then remains set until after the beginning of the next arithmetic operation. Thus, if an arithmetic operation should occur which leaves R1 = 0, NZR does not set, and a subsequent divide command would cause a lockup.

The logic used to control NZR is illustrated in Figure 4-242. Look first at the two 4-legged AND gates. The upper one is enabled by any code in the operation register that contains both a 2 bit and a 7 bit. The lower gate is enabled by all other codes - those that do NOT contain both the 2 bit and the 7 bit. Both gates require that DOST be reset. This means that the upper gate will be enabled by any arithmetic function that contains both the 2 bit and 4 bit in its command code, or by the Add, Subtract, Multiply, and Divide commands. The lower gate is enabled by all other arithmetic commands.



NZR, SGN LOGIC
(All on "DD" card)

FIGURE 4-242

TP-161SL

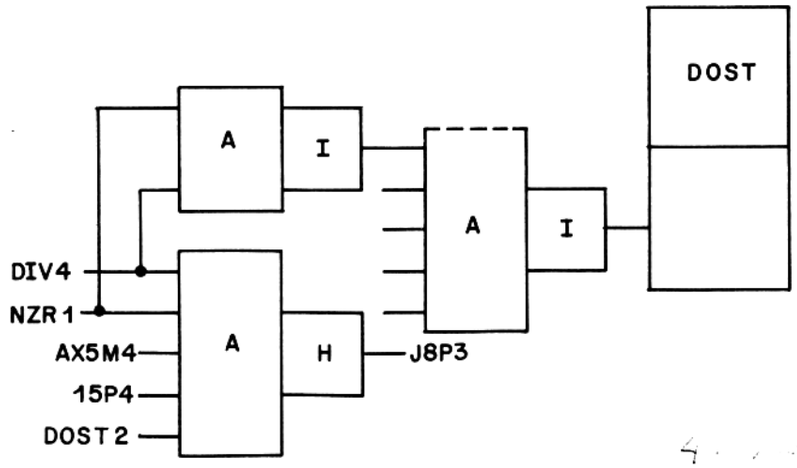
Next, notice that the "ADD-SUB-MUL-DIV" gate is further qualified by R2 time, and the other gate is qualified by R1 time. Both gates require some data to be written out onto the line to drive ALWS4 false. In other words, the output of the upper gate is true when data is written out in R2 time during Add, Subtract, Multiply, and Divide. The output of the other gate goes true when data is written out in R1 time during any other arithmetic command.

The outputs of both gates are inverted through the next OR gate and then further qualified by A0 and A1 time in the upper 3-legged AND gate: NZR cannot be set by the sync pulse or by the sign bit. In summary it can be said that NZR will be set under either of the following conditions:

(1) During Add, Subtract, Multiply, or Divide, when a bit is written out in R2 time, A2 through A14.
(2) During all other arithmetic commands, when a bit is written out in R1 time, A2 through A14.

The different register times are needed because of the different arithmetic line routings: R1 DATA must be checked, regardless of the time at which it is written. Before enlarging upon this idea, note the reset of NZR in Figure 4-242: it is reset at A0R2 of every line pass occurring while DOST is reset. This means that the only set of NZR that really matters is the one that occurs just before DOST again sets, or the one occurring in the last line pass of any particular arithmetic function. Refer to Figure 4-51, the Arithmetic Operations Chart, and note that during the last line pass in Add, Subtract, Multiply, and Divide, R1 data is written out during R2 time. In all other arithmetic functions, R1 data is last written during R1 time. Thus the gates in Figure 4-242 allow NZR to remain reset or to become set, determined by the very last line pass in each arithmetic operation. NZR then remains in this state until after DOST again resets in the following arithmetic function.

b. LOCKUP. If some previous arithmetic function has cleared R1, NZR will be reset. If, at this time, a divide command is retrieved, a jump to 8 phase is enabled, as shown in Figure 4-243, as soon as 5 mode is realized. (J8P3 is an input to the large OR gate on the P card to yield ECTR3.) At the same time, DOST is prevented from resetting, also shown in Figure 4-243.



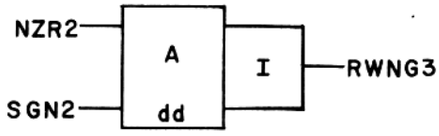
LOCKUP LOGIC IN DIVIDE
(All on "KK" card)

FIGURE 4-243

Once in 8 phase, the processor stops and waits for the Reset switch to be depressed. The operation from this point on is exactly the same as after a parity error in Enter: transfer of the Reset switch sets RST, and the marker search and removal operation is performed. Refer to Section 4G, Enter, Part 9, for more details.

c. SGN FLIP-FLOP. The operation of the SiGN flip-flop, SGN, has already been illustrated in Figure 4-242. It can be set only if a bit occurs in A1R2 during Add, Subtract, Multiply, and Divide, or in A1R1 during all other arithmetic instructions. In other words, it will be set whenever R1 contains a sign bit. Like NZR, the SGN flip-flop resets at the beginning of each line pass while DOST is reset. Although it can set in any A1 time during an arithmetic function, it is only the last line pass that really matters. The condition of SGN at the end of the arithmetic function is the important factor.

d. DEVELOPMENT OF RWNG3. If both NZR and SGN are set, RWNG3 will be true. This is shown in Figure 4-244. As explained in Section 3K, Conditional Branch, this signal is necessary in determining whether or not a Positive or Negative Branch instruction will be performed. From the development of RWNG3, it can now be seen that R1 is considered negative only if it has a sign bit (SGN) AND contains data (NZR): if either one of these qualifications is missing, RWNG3 will be false. Thus if R1 is clear, it is considered as having a positive sign.

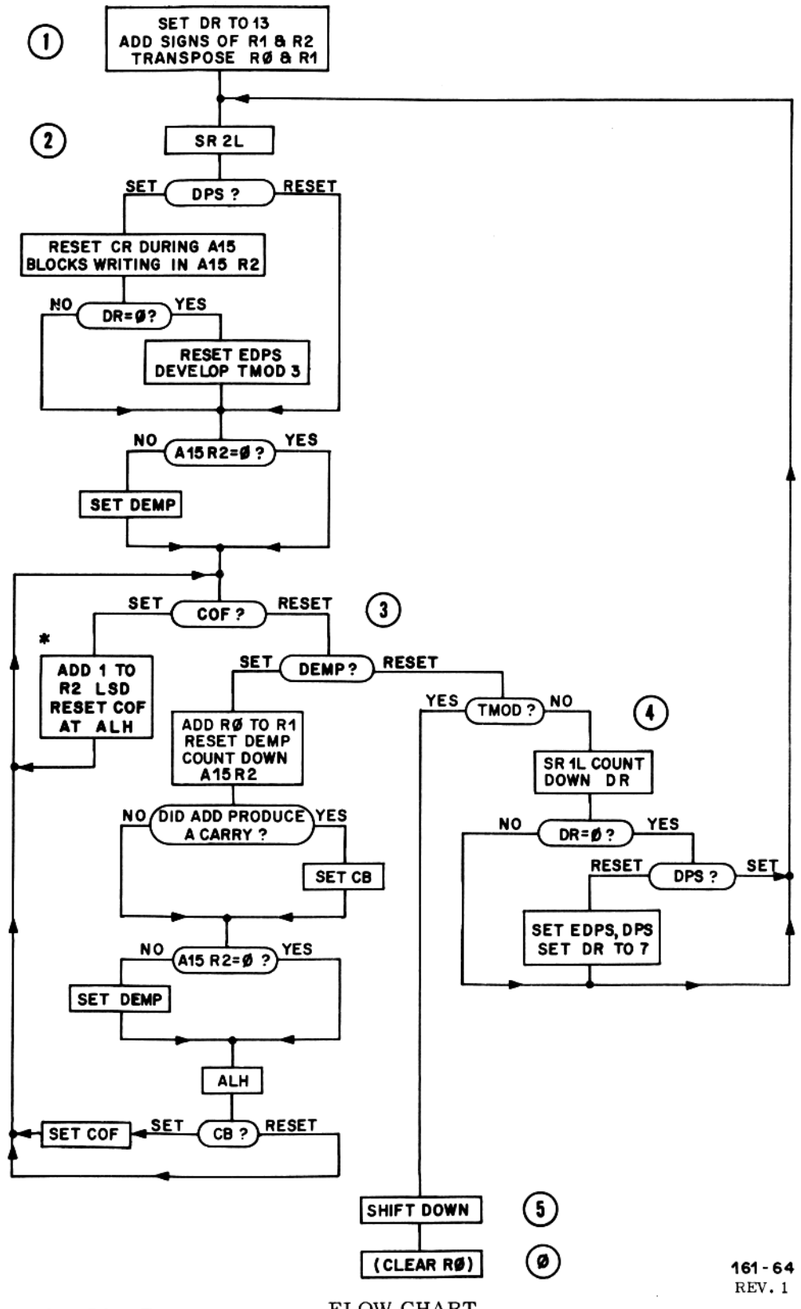


DEVELOPMENT OF RWNG3

FIGURE 4-244

8. FLOW CHARTS

The Flow Charts in Figures 4-245 and 4-246 depict the overall operation of the Multiply and Divide instructions.

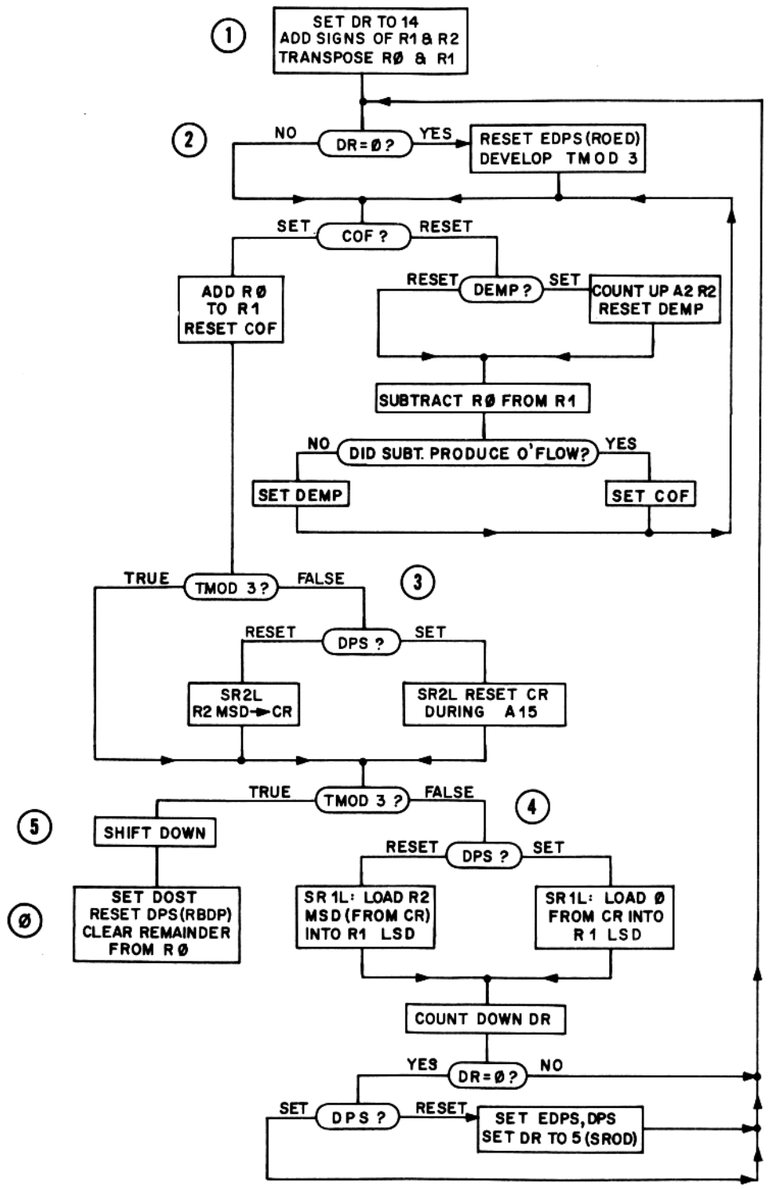


161-64
REV. 1

*Takes One Line Pass

FLOW CHART
MULTIPLY

FIGURE 4-245



FLOW CHART
DIVIDE

FIGURE 4-246

TP-161SL

TABLE OF CONTENTS

	Page
SECTION 5, STORAGE LINE LOGIC	
STORAGE DELAY LINE	5-1
Data Organization	5-1
Timing Chain	5-1
Launch Sync Pulse	5-5
Basic Read/Write Logic	5-6
STORAGE LOCATION CORRECT	5-7
DATA TRANSFER REGISTER	5-10
LINE ROUTINGS	5-10
GENERAL OPERATION	5-11
Relationship to Arithmetic Line Timing	5-11
Basic Data Transfer	5-12
DATA TRANSFER SUBROUTINE	5-13
Basic Operation	5-13
Logic	5-14
INTERLOCKING THE TWO DELAY LINES	5-17
TERMINATING THE STORAGE INSTRUCTIONS	5-18
STORE, RECALL, AND ACCUMULATE	5-21
General Operation	5-21
Instruction Retrieval	5-21
Recall	5-22
Store	5-24
Accumulate	5-27
Flow Charts	5-27

SECTION 5, STORAGE LINE LOGIC

The storage delay line and its associated logic are concerned with only three instructions; Store, Recall, and Accumulate. Since these instructions deal directly with the arithmetic delay line and its logic, thorough knowledge of the 5610 arithmetic logic is essential to the understanding of the storage line functions. Occasionally in this section reference will be made to the "recall function" or the "store function". The former pertains to either the Recall instruction or to the recall portion of an Accumulate instruction: the latter pertains to either the Store instruction or the store portion of an Accumulate instruction.

A. STORAGE DELAY LINE

The storage delay line assembly is physically identical to the two program delay lines. It has a delay time of approximately 5.5 ms and the read and write amplifiers are identical to those used with the other lines.

1. DATA ORGANIZATION

The organization of data on the storage line is similar in some respects to both the program line and the arithmetic line. Since the same oscillator (2.5 MHz) drives both the program line and the storage line, the basic timing of these two lines is the same. On the other hand, the program line data varies in length: the arithmetic line concept of columns, registers, bits, and fractional bits is used to make the storage line data fixed in length.

a. COLUMNS. The entire storage line is divided into 15 columns, numbered 1 through 15. Columns 2 through 15 are each 384 μsec. long. Column 1 is approximately 200 μsec., but it varies in length because it includes the "home" time of the timing chain.

b. REGISTERS. Columns 2 through 15 are each divided into 60 equal storage registers, numbered 00 through 59. Each register time in each column is 6.4 μsec. long. Register 59 is true throughout storage line home (column 1).

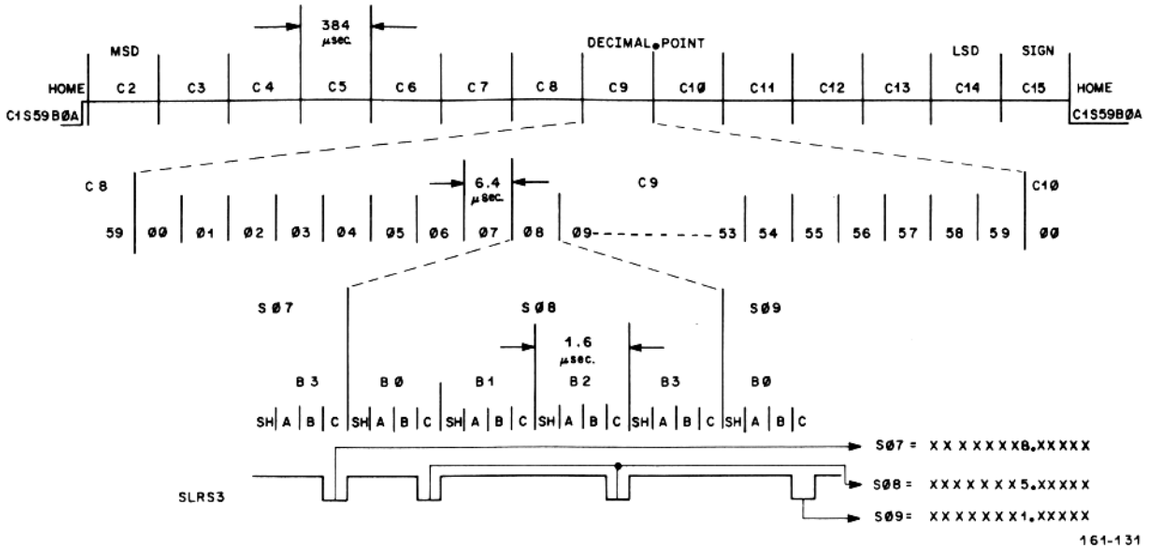
c. BITS. Each storage register is divided into four equal bits, numbered 0, 1, 2, and 3. Each bit is 1.6 μsec. long. B0 represents a binary 1, B1 represents a binary 2, B2 represents a binary 4, and B3 represents a binary 8.

d. FRACTIONAL BITS. Each bit is divided into four equal fractional bits, named SHift, A, B, and C. Each fractional bit is 400 nsec. in length. Data bits are written onto the line during "C" time, the same as on the arithmetic line.

Refer now to Figure 5-1 and note the similarity between the organization of the storage line and the arithmetic line. Note also, however, that the data is placed in the storage line in the reverse order: the MSD is in C2 and the LSD is in C14. The importance of this fact will become evident shortly. Finally, note that during storage line home, Column 1, Storage 59, Bit 0, and fractional bit "A" time (C1S59B0A) is true. When the storage line timing chain counts on from column 15, it becomes preset at this configuration.

2. TIMING CHAIN

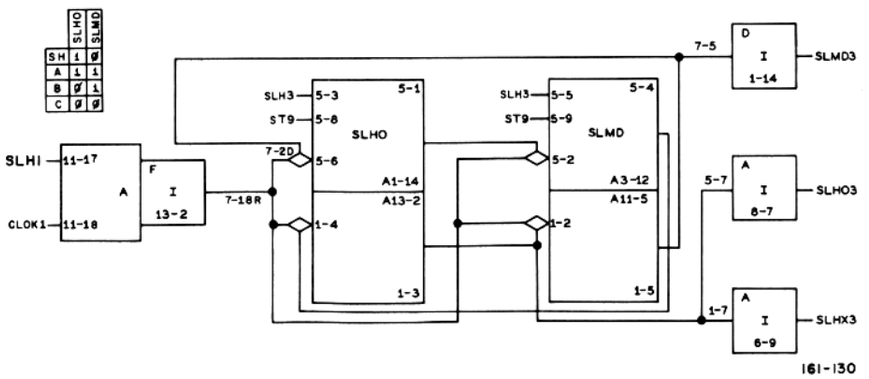
As stated earlier, the basic timing for the storage line timing chain is provided by the 2.5 MHz oscillator on the "A" card. This oscillator drives the fractional bit counter, which in turn drives the bit counter. The bit counter drives the register counter and the register counter drives the column counter. When the column counter leaves its highest state it sets the Storage Line Home flip-flop, SLH.



STORAGE LINE DATA ORGANIZATION

FIGURE 5-1

a. FRACTIONAL BIT COUNTER. The fractional bit counter is a two stage series shift counter made up of the Storage Line Half Oscillator, SLHO, and the Storage Line MiD oscillator, SLMD, flip-flops. Both flip-flops are driven by the oscillator, as long as SLH is reset, as shown in Figure 5-2. The outputs of the two flip-flops gate the setting and resetting of each other. Each time SLHO sets (SH time), SLHX3 goes true to drive the bit counter. The operation of the storage line fractional bit counter is almost identical to that of the program line and arithmetic line fractional bit counters. Note that both flip-flops are set by the Storage Line Home flip-flop (SLH), making fractional bit time "A" true during storage line home.

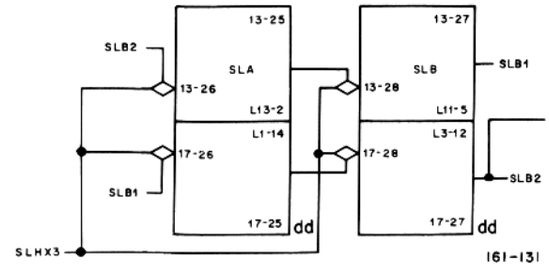


FRACTIONAL BIT COUNTER
(All on "A" card)

FIGURE 5-2

b. BIT COUNTER. The storage line bit counter is a two stage series shift register similar to the fractional bit counter and identical to the arithmetic line bit counter. The two flip-flops, SLA and SLB, are triggered by SLHX3 from the fractional bit counter. Each time SLB2 goes true (B0 time), it increments the register counter. The Storage Line Home flip-flop has no effect on this counter.

	BITS		
		SLA	SLB
*	0	0	0
	1	1	0
	2	1	1
	3	0	1

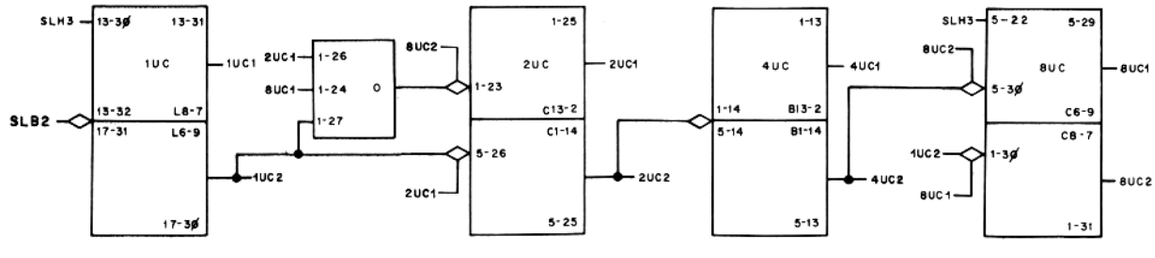


BIT COUNTER

FIGURE 5-3

c. REGISTER COUNTER. The register counter is actually made up of two separate counters, a "units" counter and a "tens" counter. The units counter is a modified binary counter which resets to the 0 state after leaving the 9 state. The tens counter is also a modified binary counter, resetting to 0 after leaving the 5 state.

(1) Units Counter. The units counter is made up of the 1UC, 2UC, 4UC, and 8UC flip-flops. It is triggered each time SLB resets, denoting B0 time. Each time the units counter returns to 0 from 9, it drives the tens counter. See Figure 5-4 and note that the operation of the 1UC and 4UC flip-flops is the same as in a regular binary counter. 2UC, however, operates "normally" only as long as 8UC is reset: with 8UC set, 2UC resets as usual (on the 9th count), but is prevented from setting again (on the 10th count). 8UC sets in the normal manner (on the 8th count), but becomes reset as 1UC resets on the 10th count. Thus the counter steps from 0 to 9 and then back to 0 again. When SLH sets, it drives this counter to its highest state.

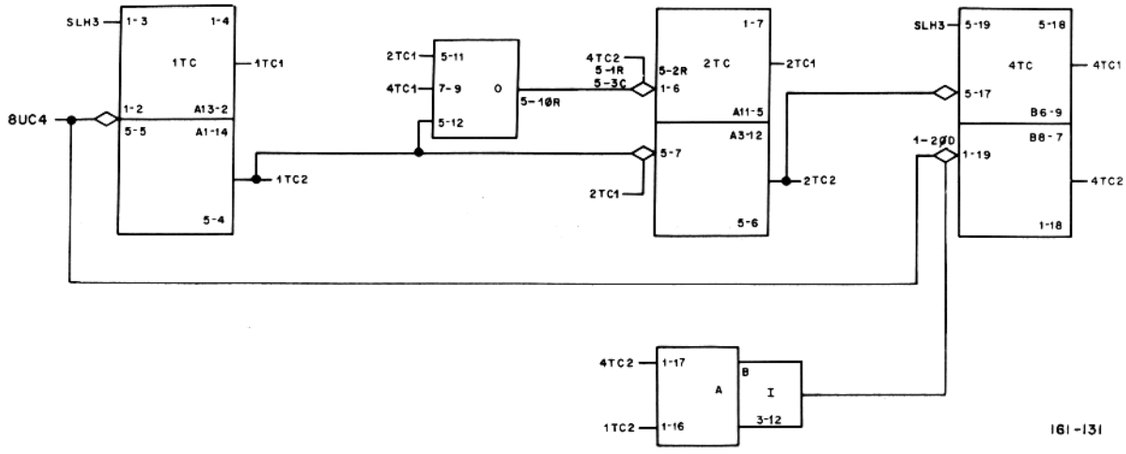


	UNITS			
	1UC	2UC	4UC	8UC
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
* 9	1	0	0	1

UNITS COUNTER
(All on "DD" card)

FIGURE 5-4

(2) Tens Counter. The tens counter is made up of the 1TC, 2TC, and 4TC flip-flops. It is triggered once every ten units by 8UC4, which goes true as the units counter steps from 9 to 0. The tens counter operation is quite similar to that of the units counter: the operation would be identical if the 4UC flip-flop were removed from the units counter. The tens counter counts up to 5 in the regular binary manner. Once it reaches the 5 state, however, the reset to 4TC is enabled and the set to 2TC is blocked. Therefore the next pulse resets 1TC and 4TC, leaving the counter in its reset, or 0, state. See Figure 5-5. Note that the Storage Line Home flip-flop also sets the tens counter to its highest state (5). Thus the register counter is at register 59 throughout storage line home time.

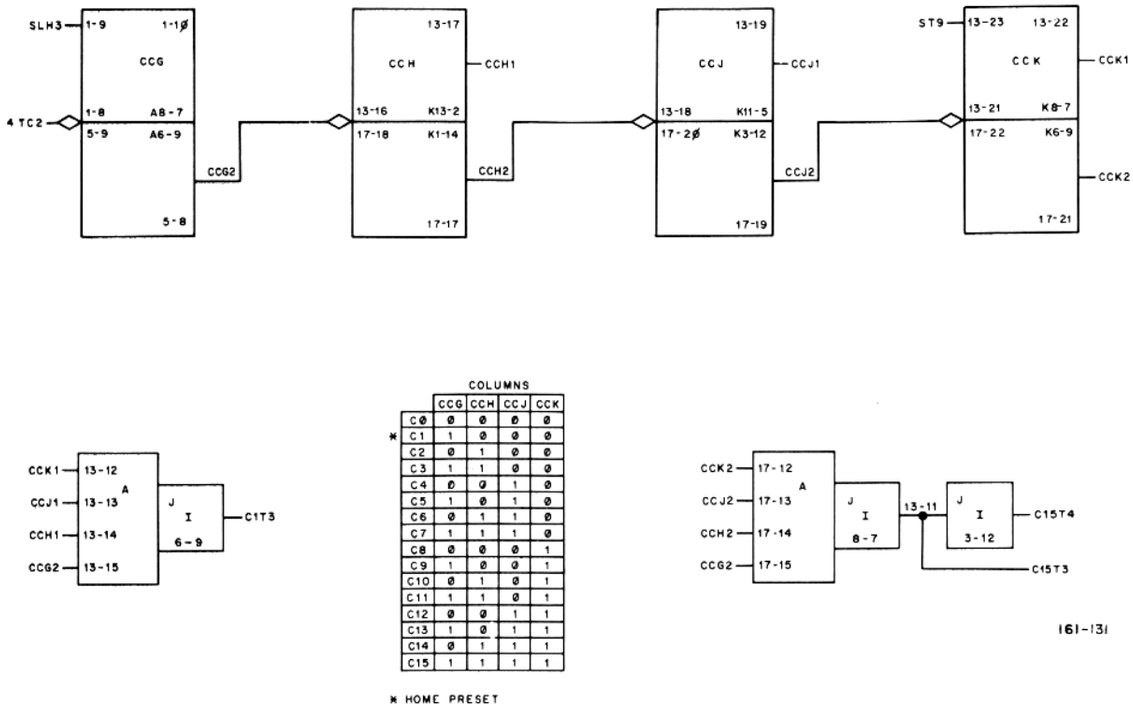


	TENS		
	1TC	2TC	4TC
0	0	0	0
10	1	0	0
20	0	1	0
30	1	1	0
40	0	0	1
* 50	1	0	1

TENS COUNTER
(All on "DD" card)

FIGURE 5-5

d. COLUMN COUNTER. The column counter is a regular scale of sixteen binary counter. It is made up of the CCG, CCH, CCJ, and CCK flip-flops. See Figure 5-6 and note that two column times are decoded, C1 and C15. This counter is triggered each time the tens counter resets. When the column counter returns to 0 from 15, CCK2 goes true and sets the Storage Line Home flip-flop. SLH3 immediately sets CCG. Thus column 0 is true only for a very short time: the propagation delay of SLH and one inverter.



COLUMN COUNTER
(All on "DD" card)

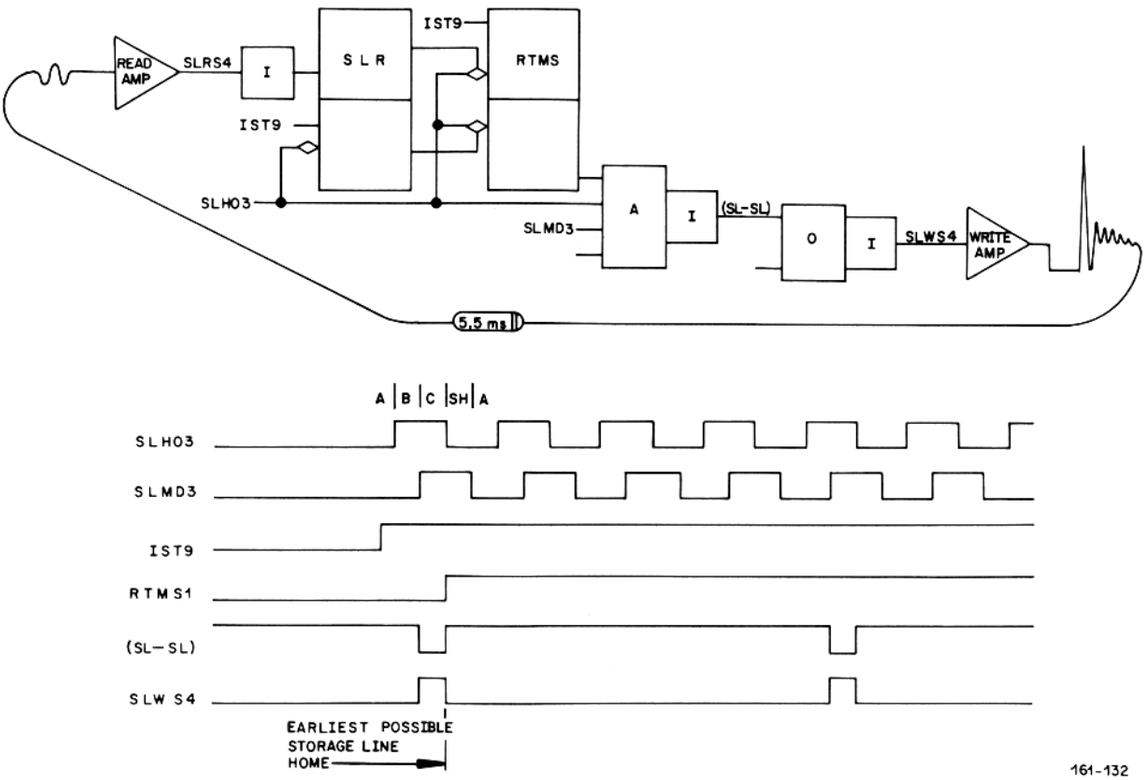
FIGURE 5-6

e. DECODING. Note that there are very few decoded outputs produced by the storage line timing chain. The SLHO and SLMD outputs are used for the basic read/write timing, in addition to driving the bit counter. Each counter has one output that drives the next counter. Besides this, the only other two outputs decoded are shown in Figure 5-6. C1T3 is used to prevent an erroneous condition at storage line home, and C15T is used to signal the end of the storage line pass at the conclusion of the store or recall function. The individual register times are not decoded: instead, a special system, which uses both outputs of each register counter flip-flop, is used to locate the proper storage register. This will be covered shortly.

3. LAUNCH SYNC PULSE

When power is first applied to the 5610, Initial STart, IST9, sets SLHO, SLMD, and CCK. This puts the fractional bit counter at "A" and the column counter somewhere between C8 and C15. IST9 also resets SLH, so that as soon as the starting pulse is withdrawn, the storage line timing chain begins to run. Since the condition of all other flip-flops in the timing chain is unknown, the length of the first cycle cannot be determined. If all other column counter flip-flops were reset, the cycle could take over 2 milliseconds. If, on the other hand, ALL other timing chain flip-flops happened to fall set, (C15S59B3), the first time SLHO became set to trigger the bit counter, all other flip-flops would reset and the cycle would end.

Refer now to Figure 5-7 and note that two other flip-flops are affected by IST9. Storage Line Read, SLR, is reset, and ReTiMe Storage, RTMS, is set. When IST9 goes false, the timing chain begins to count and the "bit" in RTMS is written out onto the storage delay line at "C" time, when the Storage Line Write Signal, SLWS4, goes false. At the next "SH" time, SLHO3 goes true, driving SLWS4 false and incrementing the bit counter. By presetting the fractional bit counter to "A" time with IST9, at least one full bit time is insured during which the sync pulse may be written out, before it is possible for the timing chain to lock up at storage line home. Most of the time, however, the timing chain will run for a much longer period of time (up to 2 ms or longer) during this first cycle. Since SLH starts out set, CCK is also set to insure that the timing chain does not start out in C0: if this were to happen, the timing chain could still be running when the sync pulse was sensed 5.5 ms later.



LAUNCH STORAGE LINE SYNC PULSE
(All on "GG" card)

FIGURE 5-7

4. BASIC READ/WRITE LOGIC

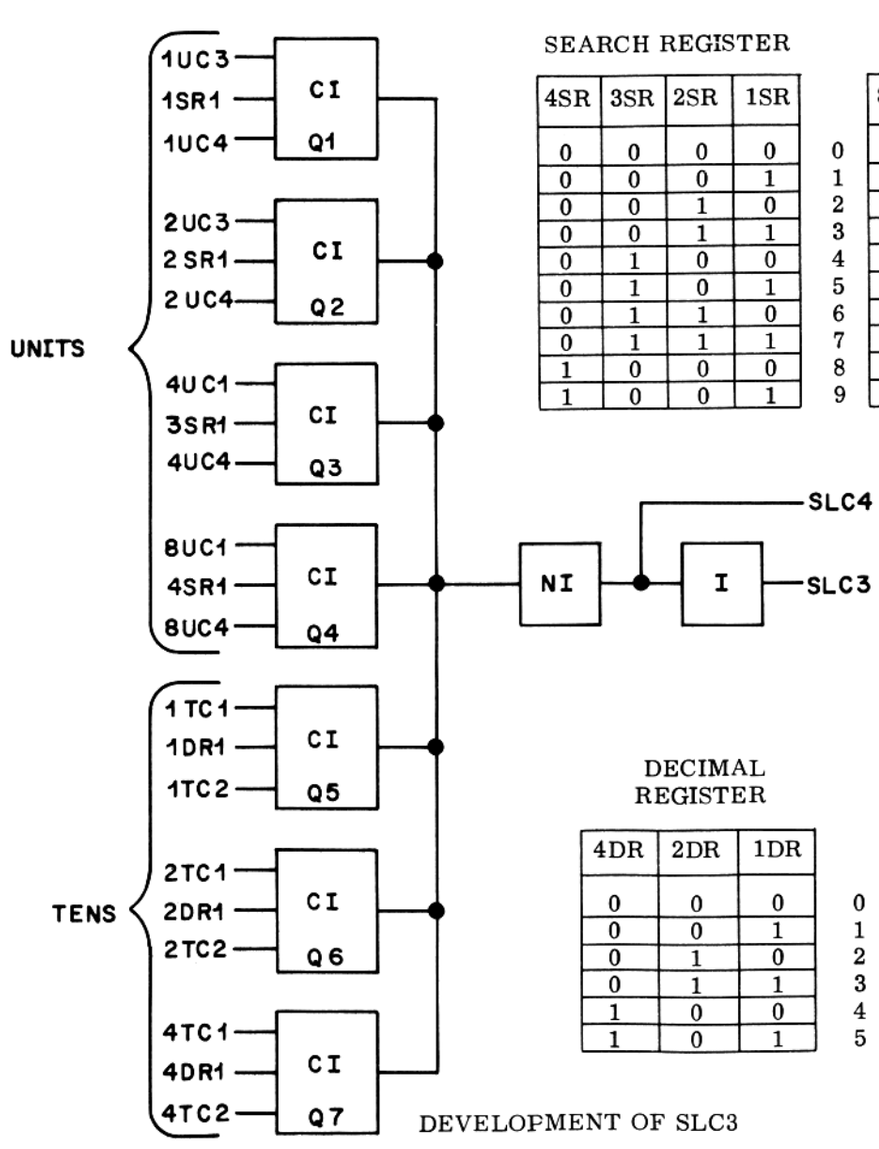
The basic operation of the storage line can also be traced in the preceding Figure 5-7. The read amplifier senses the twist emerging from the delay line and amplifies it into the normally-true Storage Line Read Signal, SLRS4. This signal is inverted, and SLRS3 sets the Storage Line Read flip-flop, SLR. Note that this occurs at some random time in relation to the timing chain. The next time SLHO3 becomes true ("SH" time) this bit transfers from SLR to RTMS. SLHO3 also resets SLR. At the following "C" time this bit is written out onto the delay line. Thus, as with the other delay lines, data is sensed randomly in relation to the timing chain, but is rewritten in sync with the timing chain.

As with the arithmetic line, fractional bit "A" is true during home time, so no data can be written during storage line home. Note also that the sync pulse, as SLRS3, resets SLH to allow the timing chain to resume operation. As stated earlier, the timing chain is preset to C1S59B0A time during program line home. Thus, when the sync pulse is sensed, one full register time is counted out (6.4 μsec.), and then the column counter steps to C2, the first column capable of holding data.

B. STORAGE LOCATION CORRECT

The gating of all data onto or off of the storage line is controlled by the Storage Location Correct signal, SLC3. This signal is developed at the proper storage time in each column of the storage line, by comparing the address of the instruction with the storage line timing chain. SLC3 is true for only one storage register time (6.4 μsec.) in each column.

The comparison is made in a group of seven "Compare Inverters", shown in Figure 5-8. These CI's are identical in both construction and operation to those used in the "Decimal Compare" function, which is covered in Section 8, Electronic Circuits. The schematic in Figure 5-9 is included here for reference.

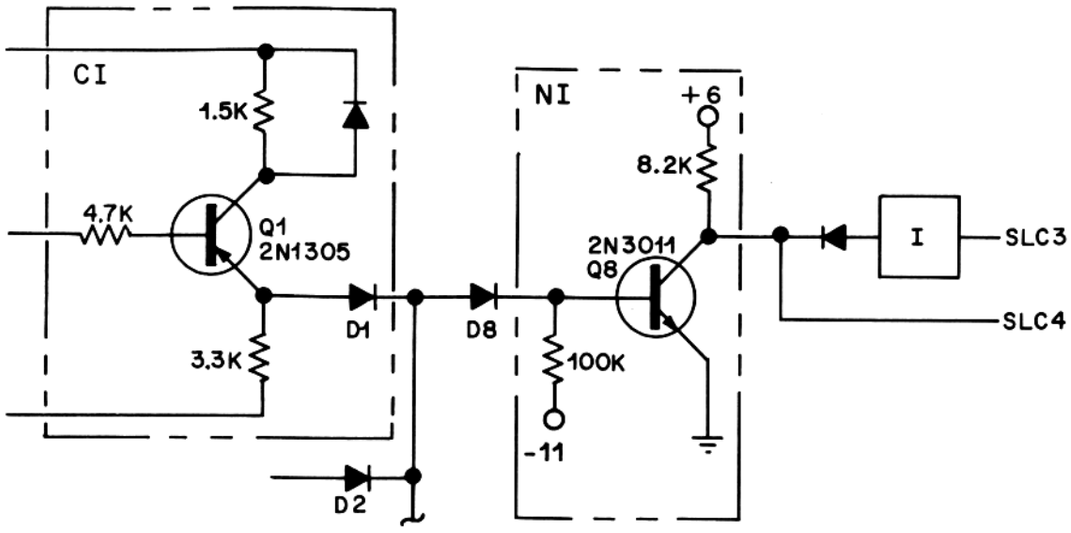


4SR	3SR	2SR	1SR		8UC	4UC	2UC	1UC
0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	1
0	0	1	0	2	0	0	1	0
0	0	1	1	3	0	0	1	1
0	1	0	0	4	0	1	0	0
0	1	0	1	5	0	1	0	1
0	1	1	0	6	0	1	1	0
0	1	1	1	7	0	1	1	1
1	0	0	0	8	1	0	0	0
1	0	0	1	9	1	0	0	1

4DR	2DR	1DR		4TC	2TC	1TC
0	0	0	0	0	0	0
0	0	1	1	0	0	1
0	1	0	2	0	1	0
0	1	1	3	0	1	1
1	0	0	4	1	0	0
1	0	1	5	1	0	1

DEVELOPMENT OF SLC3

FIGURE 5-8



SCHEMATIC, COMPARE INVERTER AND NPN INVERTER

FIGURE 5-9

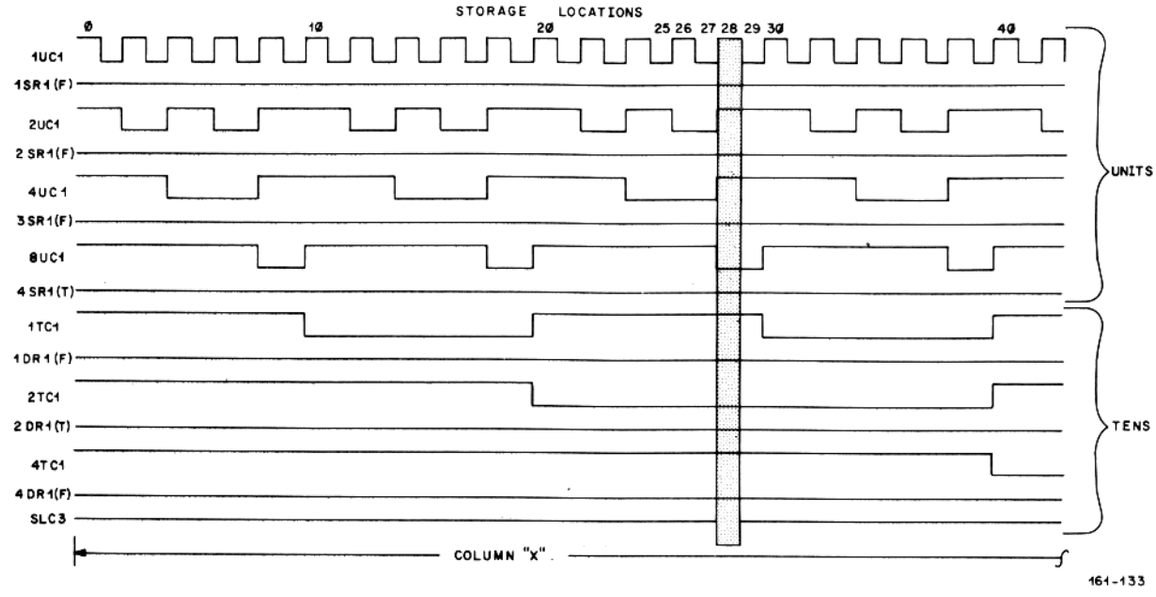
After the Store, Recall, or Accumulate command is given, the following two digits on the program line (the address) are put into the decimal register and the search register. The first digit (tens) is put into the decimal register and the second digit (units) is put into the search register. Then, as the register counter steps from 00 to 59, its output is compared with the outputs of the DR and SR flip-flops. When the storage line register counter is exactly equal to the settings in both the decimal register and the search register, SLC3 becomes true.

When the storage line register counter takes one more step, the output of at least one of the CI's will become false, and the input to the NPN inverter (NI) will be false. This will make SLC4 true and SLC3 false. These conditions remain until SLC3 again goes true at the proper storage time in the next column.

Note that SLC3 can become true at random times when a Store, Recall, or Accumulate instruction is NOT being performed. For example, if both the search register and the decimal register are reset (clear), SLC3 will become true each time the storage line register counter reaches 00. While other 5610 functions take place, short "slivers" can appear on SLC3 as data is shifted through the search register or as the decimal register is counted down. This causes no concern, however, because SLC can affect machine operation only during the one line pass in a store or recall function that includes the actual data transfer.

The logical function of the CI is to provide a true output whenever the inputs compare. That is, whenever the upper inputs (collector and base) compare (both true or both false), the CI output (emitter) is true. If the two upper inputs are not at the same logic level, the output will be false.

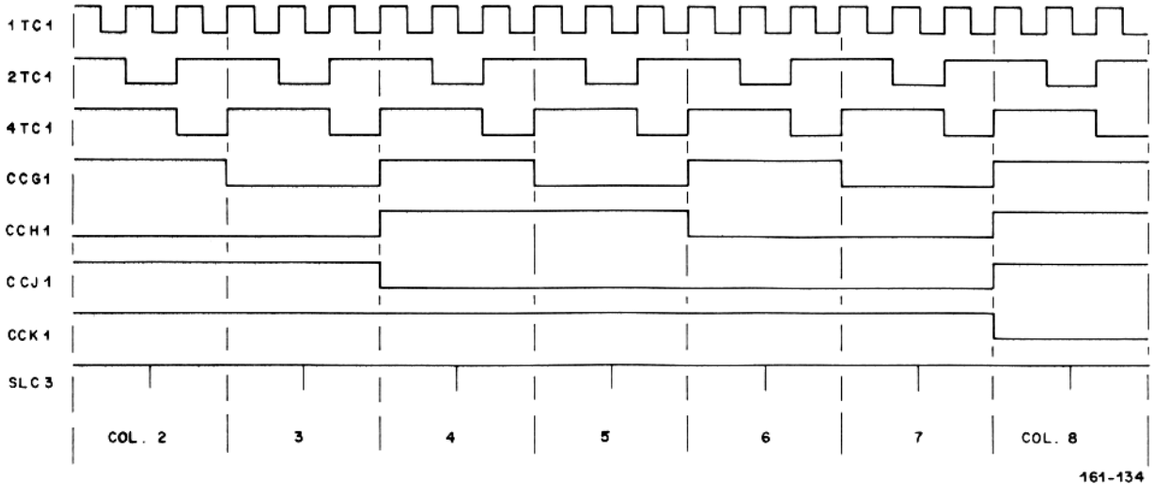
The storage line units counter and tens counter flip-flops are connected across the CI's; the set and reset outputs of each flip-flop go to the collector and emitter inputs, respectively, of the same CI. The base input to the CI is connected to the set output of the corresponding DR or SR flip-flop.



TIMING CHART
STORAGE LOCATION CORRECT

FIGURE 5-10

Figure 5-10 illustrates the storage line timing chain flip-flop waveforms used in developing SLC3 in one particular column. Figure 5-11 contains some of the same waveforms, repeated in each storage column to show that SLC3 occurs once in each column.



SLC3 OCCURS ONCE IN EACH COLUMN
(EXAMPLE: S 28)

FIGURE 5-11

C. DATA TRANSFER REGISTER

Since the basic oscillator speed of the storage line differs from that of the arithmetic line, and because each delay line is slightly different in length from the next, it is impossible to synchronize the storage and arithmetic timing chains. For this reason, data must be taken off one delay line at the proper time in relation to its timing chain, stored temporarily, and then written onto the other delay line in sync with the other timing chain.

It has already been shown that all data entering or leaving the arithmetic line must pass through the common register. Another register, called the DATA TRANSFER REGISTER, acts as a buffer for all data entering and leaving the storage line. This register is a 4-stage series shift register located on the "GG" card. It is made up of the 1DTR, 2DTR, 3DTR, and 4DTR flip-flops. It shifts data in exactly the same manner as the common register, the search register, and the operation register. It has no decoded outputs, however, and is used ONLY during Store, Recall, and Accumulate. Its input is from either RTMS (during a recall function) or 1CR (during store functions). Its only output is 1DTR2, which controls SLWS during store and 6CR during recall.

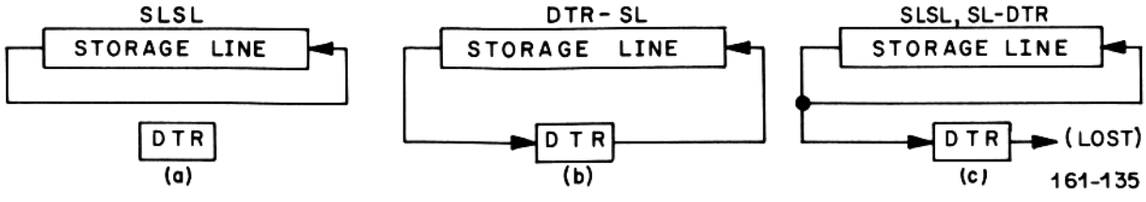
D. LINE ROUTINGS

As data emerges from the storage line read amplifier, it passes through the SLR and RTMS flip-flops to be re-timed in sync with the timing chain. From RTMS it may either be written directly back onto the delay line or it may be shifted into the data transfer register, or both. The first of these line routings, termed SLSL, can be likened to PLPL or ALAL. The sync pulse is ALWAYS recirculated SLSL. In fact, SLSL is the ONLY line routing possible at all times other than during SLC3 in a Store, Recall, or Accumulate instruction.

During a recall function, the line routing remains SLSL until storage location correct is reached. Then the line routing changes to SLSL, SL-DTR, and the contents of the proper storage register in that column are shifted into the data transfer register. Note that SLSL remains true during SLC3 to allow the data to be rewritten onto the line: this is what makes the recall function non-destructive. When the data from the proper storage location is fully shifted into the data transfer register, SLC3 goes false and the line routing returns to just SLSL. It remains this way until SLC3 goes true in the next column.

During a store function, the line routing again remains SLSL until SLC3 becomes true. In this case the line routing changes to DTR-SL ONLY. SLSL must be blocked because two sources of data cannot be used simultaneously to write onto the delay line. When SLC3 goes false, the line routing reverts to SLSL, and remains that way until SLC3 time in the next storage column. DTR-SL is accompanied by an SL-DTR shift. This has no function during store, but is allowed because it does no harm.

To summarize, there are three line routings associated with the storage line: SLSL; SLSL, SL-DTR; and DTR-SL, SL-DTR. See Figure 5-12. DTR-SL is used only when SLC3 is true during one storage line pass in a store function, SLSL, SL-DTR is used only when SLC3 is true during one storage line pass in a retrieve function. SLSL is used at ALL other times.



STORAGE LINE ROUTINGS

FIGURE 5-12

E. GENERAL OPERATION

There are many similarities between the Store, Recall, and Accumulate instructions. Their retrieval in 4 mode is identical. Their 5 mode operation is similar, even though Recall uses only two action steps, and Store and Accumulate use all five action steps. (No action steps are repeated, as they are in Multiply and Divide.) A certain amount of arithmetic logic operation precedes the actual transfer of data in each case. When the action step is finally reached during which the data transfer occurs, all data is transferred in one storage line pass. During this action step, the Storage Action Step Stop, signal, SASS, is developed to lock the action step counter until the storage line pass is complete. Whereas other instructions that utilize the arithmetic line work with R1 and R2, the three storage-oriented instructions are concerned only with R0.

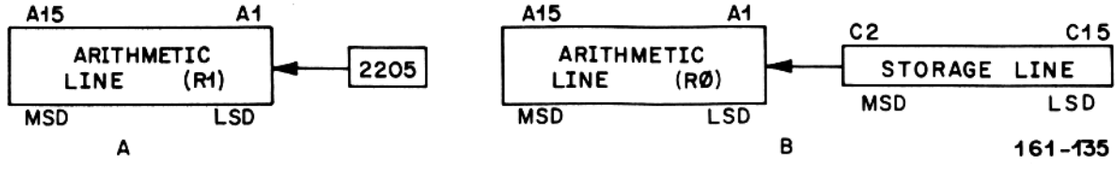
Before data can pass from the storage line to the arithmetic line or vice versa, the two lines must, in effect, become "interlocked". The information to follow will show that the two lines are not actually synchronized time-wise, but that certain operations on each line are prevented from occurring until certain other operations have finished on the other line.

Once the two lines are interlocked, a process called the "data transfer subroutine" either transfers the contents of the proper storage register to R0 (recall function), or INTERCHANGES the contents of the two registers (store function). Note that when a number has been fully stored on the storage line, the number previously in that storage location has been transferred to R0. This number is then cleared from R0 when the instruction is completed. Note that this "recall during store" has no function: it is allowed because it does no harm.

Note also in the following material that many of the flip-flops used are those already used during Add, Subtract, Multiply, and Divide: since the mathematics portion of the machine is idle at this time, these flip-flops are used instead of adding new ones. Bear in mind that these flip-flops are named for their functions during the computational instructions, and that these names do not necessarily pertain to their functions during Store, Recall, and Accumulate.

1. RELATIONSHIP TO ARITHMETIC LINE TIMING

As stated earlier, storage line data is organized in the reverse of arithmetic line data. To better understand the reason for this, picture the contents of a register on the storage line as a number being entered from the 2205. The arithmetic line action is quite similar during both Enter and Recall. Refer to Figure 5-13 (a) and note that as each digit is entered by the 2205, it is shifted into R1 in the LSD position and all previous digits are shifted left. Thus the MSD, which was entered first, is in the MSD position of R1 after all shifts left have occurred.



Enter each digit, shift R1 left (a)

Retrieve each digit, shift R0 left (b)

COMPARISON BETWEEN ENTER AND RECALL

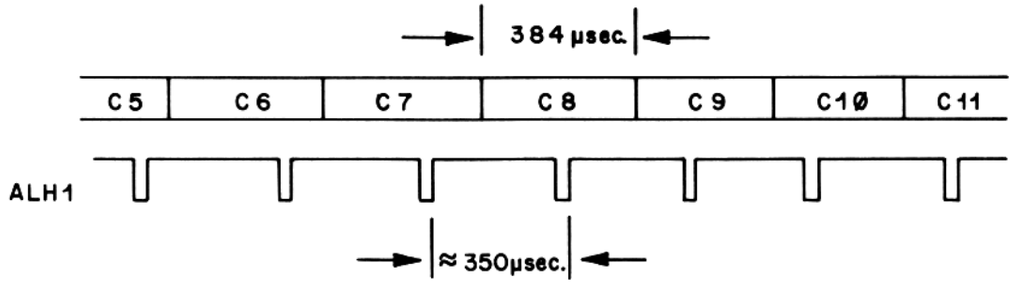
FIGURE 5-13

Now refer to Figure 5-13 (b) and note the similar arithmetic line action during Recall: each digit retrieved from the storage line causes a shift left on the arithmetic line, this time in R0. Since the MSD is in C2 on the storage line, it is shifted into R0 first. As each digit is retrieved from storage and written onto the arithmetic line, it is accompanied by a shift left of R0. After all 13 digits are retrieved, the MSD has been shifted into A13. Then the sign is retrieved, and as it is shifted into A1 the MSD shifts into A14. Note that the number on the storage line is NOT shifted left. Instead, after each digit is retrieved, it is simply recirculated on the delay line.

Another parallel can be drawn between the two print instructions and the Store instruction. During DPR or NDP, each digit, MSD first, is shifted into the common register and then sent to the writing machine. The printing of each digit is followed by a shift left of R1. During Store the same basic arithmetic line operation occurs (except that R0 is used): the MSD is shifted into the common register and from there (eventually) into C2 on the storage line. Then R0 is shifted left, and the next digit is shifted into the common register. This digit is soon written into C3 on the storage line. R0 is shifted left once for each digit that is stored, and each digit is stored in the next higher column on the storage line (at SLC3).

Note one basic fact in the preceding discussion: for each digit on the storage line, an arithmetic shift left occurs. Since an arithmetic shift left takes one full line pass (≈350 μsec.), it is complete before the next digit can be retrieved from (or stored in) the next SLC3, which is 384 μsec. away.

Figure 5-14 shows the basic time relationship between the two delay lines. Note that more than one full arithmetic line pass occurs during each storage column time. This makes it possible for a digit to be stored in or retrieved from each storage column, and still allows time for an arithmetic shift left before the next SLC3.

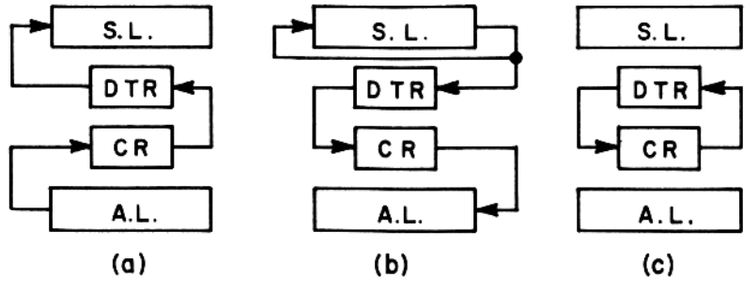


TIME RELATIONSHIP BETWEEN STORAGE AND ARITHMETIC LINES

FIGURE 5-14

2. BASIC DATA TRANSFER

The basic transfer of data is also the same in either a store or recall function. This is illustrated in Figure 5-15. The store function, shown in Figure 5-15 (a), first shifts the data (one digit at a time) from the arithmetic line into the common register. Then the data is transferred from the common register into the data transfer register, and from there it is written out onto the storage line. The recall function is pictured in Figure 5-15 (b). In this case the data flows from the storage line into the data transfer register, then to the common register, and finally onto the arithmetic line. In each case, three separate and distinct operations actually take place. First, (1) the data must be obtained from its delay line in sync with its timing chain, then (2) be transferred to the other register, and (3) finally be written onto the other delay line in sync with the other timing chain. Bear in mind that these three events occur 14 times during each store or recall function: once for each digit and once for the sign.



BASIC DATA TRANSFER

FIGURE 5-15

The actual transfer of data between the common register and the data transfer register is the same in either case. When this transfer occurs, the contents of the two registers are simply interchanged. This is illustrated in Figure 5-15 (c). Before the transfer is made during a recall function, the common register is clear (because all of R0 is clear), so when the transfer occurs this clear condition shifts into the data transfer register. During a store function, the digit formerly in the previous column of the storage register is in the data transfer register. This digit is shifted into R0, which is cleared after the instruction is completed. The signal which allows the transfer to occur is called Enable TRansfer, ETR3. During a store function, it is developed at A0R1 time after the data has been shifted from the arithmetic line into the common register. During a recall function it is also developed at A0R1 time, after the data has been shifted from the storage line into the data transfer register. This is the basic idea behind the "interlocking" of the two timing chains: during a store function, the transfer of data cannot occur until the digit has been retrieved from the arithmetic line; during a recall function the transfer is not allowed until the digit has been retrieved from the storage line.

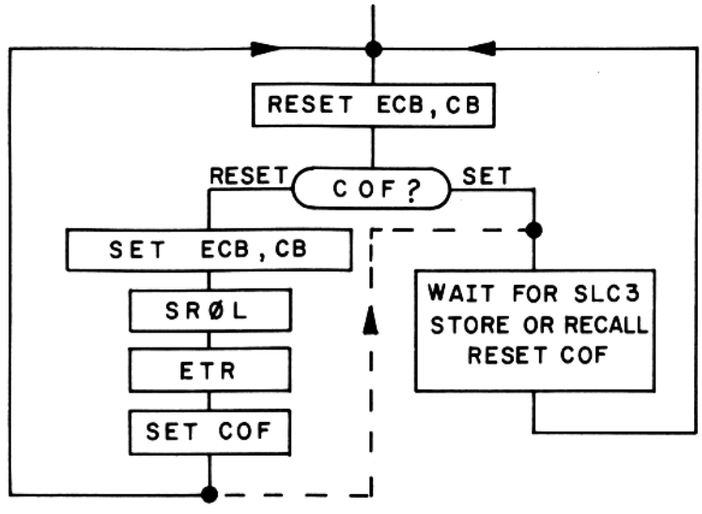
F. DATA TRANSFER SUBROUTINE

The transfer of data from one delay line to the other is, for the most part, identical in both the store and recall functions. A definite routine is followed for the transfer of each individual digit. This routine is called the "data transfer subroutine".

The basic data transfer subroutine is pictured in Figure 5-16. It is needed 14 times during each store or recall function: once for each of the 13 digits and once for the sign. It actually occurs a few extra times, however, because certain parts of it are necessary in initiating and terminating the two functions. This aspect will be covered later, as the functions are studied individually.

1. BASIC OPERATION

Enable Carry/Borrow, ECB, sets at A0R0B1 and Carry/Borrow, CB, sets at A0R0B2. With CB set, Shift R0 Left, SR0L, is developed. Before R0 is actually shifted, however, ETR3 is developed (at A0R1), and this causes COF to set. ETR3 also triggers the data transfer between the common register and the data transfer register. During the remainder of the arithmetic line pass R0 is shifted left. This shift either puts data into R0 (recall) or takes data out (store). The storage line meanwhile waits for SLC3 to become true: at storage location correct a data transfer occurs between the data transfer register and the storage line. SLC also resets COF so that this process can repeat on the next arithmetic line pass: ECB and CB become set, ETR sets COF and triggers the DTR-CR and CR-DTR swap, and R0 is shifted left.



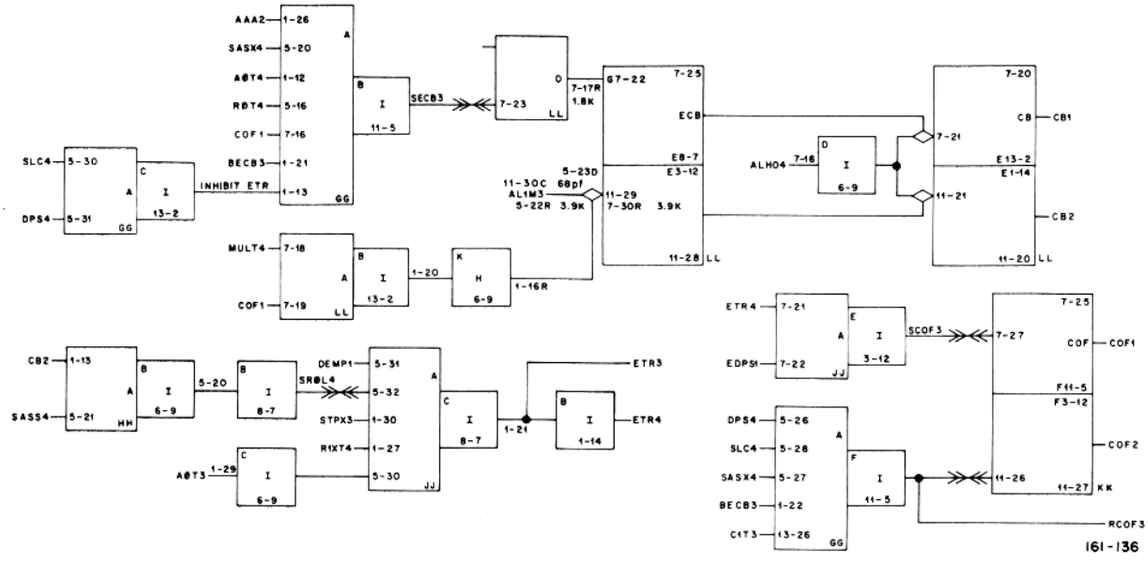
FLOW CHART
DATA TRANSFER SUBROUTINE

FIGURE 5-16

This same operation works in both a store and a recall operation. In store, the digit to be stored is shifted into the common register as a result of the R0 shift left. It is transferred to the data transfer register on the next line pass, and at SLC it is shifted onto the storage line. In recall, the digit being brought from storage is shifted into the DTR during SLC, transferred to the common register by ETR3, and then shifted into R0 by SR0L. Note that the operation is the same in either case, and that only the initial starting point is different. Once the operation begins, it occurs over and over, the only difference between store and recall being in the storage line routing during SLC3.

2. LOGIC

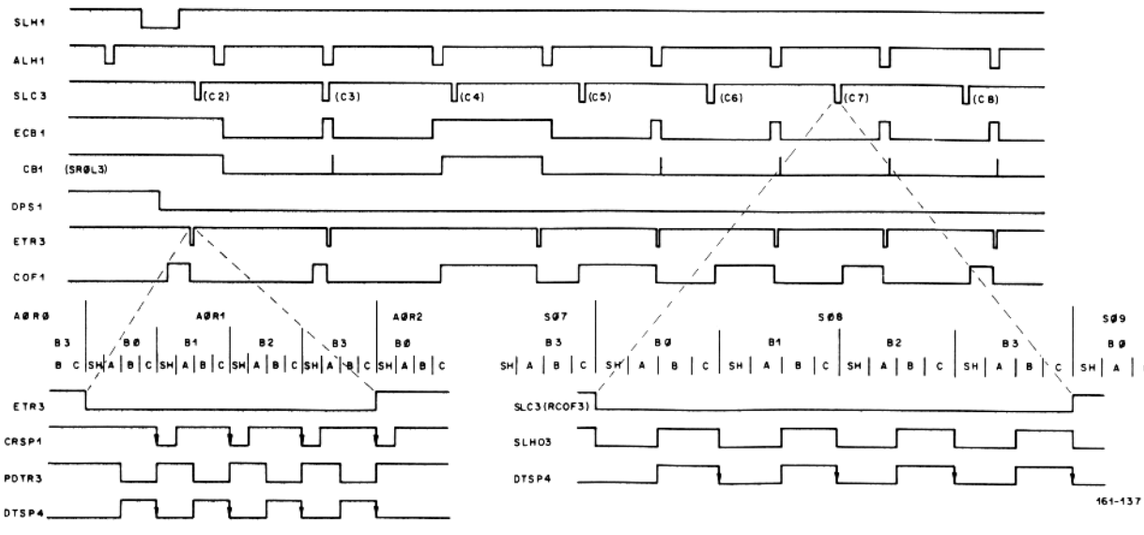
The logic involved in the data transfer subroutine is pictured in Figure 5-17. It begins at A0R0B1 time when AAA2 goes false to develop Set Enable Carry/Borrow, SECB3. CB sets at the next bit time (A0R0B2) to produce SR0L. Before the shift of R0 actually begins, however, ETR3 is developed, which sets COF. ETR3 also triggers the transfer of data between the common register and the data transfer register, which will be covered shortly. As the arithmetic line pass continues, R0 is shifted left, and the storage line waits for storage location correct. When this occurs, SLC develops Reset Carry OverFlow, RCOF3, which not only resets COF, but also enables the Data Transfer Shift Pulse, DTSP, which shifts data into or out of the data transfer register. (This last item is not shown in Figure 5-17, but will be covered shortly.) ECB resets at arithmetic line home, when the R0 shift left is complete. Note that CB remains set through arithmetic line home, and resets at the next bit time, which occurs after the sync pulse is sensed and the timing chain restarts.



DATA TRANSFER SUBROUTINE LOGIC

FIGURE 5-17

At A0R0B1, when CB resets, ECB will set again (provided COF is reset), and begin the entire subroutine over again. In actual machine operation, this is the true sequence of events most of the time. However, at least once during each store or recall function, an arithmetic line pass begins at a time when SLC has not yet occurred to reset COF. If COF is set at A0R0B1, it blocks the set of ECB, and this causes the beginning of the subroutine to wait for one more arithmetic line pass. This is caused by the difference in the length of each storage column (384 μsec.) and the length of the arithmetic line ($\approx$350 μsec.). See Figure 5-18.

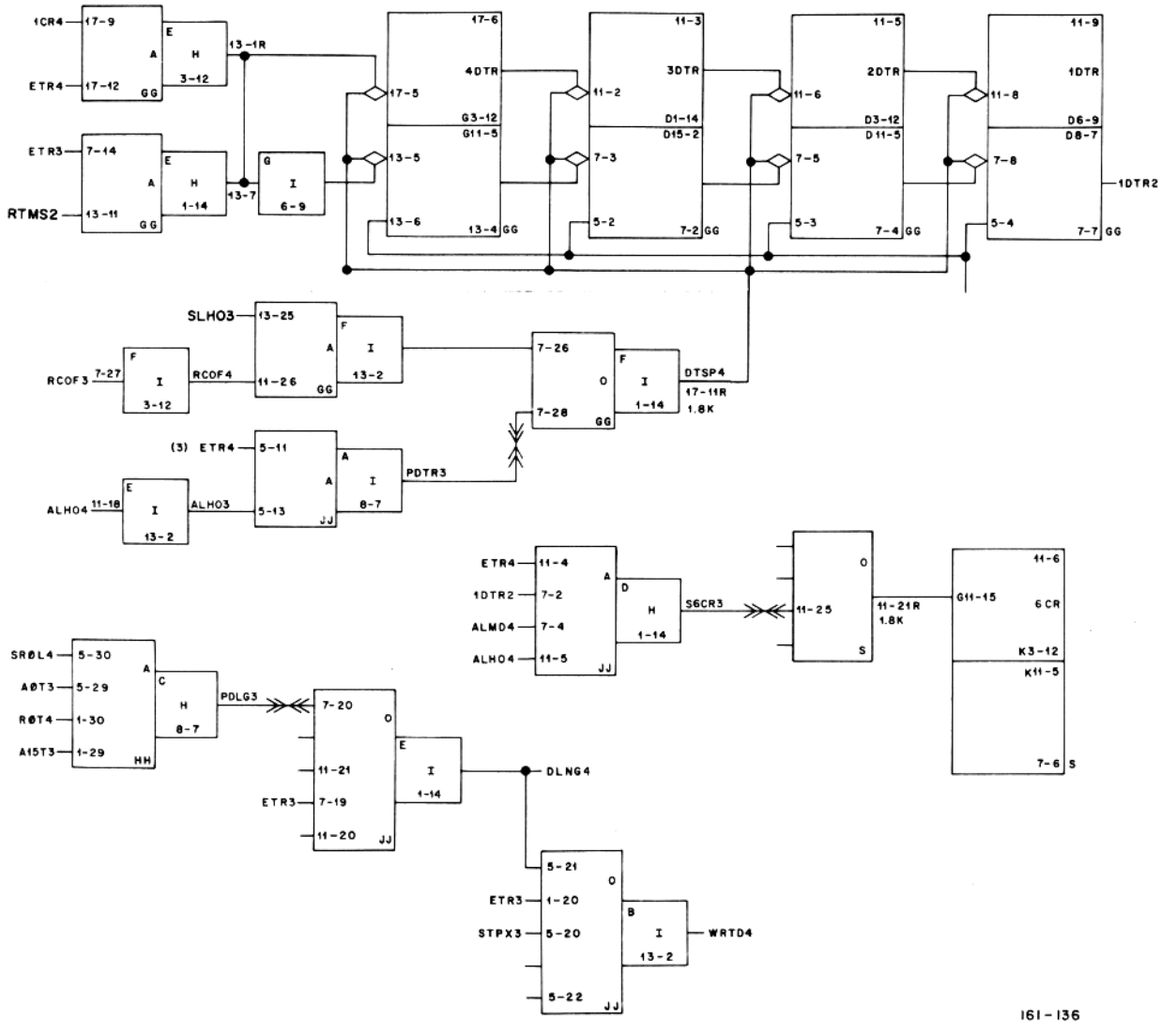


TIMING CHART
DATA TRANSFER SUBROUTINE

FIGURE 5-18

Figure 5-18 also shows the time relationships between some of the various pulses that occur during ETR3. Refer also to Figure 5-19, which shows the logic that develops these signals. When ETR3 goes true it develops DLNG. The ONLY action that ALWAYS occurs during DLNG is the shifting of data through the common register. This is the only action caused by DLNG now: CRSP is enabled by DLNG4, and each ALHO3 sets CRSP, shifting the CR flip-flops.

At the same time, ETR4 enables the Pulse Data Transfer signal, PDTR3, which becomes true at each ALHO4. Each time PDTR3 goes false, DTSP4 goes true and shifts the data transfer register. Note that in this case the DTR flip-flops shift in sync with the arithmetic timing chain.



LOGIC USED DURING ETR3

FIGURE 5-19

While both registers are being shifted, ETR gates the output of 1CR into 4DTR and the output of 1DTR into 6CR. Since ETR3 remains true for one register time (A0R1), ALHO sets four times and provides four shifts. This allows the entire contents of each register to be shifted into the other.

Note also that ETR holds WRTD4 false while DLNG4 goes false. This is to ensure that the arithmetic line routing remains ALAL during the transfer.

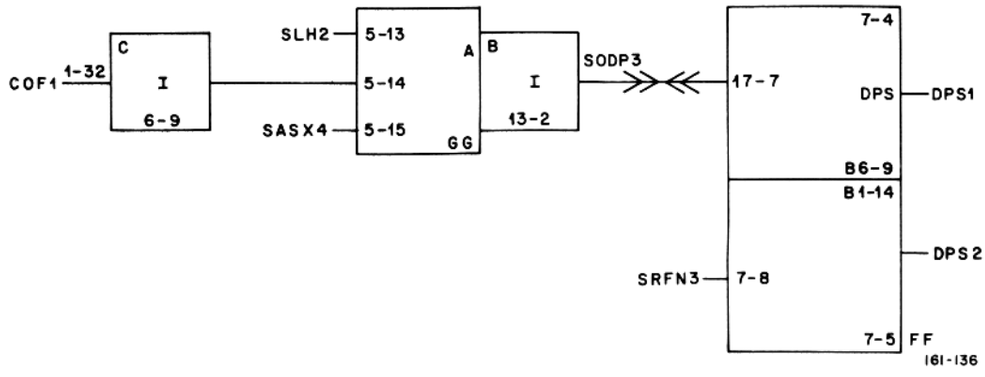
Further, note in Figure 5-19 that DTSP4 is also developed by SLHO3 and RCOF4. When RCOF4 goes false, it allows pulses to appear on DTSP4 in sync with the storage line timing chain. These pulses shift data from the storage line into the data transfer register, or vice-versa. Since RCOF is developed by SLC, it allows four pulses to appear on DTSP4, thus shifting 4 bits, or one full digit, into or out of the data transfer register. These timing relationships are also shown in Figure 5-18.

Finally, note that the RTMS input to 4DTR is always enabled (except during ETR). Therefore it is only a matter of developing pulses on DTSP4 to shift data into the data transfer register from the storage line. This actually occurs as a digit is being stored on the storage line: SL-DTR occurs with DTR-SL. This SL-DTR line routing is necessary in a recall function, and it is allowed to occur during a store function because it is harmless.

G. INTERLOCKING THE TWO DELAY LINES

It was stated earlier that the contents of an entire storage register are recalled or stored in one storage line pass. Once storage line home is reached, the data transfer subroutine begins to transfer the number, digit-by-digit, from one line to the other. In the case of a recall function, the arithmetic line needs simply to wait for the first digit to be retrieved from the storage line. In a store function, however, the first digit to be stored must be retrieved from the arithmetic line BEFORE the storage line pass begins, so that the digit is ready to be stored when the first storage location correct occurs.

In both cases, the Decimal Point Storage flip-flop, DPS, is used to signify the fact that storage line home has been reached and the important storage line pass has begun. During a store function, DPS cannot set until after the first digit has been retrieved from the arithmetic line. This is accomplished by qualifying the setting of DPS with COF (see Figure 5-20). A data transfer subroutine is then required to set COF. At the storage line home after COF becomes set, Set Only Decimal Point storage, SODP3, is developed, setting DPS and signifying the beginning of the line pass during which the data is actually transferred. A data transfer subroutine also occurs during a recall function, but only to set COF: no data is actually transferred. In either case, COF is prevented from resetting until after DPS sets (see Figure 5-19). This prevents any more data transfer subroutines from occurring until after storage line home.



SETTING DPS AT STORAGE LINE HOME

FIGURE 5-20

H. TERMINATING THE STORAGE INSTRUCTIONS

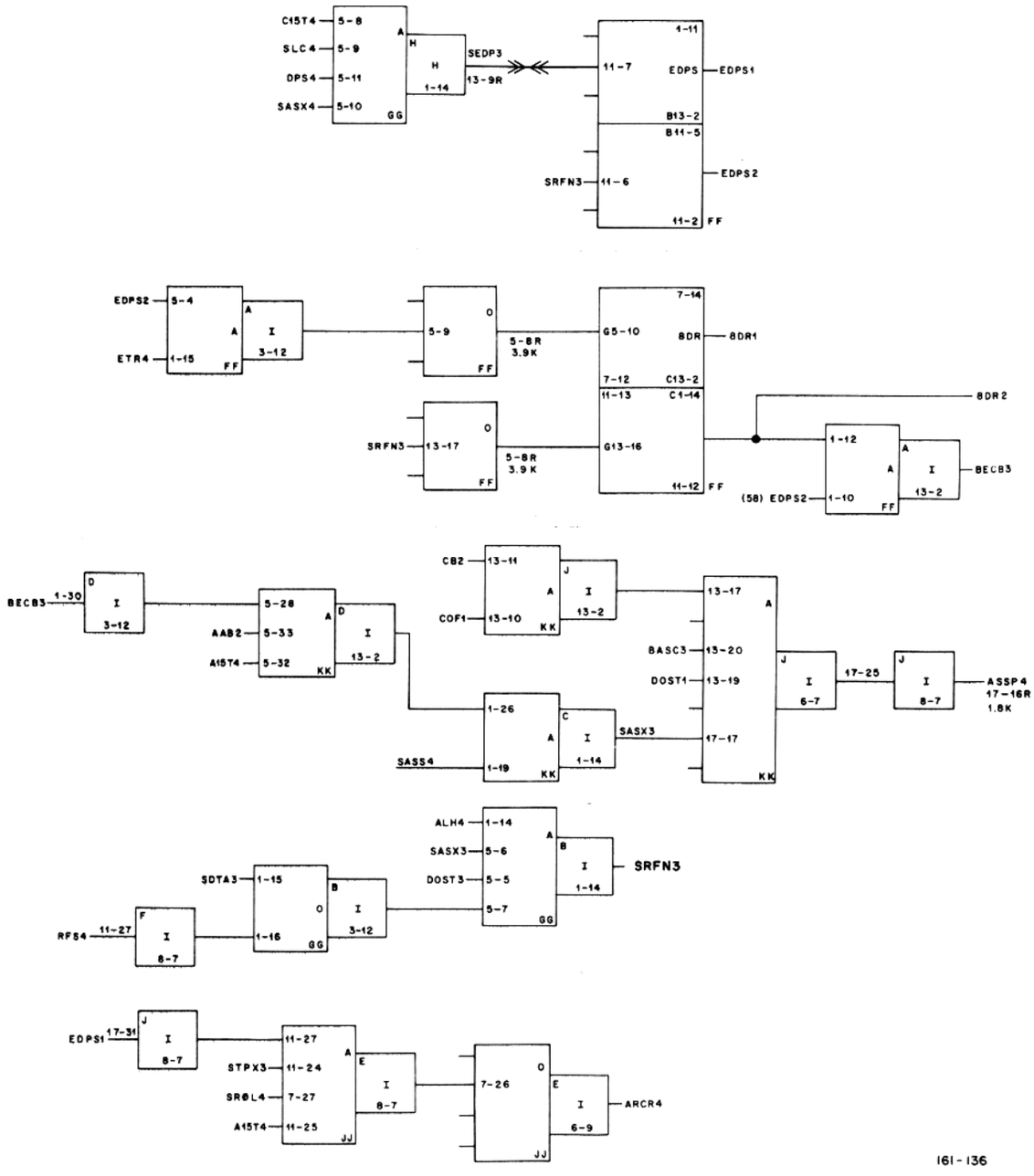
The method used to terminate the three storage-oriented instructions is the same in all cases. Once the storage line pass is completed, the SASS signal is terminated and the action step counter is allowed to proceed.

In a store function the required action is complete after the last digit (sign bit) is stored in storage location correct in C15. The action step counter could proceed immediately. In a recall function, though, the last digit is retrieved from the storage line at SLC in C15, and must still be transferred to the arithmetic line. Therefore, one more ETR3 and one more shift left of R0 are required before the action step counter can proceed. Actually, two more arithmetic line passes occur, the last one being a "dummy". This will be covered shortly. Since these extra line passes do no harm, they are allowed to occur in a store function, even though they are not needed.

At storage location correct in C15, Enable Decimal Point Storage, EDPS, becomes set. See Figure 5-21. SLC resets COF and EDPS blocks it from any further setting (see Figure 5-17). One more transfer is needed during a recall function, so ETR is developed on the next line pass in both the store and recall functions to set 8DR. With both 8DR and EDPS set, Block Enable Carry/Borrow, BECB3, is developed. This signal performs two important functions: it blocks the set of ECB on the next arithmetic line pass, thus preventing any further data transfer subroutines, and it terminates SASS (drives SASX3 false), so that the action step counter can proceed. Note, however, that the action step counter cannot step the first time SASX3 goes false because ASSP4 is held true by CB being set. ECB resets at arithmetic line home, CB resets at the next bit, and one more arithmetic line pass takes place. During this last line pass the line routing remains ALAL. At A15R4B2 time ASSP4 finally goes false and then steps the action step counter as it returns to true at arithmetic line home. Once the action step counter steps, SASX3 remains false, so StoRage FiNished, SRFN3, is developed, which resets EDPS, DPS, 8DR, and the four DTR flip-flops.

Note the development of ARCR during A15 time of the last shift left. When a Recall (or Accumulate) is the first storage instruction performed after machine turn-on, the number in the data transfer register resulting from the turn on is shifted all the way through R0 and into the common register at A15 time of the last shift. ARCR is needed to clear this number (which may be any number from 0 to 15) and leave the common register reset.

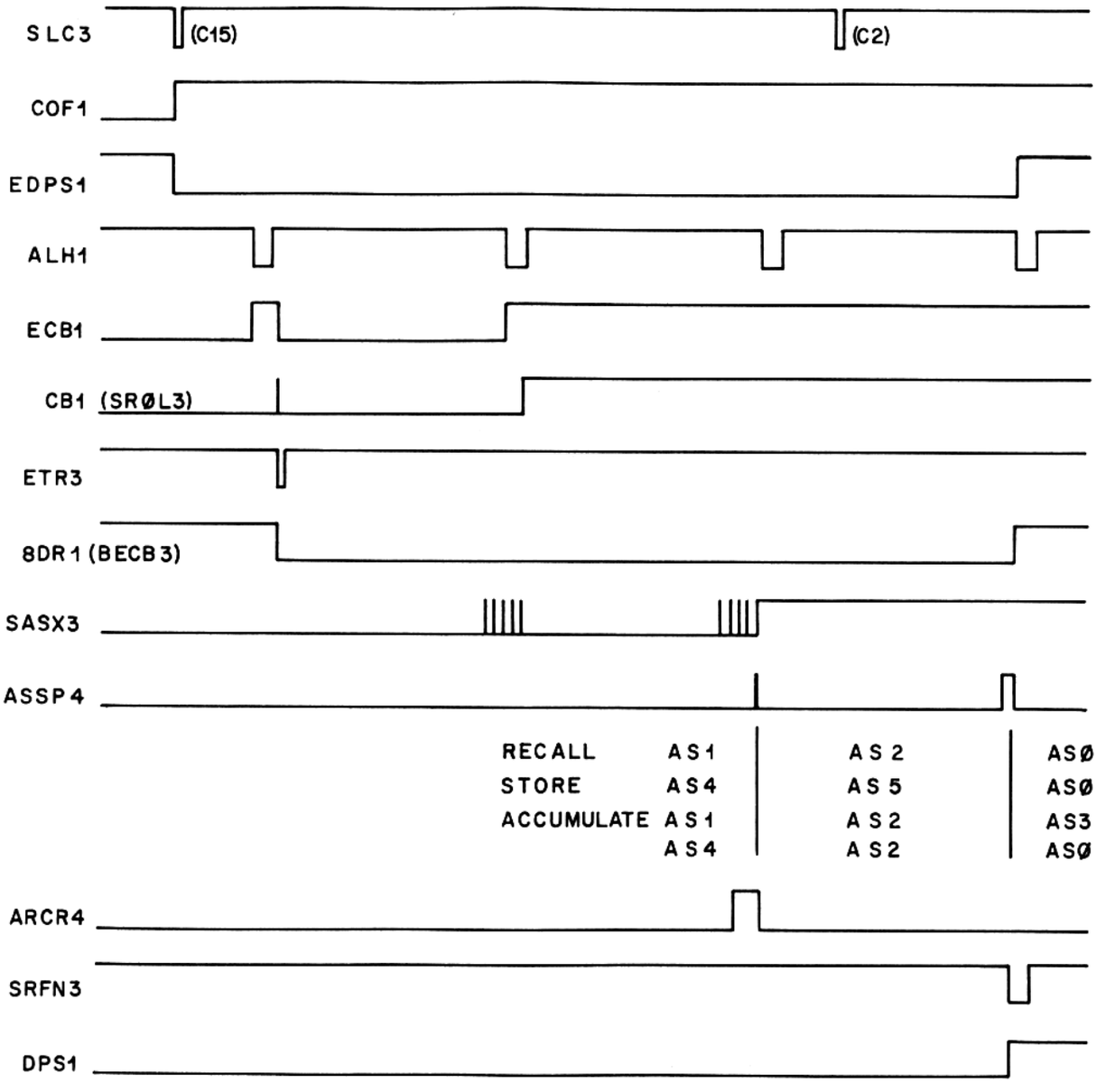
The timing chart in Figure 5-22 shows the timing relationships of the various signals involved in the termination of a Store, Recall, or Accumulate instruction.



161-136

STORAGE TERMINATION LOGIC

FIGURE 5-21



TIMING CHART
TERMINATING STORAGE INSTRUCTION

FIGURE 5-22

I. STORE, RECALL, AND ACCUMULATE

Many of the details pertaining to the Store, Recall, and Accumulate instructions are covered earlier in this section. Most of these are functions that are common to all three instructions, and are performed similarly in all three. The preceding information should be understood thoroughly before going on in this study.

1. GENERAL OPERATION

All three instructions operate in conjunction with a storage register designated by the 2-digit address of the instruction. The Store instruction transfers the contents of R1 to this storage. The Recall instruction transfers the number in this storage register to R1. The Accumulate instruction algebraically adds the contents of R1 to the number already in this register.

a. INSTRUCTION RETRIEVAL. The instructions are retrieved in 4 mode the same as the other 2-address instructions (Enter and Print). The command is shifted into the operation register. The first of the two address digits is set into the decimal register and the second is shifted into the search register.

b. EXECUTION. After instruction retrieval, a jump to 5 mode 15 phase occurs, and subsequent 5 mode operation of the phase counter and related program line logic is the same as for other arithmetic instructions.

(1) Recall. During action step 1 the number in the designated storage register is transferred to R0. In action step 2 the operating stack is shifted up, and the instruction is complete.

(2) Store. Action step 1 is a "dead" step during which no arithmetic line action occurs and the data is simply recirculated ALAL. In action step 2 a "dummy" add occurs. This is a regular add function, but since R0 is clear the contents of R1 remain unchanged. A shift down of the operating stack occurs in step 3. In step 4 the actual transfer of data from R0 to the storage line takes place. The last action step is another "dead" step.

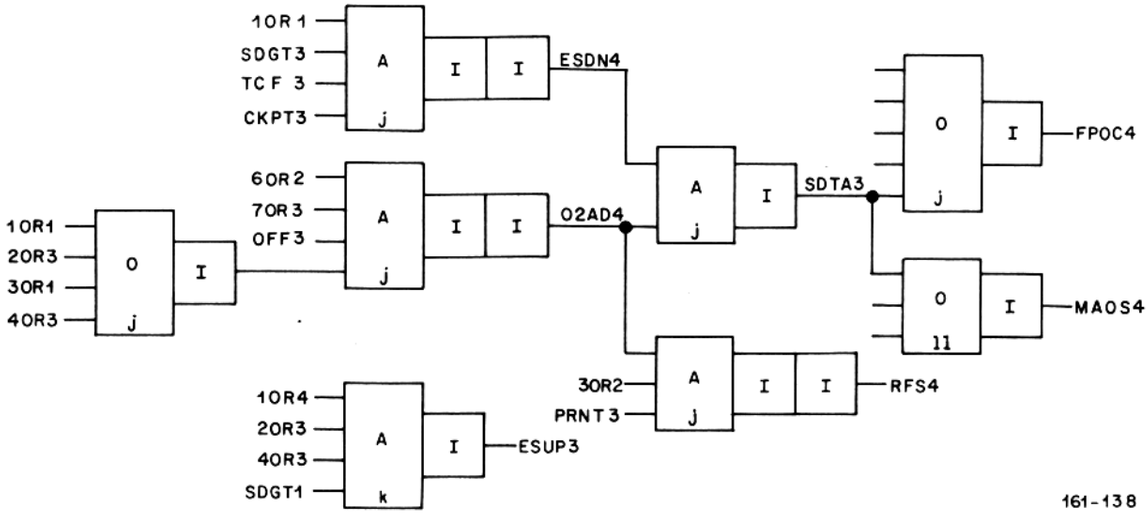
(3) Accumulate. In action step 1 the number in the designated storage register is transferred to R0. In step 2 the contents of R1 and R0 are algebraically added, the answer appearing in R1. The operating stack shifts down in action step 3. In step 4 the answer, now in R0, is transferred to the storage line, thus completing the accumulation. Action step 5 is a "dead" step.

2. INSTRUCTION RETRIEVAL

The retrieval of all three instructions is virtually identical: the only difference is in the master signals developed. The Store command (letter "S", 2-5-6 code), the Recall command (letter "V", 1-3-6 code), and the Accumulate command (letter "W", 2-3-6 code) are shifted into the operation register and the Operation 2 ADdress signal, O2AD, is immediately developed. Refer to Figure 5-23. Note that both Store and Accumulate develop Enable Shift DowN, ESDN, Store DaTA, SDTA, Machine Add Or Subtract, MAOS, and Five Phase Op Code, FPOC. Accumulate also develops Recall From Storage, RFS. The Recall command develops only RFS and Enable Shift UP, ESUP, in addition to O2AD.

Being 2-address commands, all three cause jumps from 3 phase to 6 phase, where the marker is written back out onto the line and the first address digit is shifted into the common register. At T6A of 6 phase this digit is strobed into the decimal register. In 7 phase the second address digit is shifted into the search register. The phase counter then steps to 8, where it remains until program line home, and a jump to 5 mode 15 phase occurs.

Once 5 mode is reached, DOST resets and the action step counter steps to 1. From this point on, the different master signals cause different operation for each of the commands. Each of the three will now be discussed individually.

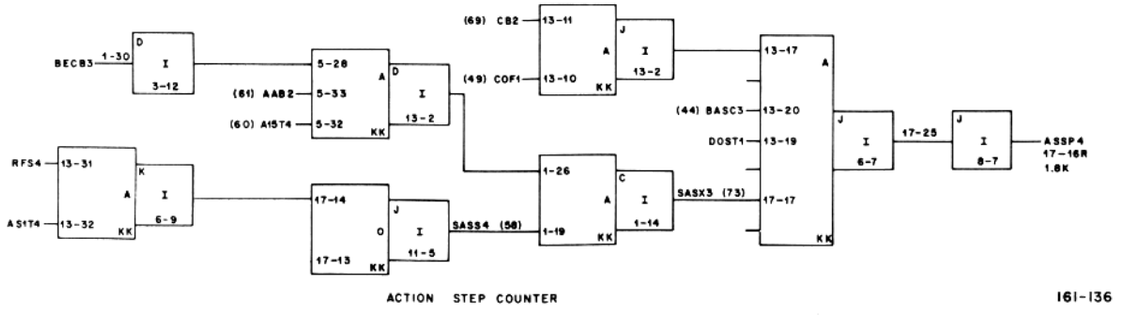


STORAGE MASTER SIGNALS

FIGURE 5-23

3. RECALL

As soon as action step 1 is reached, RFS develops SASS4. This enables SASX3, which holds ASSP4 true, thus locking the action step counter at 1. It will remain in this state until Block Enable Carry/Borrow, BECB3, is developed after the entire transfer of data is complete. See Figure 5-24.



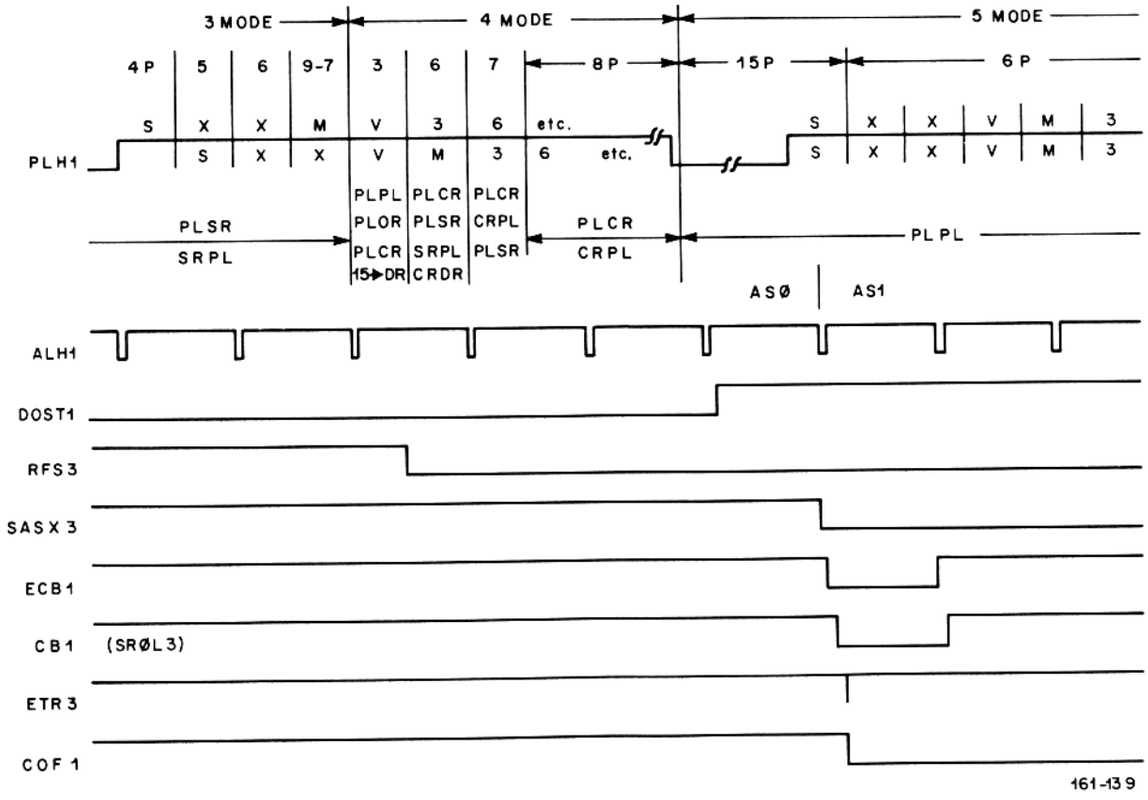
LOCKING IN ACTION STEP 1

FIGURE 5-24

When the timing chain restarts, SECB3 is developed, which initiates a data transfer subroutine (see Figure 5-17). Note that the shift left of R0 during this first line pass in action step 1 has no function because R0 is still void of data. ETR occurs, however, and sets COF, so no further subroutines can occur. Note also that when ETR occurs, a transfer actually takes place between the common register and the data transfer register. This transfer is redundant, but the ETR is necessary to set COF. With COF set, the data transfer subroutine is disabled until after storage line home. The action to this point is shown in the timing chart in Figure 5-25.

At storage line home DPS sets, and at the next storage location correct (C2) COF resets and enables the start of another data transfer subroutine. Also during SLC3 the data in column 2 of the storage line is shifted into the data transfer register. RCOF, which is developed during SLC, allows the development of pulses on DTSP4 that are in sync with the storage line timing chain. This shifts the output of RTMS into the data transfer register at the same time that it is rewritten onto the storage line. Note the similarity between the retrieval of a digit from the storage line and retrieval of a character from the program line.

As the next data transfer subroutine begins, ETR transfers the first digit to the common register, and SR0L shifts it into A1R0. ETR also sets COF so that another subroutine cannot begin until after the next digit is retrieved from the storage line. When this occurs SLC resets COF. This procedure repeats 13 more times, transferring all 13 digits and the sign bit. Remember that at least once during the storage line pass, two arithmetic line homes will occur between SLC's, resulting in one ALAL (dummy) line pass. After all 14 columns of data have been transferred, the action step counter steps to 2. (Refer to Section 5H on Terminating Storage Instructions.)



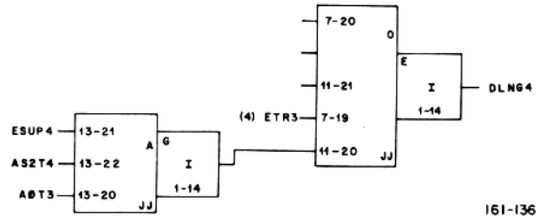
TIMING CHART

RECALL
PRELIMINARY OPERATIONS

FIGURE 5-25

TP-161SL

In action step 2, ESUP provides DLNG at all times except A0, so the sync pulse is rewritten direct and ALL data is written long. See Figure 5-26. This provides the shift up of R0 data into R1 and the loss of R4 data. At the end of step 2, the action step counter returns to 0. DOST sets at the next A1 time, enabling the phase counter's step to 7 phase at the next T6B. At program line home a jump to 3M6,4 occurs and the search for the next instruction begins.



SHIFT UP IS AS2

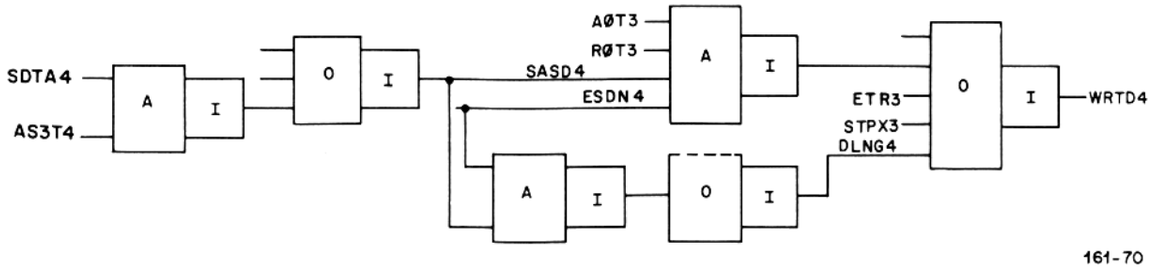
FIGURE 5-26

4. STORE

Action step 1 during Store is a "dead" step, with no action. The arithmetic line idles through step 1 ALAL.

a. "DUMMY" ADD. At the next arithmetic line home a step to action step 2 occurs, where an add line pass takes place. MAOS allows all of the operations described in Section 4K, Add or Subtract, to occur during this line pass, but since R0 is clear to begin, zero is added to R1 and its contents remain unchanged. In other words, this add line pass is of no value. It is necessary, however, in executing the Accumulate instruction.

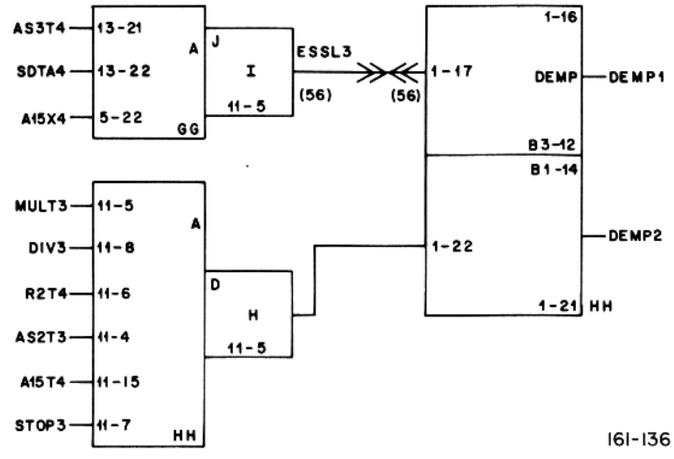
b. SHIFT DOWN. At the end of the "dummy add" line pass, the action step counter steps to 3 (because FPOC has been developed), where a shift down of the operating stack occurs. Refer to Figure 5-27 and note that SDTA in action step 3 produces SASD4, which is ANDed with ESDN to produce DLNG4 for the entire line pass. Another gate produces "write direct" at all times other than A0 and R0. Thus the sync pulse is written long at A0, and R4 data is written long at R0 time, but all other data goes direct. This gives the effect of shifting R1, R2, R3, and R4 down by writing them "closer" to the sync pulse, and repeats R4 data. Note that this is identical to the shift down that occurs during Delete and during Add or Subtract.



SHIFT DOWN DURING AS3
(All on "JJ" card)

FIGURE 5-27

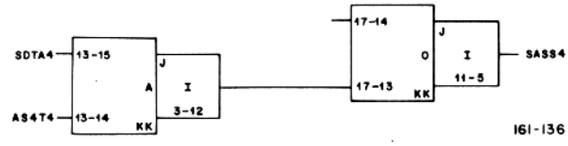
Near the end of this line pass, Enable Store Shift Left, ESSL3, is developed, which sets the DEMP flip-flop. See Figure 5-28. DEMP prevents the development of ETR3 during the first data transfer subroutine so that the first digit to be stored can be retrieved from the arithmetic line without causing an actual transfer between the common register and the data transfer register. At the end of this line pass a step to action step 4 occurs.



DEVELOPMENT OF ESSL3

FIGURE 5-28

c. LOCKING IN ACTION STEP 4. Action step 4 and SDTA develop SASS4, which locks the counter in step 4 until BECB3 is developed at the end of the transfer. See Figure 5-29 and refer to Figure 5-24 if necessary.



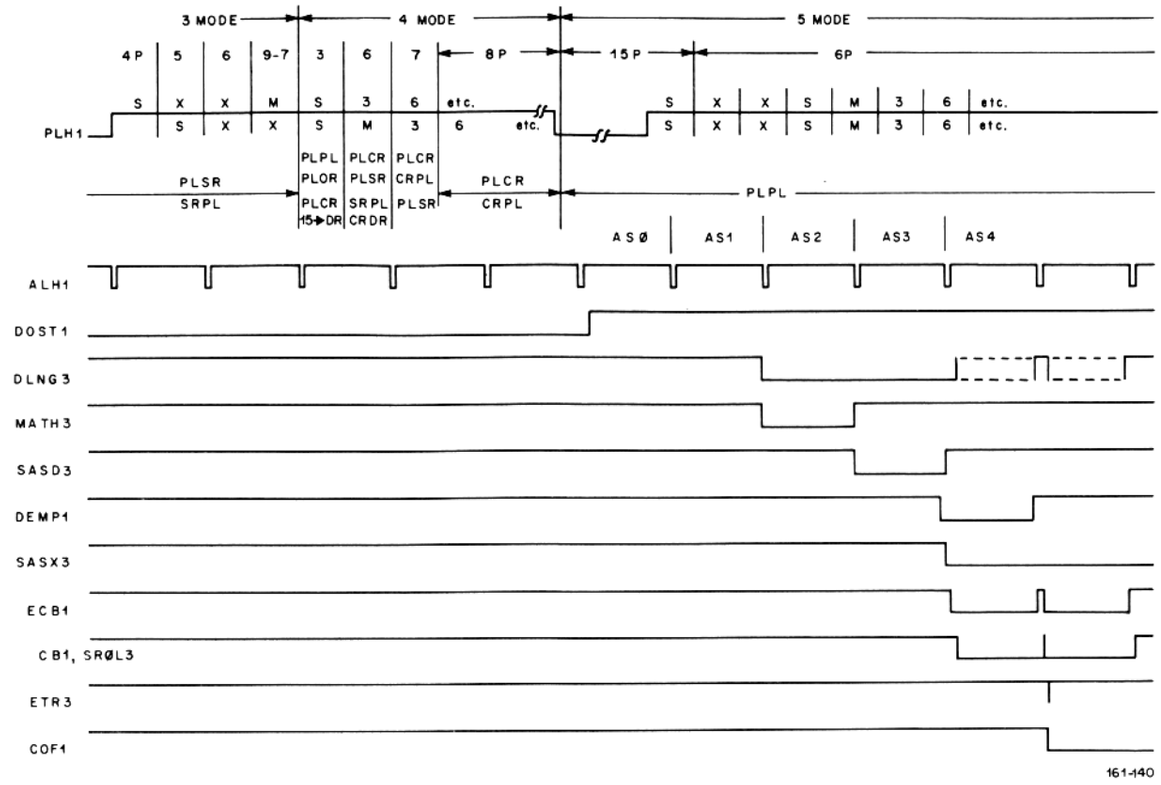
DEVELOPMENT OF SASS4 IN ACTION STEP 4

FIGURE 5-29

d. ACTION BEFORE STORAGE LINE HOME. Before any data can be stored, it must be put into the data transfer register. Note that TWO arithmetic line passes are necessary to do this: the first to shift the digit out of A14, and the second to provide the ETR for the transfer. Note that an ETR is not necessary during the first shift left (as the first digit is shifted out of A14 into the common register), so ETR is blocked by DEMP being set. DEMP receives its usual reset at A15R2 (see Figure 5-28), so does not inhibit ETR on the second shift left, and no more data transfer subroutines can occur until the first SLC after storage line home.

At this point the first digit to be stored is in the data transfer register and the next digit is in the common register. Compare this operation to a Recall, when the first shift left was unnecessary and the ETR was needed only to set COF. In this case, TWO shifts left are necessary before the action can be stopped by COF.

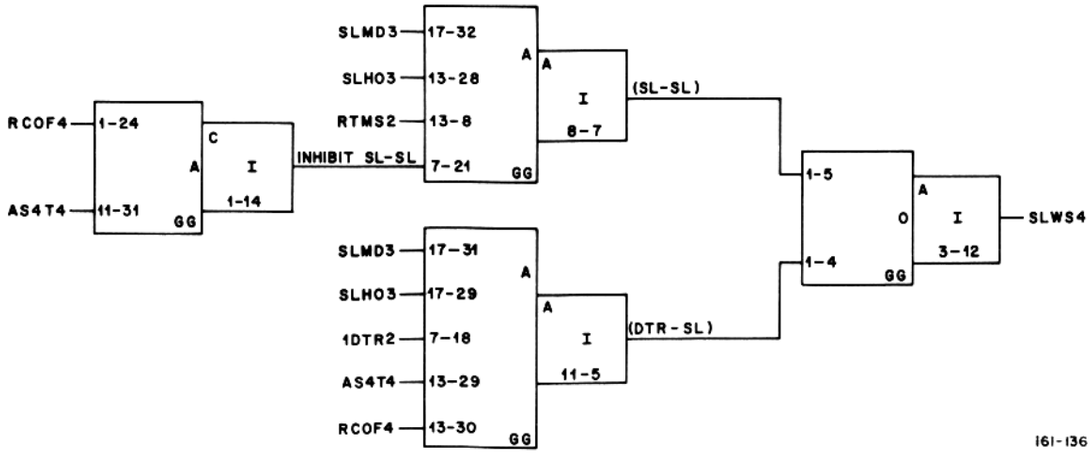
The timing chart in Figure 5-30 summarizes the operation to this point.



TIMING CHART
STORE, PRELIMINARY OPERATIONS

FIGURE 5-30

e. THE STORE PROCESS. At storage line home DPS sets, enabling the reset of COF. This occurs when RCOF is developed at storage location correct. Refer to Figure 5-31 and note that RCOF in action step 4 blocks the SLSL line routing and enables the DTR-SL routing. RCOF also enables the development of pulses on DTSP4, the same as during Recall, so that data can be shifted out of the data transfer register onto the storage line.



DTR-SL LINE ROUTING

FIGURE 5-31

Remember that at this time the normal Recall procedure also causes data to be shifted from the storage line into the data transfer register. This data is shifted into the common register during the next ETR and then shifted into R0 by the subsequent SR0L. After 14 such transfers, the number originally in storage has been completely transferred to R0. In this way a recall function actually occurs during a Store, but the number is left in R0 at the end of the instruction and then cleared out.

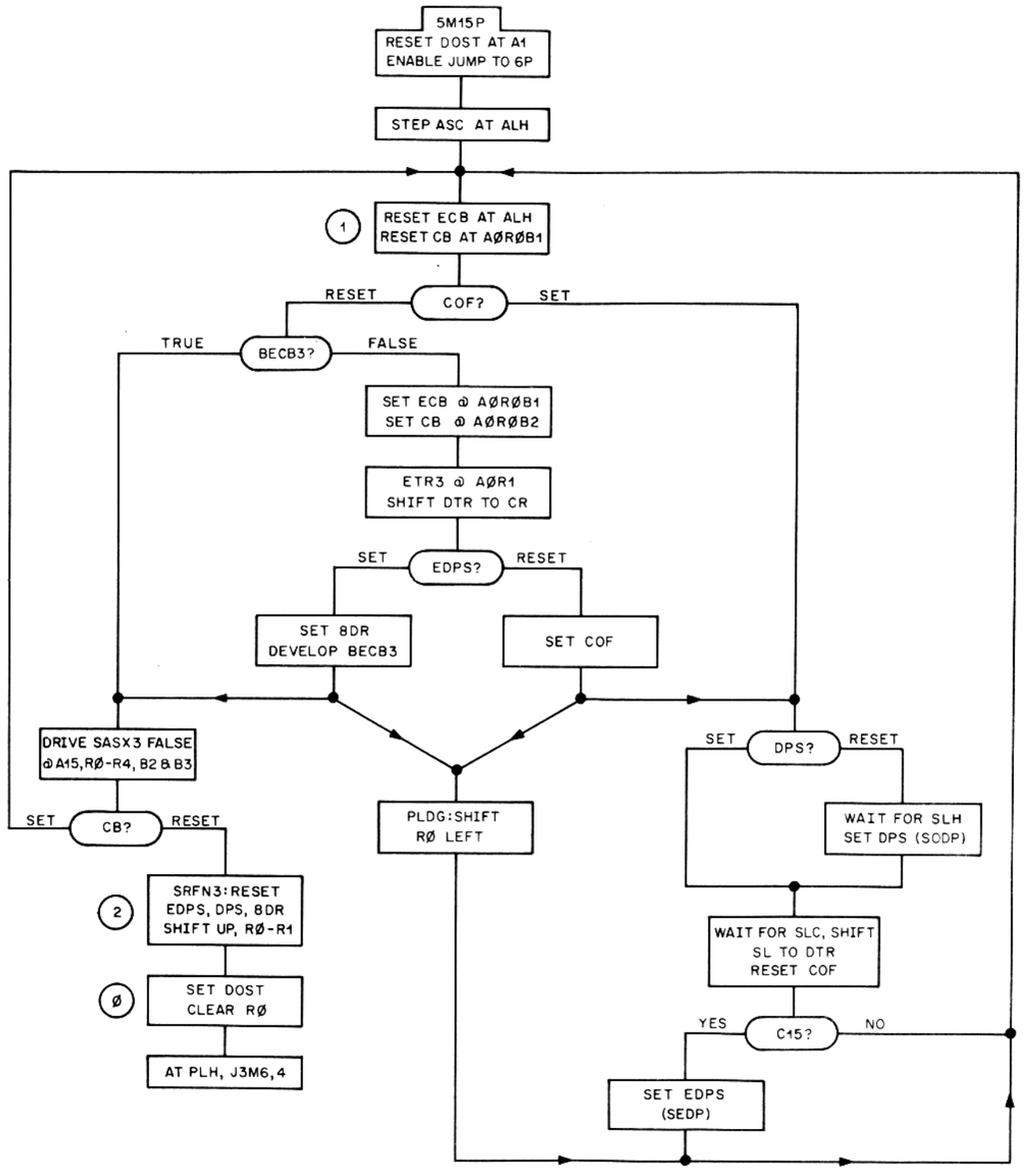
The data transfer subroutine occurs once for each storage column to shift the next digit out of A14R0. After the last SLC (in C15), the action step is terminated as explained earlier, and the counter steps to 5. Action step 5 is another "dead" step during which the arithmetic line routing is ALAL. After this the counter returns to 0 and the program line logic takes over.

5. ACCUMULATE

Accumulate is simply a combination of Recall and Store. The Accumulate command produces the same master signal as does the Store command, so the store portion of Accumulate is identical to a Store instruction. Accumulate also develops RFS, which provides SASS in action step 1 to perform a recall function. The number recalled is algebraically added to R1 in action step 2. Since this cycle leaves the answer in R1, the shift down is still required in action step 3. The answer is then stored during action step 4, and step 5 is again a "dead" step. Note that the same storage location is used in the recall and store portions because the same address digits remain in the decimal and search registers.

6. FLOW CHARTS

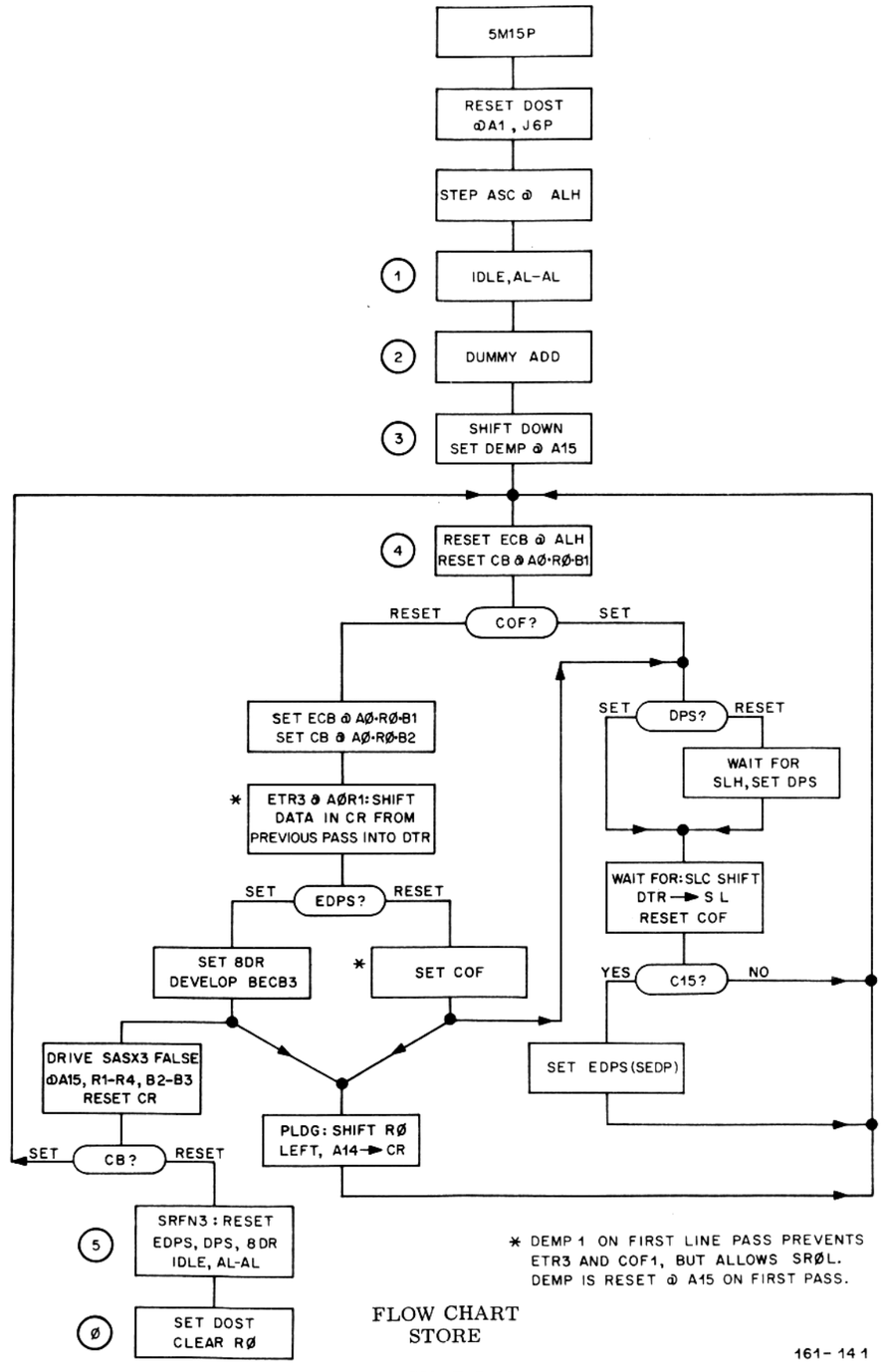
The flow charts in Figures 5-32 and 5-33 depict the Recall and Store instructions. They may also be used to study the recall and store portions of the Accumulate instruction.



FLOW CHART
RECALL

FIGURE 5-32

TP-161SL



FLOW CHART
STORE

161-141

FIGURE 5-33

TABLE OF CONTENTS

	Page
SECTION 6, FLEXOWRITER COMMANDS	
"BUSY CIRCUIT"	6-1
2205	6-1
Noise Filter and 5 ms Delay.	6-2
Second Filter.	6-2
Logic	6-2
OUTPUT SUBROUTINE	6-4
Basic Operation	6-4
Development of ECTR3	6-5
Arithmetic Line	6-7
PRINT SIGN	6-7
General Operation	6-7
Instruction Retrieval.	6-7
5 Mode Operation	6-7
FLEXOWRITER FUNCTIONS	6-8
General Operation	6-8
Instruction Retrieval.	6-8
Output Subroutine	6-9
Variations	6-9
TYPE INSTRUCTION	6-9
General Operation	6-10
Instruction Retrieval.	6-10
Execution.	6-10
SPECIAL FLEXOWRITER FUNCTIONS.	6-10
General Operation	6-10
Instruction Retrieval.	6-11
6 Mode Logic.	6-11
HALT	6-13
General Operation	6-13
Instruction Retrieval.	6-13
"Halt" Condition	6-14
Terminating Halt.	6-15

SECTION 6, FLEXOWRITER COMMANDS

There are basically five types of Flexowriter-oriented commands. Three of these, Print Sign, Type, and Halt, are individual instructions. The others are groups of instructions, known as "Flexowriter Functions" and "Special Flexowriter Functions". The first group consists of the Tab, Carriage Return, Backspace, Upper Case, Lower Case, and similar codes. The second group includes the eight "F" codes that control punches, readers, and the non print function. These are sometimes referred to as the "direct pick" Flexowriter Functions.

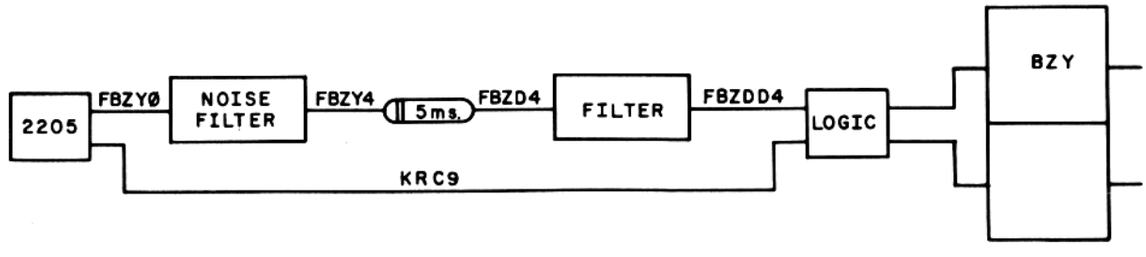
The Enter, Decimal Print, and Non Decimal Print instructions are covered in Section 4, Parts G and H, because they operate in conjunction with the arithmetic line. The instructions discussed in this section operate independently of the arithmetic line: each of them develops a master signal which prevents DOST from resetting once 5 mode is reached. The signals which block the reset of DOST are OFF3 (Flexowriter Functions and Special Flexowriter Functions), OP3 (Print Sign), OTO3 (Type), and OQ3 (Halt).

The "busy" circuit is covered first in this section, since it affects most of the other instructions. The Print Sign and Type instructions are discussed together with the Flexowriter Functions because the basic 5 mode operation is the same in each case.

A. "BUSY CIRCUIT"

Most of the logic in the "Flexowriter busy" circuit is associated with the BZY flip-flop, and sometimes the entire operation is referred to simply as "BZY". The logic operation involved is important during all input and output instructions. Its basic purpose is to delay the performance of any input or output until a previous input or output is complete.

A block diagram of the busy circuit is shown in Figure 6-1. The main "busy" signal produced by the 2205 is FBZY0, which is true whenever the 2205 is NOT busy. This signal is filtered, delayed, and filtered again before it finally enters into the logic. The KRC9 signal is also applied to the logic to indicate when the KRC relay is energized, a condition which requires modification of the busy circuit operation.

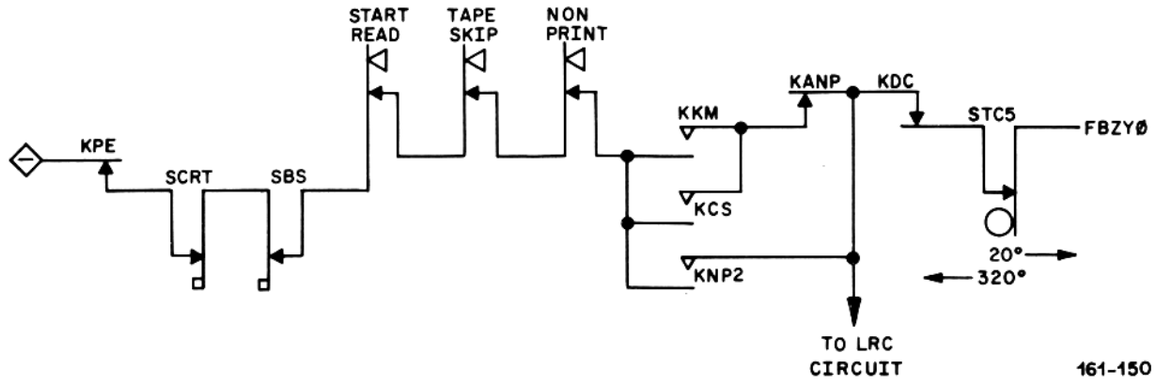


"BUSY" BLOCK DIAGRAM

FIGURE 6-1

1. 2205

The 2205 produces a true signal whenever it is NOT busy, so this signal is called FBZY0. It is developed as shown in Figure 6-2. Note that any of the standard 2205 delay circuits, such as SCRT, SBS, KPE, etc., will open the busy circuit. Note also that the circuit is opened by STC5 on each translator cycle. This latter set of contacts opens the circuit on ALL outputs other than Special Flexowriter Functions, and the other contacts perform when their respective functions are executed. Last, note that a set of normally open KKM contacts keeps the 2205 "busy" if its power switch is "off" and KKM is not energized.



DEVELOPMENT OF FBZY0 IN 2205

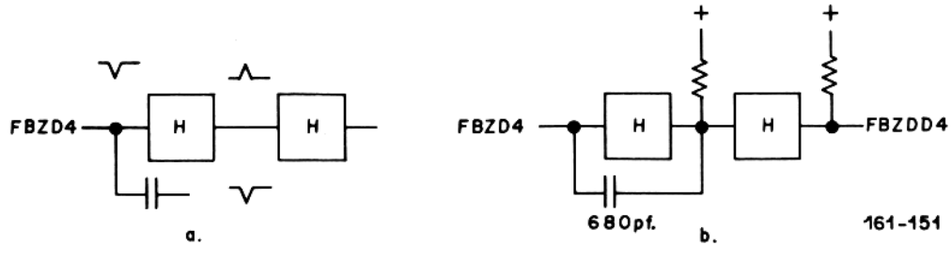
FIGURE 6-2

2. NOISE FILTER AND 5 MS DELAY

The first noise filter uses the same circuit employed by the input drivers, and the 5 ms delay circuit is the same as that used in the Flexowriter Common Bit (FCB3) circuit. These two circuits are covered in Section 8, Electronic Circuits. Note at this time, however, that the 5 ms delay is produced only at the end of "busy", when the 2205 contacts reclose. Most of the time the logic is waiting for busy to end before going ahead, so this is the critical point in busy: the beginning is of secondary importance.

3. SECOND FILTER

The second filter is simply two H-inverters with a capacitor around the first. The actual circuit is shown in Figure 6-3 (b). Figure 6-3 (a) shows that a noise spike on the input will cause an equal but opposite spike on the inverter output. By coupling the input to the output capacitively, short changes (spikes) on the input cancel the reaction on the output. Thus the capacitor can be called a "spike eater". Note that this coupling also slows down the overall operating speed of the inverter, which in this case is relatively unimportant.

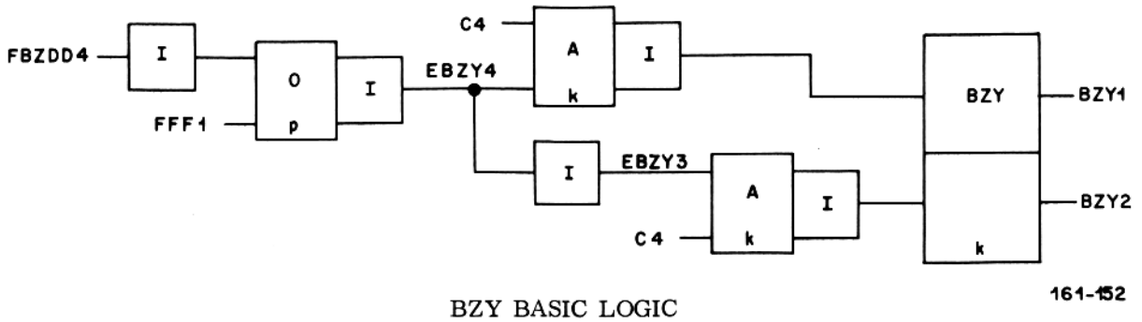


SECOND FILTER IN BUSY CIRCUIT

FIGURE 6-3

4. LOGIC

The basic logic associated with the BZY flip-flop is shown in Figure 6-4. Note that whenever the 2205 is busy, EBZY4 will be false and BZY will set at the next "C" time (of the program line timing chain). When the 2205 is no longer busy, EBZY3 becomes false and BZY resets at the next "C" time. The setting and resetting of BZY is triggered by C4 so that BZY will not change state just as the phase counter is becoming enabled, thus causing a partial jump or step.



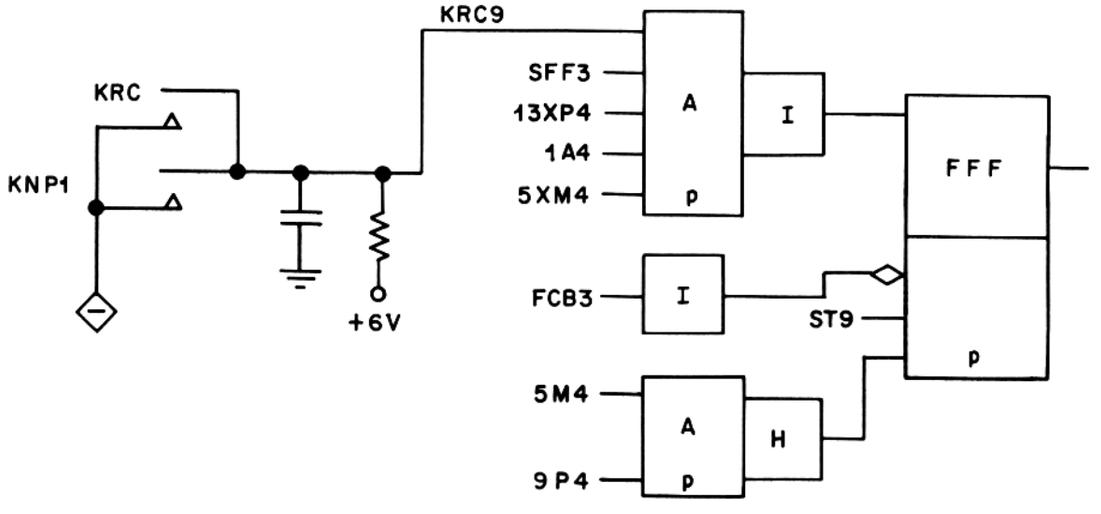
BZY BASIC LOGIC

FIGURE 6-4

In resetting BZY, note that the OR gate in Figure 6-4 actually becomes an AND gate, requiring FFF1 to be false. If the Flexowriter Flip-Flop, FFF, is set, it prevents the reset of BZY. FFF is needed in two cases, both occurring only when the KRC relay in the 2205 is not energized. The first occurs when the Carriage Return, Tab, or Backspace output is performed, and the second when an output is followed immediately by an input or when an output is the last instruction in a sequence.

a. "HOLE IN BUSY". At first glance it may seem unusual that KRC should be of concern during an output instruction. Note, however, that the operation of the STC5 contacts and the SCRT contacts does not overlap: at 320 degrees of the translator cycle, the STC5 contacts remake, ending the busy condition. A short time later the SCRT contacts open, reestablishing the busy condition. This time between the reclosing of the STC5 contacts and the opening of the SCRT contacts is referred to as a "hole in busy".

When the KRC relay is energized, it allows the KDC relay to operate, thus filling this "hole". When KRC is not energized, KDC cannot operate, so FFF is used to fill the "hole". With KRC down, FFF sets on every output (in 5 mode 13 phase). See Figure 6-5. FFF resets when the selector contacts restore and FCB4 goes true. By this time the SCRT contacts are open (during TAB, CRE, and BAC), maintaining the busy condition until the end of the function.



FFF LOGIC

FIGURE 6-5

Note that FFF is also reset each time 5 mode 9 phase is reached. This is to speed up the execution of the Decimal Print and Non Decimal Print instructions, which will never output a tab, carriage return, or backspace.

b. INPUT FOLLOWING OUTPUT. When an output immediately precedes an input, the selector slide corresponding to the output is operated (and its associated contacts are transferred) when the input begins. When KRC is energized, it opens the 2205 negative supply to the selector contacts that feed the input drivers, thus preventing any erroneous input. When KRC is not energized, FFF delays the beginning of the input instruction until the selector contacts have restored.

Note that this latter function of FFF is important not only in the case of an Enter instruction following an output, but also when the output is the last instruction in a given sequence. In this case a jump to 3 mode 0 decimal follows and the processor is ready for a sequence entry to be made. Without FFF the last output would become the sequence entry.

Last, note that in this discussion "output" means any output instruction other than a Special Flexowriter Function. This includes Flexowriter Functions, Type instructions, Decimal Print and Non Decimal Print instructions, and the Print Sign instruction.

c. USES. The following list itemizes the places in the logic where BZY is used:

4M15P	(Enter, ONOM, RST)
4M8P	Only if Op=Print
6M15P	Special Flexowriter Functions
5M15P	TYP, PSI, Flexowriter Functions
5M9P	Print
5M12P	All outputs other than SFF

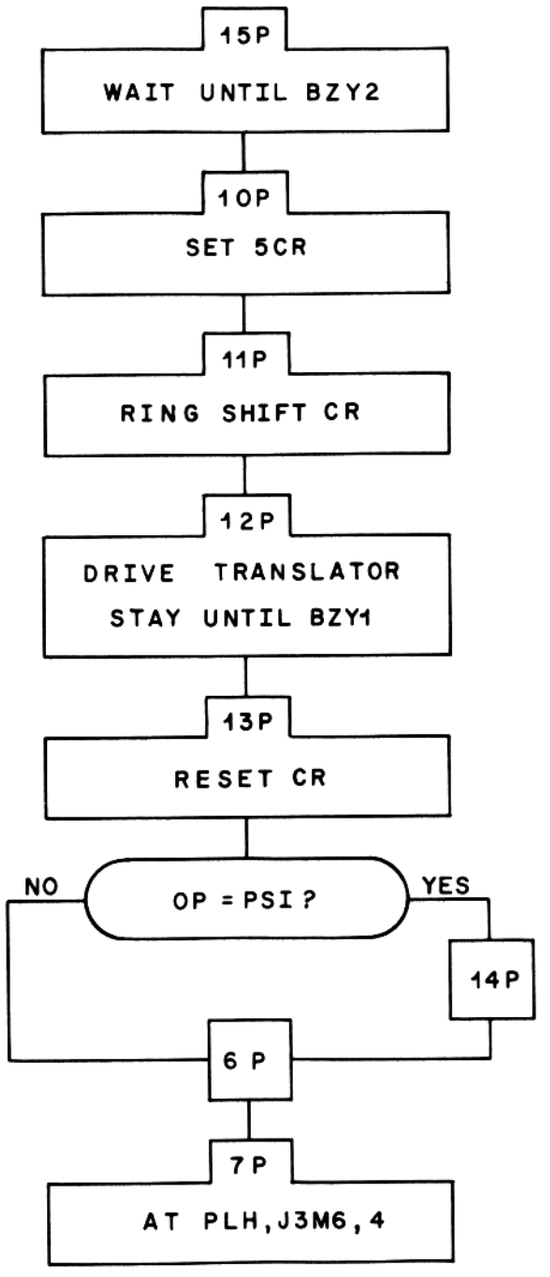
B. OUTPUT SUBROUTINE

There are certain 5 mode operations that are common to all output instructions except the Special Flexowriter Functions. These operations are included in the "output subroutine", which utilizes phases 10, 11, 12, 13, and sometimes 14. This subroutine utilizes much of the logic used in the "repetitive operations" (covered earlier in Section 4H, Print) but it occurs only once for each instruction.

1. BASIC OPERATION (Refer to flow chart in Figure 6-6)

During Print Sign, Type, and Flexowriter Functions, this subroutine begins in 5 mode 15 phase, after instruction retrieval. During Print Sign, the sign of R1 is set into the common register during 15 phase: during Type and Flexowriter Functions, the code to be sent to the 2205 is already in the common register as a result of the retrieval. In either case, the logic checks in 15 phase to determine the state of BZY. If BZY is still set from some previous operation, the phase counter waits. When BZY is reset a jump to 10 phase occurs.

In 10 phase the 5CR flip-flop becomes set. In 11 phase the common register is ring-shifted, and in 12 phase the translator is driven. The common register resets in 13 phase. This same procedure is followed during Print Sign, during execution of the Type instruction, and during all Flexowriter Functions except Carriage Return. The only difference in this latter case is that 5CR does not set in 10 phase.



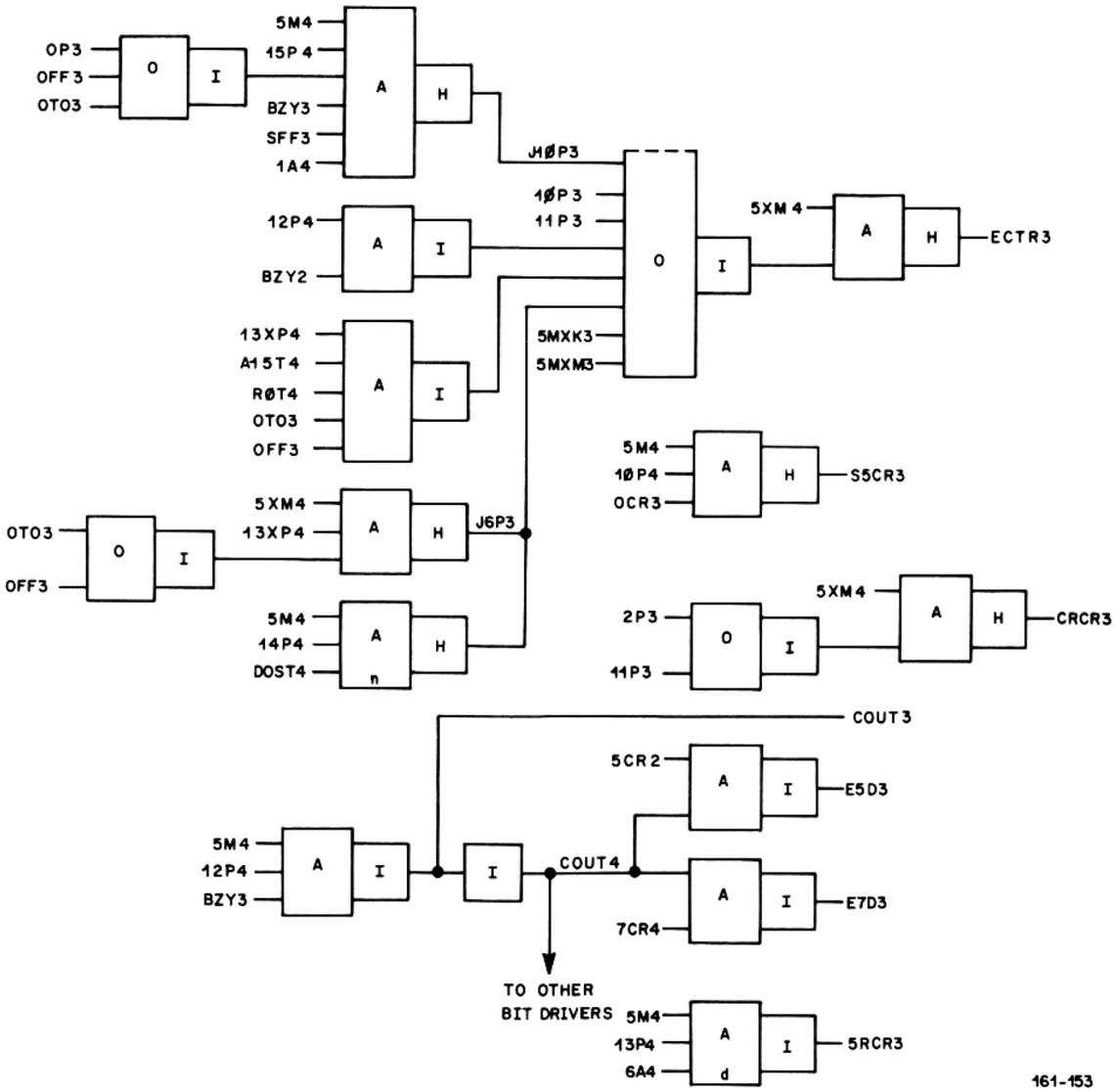
FLOW CHART
OUTPUT SUBROUTINE

FIGURE 6-6

2. DEVELOPMENT OF ECTR3

Figure 6-7 shows most of the logic involved in the 5 mode phases used in Print Sign, Type, and the Flexowriter Functions. OP3 is developed by Print Sign, OFF3 by the Flexowriter Functions,

and OTO3 by the Type instruction. Note that unlike Decimal Print and Non Decimal Print, these instructions do not enter 5 mode in 9 phase, but in 15 phase. When the 2205 is not busy, a jump to 10 phase occurs. ECTR3 is automatic in phases 10 and 11, but does not occur in 12 phase until the translator begins to rotate, causing "busy" to become true and driving BZY2 false. In 13 phase the operation changes slightly: Print Sign simply provides ECTR3 for a step to 14 phase, and the Type instruction and the Flexowriter Functions cause a jump to 6 phase. In 14 phase, Print Sign causes a jump to 6 phase because DOST was never set, so the operation is essentially the same. (See Figure 6-6). In 6 phase 5MXK3 provides ECTR3 for the step to 7 phase, and in 7 phase 5MXM3 develops it at program line home.



OUTPUT SUBROUTINE LOGIC
(All on "P" card except as noted)

FIGURE 6-7

3. ARITHMETIC LINE

Although not illustrated here, note that OP3, OFF3, and OTO3 all block the reset of DOST, thus inhibiting all arithmetic line action.

C. PRINT SIGN

Performance of the Print Sign instruction causes the sign of R1 to be printed and/or punched into tape. If R1 is negative, a dash (minus) is printed, and if R1 is positive a space is executed. In order to print the minus sign, R1 must be negative AND contain data: if R1 is clear, its sign is always considered positive.

If the 2205 is in non print, the space or dash code will be punched into tape without printing, provided the punch is "on".

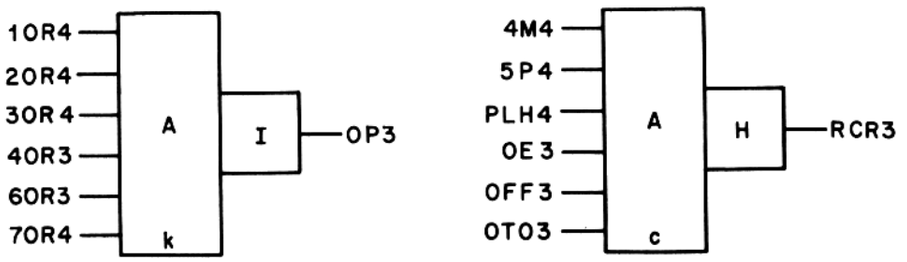
1. GENERAL OPERATION

The Print Sign command (letter P, 1-2-3-5-7 code) is retrieved in 4 mode and at program line home a jump to 5 mode 15 phase occurs. In 15 phase two operations MAY occur. First, the sign and contents of R1 are checked to determine the sign to be printed: if negative, the 7CR flip-flop is set. Second, if the 2205 is still busy from a previous instruction (or sequence entry), the phase counter is blocked.

When the 2205 is not busy, the output subroutine resumes with a jump to 10 phase. The code is ring-shifted in 11 phase and sent to the translator in 12 phase. The subroutine continues through phases 13, 14, 6, and 7, and then concludes with a jump to 3M6,4 at the end of 7 phase.

2. INSTRUCTION RETRIEVAL (Figure 6-8)

The Print Sign command is shifted into the operation register in 4 mode 3 phase and immediately develops the O0AD3 and OP3 signals. The fact that a 0-address code is retrieved causes a jump to 5 phase where the balance of the program is recirculated PLSR, SRPL. Since all commands are shifted into the common register in 3 phase, the Print Sign command must be cleared out before the sign itself can be set in. This occurs at program line home, which also enables a jump to 5 mode 15 phase. Note that this reset is blocked when the operation register contains a Type command or a Flexowriter Function.

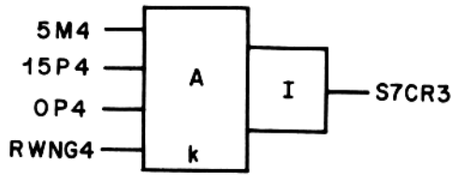


PRINT SIGN SIGNALS IN 4 MODE

FIGURE 6-8

3. 5 MODE OPERATION

The operation in 5 mode 15 phase occurs independently of the busy condition, which governs only the jump to 10 phase. In 15 phase the 7CR flip-flop is set if R1 contains a sign bit and some other data. The same signal that determines the status of a Positive or Negative Branch is used to develop S7CR3, namely RWNG. See Figure 6-9. (The development of RWNG is covered in Section 4L, Divide, Part 7.) If R1 does not contain a sign bit, or if R1 is clear, S7CR3 is not developed, and the common register remains clear until 10 phase.



DEVELOPMENT OF S7CR3

FIGURE 6-9

In either case, when the 2205 is not busy, a jump to 10 phase occurs. This was shown previously in Figure 6-7.

Once the phase counter reaches 10 phase, the operation is the same as during a Print instruction: in 10 phase 5CR sets, in 11 phase the common register is ring-shifted, in 12 phase the translator is driven, and in 13 phase the common register is reset. If 7CR was not set, however, 5CR remains set at the end of the ring shift. Thus when the translator is driven in 12 phase, either the 5 bit or the 7 bit is driven along with the clutch.

D. FLEXOWRITER FUNCTIONS

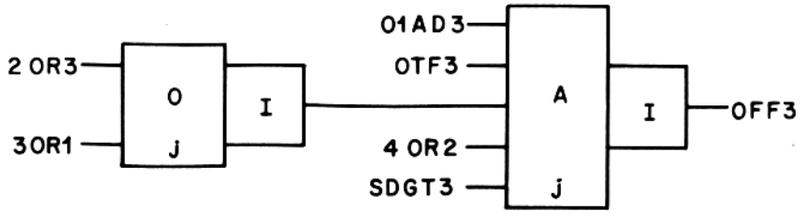
The Flexowriter Functions are a group of commonly-used writing machine functions which the 5610 can execute without requiring a Type command preceding. The more common of these are included in the SWIFT translator: Carriage Return, Tab, Upper Case, Lower Case, and Backspace. All of the 2205 function keys ("F" keys) represent Flexowriter Functions, but F1 through F6, F11, and F13 are also "Special Flexowriter Functions" (which will be discussed separately). A few others (percent, comma, and degree) are special cases, as is Carriage Return; these will be discussed separately.

1. GENERAL OPERATION

The overall operation of the Flexowriter Functions is very similar to that of the Print Sign instruction. The only real difference lies in the fact that the common register is not reset at the end of 4 mode and that 7CR is not affected by the sign of R1 during 5 mode 15 phase. All other operations are virtually identical.

2. INSTRUCTION RETRIEVAL

Each Flexowriter Function that is retrieved in 4 mode 3 phase immediately develops OFF3 (Figure 6-10), which allows the output subroutine to take place. Also, the PLCR line routing in 4 mode 3 phase becomes valuable for the second time (the first time was during Constants). Thus, as the command is shifted into the operation register, the address (the same code) is shifted into the common register. The balance of the retrieval is the same as for any other 0-address code, and at program line home a jump to 5 mode 15 phase occurs.



DEVELOPMENT OF OFF3

FIGURE 6-10

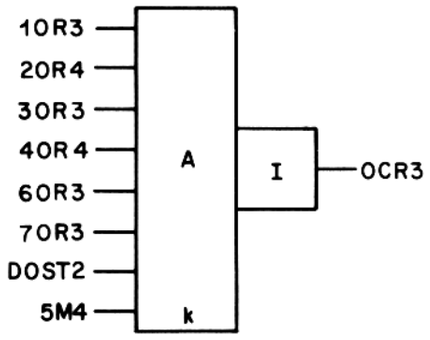
3. OUTPUT SUBROUTINE

The output subroutine takes place normally in phases 15, 10, 11, 12, 13, 6, and 7, in that order. 14 phase is not used (refer to Figure 6-7). The code in the common register at the beginning of the output subroutine (the same code that is retrieved in 4 mode) is sent to the translator in 12 phase.

4. VARIATIONS

The one allowable variation to the foregoing procedure is the Carriage Return instruction. There are many other variations which are not normally allowed.

a. CARRIAGE RETURN. The Carriage Return code retrieved from the program line (2-4-5) must be converted to a lone 8 bit before being sent to the 2205. This is accomplished by decoding the 2-4-5 after retrieval into the Operation Carriage Return signal, OCR3, as shown in Figure 6-11. In 5 mode OCR3 first resets 2CR and 4CR (which were set by the 2-4-5 code during retrieval), and sets 8CR. Then, in 10 phase, it blocks the development of S5CR3, so when the ring shift occurs in 11 phase, 8CR is the only flip-flop set and 5CR cannot be toggled. Note that during the course of this instruction, the 2-4 code remains in the operation register while the 8 bit is set into the common register.



DEVELOPMENT OF OCR3

FIGURE 6-11

b. OTHER VARIATIONS. Codes such as Comma, Decimal, Percent, Degree, M, U, and Y are not a part of the SWIFT translator because they can cause erroneous operation under certain conditions. For this reason, these codes should never be used except as an address of a Type command.

E. TYPE INSTRUCTION

The Type instruction can be used to perform any 2205 function or print any character: it causes its single address character to be sent to the 2205 for printing or for performing whatever function the 2205 is programmed to do with that code. If the 2205 is in non print, the code will be punched in tape.

The Type instruction can be used to perform any of the Flexowriter Functions, and, through the 2205 patchboard, any of the Special Flexowriter Functions. It requires two characters on the program line, however, while the other functions require only one. The main use of the Type instruction, then, is to type characters that are not normally available through other instructions.

1. GENERAL OPERATION

The only difference between the Type instruction and the Flexowriter Functions is in instruction retrieval. When the Type command (1/2 symbol, 1-3-4-6-7 code) is retrieved, it is decoded as a 1-address command, and the next program character is recognized as its address and shifted into the common register. Then, in 5 mode, the standard output subroutine takes this code in the common register and sends it to the 2205.

2. INSTRUCTION RETRIEVAL

When the Type command is retrieved in 4 mode 3 phase, O1AD3 and OTO3 are developed. O1AD3 causes a step to 4 phase where the address digit is shifted into the common register. This is the normal retrieval path for 1-address codes: next the step to 5 phase occurs, and at program line home a jump to 5 mode 15 phase takes place.

3. EXECUTION

All 5 mode operations are exactly the same as described under Flexowriter Functions. OTO3 blocks the reset of DOST, preventing any arithmetic line operation.

F. SPECIAL FLEXOWRITER FUNCTIONS

Eight of the function key codes on the 2205 are used for the Special Flexowriter Functions. These instructions control the punches and readers on the 2205 and auxiliary equipment and the non print function in the 2205. The following chart lists each of the Special Flexowriter Functions and the more pertinent information pertaining to each:

COMMAND	MNEMONIC	2205 KEY	CODE	KEY SIGNAL	FUNCTIONS
Readers Off	ROF	F1	1-2-4	OSTP3	De-energize KRC (2205 & Aux)
Non Print	NPR	F2	3-4-5	ONP3	Energize KANP (2205)
Print Restore	PRE	F3	3-4-6	OPR3	De-energize KANP (2205)
Punch 1 On	P1O	F4	3-4-7	O1P3	Energize KPS (2205)
Punch 2 On	P2O	F5	2-3-4-6-7	O2P3	Energize KPS (Aux)
Punches Off	POF	F6	1-2-3-4-6	OPO3	De-energize KPS (2205 & Aux)
Reader 1 On	R1O	F11	1-2-3-4-7	ONR3	Energize KRC (2205)
Reader 2 On	R2O	F13	1-3-4-5-6	O2R3	Energize KRC (Aux)

In addition to the information given in the chart, note that either the Reader 1 On or the Reader 2 On instructions will turn off the opposite reader while turning on the reader specified.

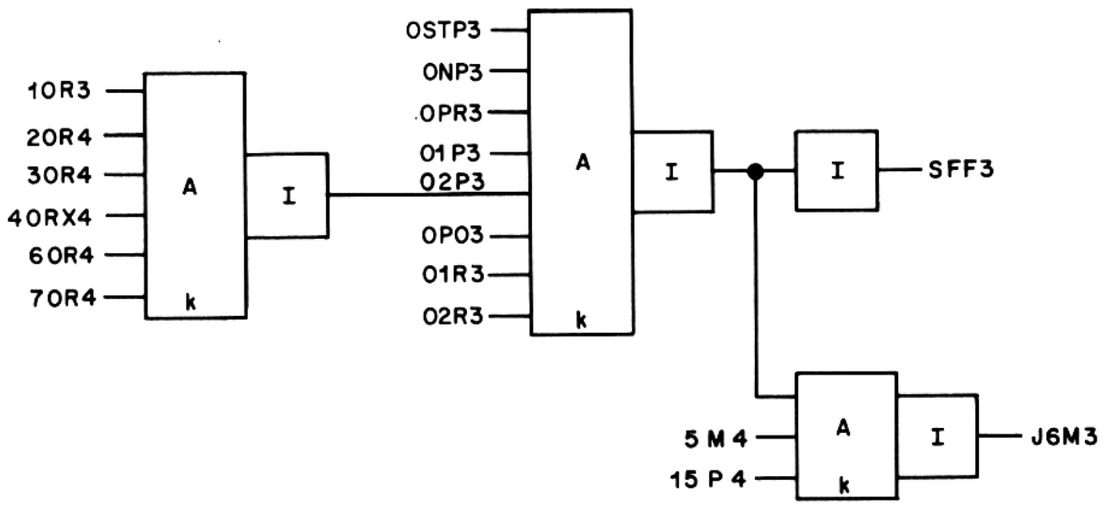
All eight of these instructions are performed without going through the 2205 translator: each has its own "driver" circuit which energizes or de-energizes the specified relay directly, independently of the 2205 program panel patching. For this reason, these instructions are sometimes referred to as "direct pick" Flexowriter Functions. Because these instructions affect the relays directly, and do not go through the translator, they are all non-reproducing. For this same reason, they always perform their named function, regardless of the 2205 program panel patching.

1. GENERAL OPERATION

After retrieval in 4 mode, the Special Flexowriter Functions cause a jump first to 5 mode and then to 6 mode. In 6 mode an SCR is turned on, corresponding to the particular instruction, for up to 50 ms, long enough to ensure a reliable operation of the affected relay. This time is determined by a cycle through phases 1-8 in 6 mode, the stepping being gated by program line home. This makes the time relatively constant from one instruction to the next. From 6 mode the logic jumps back to 5 mode 13 phase, and the balance of the operation is the same as during the output subroutine.

2. INSTRUCTION RETRIEVAL

In 4 mode 3 phase the code representing the Special Flexowriter Function is shifted into the operation register and it immediately develops the corresponding "key signal" and the SFF3 signal, as shown in Figure 6-12. Note that OFF3 is also developed, which prevents any arithmetic operations from occurring. Because it is a 0-address code, the code also follows the normal 4 mode routine and provides a jump to 5 mode 15 phase at program line home. Here J6M3 is developed and the jump to 6 mode 15 phase occurs when the timing chain restarts.

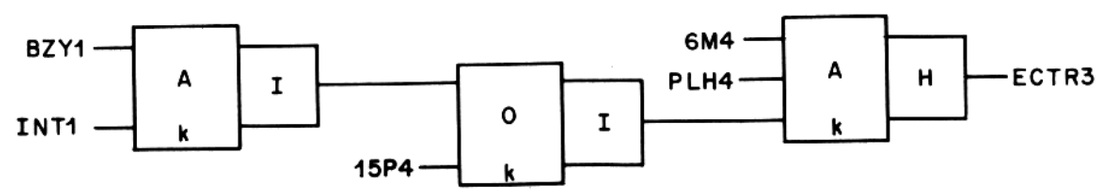


DEVELOPMENT OF J6M3

FIGURE 6-12

3. 6 MODE LOGIC

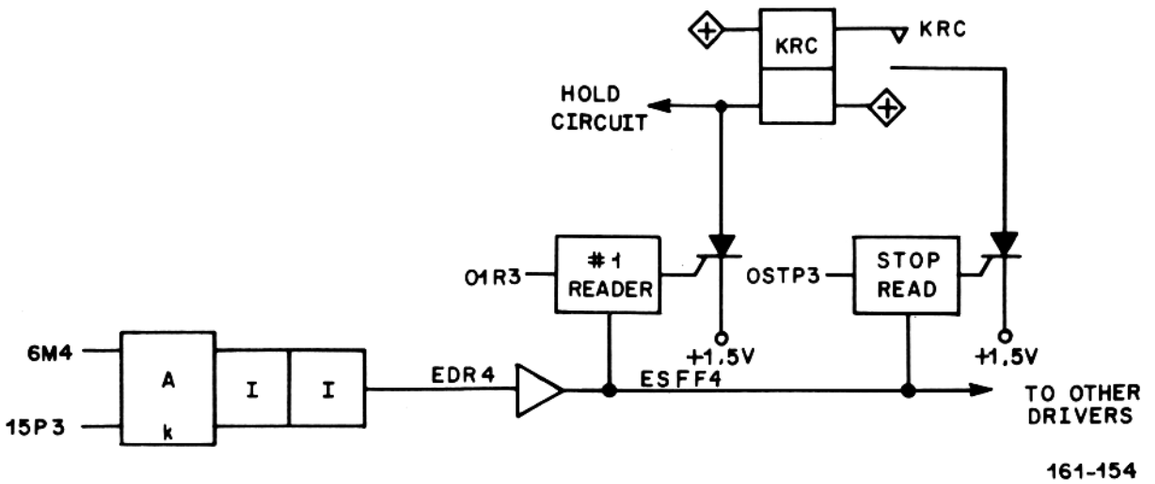
In 6 mode 15 phase a check is made to see if the 2205 is busy. When it is not busy, ECTR3 is developed at the next program line home to enable a step to 1 phase. See Figure 6-13. The qualification of INT1 is added to ensure that no output will occur if some selector slide is still operated (from a prior entry).



DEVELOPMENT OF ECTR3 IN 6 MODE

FIGURE 6-13

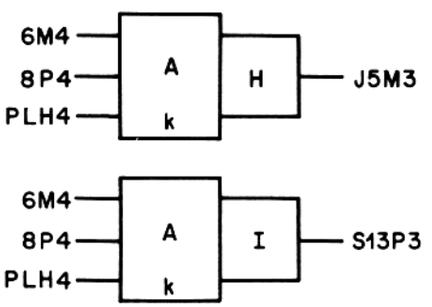
Once the phase counter leaves 15 phase, Enable DRivers, EDR4, is driven false. This signal is inverted through a special driver circuit to produce Enable Special Flexowriter Functions, ESFF4, which is applied to eight "control drivers", one for each of the functions. The other input to each of the control drivers is one of the eight "key" signals developed by the operation register. With both ESFF4 and one of the "key" signals true, the appropriate SCR is turned on, which either "picks" or "drops" one of the relays in the 2205. Figure 6-14 presents a simple block diagram of the preceding: the details of the driver circuits are covered in Section 8, Electronic Circuits.



BLOCK DIAGRAM
CONTROL DRIVERS

FIGURE 6-14

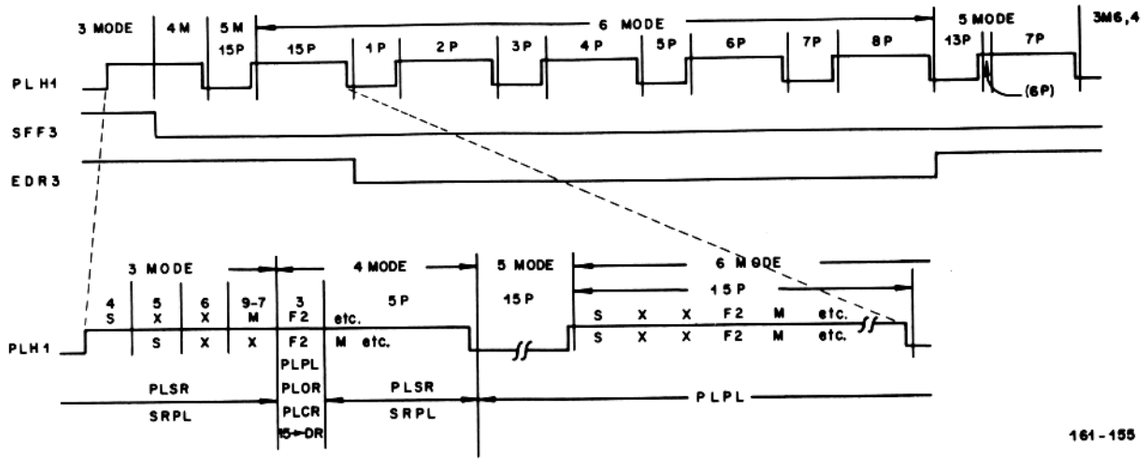
Each SCR is held "on" only as long as the mode register remains in 6 mode. This time is determined by the phase counter, which must step through each phase, 1 through 8, until a jump back to 5 mode is enabled.



LEAVING 6 MODE

FIGURE 6-15

As shown in Figure 6-13, ECTR3 is developed only during program line home. Note that the phase counter takes TWO steps for each program line pass: the first step occurs at T6B during program line home, and the second at the first T6B after the timing chain restarts. SPC sets for the first step at T4A when PLH sets. The next T6B steps the phase counter, and then T6C resets SPC. The timing chain stops at T1A, which also sets SPC for the second time. Then, when the timing chain restarts after the sync pulse is sensed, the first T6B steps the phase counter the second time, and T6C resets SPC. At the end of the program line the process repeats. Thus the SCR's are "on" for approximately 48 ms. When program line home is reached in 8 phase, a jump to 5 mode and 13 phase is enabled, and as soon as this jump occurs, 6M4 goes true and cuts off the SCR's. Note, however, that when relays are being "blown down", the SCR turns off when its circuit is opened by the relay dropping out. This entire routine is summarized in the timing chart in Figure 6-16.



TIMING CHART
SPECIAL FLEXOWRITER FUNCTIONS

FIGURE 6-16

G. HALT

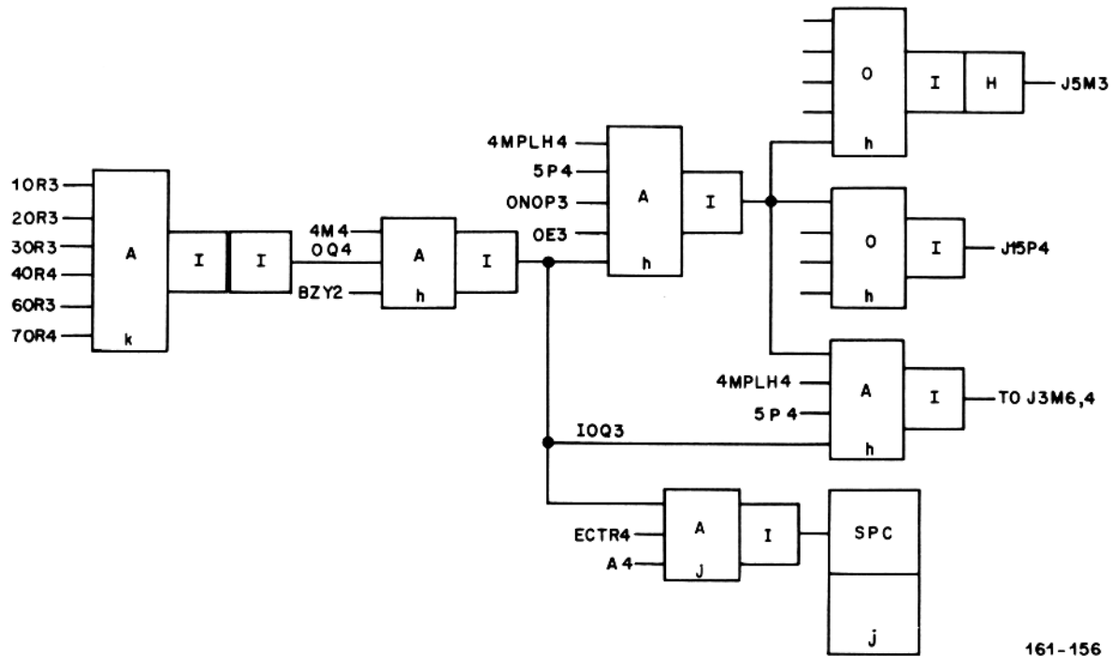
The Halt command, when executed, stops all processor operation and allows the 2205 to operate in a condition identical to "Type Only" until a "Terminate Halt" code is read in a reader or entered on the keyboard. Any code can be made to terminate the halt condition by programming it to do so in the 2205 patch panel: it is only necessary to provide ground (minus) to patch point R16 in the patch panel (for 12.5 ms or longer) to terminate halt. The means used to provide this potential to R16 depends upon the application being programmed.

1. GENERAL OPERATION

Instruction retrieval is slightly different during Halt: the logic remains in 4 mode 5 phase as long as the 2205 is "busy". Only after the busy condition is terminated does the logic go on to 5 mode 15 phase. The phase counter remains locked in 5 mode 15 phase until the Halt is terminated by a pulse on R16: then a jump to 6 phase and a step to 7 phase follow, and at the next program line home a jump to 3M6,4 occurs. While waiting for the terminating pulse in 5 mode 15 phase, the Halt command in the operation register develops RR3 to allow the reader to run, thus allowing the 2205 to run as if it were in "Type Only".

2. INSTRUCTION RETRIEVAL

As soon as the Halt command (letter Q, 4-5-7 code) is completely shifted into the operation register in 4 mode 3 phase, OQ3 is developed. If the 2205 is still "busy" at this time (from a prior input or output), Inhibit Halt, IOQ3, is immediately developed. IOQ3 has no effect in 4 mode 3 phase: note that SPC will become set BEFORE IOQ3 is developed, so that the phase counter remains in 5 phase until after BZY resets. The signal which normally provides the jump to 5 mode 15 phase also inhibits the jump to 3 mode 4 phase, and since this signal is blocked when BZY is set, IOQ3 must also be used to prevent the jump to 3 mode at this time. When BZY resets, IOQ3 returns to false and at the next program line home the jump to 5 mode 15 phase occurs. Of course, this jump occurs at the first program line home if the 2205 is not busy. See Figure 6-17. This "busy" check in 4 mode is needed to prevent erroneous operation of some auxiliary input equipment.

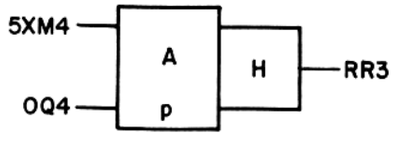


HALT RETRIEVAL LOGIC

FIGURE 6-17

3. "HALT" CONDITION

In 5 mode 15 phase the system is said to be in the "Halt" condition. This condition is identical to the "Type Only" condition, for all practical purposes: the 2205 operates exactly the same as it does during Type Only, until the Halt is terminated. This is accomplished by developing RR3 (Figure 6-18) so that the reader may operate if desired. (Remember that developing RR3 does not start the reader: it only permits it to run if KRC is energized through one of its normal channels.) Since INQ is not developed, the 2205 will not enter any codes into the common register, and no other processor action will occur.

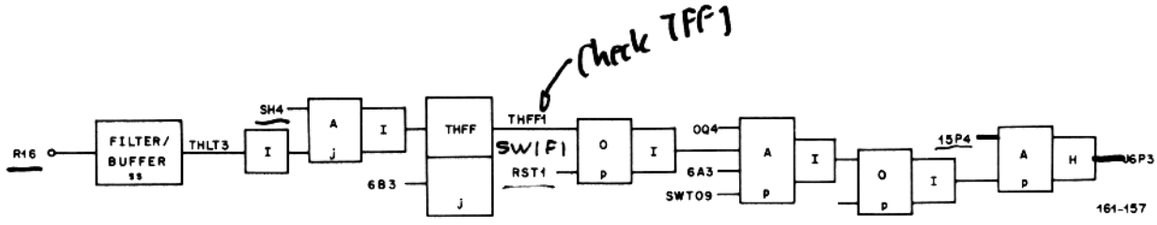


DEVELOPMENT OF RR3 DURING HALT

FIGURE 6-18

4. TERMINATING HALT

The termination process is illustrated in Figure 6-19. The "Filter/Buffer" is described in Section 8, Electronic Circuits. When R16 is grounded, Terminate HaLT, THLT3, becomes true, and at the next SH time the Terminate Halt Flip-Flop, THFF, sets. A minimum pulse width of 12.5 ms is needed to insure a SH time before the pulse on R16 goes away. This immediately provides J6P3 for the jump to 6 phase. The same T6B that provides the actual jump to 6 phase also resets THFF. After 6 phase is reached, the next T6B provides a step to 7 phase. At program line home a jump to 3M6,4 occurs to initiate the search for the next instruction.



TERMINATING HALT

FIGURE 6-19

Note in Figure 6-19 that RST1 can also provide the jump to 6 phase. This means that the halt condition can also be terminated by operation of the Reset switch, but in this case the entire instruction sequence will be terminated because the ensuing marker search results in the destruction of the marker. 6A3 is on the large AND gate to prevent "short pulsing" the phase counter if the RST flip-flop sets during "6A" time.

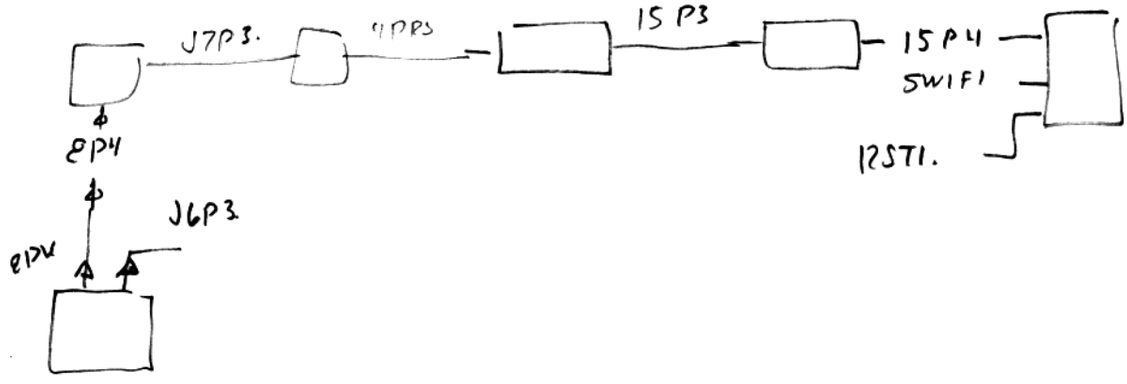


TABLE OF CONTENTS

	Page
SECTION 7, TRANSLATE	
GENERAL OPERATION	7-1
Software	7-1
Hardware	7-1
2 MODE OPERATION	7-4
Translate Switch	7-4
Basic Logic	7-4
Decimal Settings	7-6
Mode and Phase Jumps	7-6
Register Resets	7-8
Parity Error	7-9
3 MODE OPERATION	7-9
General Operation	7-10
Translate Ring Shift	7-10
Compare Logic	7-12
Unsuccessful Compare	7-13
Software Precautions	7-14
VARIATIONS IN "NORMAL" OPERATION DURING TRANSLATE	7-14
Conditional Branch	7-14
Enter	7-15

SECTION 7, TRANSLATE

The main purpose of the translate feature of the 5610 is to convert English-language (or some other language) statements into "machine language" codes which the 5610 can use later in performing a program. In some advanced programs, the 5610 operates in the translate mode, in which case there are minor variations in the execution of the Branch and the Enter instructions. These variations will be covered near the end of this section. For the most part, however, this section will deal with the translate feature as it is more commonly used: in conjunction with the SWIFT translator.

A. GENERAL OPERATION

As noted in Section 2, Programming, each 5610 instruction has its own unique "mnemonic". When this three-character mnemonic is entered on the 2205 during translate, a search of the program line occurs, which locates the "translate" sequence having a three-character address corresponding to the mnemonic. Once the sequence is located, the instructions contained within it are executed, the same as during "regular" operation. In other words, translation is concerned basically with locating a particular "translate" sequence, and once the sequence is found, normal operation resumes.

A translate sequence is different from a normal sequence in that it has three address characters instead of only one. To differentiate from a normal sequence, the translate sequence uses the letter "L" instead of the letter "H" as a sequence separator.

1. SOFTWARE

A typical translate sequence (from SWIFT) is as follows:

MNEMONIC FOR DECIMAL PRINT COMMAND	LDPR
HALT FOR DESCRIPTION	Q
PUNCH 1 ON FOR OBJECT CODE	F4
TYPE OBJECT CODE	½X
ENTER 2-DIGIT ADDRESS	/20
BRANCH FOR LINE AND OBJECT COUNT	EZ
END OF SEQUENCE	

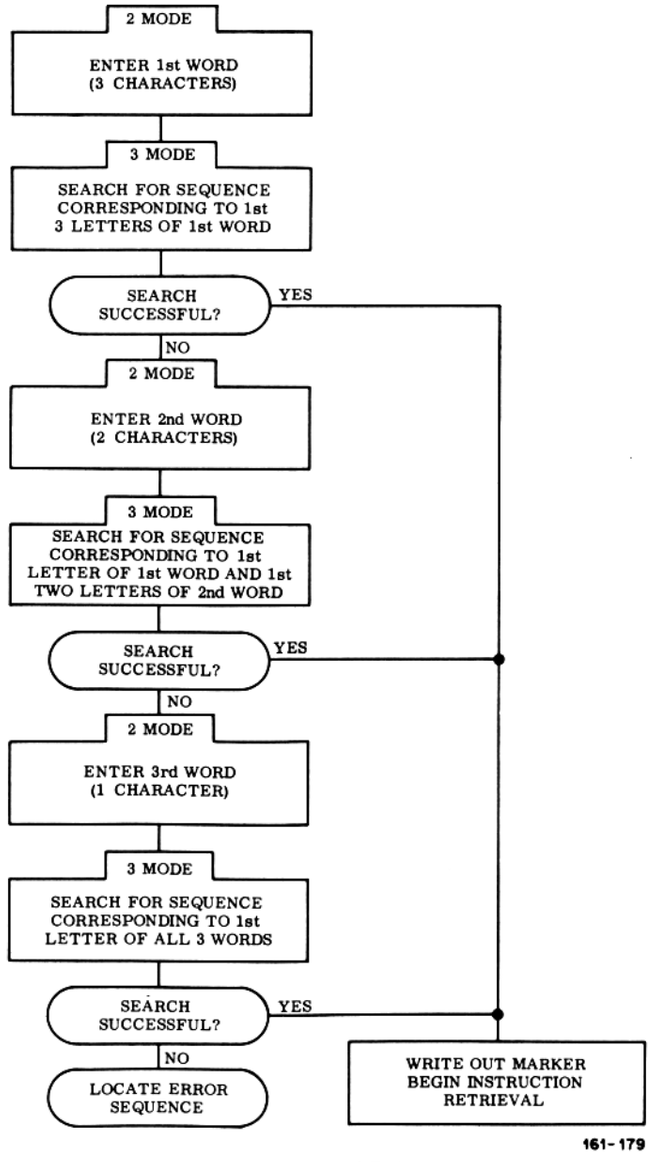
The sequence separator ("L") is followed by the mnemonic representing the Decimal PRint instruction, DPR. The first instruction in the sequence is a Halt to allow for typing of a description (e.g. "Decimal Print net total"). Next the no. 1 punch is turned on and the object code representing the Decimal Print command, the letter "X", is punched into the tape. An Enter 20 instruction follows so that the 2-digit address of the Decimal Print instruction may be entered directly into the object tape. The final Branch instruction allows a branch to a sequence which updates the line and object counts, turns off the punch, etc.

Each instruction in the SWIFT translator has a similar sequence. Some are, of course, more complex, but their basic operation is the same: the punch is turned on and the object code for the command is punched, followed by any necessary address. The source tape used for this translation must contain a code to terminate the Halt, and two digits for the instruction address. If the translation is being performed manually, with no source tape, these codes must be entered on the 2205 keyboard. Further details of the software requirements are covered in Section 2: the remainder of this section will be concerned only with the hardware aspects of translation.

2. HARDWARE

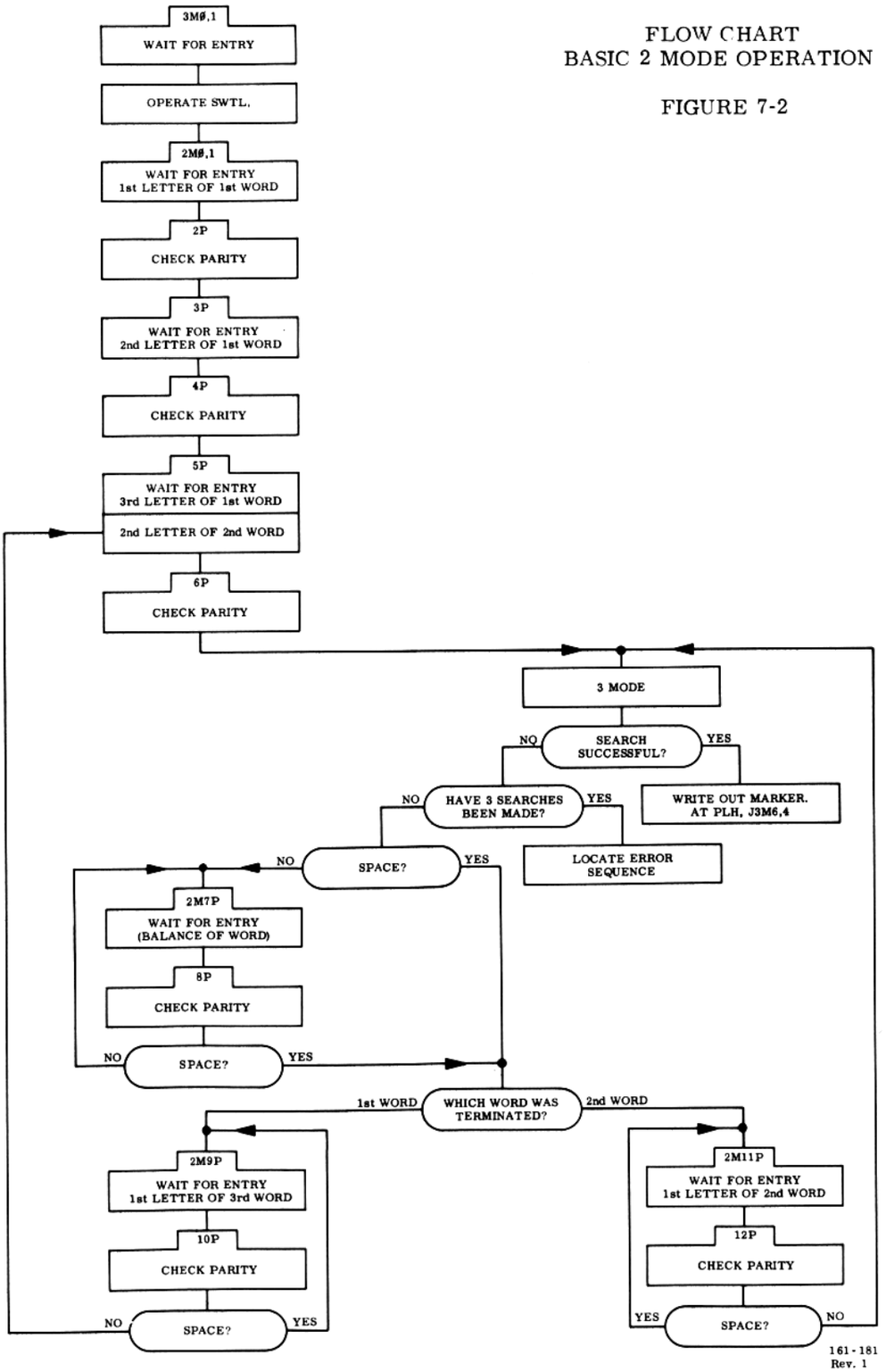
Although 2 mode is called the "translate mode", translation is not restricted to 2 mode: 2 mode and 3 mode work together to identify and locate the desired three-character sequence title, and once the sequence is located, operation is back to "normal", utilizing modes 3, 4, 5, and 6. Since the Translate switch is operated, though, termination of a sequence causes a jump to 2 mode (instead of 3 mode), where the next sequence title is entered.

Whereas a normal sequence title is entered in 3M0,1 and the sequence is located in the remainder of 3 mode, translate allows three title characters to be entered in 2 mode, and modifies the 3 mode search so that all three characters are checked. If the 3 mode search fails to locate the sequence corresponding to the three characters entered, a jump occurs back to 2 mode where a second entry is allowed. After the second entry, 3 mode takes over and searches for a sequence identified by the first letter of the first entry and the first two letters of the second entry. If such a sequence still cannot be located, another jump occurs back to 2 mode for a third entry. Then 3 mode searches for a sequence identified by the first letter of each of the three entries. If the correct sequence has not been found after three searches, the error sequence is selected instead. If the correct sequence is located at any time, a marker is written out and operation returns to normal, the only difference being the jump to 2 mode upon completion of the sequence. See Figure 7-1.



FLOW CHART
BASIC TRANSLATE OPERATION

FIGURE 7-1



FLOW CHART
BASIC 2 MODE OPERATION

FIGURE 7-2

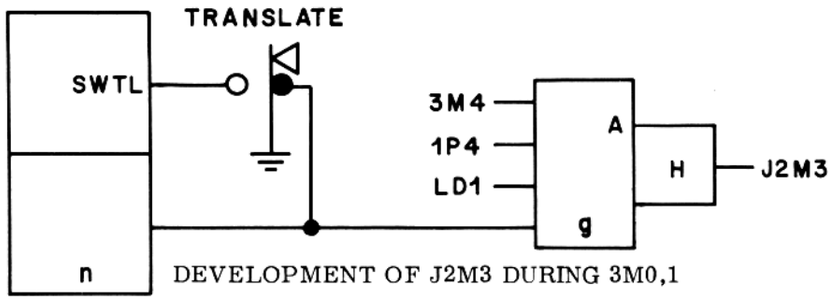
TP-161SL

B. 2 MODE OPERATION

The only function of 2 mode is to allow the operator to enter a parity-checked address of more than one character. Review the basic 2 mode operation in Figure 7-2. Most odd-numbered phases allow for entries, and most even-numbered phases provide parity checks. Note that phases 7 and 8 allow for the entry of the balance of the first and second words when the search has not been successful. (When the search is successful, a Halt instruction must be executed to allow the balance of the word to be printed, as is the case with the third word.)

1. TRANSLATE SWITCH

Operation of the Translate switch (locking type) enables a jump from 3M0,1 (the sequence address entry point in normal operation) to 2 mode, as shown in Figure 7-3. The decimal register and phase counter are not affected.



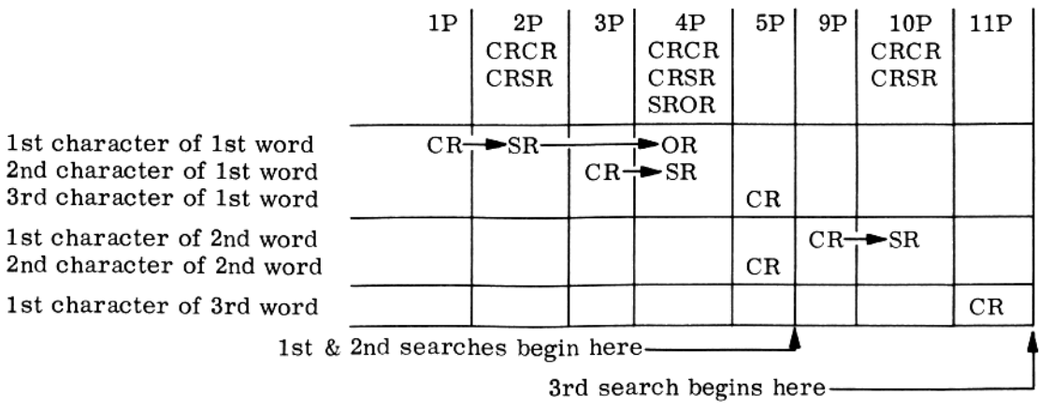
DEVELOPMENT OF J2M3 DURING 3M0,1

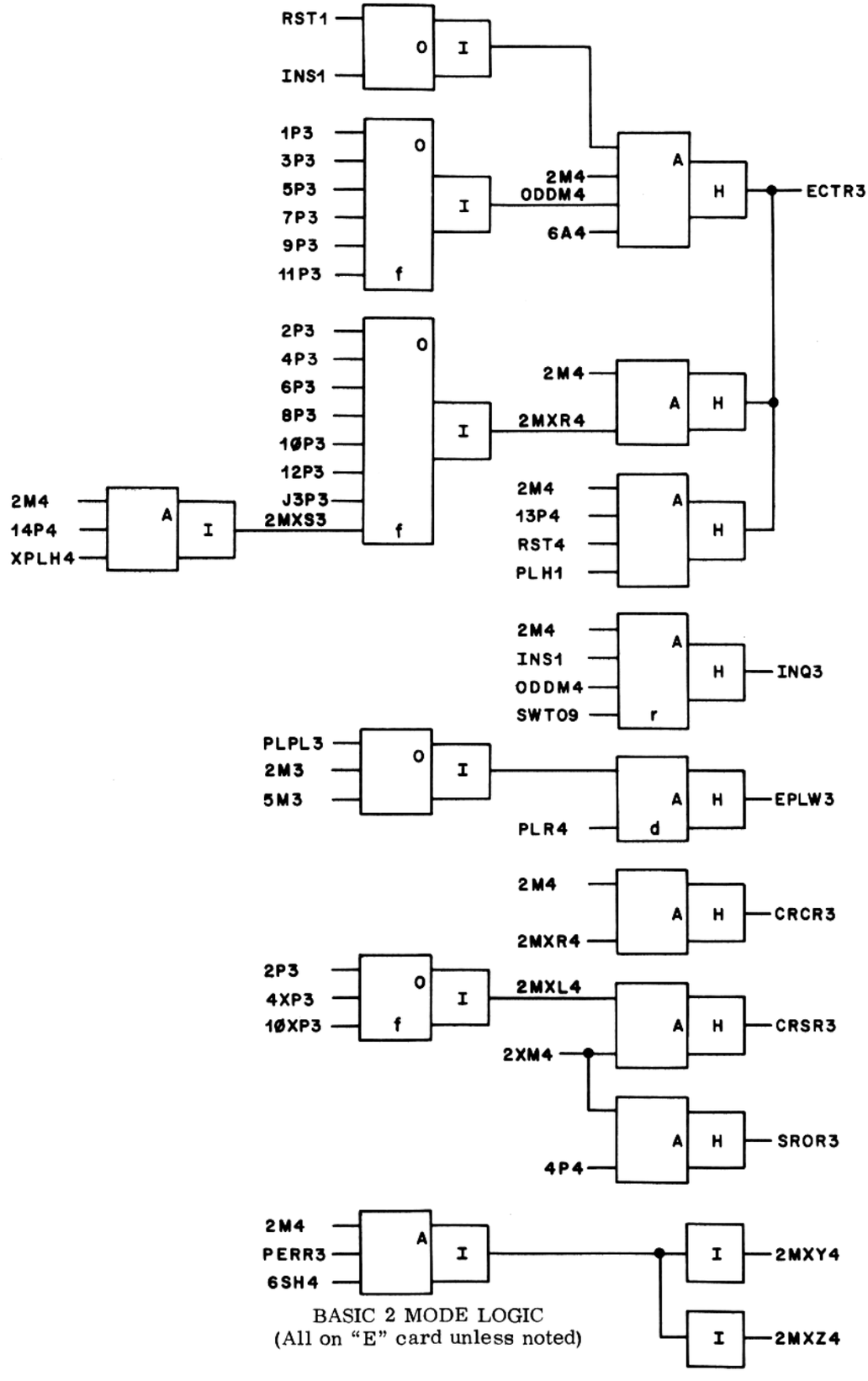
FIGURE 7-3

2. BASIC LOGIC

Figure 7-4 illustrates the line routings used in 2 mode, the development of ECTR3, and other basic 2 mode logic. Note that the program line routing is always PLPL in 2 mode. Note also that INQ3 is developed in all odd-numbered phases, 1 through 11, so that characters can be entered into the common register, and that CRCR3 is developed in all even-numbered phases, 2 through 12, so that parity can be checked. ECTR3 is developed by INS when the character is entered in the odd phases, and it is "automatic" in the even phases.

Still referring to Figure 7-4, note that in addition to the CRCR line routing in 2, 4, and 10 phases, CRSR3 is also developed, and SROR3 is also developed in 4 phase. These line routings are needed to shift the three mnemonic characters into the proper registers for the 3 mode search. The chart below lists the different mnemonic characters and relates them to the line routings in the various phases. Note that after each group of characters is entered (as each search begins), the first character is in the operation register, the second is in the search register, and the last is in the common register.





BASIC 2 MODE LOGIC
(All on "E" card unless noted)

161-182

FIGURE 7-4

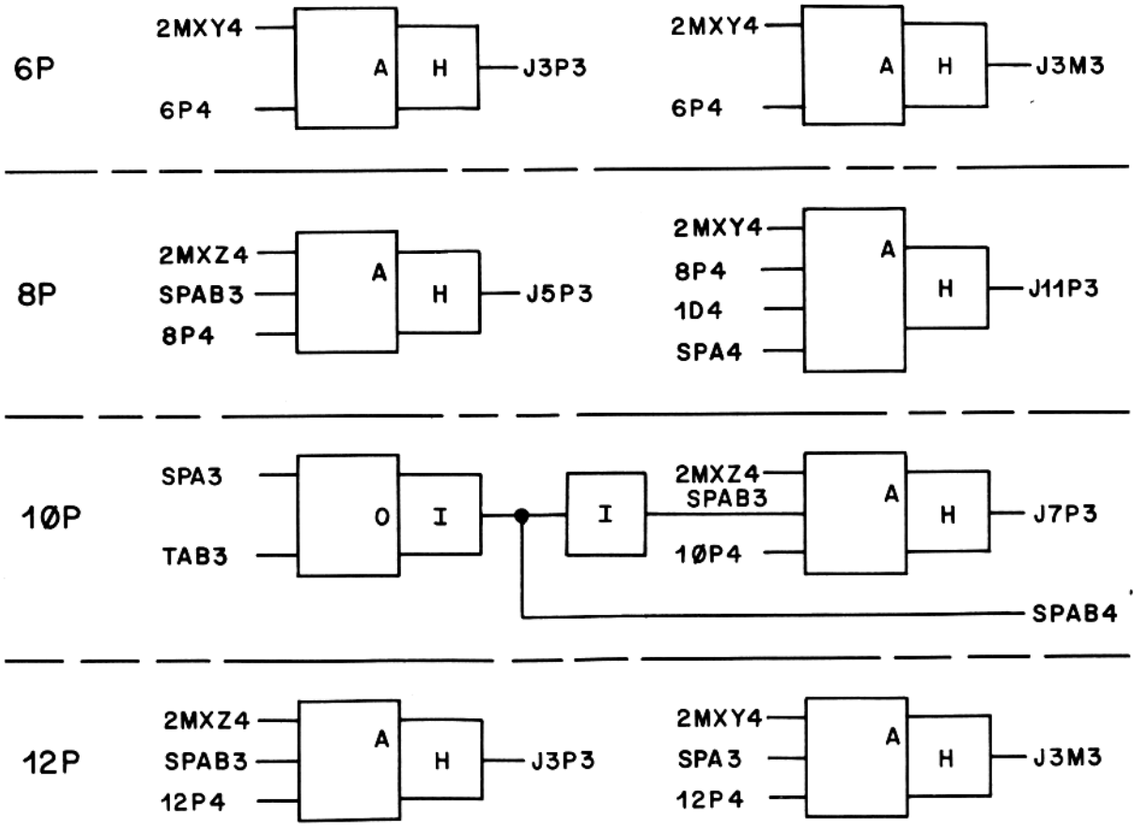
3. DECIMAL SETTINGS

The decimal register settings of 1, 2, and 3, otherwise known as "Translate Decimal", are used throughout 2 mode and 3 mode to keep track of the type of mnemonic being serviced. As soon as 2 mode 2 phase is reached, the decimal register is set to 3, where it remains until the first search is completed. If this search is unsuccessful, the decimal register is counted down to 2, and this setting allows the second mnemonic word to be entered and the second search to occur. If this search is also unsuccessful, the decimal register is counted down to 1, and the third word is entered, followed by the third search. When this search is also unsuccessful, the decimal register is counted down to 0, and this fact resets the common register and causes a search for "sequence space". The different decimal register settings enable the various jumps between 2 mode and 3 mode, and allow the search and compare of the translate sequence titles in 3 mode.

4. MODE AND PHASE JUMPS

Different mode and phase jumps occur as the result of entering a certain number of characters (attaining certain phases) or as the result of entering space or tab codes. The following discussion will outline the jumps in their normal order of occurrence as a three-word mnemonic is entered. Variations caused by the tab and space will be covered last.

a. "STANDARD" JUMPS. The "standard" jumps are shown in Figure 7-5. Once 2 mode is begun in 1 phase, the phase counter only steps until reaching 6 phase. From here a jump occurs to 3 mode and 3 phase, where the first search begins.



"STANDARD" JUMPS IN 2 MODE
(All on "E" card)

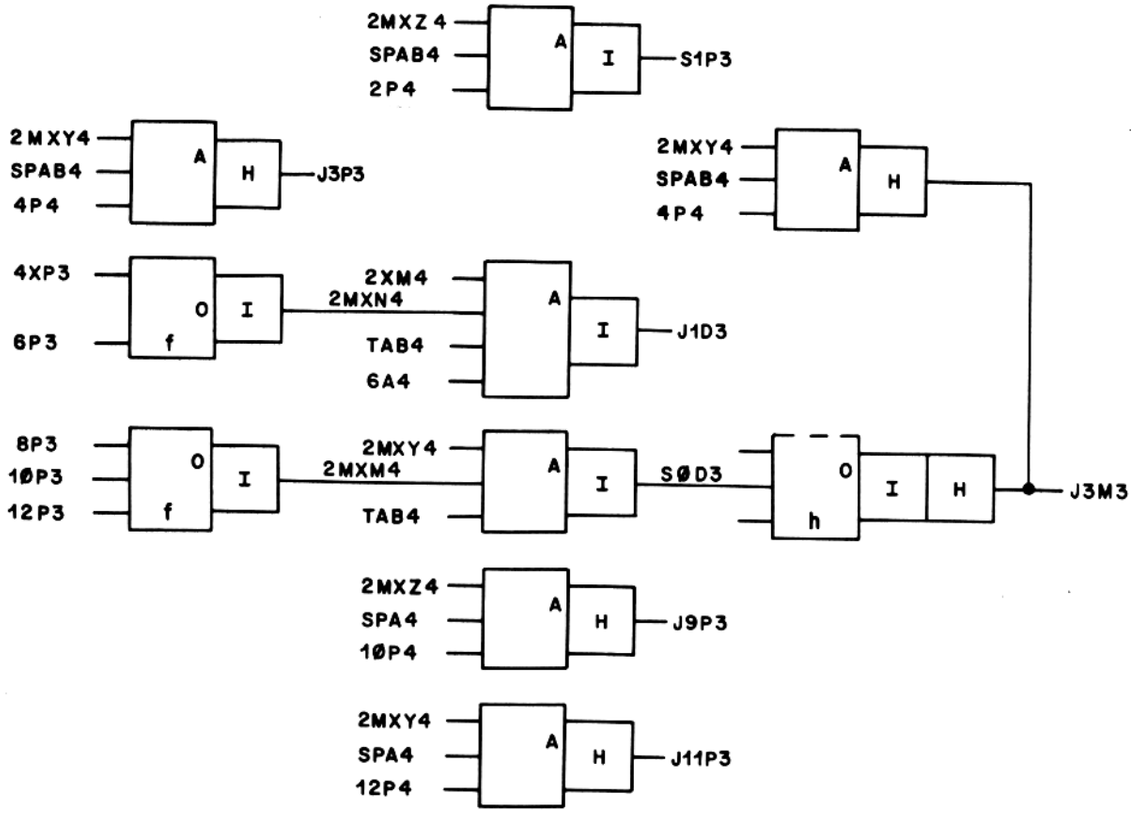
FIGURE 7-5

If the first search is successful, the logic does not return to 2 mode again until the sequence is terminated, and then 2 mode is again begun at 1 phase. If the first search is not successful, however, a jump back to 2 mode 7 phase occurs after the search, and each successive character of the first word is entered in 7 phase and parity-checked in 8 phase. At the end of 8 phase a jump occurs back to 7 phase to allow the next character to be entered, and this process repeats until a space is entered in 7 phase, thus terminating the first word. This space allows a step from 8 phase to 9 phase, where the first character of the second word is entered. After this character is parity-checked in 10 phase, a jump occurs to 5 phase, where the second character is entered. After the parity check in 6 phase, another jump to 3 mode 3 phase occurs, and the second search takes place.

If the second search is unsuccessful, the logic returns to 2 mode 7 phase to allow the entry of the balance of the second word. Again phases 7 and 8 are repeated for each character until a space is entered, signifying the end of the second word. Since the decimal register was decremented to 1 at the unsuccessful end of the second search, entry of a space in 2 mode 7 phase allows a jump from 8 phase to 11 phase. In 11 phase the first character of the third word is entered, and in 12 phase it is parity-checked. Then 12 phase causes a jump to 3 mode 3 phase, where the third search begins.

b. SPACE AND TAB. In general, a space entry in 2 mode indicates the end of the "word" and a tab entry indicates the end of the entire mnemonic entry. However, a space or tab entered before a significant character has no real effect. The common, search, and/or operation registers may or may not be reset by a tab or space entry, depending upon the phase and decimal settings in effect at the time.

One of the jumps caused by the space (J11P3 during 8 phase) has already been discussed (Figure 7-5). The rest of the jumps caused by the space and tab are shown in Figure 7-6.



JUMPS CAUSED BY SPACE AND TAB IN 2 MODE
(All on "E" card unless noted)

161-183

FIGURE 7-6

After a space or tab is entered in 1 phase, it is parity-checked in 2 phase, where S1P3 is developed, causing a jump back to 1 phase at the end of 2 phase. This allows the operator to begin the entry of the first word again.

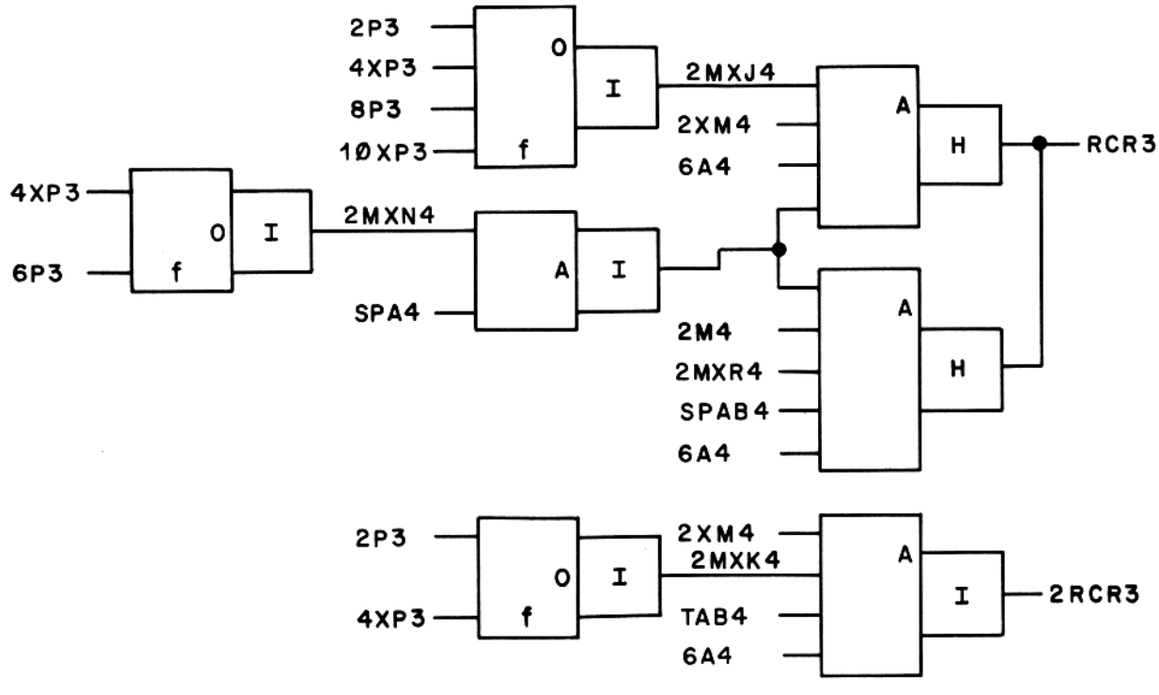
A space or tab entered in 3 phase, after being parity-checked in 4 phase, terminates the first word and causes jumps to 3 mode and 3 phase, where the first search begins. If a tab is entered in 3 phase, however, it not only denotes the end of the word, but also the end of the entire mnemonic entry. This means that the search initiated must be the "third" search. This is accomplished by setting the decimal register to 1 (via J1D3).

A tab detected in either 8, 10, or 12 phase means that an error has occurred: at least one unsuccessful search has occurred, and the fact that the tab was entered denotes that the entry is complete and no more words will be entered. In this case S0D3 is developed to set the decimal register to 0, and this signal sets 3 phase directly into the phase hold flip-flops and causes a jump to 3 mode. This begins the search for the error sequence.

A space detected in 10 phase or 12 phase simply returns the logic to 9 phase or 11 phase, respectively, the same as a tab or space detected in 2 phase: this would occur when two spaces (or more) were entered successively.

5. REGISTER RESETS

At certain times during 2 mode the common and search registers have to be reset in order to eliminate unwanted codes and to make room for new codes. Two signals are used to do this: RCR3 resets the common register alone, and 2RCR3 resets both the common and search registers. (The operation register must also be reset in the case of an error, and this will be covered shortly.) The development of these two signals is shown in Figure 7-7.



DEVELOPMENT OF RCR3 AND 2RCR3 IN 2 MODE
(All on "E" card unless noted)

FIGURE 7-7

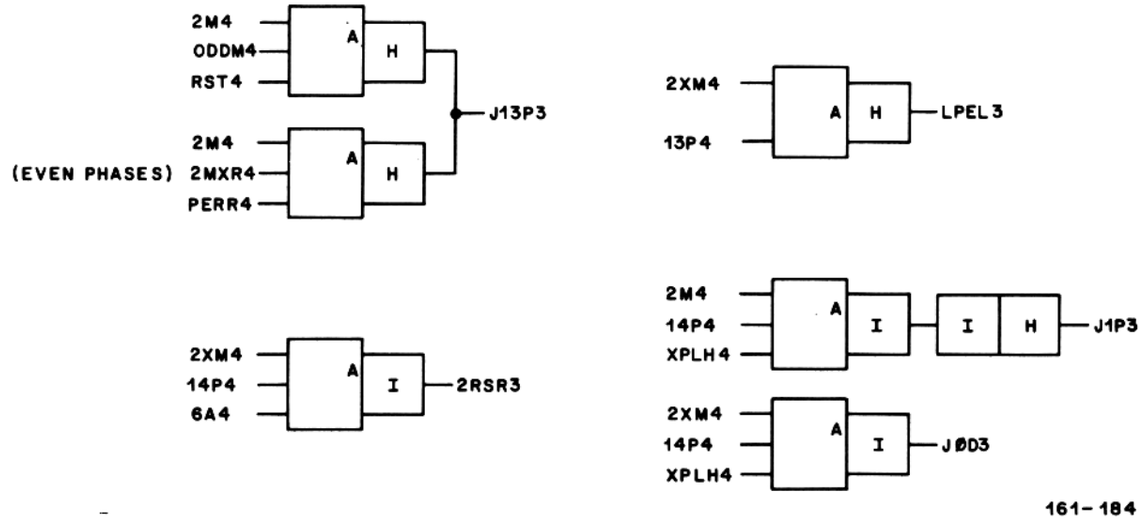
The top gate in Figure 7-7 is used to reset all codes out of the common register in phases 2, 4, 8, and 10: in 2, 4, and 10 the code just shifted into the search register must be reset out of the common register to allow for entry of the next code. In 8 phase each code making up the balance of the word must be reset out.

An exception to the preceding occurs when a space is detected in 4 phase: the common register is not reset when this occurs, because if the ensuing search is unsuccessful, the SPA3 output from the CR Decoder is needed to cause a jump back to 2 mode 9 phase (for entry of the second word), instead of to 7 phase, where the balance of the word is normally entered.

The bottom gate in Figure 7-7 allows the common register to reset when a tab is entered in 6 phase or when a space or a tab is entered in 12 phase. Note that a space detected in 6 phase prevents the reset of the common register through this gate, for the same reason noted above.

6. PARITY ERROR

Detection of a parity error in any even numbered phase of 2 mode causes a jump to 13 phase where the red error lamp is illuminated. See Figure 7-8. The phase counter remains in 13 phase until the Reset switch is operated, which causes all previously-entered characters to be "erased" and the entry of the mnemonic to start anew.



ERROR LOGIC
(All on "E" card)

FIGURE 7-8

The only way out of 13 phase is via depression of the Reset switch, which sets the XRST flip-flop. XRST and LPEL combine to set RST, which provides ECTR3 (see Figure 7-4) for the step to 14 phase. In 14 phase, 2RSR3 is developed to reset the search and operation registers. At the next program line home a jump back to 1 phase and 0 decimal occurs, J1P3 also setting CRR to clear the error code out of the common register and J0D3 also resetting the RST flip-flop. This leaves the processor in 2M0,1 awaiting the first entry in the next mnemonic.

C. 3 MODE OPERATION

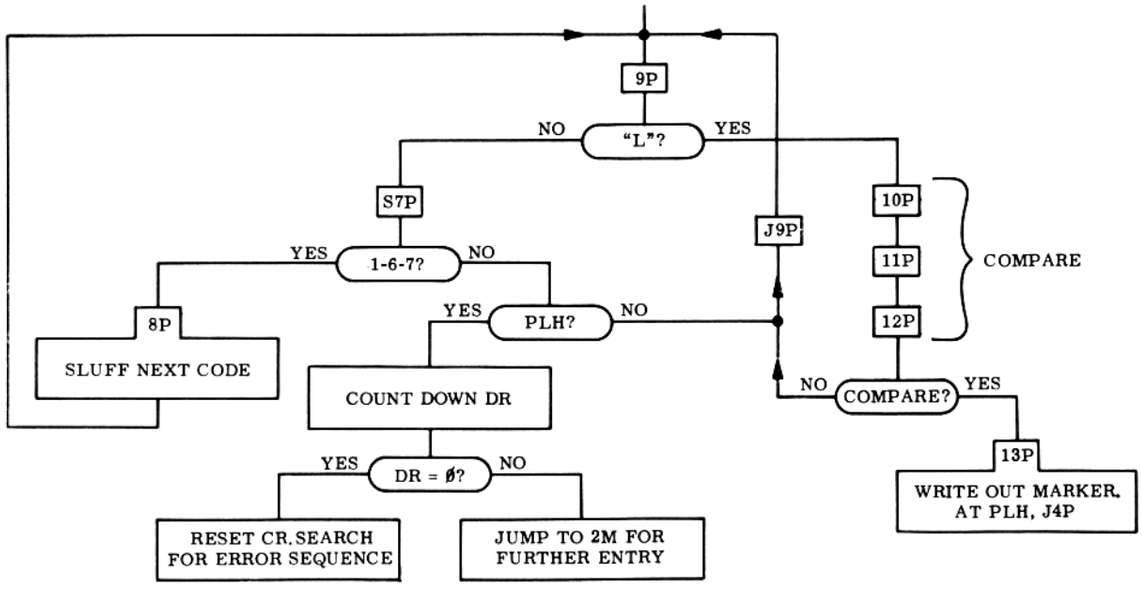
The basic search patterns operate much as before in 3 mode during Translate, but the Translate Decimal settings of 1, 2, and 3 decimal modify the search so that only 3-character sequence titles are compared. Normal sequences are ignored. After each unsuccessful search the decimal register is counted down once so that after three searches the 0D3 condition initiates a search for the error sequence (Sequence Space).

1. GENERAL OPERATION

The three-character title of the sequence being sought is stored in the operation, search, and common registers. When an "L" (translate sequence separator) is detected in 9 phase, the 9-7 jump is blocked and the phase counter steps to 10, 11, and 12 phase. In each of these phases the character coming off the program line is compared to the corresponding character entered via the 2205. If at least one of the three characters does NOT compare, a jump to 9 phase occurs at the end of 12 phase and the search for another "L" resumes. If all three characters DO compare, a step to 13 phase occurs, at which time a marker is manufactured and written onto the program line. 13 phase remains until program line home, at which time a jump to 4 phase and 6 decimal occurs to initiate marker search.

If program line home is reached without having made a successful compare, the decimal register is counted down and a jump occurs back to 2 mode to allow for the entry of additional characters to modify the three-character address. If program line home is reached and the decimal register is counted down to 0, the common register becomes reset and a NORMAL sequence search takes place for "Sequence Space".

The Flow Chart in Figure 7-9 summarizes the search action in 3 mode, 1-2-3 decimal. The basic search patterns cause an immediate jump to 7 phase for all codes other than "L", and the code immediately following any code containing the 1-6-7 bits is "sluffed" in 8 phase.



FLOW CHART
BASIC TRANSLATE SEARCH

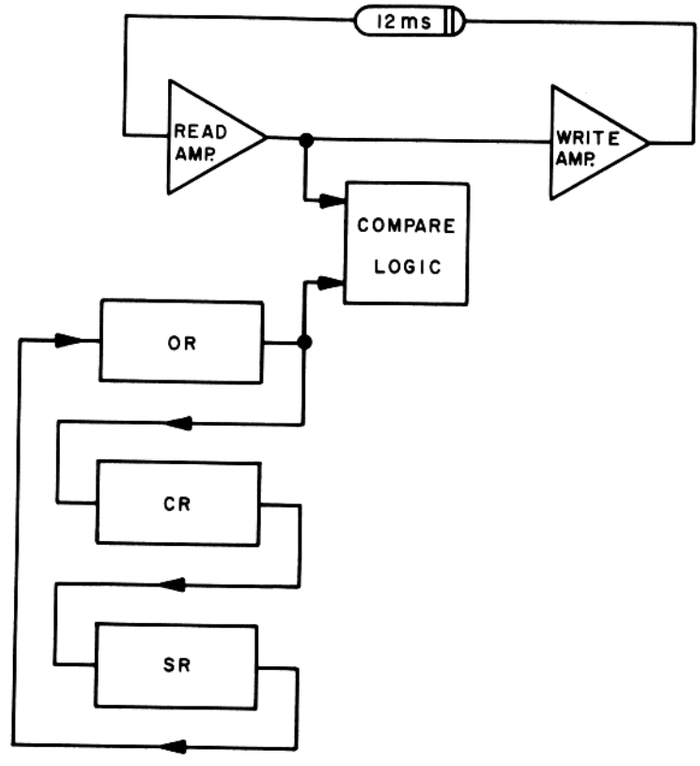
FIGURE 7-9

2. TRANSLATE RING SHIFT

During 10, 11, and 12 phases of 3 mode the three characters entered in 2 mode are compared to the three characters following each "L" on the program line. The compare logic actually compares the program line read amplifier output to the output of the 1OR flip-flop. During these phases the operation, search, and common registers perform a "Translate Ring Shift", during which the contents of each register are shifted through the other two registers and back into the original register. This ring shift is similar to the regular common register ring shift (CRCR), but since the data is passing through three registers, the ring shift requires three character times to be completed.

The block diagram in Figure 7-10 shows the output of the read amplifier being compared to the output of the operation register. As the code in the operation register is shifted out and compared, it is also shifted into the common register (ORCR), the code formerly in the common

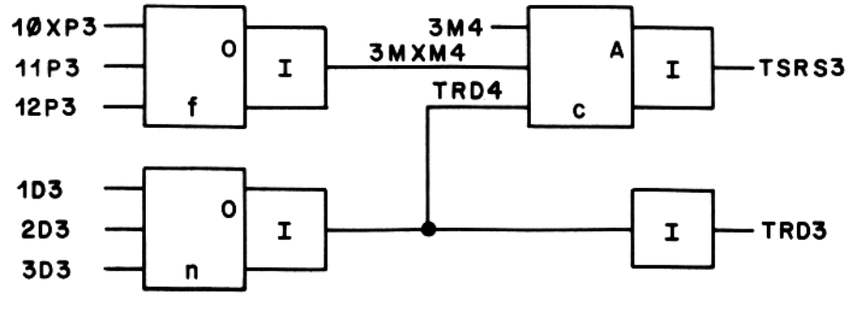
register is shifted into the search register (CRSR), and the code formerly in the search register is shifted out, to take the place of the code being shifted out of the operation register (SROR).



BLOCK DIAGRAM
TRANSLATE RING SHIFT

FIGURE 7-10

Figure 7-11 pictures the development of TRanslate Decimal, TRD, and TranSlate Ring Shift, TSRS. The latter signal is applied to the inputs and the shifting logic of the three registers to provide the ORCR, CRSR, and SROR "line routings" needed for the translate ring shift. The ring shift begins in 10 phase, and one character is compared in each phase. Thus, at the end of 12 phase the three characters are back in the register where they started. This fact is of no importance if a compare occurs, but if the compare was not successful, the three codes must be in the proper register to allow the next compare to take place.

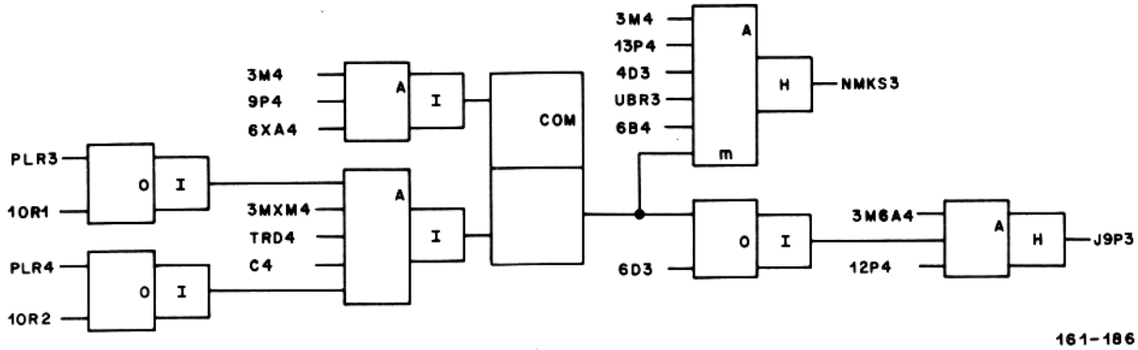


TRANSLATE RING SHIFT LOGIC

FIGURE 7-11

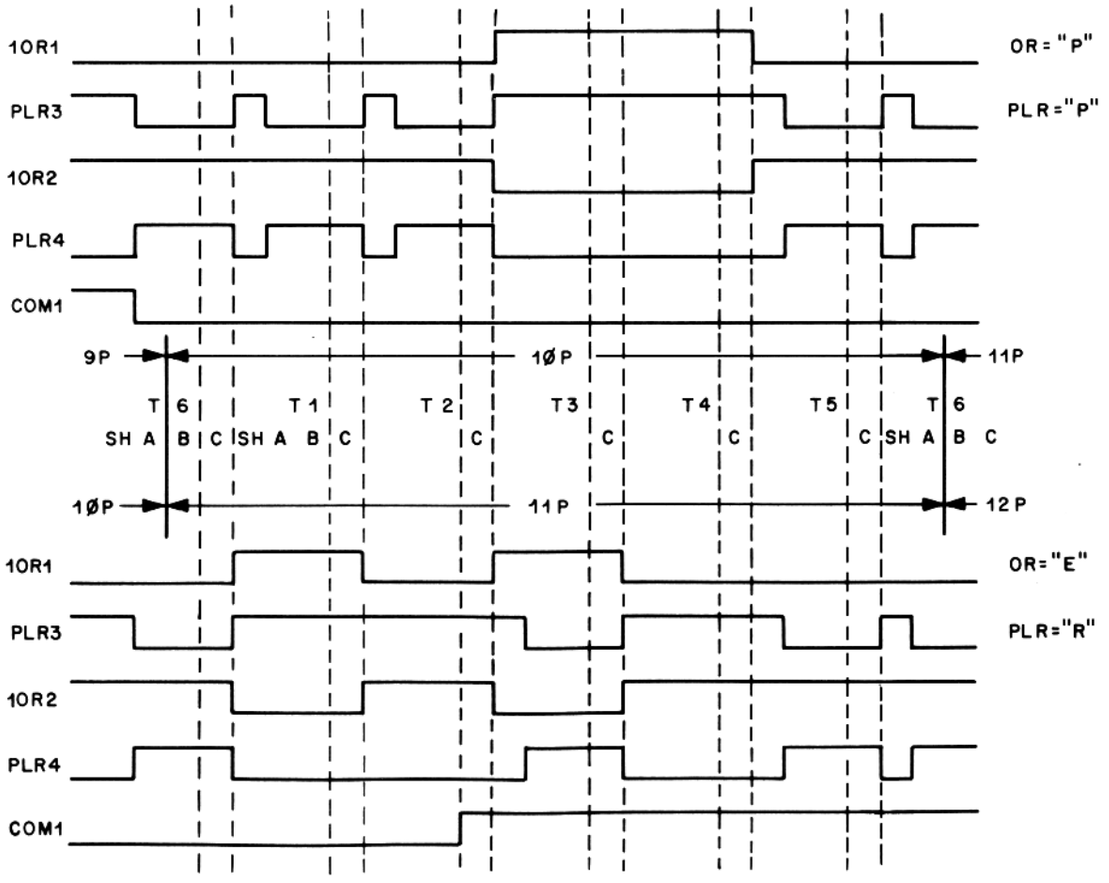
3. COMPARE LOGIC

The logic associated with the COMpare flip-flop in phases 10, 11, and 12 is shown in Figure 7-12. COM is set in 9 phase. At each fractional bit "C" time of the program line timing chain, the condition of PLR is compared to the condition of 1OR. If the outputs always compare, COM remains set: the first time the outputs vary, COM resets. The comparison is made at "C" time, just before the next shift occurs.



TRANSLATE COMPARE LOGIC

FIGURE 7-12



TIMING CHART
UNSUCCESSFUL TRANSLATE COMPARE

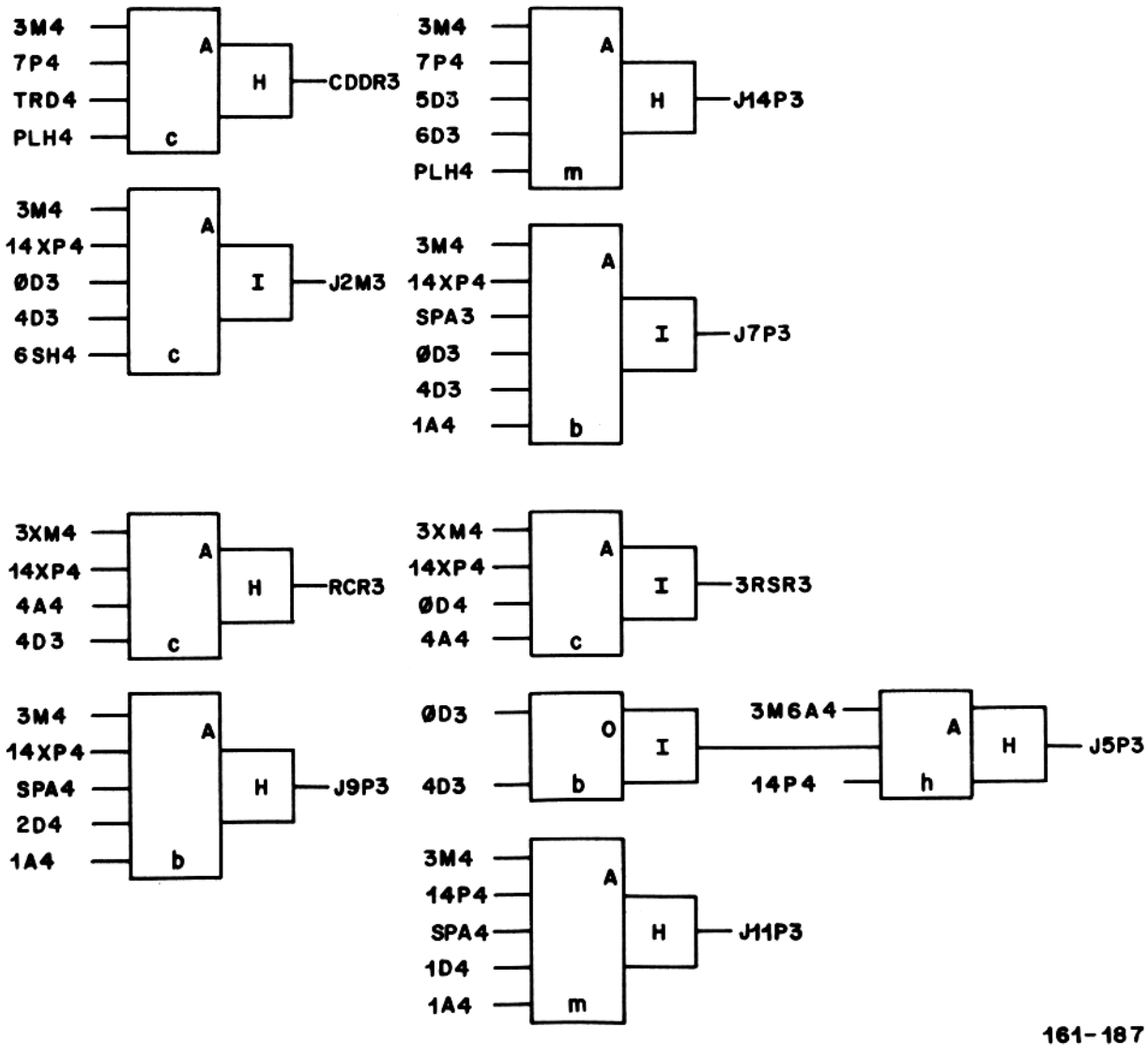
FIGURE 7-13

If COM is reset at the end of 12 phase, a jump to 9 phase is enabled. If COM is still set, a step to 13 phase occurs, where NMKS3 is developed to set a marker into the search register. Then the line routing changes to PLSR, SRPL, and the marker is written out onto the program line. (This is the same line routing and logic used after a normal sequence search and compare: refer to Section 3G for details.)

A sample comparison is shown in the timing chart in Figure 7-13. An "(L)PRE" on the program line is being compared to "PER" entered via the 2205. The top half of the figure shows the comparison in 10 phase. Since the "R" in PRE and the "E" in PER are different, COM resets in 11 phase, as shown in the bottom half of the drawing. Depending upon the codes involved, COM can reset at any "C" time during 10, 11, or 12 phase.

4. UNSUCCESSFUL COMPARE

If the entire program line is searched without locating a sequence corresponding to the title previously entered, the decimal register is counted down at program line home and a jump to 14 phase is enabled. See Figure 7-14. In 14 phase the common register is reset, and if three searches have not yet been attempted (0D3 is false: the decimal setting is either 1 or 2), a jump to 2 mode 7 phase is enabled.



UNSUCCESSFUL COMPARE LOGIC

FIGURE 7-14

An exception to the preceding is found when the first word entered in 2 mode is terminated with a space: in this case, SPA3 is true in 3 mode 14 phase, blocking the jump to 7 phase and allowing a jump to 9 phase instead. If the second word entered in 2 mode is terminated with a space, and the second search is unsuccessful, SPA4 enables a jump to 11 phase to allow for entry of the third word. Last, if three searches have occurred without a valid compare, the decimal register is at 0, allowing 3RSR3 to be developed in 14 phase to reset the operation and search registers. 0D3 also blocks the jump to 2 mode and enables a jump to 5 phase, which occurs as the timing chain restarts. This is the beginning of the normal sequence search for the error sequence.

Note that the common register is reset at 4A time in 14 phase, and that all of the jumps enabled by SPA3 or SPA4 are gated by 1A time. This allows the SPA signal to enable the necessary jump before the space is reset out of the common register.

5. SOFTWARE PRECAUTIONS

During Translate search, the "H" is treated the same as any other single-character command: its address (title) is not sluffed. For this reason all NORMAL Sequence Separator codes should be placed after the LAST "L" in the program, and "Sequence L" should be avoided so that the codes following it will not be recognized as a Translate sequence title.

D. VARIATIONS IN "NORMAL" OPERATION DURING TRANSLATE

The operation of the conditional branch and the Enter instructions is affected when these instructions are executed in the Translate mode. Since a conditional branch causes a search for a Resume to start at the beginning of a "regular" sequence, the conditional branch instruction cannot be located in a "translate" sequence. Also, provision is made in translation for the entering of alphabetic information (for addresses) under the Enter instruction.

1. CONDITIONAL BRANCH

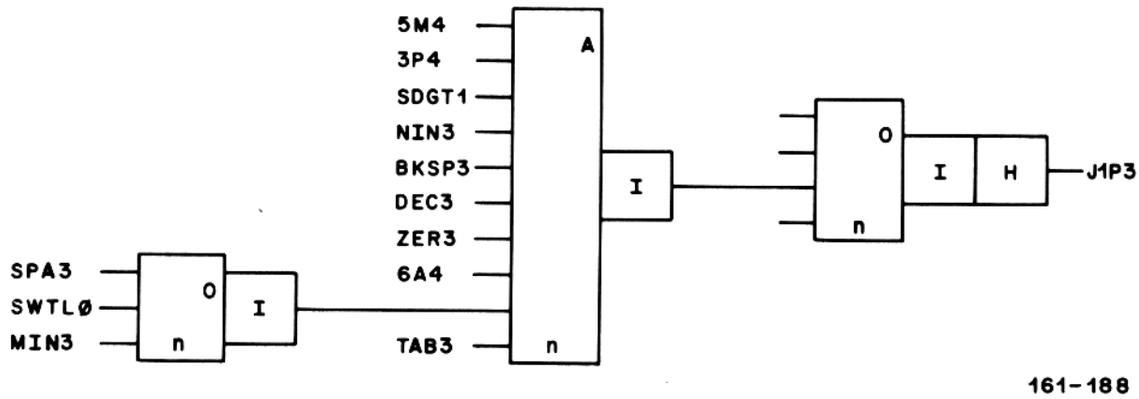
As noted in Section 3K, Conditional Branch, execution of a conditional branch instruction (with the sign of R1 in the right state) causes a search for a Resume command to start at the beginning of the sequence containing the conditional branch. Before this Resume search can begin, a normal sequence search occurs, the address of the sequence to be located being taken from one of the "blanks'' at the beginning of the program line. Since these "blanks" can hold only one character each, and since the normal sequence search occurs in 5 decimal, a "translate" sequence cannot be located at this time: only a "normal" sequence, having an "H" for a sequence separator, can satisfy the 5 decimal search.

The CRPL line routing that loads the sequence title into the first blank on the program line after a sequence entry is not inhibited during the translate sequence search, so the third character of the mnemonic (which happens to be in the common register) is loaded into the first blank during the translate sequence search. (If the first search is not successful, a new code is loaded into the first blank for each subsequent search.) Thus, when a translate sequence is finally located, the first blank contains a code representing the third character of the mnemonic. If a conditional branch instruction is executed from within this translate sequence, a search for the NORMAL sequence having an address the same as the code in the first blank is carried out. If such a sequence exists, it will be located and the Resume search will start at that point. If such a sequence does not exist, however, the processor falls into an endless loop looking for the non-existent sequence. When this occurs, the "panic button" must be depressed, with resulting loss of the program. To prevent such an occurrence, it is recommended that no conditional branch instructions be located in a "translate" sequence. Instead, when the conditional branch is necessary to the proper operation of the program, an UNCONDITIONAL branch should first be made to some NORMAL sequence, and the conditional branch then made from the normal sequence. Any number of normal sequences (having "H" sequence separators) may be included in the translate program, but, as noted earlier, they should be loaded onto the program line AFTER all of the translate ("L") sequences.

2. ENTER

During translate the Enter instruction requires a 2-digit address specifying the LRD pattern to be used, the same as in regular operation. In translation, however, alpha characters, as well as numerics, must decrement the decimal register, indicating entry of a character. This requirement is necessary to allow the entry of alpha characters for addresses of sequence separators, Type instructions, Resumes, etc., during translation.

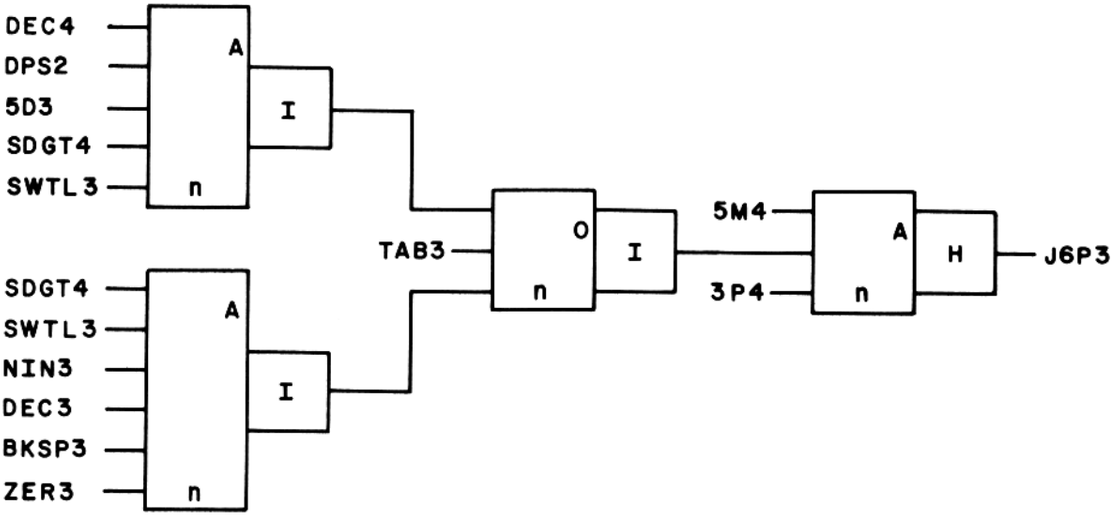
Refer to Figure 7-15 and note that the gate normally producing J1P3 after entry of an alpha character (and before SDGT is set) is disabled by SWTL0, unless the entry is a space or a minus. In other words, only the space and minus will cause a jump back to 1 phase: all other codes will cause the logic to go on and decrement the decimal register in 5 phase. (The tab still terminates the entry, the carriage return is not seen by the processor, and the decimal still fulfills the left-of-decimal requirement, as in a normal entry.)



DEVELOPMENT OF J1P3 BY SPACE OR MINUS ONLY IN TRANSLATE ENTRY

FIGURE 7-15

After entry of a significant digit, an alpha character normally provides a jump to 6 phase to terminate the entry, as shown in Figure 7-16. During translate, this jump is blocked by SWTL3 so that further characters may be entered.



BLOCKING J6P3 FROM ALPHA OR DECIMAL ENTRY DURING TRANSLATE

FIGURE 7-16

Figure 7-16 also shows the gate that normally terminates the entry (by providing a jump to 6 phase) when a decimal is entered right-of-decimal. SWTL3 again blocks this jump, allowing other codes to be entered after the decimal. An unusual feature of the decimal entry during translate is that it ALWAYS sets the decimal register to 5: even if characters have been entered right-of-decimal, a decimal entry will set the decimal register back to 5.

The numeric value of all writing machine characters entered, numeric AND alpha, is entered into R1 during execution of an Enter instruction during translate. The Enter instruction is used primarily to define field size during translation, and the numeric value of the entry is of little value unless the entry is satisfied only with digits. To simplify programming, entries to be used for computations should consist solely of digits.

The binary value of the first four bits of each code represents the number entered into R1 during a translate entry. Thus values from 0 to 15 can be entered into each digit position of R1. If these values are next stored on the storage line, they are converted to binary values between 0 and 9. No binary values >9 can be stored on the storage line because of the "dummy" add cycle that occurs during the Store operation: this add cycle detects any >9 condition and automatically "corrects" and carries 1 to the next column. (A complete table listing all of the various possibilities available through entry of alpha characters is presented on pages 4-6 and 4-7 in Section 6 of the Branch Office Library, volume DP11.)

The rules for entries made during translation are summarized in the following list:

a. The entry of any character except minus, tab, decimal, and carriage return fulfills the requirements for one digit of the entry and causes its numeric equivalent to be entered into R1.

b. Prior to the entry of a character, the entry of a space fulfills the requirement for one digit of the entry, but the entry of a minus is ignored unless the left-of-decimal requirement is 0.

c. After the entry of a character, entry of a space or minus terminates the entry. The minus also changes the sign of R1. (Note: minus has no effect if preceded only by spaces and the left-of-decimal requirement is not yet satisfied.)

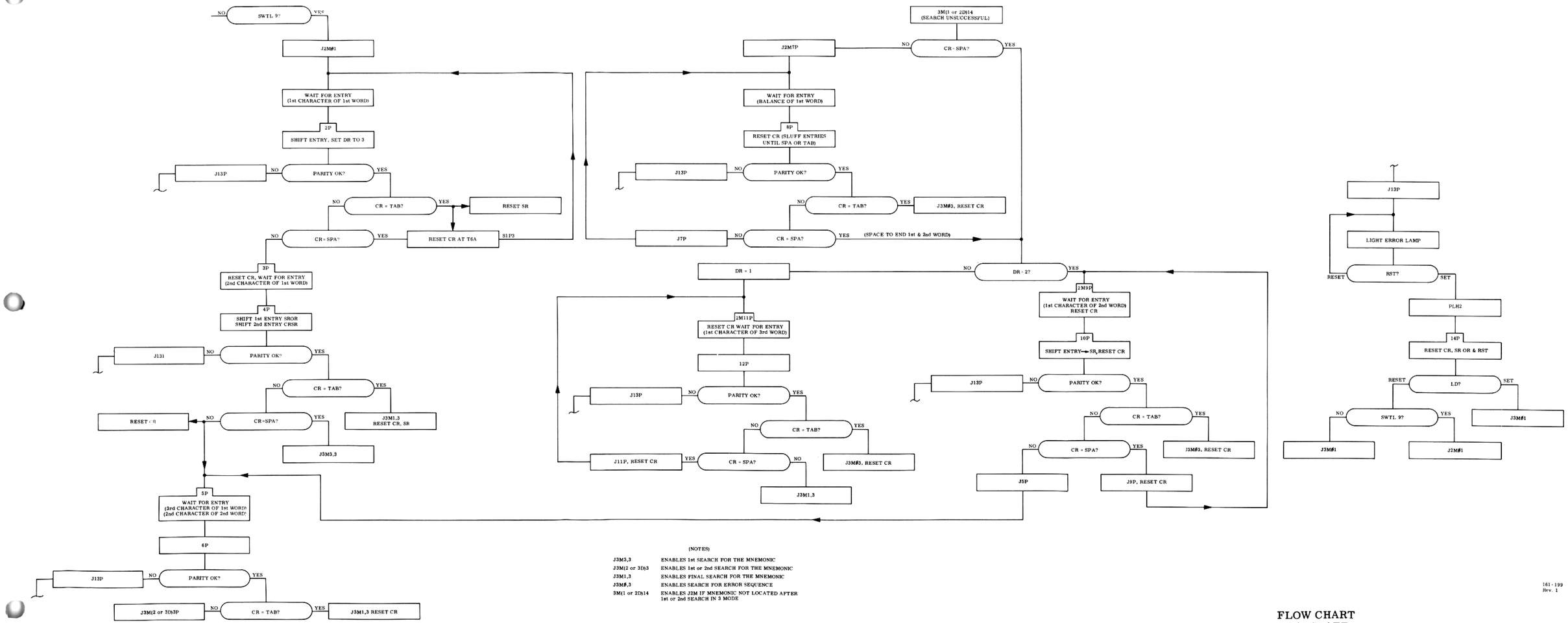
d. Entry of a decimal terminates the left-of-decimal requirements, and always prepares for the full right-of-decimal requirement, even if characters have already been entered right-of-decimal.

e. Backspace is always entered as a character.

f. A tab or carriage return entered at any time will terminate the entry without entering any additional codes into R1.

g. If the right-of-decimal requirement is 6, the entry must be terminated with a space, minus, tab, or carriage return.

Only the major differences between translate and normal operation have been covered here. Since operation in the translate mode is restricted to only a few widely scattered applications, few technicians need concern themselves with all of the details involved. For those who need more information, it is recommended they read the chapter on Advanced Programming in the Branch Office Library, Volume DP11, Section 6, Chapter 4.



(NOTES)

J3M3,3	ENABLES 1st SEARCH FOR THE MNEMONIC
J3M(2 or 3D)3	ENABLES 1st or 2nd SEARCH FOR THE MNEMONIC
J3M1,3	ENABLES FINAL SEARCH FOR THE MNEMONIC
J3M#,3	ENABLES SEARCH FOR ERROR SEQUENCE
3M(1 or 2D)14	ENABLES J2M IF MNEMONIC NOT LOCATED AFTER 1st or 2nd SEARCH IN 3 MODE

161 - 199
Rev. 1

FLOW CHART
TRANSLATE

FIGURE 7-17

TABLE OF CONTENTS

	Page
SECTION 8, ELECTRONIC CIRCUITS	
POWER SUPPLY	8-1
Universal Power Supply	8-1
Interim Power Supply	8-12
OSCILLATORS	8-18
Basic Oscillator	8-18
Two-Stage Oscillator	8-19
Controlled Two-Stage Oscillator	8-20
Basic Crystal Action	8-20
5610 Crystal Oscillators	8-21
HEXINVERTERS	8-22
Construction	8-22
Operation	8-23
"H" Type Inverter	8-26
READ/WRITE AMPLIFIERS	8-28
Write Amplifier	8-28
Read Amplifier	8-29
END OF PROGRAM DETECTOR (EPD)	8-30
Basic EPD Logic	8-31
Circuit Action	8-32
DECIMAL COMPARE	8-33
INTERFACE	8-38
Input Circuits	8-38
Translator Drivers	8-44
Control Drivers	8-47
LRC Circuit	8-47
KCL Circuit	8-48
"Busy" Circuit	8-49
LAMP DRIVER	8-49

SECTION 8, ELECTRONIC CIRCUITS

This section covers the detailed operation of many of the electronic circuits mentioned in other parts of this manual. This section is designed to cover the ELECTRONIC operation of these circuits, and only enough LOGIC is mentioned here to acquaint the reader with the location and basic function of the various circuits. The information contained in this section is not, for the most part, necessary to the understanding of the logic operation of the 5610, but is necessary in troubleshooting malfunctions that might arise in these circuits.

Step-by-step procedures for the adjustment of the power supply, the delay line amplifiers, and other circuits are not covered in this section: they will be found in the Service Letters.

A. POWER SUPPLY

There are two power supplies used in the 5610, the "interim" and the "universal" power supplies. The universal supply is used on all 5610's after serial number 1068, plus numbers 1051, 1057, 1065 and 1066. The interim supply is used on all others.

1. UNIVERSAL POWER SUPPLY

The universal power supply derives its name from the fact that it can be used with inputs of either 115 or 220 volts (nominal): it can be used "universally", with almost any input voltage.

Two transformers are employed. One provides the voltages needed by the various low-voltage supplies (+6, +50, +11, etc.). The other serves as an isolation transformer, its secondary providing 115 volts ac to the 2205, the tape drive motor, and the K1 relay circuit.

An additional feature of the universal power supply is the "battery" option, which provides protection of all delay line data for a minimum of 5 or 6 minutes in the event of a power failure. (Although only 6 minutes of protection is guaranteed, the data is usually protected for a much longer time.)

a. INPUT. Both sides of the ac input pass through the On/Off switch (S1). The "cold" side of the line is connected directly to four auxiliary receptacles, which provide power for the auxiliary input/output units. A 6-1/4 amp fuse protects the "hot" line going to the primaries of the two transformers. Each transformer has two primary windings and an identical set of inputs to each. If the 5610 is to operate on 115 volts, the two primaries are wired in parallel: if the 5610 is to operate on 220 volts, the two primaries of each transformer are wired in series. Separate taps are available to compensate for high or low voltage conditions at the power source. Complete details are listed on the power supply schematic, drawing #356138.

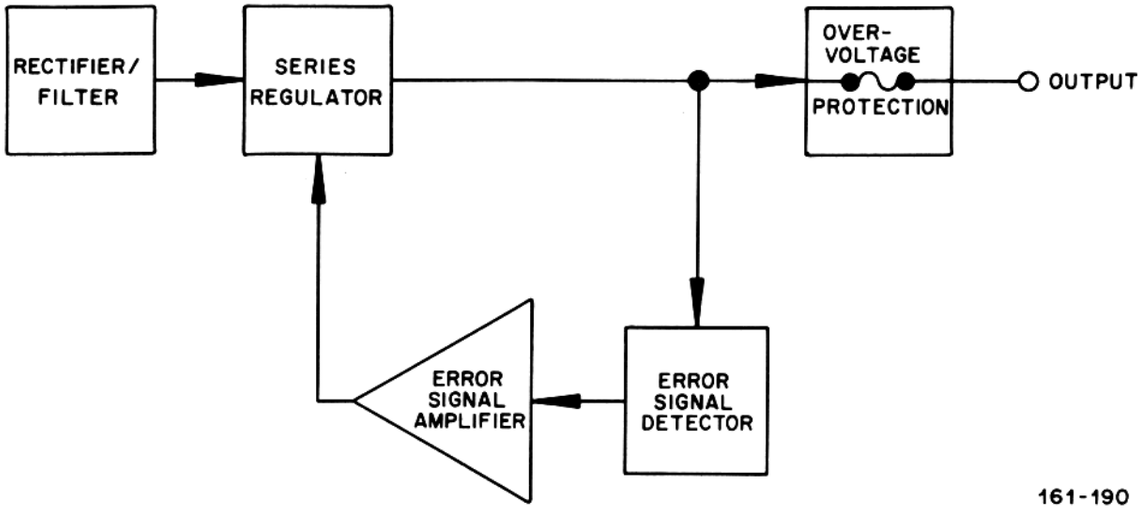
b. OUTPUT. As noted earlier, the isolation transformer (T2) provides 115 volts ac to the 2205, the tape drive motor, and the K1 relay circuit. The logic transformer, T1, provides the power necessary for the development of the following voltages:

+50 volts dc @ 10 ma, Unregulated
+24 volts dc @ 1.5 amp "
-24 volts dc @ 1.5 amp "
1 volt ac @ 1 amp "
+11 volts dc @ .5 amp, Regulated
-11 volts dc @ .2 amp "
+6 volts dc @ 4 amp "

Although many of these supplies are related to one another or dependent upon one another in some respects, they are easier to understand when studied separately. This will be the method used in the following material.

c. +6-VOLT SUPPLY. The universal +6-volt supply provides a nominal 4 amperes of current to operate all of the logic circuits in the 5610. A block diagram is shown in Figure 8-1. The rectifier/filter delivers a higher-than-necessary voltage to the series regulator, which limits this

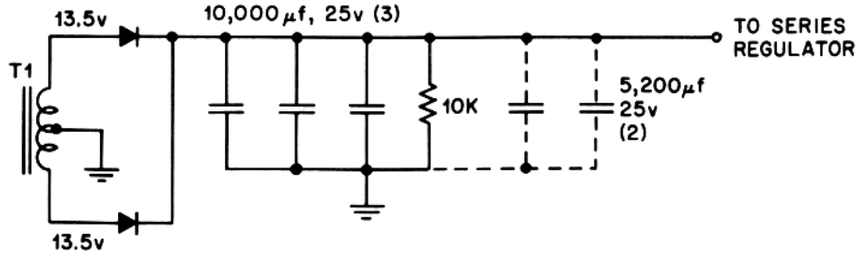
voltage to +6 volts. The error signal detector monitors the output voltage and detects any variation from +6 volts. This signal is amplified by the error signal amplifier and sent back to control the series regulator, which then adjusts to compensate for the change in the output. The overvoltage protection circuit automatically blows the fuse if the output should rise toward a value that might damage the integrated circuit hex packs.



BLOCK DIAGRAM
UNIVERSAL +6-VOLT SUPPLY

FIGURE 8-1

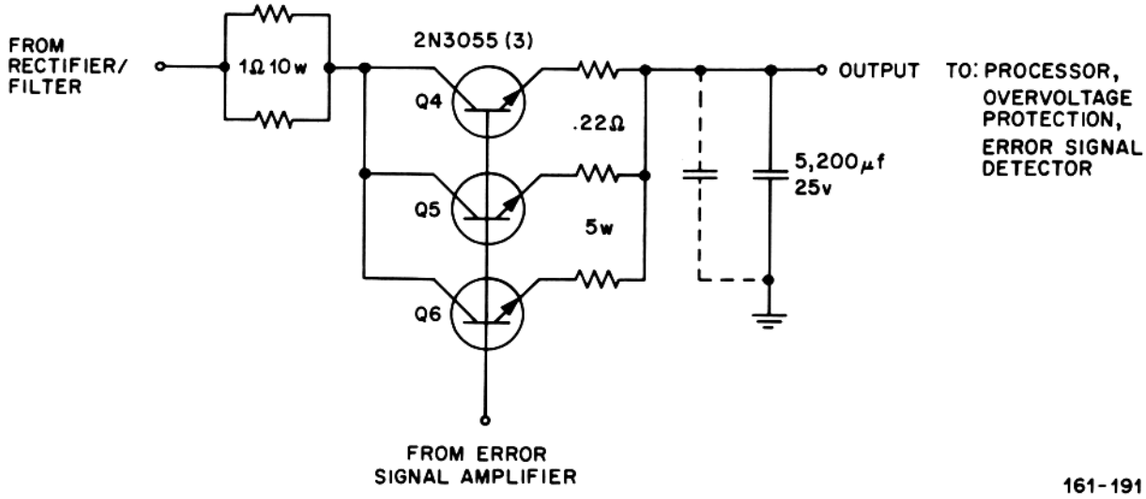
(1) Rectifier/Filter. The rectifier/filter section of the +6 supply is the same section used to provide filtered DC to the +11-volt supply, so the voltage developed (about 18 volts) is slightly higher than needed for 6-volt operation. Refer to Figure 8-2 and note that the rectifier consists of a conventional center-tapped transformer and full-wave rectifier. The filter is made up of a number of large capacitors. (The dotted lines represent additional capacitance added when the battery option is not used.) The 10,000 ohm resistor is used to discharge the capacitors when the power is turned off. Note that there is no series resistance between the output of the rectifier and the input to the filter: it might seem that the rectifier diodes would be damaged by excessive current when power is first applied. Such a series resistance is not needed, however, because the particular rectifier diode used (1N3492) is capable of withstanding a remarkably high surge current.



RECTIFIER/FILTER
UNIVERSAL +6-VOLT SUPPLY

FIGURE 8-2

(2) Series Regulator. The series regulator is pictured in Figure 8-3. Three transistors are used instead of one to share the current load. The series resistors drop part of the voltage, thus lessening the load on the transistors.



SERIES REGULATOR
UNIVERSAL +6-VOLT SUPPLY

FIGURE 8-3

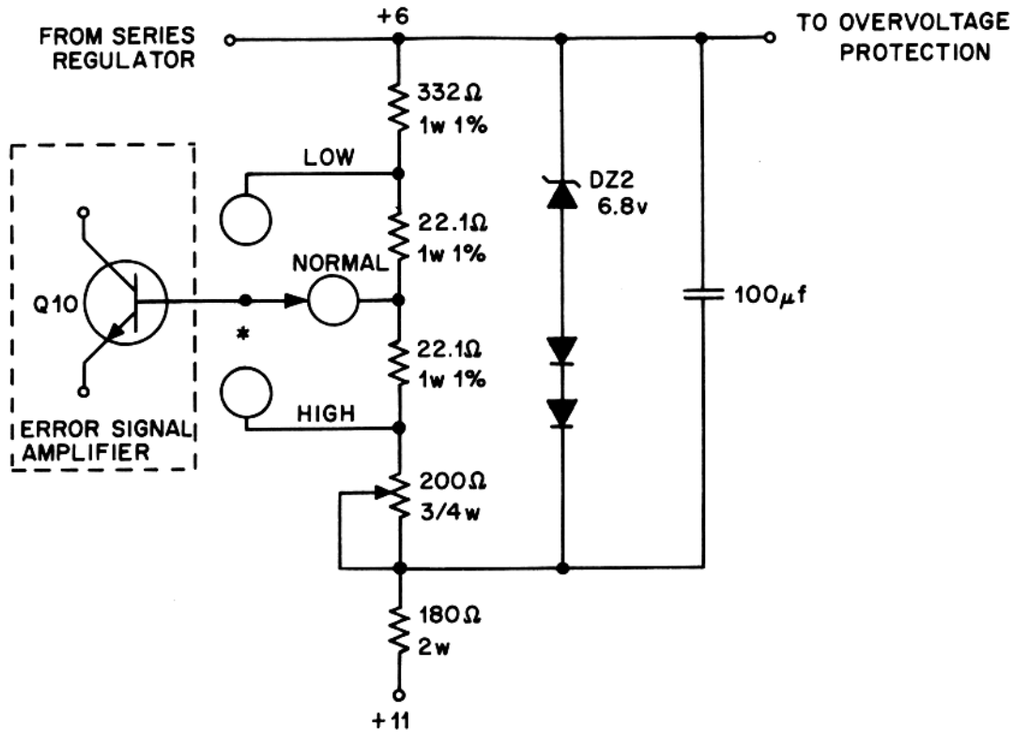
The signal from the error signal amplifier controls the amount of current flowing through the three transistors. If the load resistance should drop, thus causing an increase in load current, the output voltage starts to drop. This small drop is detected by the error signal detector, boosted by the error signal amplifier, and applied to the base of the three regulator transistors, causing them to increase conduction. This provides more current to the load and allows the voltage to remain at +6 volts. An equal but opposite reaction occurs with an increase of load resistance. Similar reactions also occur as the result of changes in the input voltage from the rectifier/filter. This will be more easily understood after the error signal detector is covered.

The capacitor absorbs noise spikes and holds the output up through very short fluctuations in the input voltage.

(3) Error Signal Detector. The error signal detector is shown in Figure 8-4. The base of transistor Q10 (part of the error signal amplifier) is held at a constant voltage in relation to the output by the action of zener diode DZ2. As minute changes occur in the +6-volt output, these changes are seen at the base of Q10, when observed in relation to ground or minus. The effect of changes will be shown shortly.

The 200 ohm potentiometer is used to adjust the output voltage to precisely 6 volts: by varying the voltage drop across the lower leg of the voltage divider, the operating point for Q10 may be varied.

The HIGH-NORM-LOW switch is a service tool which allows the CSRepresentative to increase or decrease the output when testing for marginal operation. Note that setting the switch to the HIGH position lowers the voltage at the base of Q10. It will be seen shortly that this increases the output voltage level. Note that this switch does not compensate for high or low input voltage: if such conditions are present and too extreme for the regular adjustment (the 200 ohm potentiometer) to compensate, the primary connections to the transformer should be changed. Do not operate the HIGH-NORM-LOW switch with the power on: this usually triggers the overvoltage SCR and causes the fuse to blow.



*Switch is not in all machines

ERROR SIGNAL DETECTOR
UNIVERSAL +6-VOLT SUPPLY

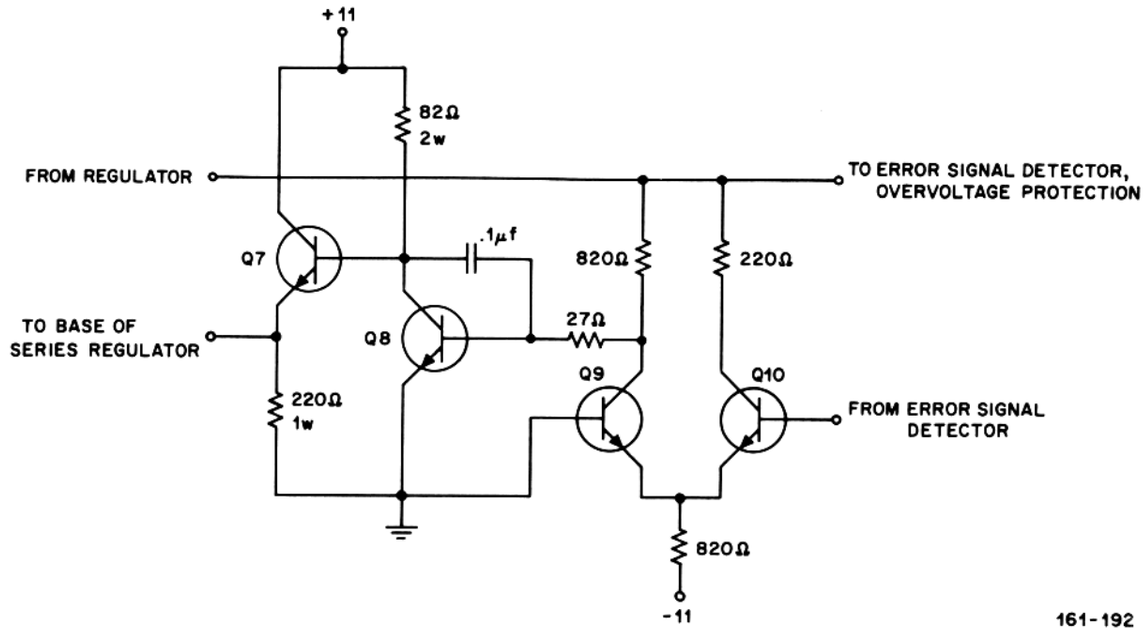
FIGURE 8-4

(4) Error Signal Amplifier. The error signal amplifier is shown in Figure 8-5. Changes at the base of Q10, provided by the error signal detector, are transferred to the collector of Q9, further amplified by Q8 and Q7, and finally applied to the bases of the series regulator transistors to effect a change in the total current flow in the +6-volt supply.

Q9 and Q10 function as a comparator: the current through their emitter resistor remains relatively constant, so any change of current flow through Q10 is accompanied by an equal but opposite change in Q9 current. For example, assume a rise in input voltage. This tends to drive the base of Q10 more positive, increasing its conduction. This increased conduction tends to drive the Q10 emitter more positive, which only decreases the forward bias on Q9, causing it to conduct less heavily. This drives the Q9 collector voltage higher, causing Q8 to turn on more heavily and driving the base of Q7 more negative. This decreases Q7 conduction, causing the voltage sent to the bases of the series regulator transistors to drop proportionally. This lower base voltage causes the series regulators to conduct less heavily, decreasing the current flow through the load and lowering the output voltage.

The actual voltage at the emitters of Q9 and Q10 varies very little because the base of Q9 is grounded and the current through the emitter resistor remains very stable. However, the CURRENT through the two transistors does vary.

A drop in output voltage has an equal but opposite effect on the entire amplifier, ultimately causing the series regulators to conduct more heavily to bring the output back up to 6 volts. Note that a change in the output will cause this corrective action regardless of the reason for the change: the change can be due to changes in either the load resistance or the input voltage, and the corrective action will still be the same. This is aided by the fact that the error detection and amplification is referenced to a regulated + and - 11 volts, making these circuits relatively insensitive to changes in input voltage (except as detected on the +6-volt output).



ERROR SIGNAL AMPLIFIER
UNIVERSAL +6-VOLT SUPPLY

FIGURE 8-5

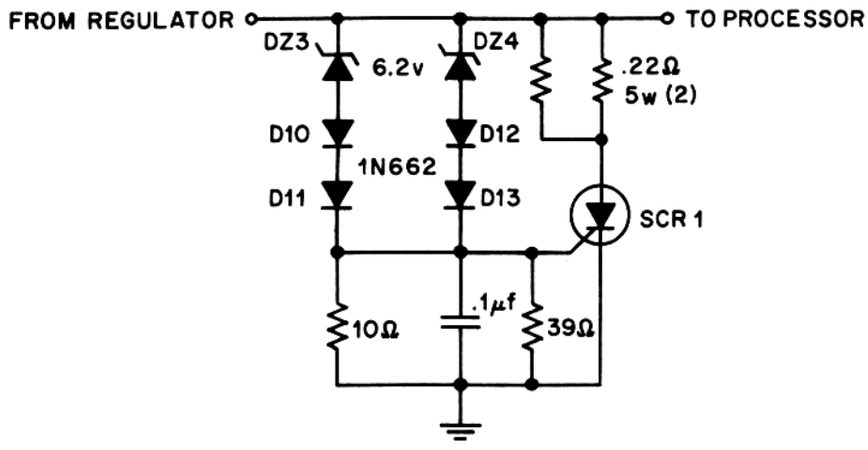
Notice that Q9 and Q10 make up a differential amplifier. This minimizes any tendency for drift to occur in this critical first stage of amplification: any temperature effects will be felt by both transistors, and the effect on one will be canceled by the other. Since normal circuit action is for a change in Q10 current to cause an opposite change in Q9, changes caused by temperature are canceled because they affect both transistors equally.

The .1 μf capacitor around Q8 serves to slow down the operation of Q8, thus preventing any changes in the output from causing corrective action at the series regulators fast enough to allow oscillation to occur.

(5) Overvoltage Protection. The overvoltage protection circuit disables the +6-volt supply in the event of a malfunction that would allow the voltage to reach a value high enough to damage the IC hex packs. It is illustrated in Figure 8-6. With the output at 6 volts, the gate circuit to the SCR is open due to the zener action of DZ3 & 4. If some malfunction should occur which drives the output up toward about 8.4 volts, the zener diodes conduct, driving the gate of SCR1 positive, causing it to turn on. SCR1 then remains on until the F9 fuse (6-1/4 a.) blows. This opens the anode circuit of SCR1 (and the +6-volt output), causing it to turn off.

Occasionally some of the 1N662 diodes are shorted out with bus wire to insure that the SCR will fire before +8.4 volts.

During troubleshooting, when the load is disconnected from the power supply, the SCR may turn on due to noise spikes. Operation of the HIGH-NORM-LOW switch may also trigger the SCR: it is best to turn off the ac power before operating this switch. If the SCR should turn on for any reason, ac power must be turned off when replacing the fuse in the +6-volt supply.



OVERVOLTAGE PROTECTION CIRCUIT
UNIVERSAL +6-VOLT SUPPLY

FIGURE 8-6

d. 11-VOLT SUPPLIES. Two interrelated power supplies provide +11 volts at .5 amp and -11 volts at .2 amp. These voltages power the delay line read and write amplifiers and the end of program detector. The two supplies are dependent upon each other because the read/write amplifiers require both voltages to remain constant for reliable operation.

(1) +11-Volt Supply. The same rectifier/filter circuit that provides the input to the +6-volt supply also serves the +11-volt supply. A single series regulator transistor controls the current, maintaining the output at 11 volts. An 8.2-volt zener diode in a voltage divider provides a reference from which variations in the output are detected and sent to a two-transistor error signal amplifier. From here the signal is applied to the base of the series regulator transistor to control the output current. There is no overvoltage protection. The entire circuit is somewhat less sophisticated than the +6-volt supply, mainly because the current drain is less, but also because the accuracy of the regulation need not be quite so exact.

Refer to the schematic of the 11-volt power supplies in Figure 8-54 (a fold-out in the back of this section). Note that the error signal detector is quite similar to that used in the 6-volt supply. A minute change in the +11-volt output is detected at the wiper of R11. This affects the bias on Q3, which in turn affects the bias on Q2, causing a change in Q1 current to bring the output back to +11 volts. The dotted lines in Figure 8-54 again represent additional capacitance added when the battery option is not used, to help hold the output up through very short dropouts in input voltage.

(2) -11-Volt Supply. The -11-volt supply is quite similar to the +11-volt supply, the basic operating principles being the same. Still referring to Figure 8-54, note that the rectifier/filter section receives its power from the same windings of T1 as do the +6 and +11-volt rectifiers. The rectifier diodes in the -11-volt supply are of a lower current type, so a series resistor (.22 ohms, 5 watts) is added to limit the initial current flow when power is first applied. Only two filter capacitors are used because the demand on the supply is not as great as that on the +11-volt supply.

Note that the error signal detector does not contain a voltage reference element of its own, but uses the zener diode in the +11-volt supply. This is made possible by the fact that the two error signal detectors make up one long voltage divider from +11-volts to -11-volts and back to +11-volts. Because of this arrangement, any change in the output of one supply will affect the other to a certain extent. This is desirable in that, as pointed out earlier, both supplies are used to drive the delay line amplifiers, and the ratio of the two voltages in respect to ground must remain constant.

The 270 ohm resistor around Q11 is necessary to get the two supplies started. Since each supply is biased by the other, neither could start up without bias from the other: the resistor allows negative voltage to appear on the -11-volt supply as soon as power is applied. This negative voltage provides the bias for the +12-volt regulator, which eventually provides the bias for the -11-volt regulator.

At this time it is also well to note the two 220 pf capacitors in the -11-volt regulator circuit. Their only purpose is to slow down the regulating action of the -11-volt regulator when power is first applied and all of the various voltages are first building up. Since -11 volts is needed in developing +11 volts, and +11 volts is needed in developing +6 volts, it is desirable to have the -11-volt supply become negative as early as possible. By disabling the regulator immediately after power is applied, the -11-volt supply is allowed to build up almost immediately, and the other voltages are thus developed quicker also.

The capacity between the emitter of Q3 and the base of Q11 couples the rising positive voltage from the +11-volt supply to the -11-volt supply regulator, thus forward biasing Q11 more heavily. At the same time, the capacitor between the emitter of Q11 and the base of Q12 holds the base of Q12 negative momentarily so that it cannot turn on to inhibit the forward bias on Q11. Both of these capacitors thus speed up the development of a negative voltage on the -11-volt supply.

The 270 ohm resistor also makes it unecessary to include a bleeder resistor for the filter capacitors by providing a discharge path through the 330 ohm resistor across the output. This resistor is smaller in value to limit the peak voltage, should some malfunction occur, to a safe value, at least until the fuse blows. It also serves as a temporary load when the power supply is out of the processor and disconnected from the regular load.

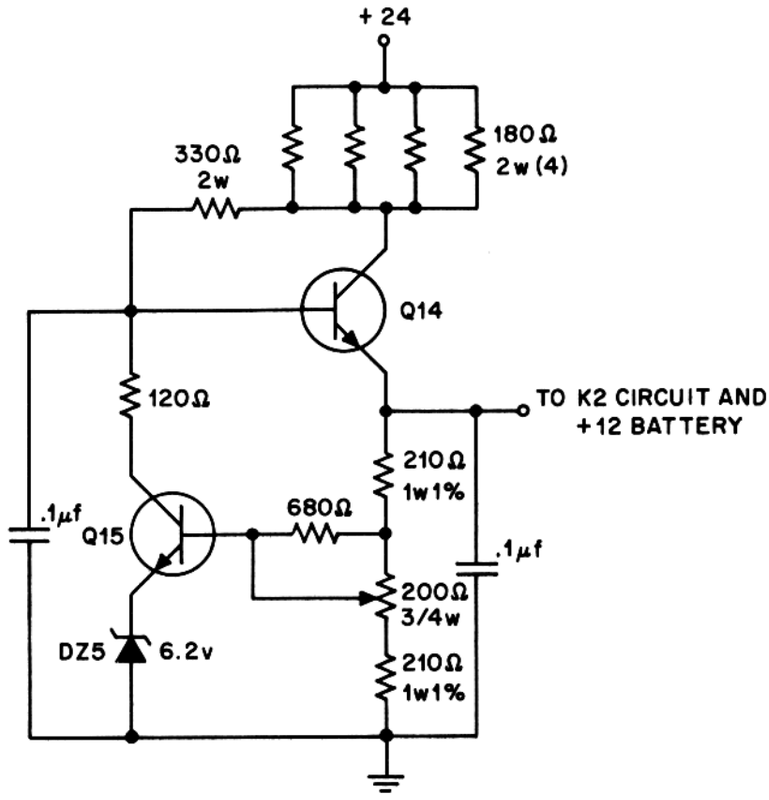
e. BATTERY OPTION. The battery option consists mainly of three parts: two pairs of 6-volt rechargeable batteries (one pair for each of the two 11-volt supplies), two battery charging circuits, and a relay which switches the batteries into or out of the circuit. This relay is controlled by a circuit which automatically disconnects the batteries from their load when their voltage drops too low.

Figure 8-54 at the end of this section shows the way the batteries are connected into the +11- and -11-volt circuits. Relay K2 must be "picked" for the batteries to power these two supplies. Note that the .22 ohm series resistor in the -11-volt supply is bypassed when the batteries are used: this is because the filter capacitors will always be charged by the time relay K2 picks, so there is no danger of damaging the diode.

(1) Battery Charger. The schematic of the +11-volt battery charger is shown in Figure 8-7. The emitter of Q15 is clamped to +6.2 volts by zener diode DZ5. The setting of the 200 ohm potentiometer controls the base current (and thus the conduction) of Q15, which determines the bias on Q14. Thus, the potentiometer is adjusted to provide +14.6 volts at the emitter of Q14, which is the output to the battery.

The -11-volt charger is identical with the exception of the use of PNP transistors instead of NPN's, the connection to -24 volts instead of +24 volts, and the reversal of the zener diode in the circuit. Also, the 330 ohm resistor connected to the base of Q14 is replaced with a 470 ohm resistor (at the base of Q16) because the -11-volt charger is not required to deliver as much current.

The battery charger connections to the batteries are slightly different in each of the two cases. The +11-volt charger output goes to three places: it is connected directly to the K2 relay circuit; it is connected to the +11/+6-volt power supply input through a diode; and it is connected to the +12-volt battery pair by a set of normally open K2 relay contacts. The -11-volt charger output is connected to the battery through a diode, and to the -11-volt power supply input through both a diode and a normally open K2 contact. Refer to Figure 8-54.



+11-VOLT BATTERY CHARGER

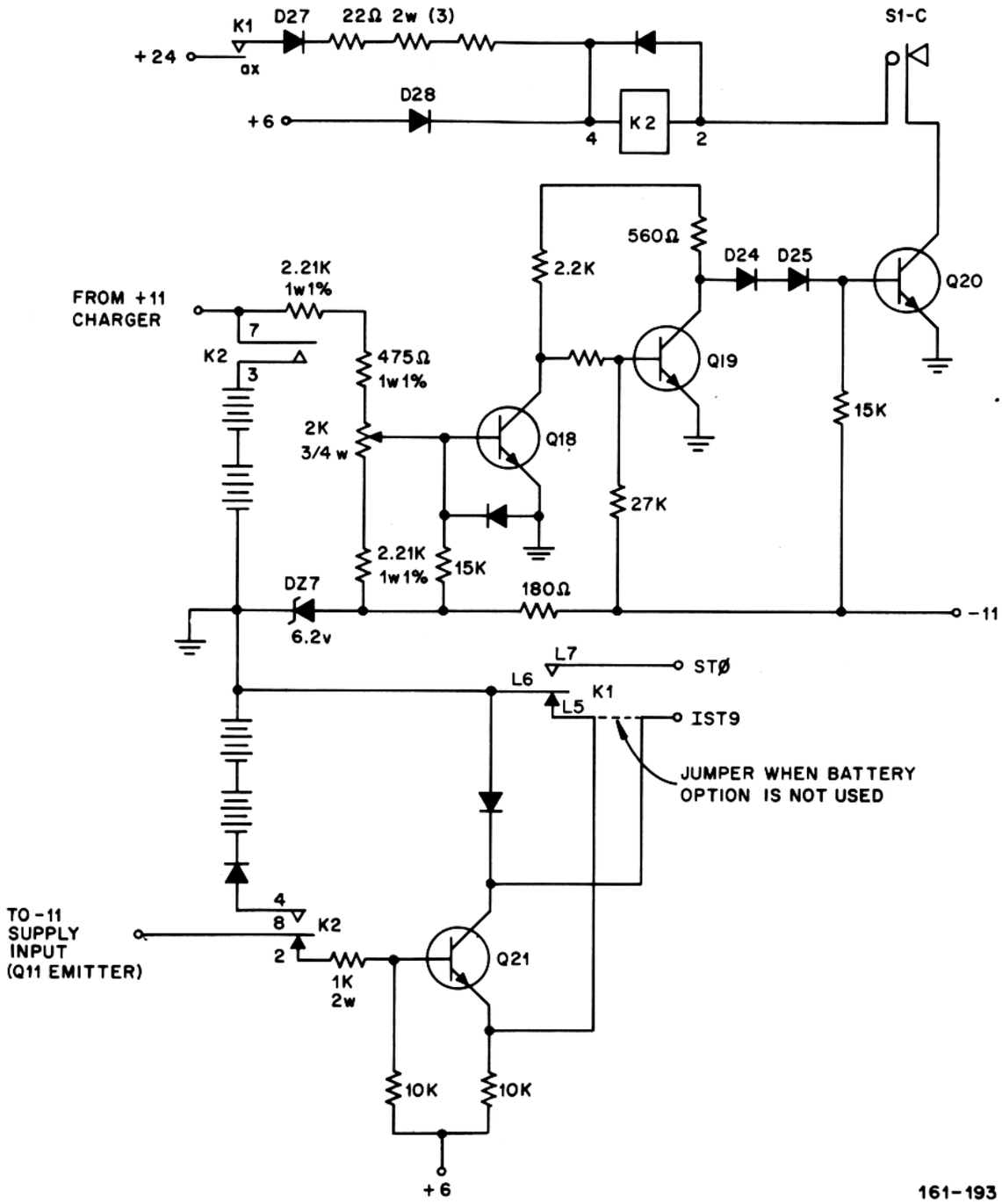
FIGURE 8-7

(2) K2 Relay. The purpose of the K2 relay is to open up the IST9 (Initial STart) line when the K1 relay drops out as a result of line voltage failure. Without the battery option, any line voltage dropout long enough to allow K1 to drop will result in loss of all delay line data. With the battery option, the +6- and the 11-volt supplies will remain active, thus saving all delay line data until power is restored. Thus, when K1 drops, K2 holds the IST9 signal false to prevent a general reset of the entire processor.

Figure 8-8 illustrates the K2 relay circuit. When power is first applied, the +6-volt supply provides the high initial current required to first energize K2. After 1 second, when K1 picks, the +24-volt supply provides the "hold" current for K2, reverse biasing the diode D28. K2 remains energized through D28 when a power failure occurs and K1 drops out.

The ground return for K2 is provided by an electronic circuit which monitors the battery voltage and causes K2 to drop when this voltage drops to a certain level. Note that the +11-volt charger circuit must be in operation before K2 can pick. When the charger output is developed, it applies some positive voltage at the base of Q18, turning it on. This turns off Q19, which turns on Q20, providing ground to the coil of K2.

As soon as K2 picks, its contacts 3 and 7 close to allow charger current to flow into the battery. Also, its contacts 2 and 8 open, causing Q21 to become reverse-biased. This opens the IST9 line, disabling the start-up conditions.



K2 RELAY CIRCUIT
UNIVERSAL POWER SUPPLY

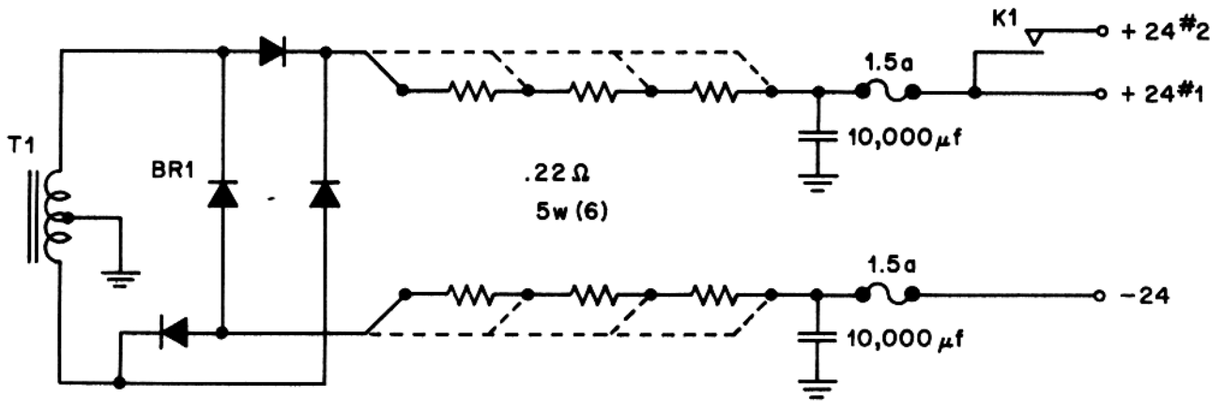
FIGURE 8-8

If a power failure should occur, K2 holds the IST9 line open even though K1 drops, and its other contacts keep the battery in the circuit to provide the +6 and +11 voltages. As long as the battery voltage remains high enough to keep Q18 turned on, K2 remains picked. When the battery voltage drops to about 10.8 volts, Q18 turns off, turning on Q19, which turns off Q20 and allows K2 to drop. When K2 drops, it removes both batteries from the circuit and provides a circuit to the base of Q21 so that IST9 may be produced when power is restored.

The K2 circuit is opened by a set of contacts on the main power switch so that the batteries will not hold power up when the 5610 is shut off.

When the battery option is not installed, a series of 5200 μf capacitors are installed at strategic points in the circuit (as noted by the dotted lines in the various drawings) to help in maintaining the voltages through short power dropouts.

f. 24-VOLT SUPPLIES. The +24- and -24-volt supplies are virtually identical. They are used to power the indicator lamps on the front of the processor and as a source for the 2205 interface circuits. They are not regulated, and they are adjustable only within certain limits. See Figure 8-9. Each supply is fed by a 19.5-volt center-tapped winding of T1, rectified conventionally, and boosted up to 24 volts by a 10,000 μf capacitor. Note that although the rectifiers used are physically part of a bridge, they are not actually connected in a bridge configuration: each half of the bridge supplies one of the 24-volt supplies.



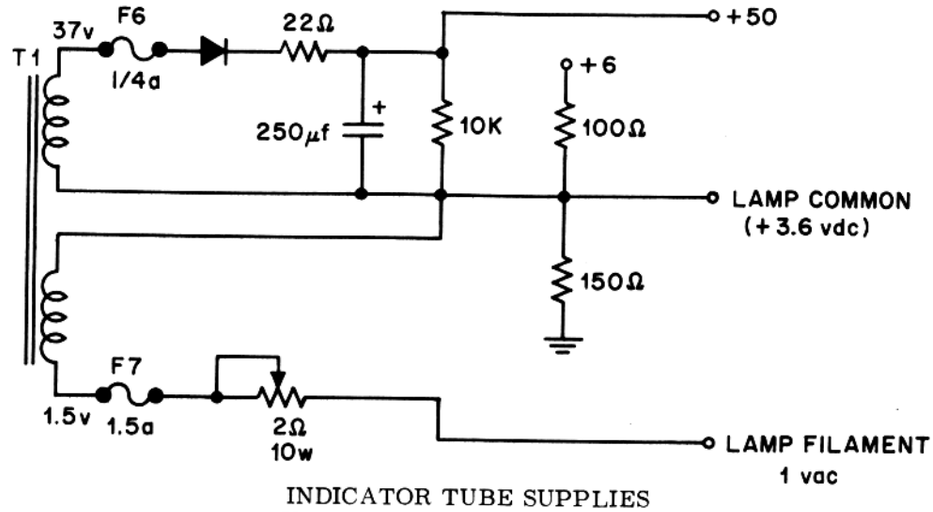
24-VOLT SUPPLIES
UNIVERSAL POWER SUPPLY

FIGURE 8-9

The three different resistors allow a choice of four voltages on the output, one of which should be within .8 volt of 24 volts. The K1 relay contact in the +24#2 line prevents the premature firing of the SCR's in the 2205 interface when the many voltages are building up immediately after machine turn-on. As noted earlier, it also provides +24 volts to the K2 relay circuit.

g. INDICATOR TUBE SUPPLIES. The indicator tubes in the test panel require three voltage levels: a relatively high anode supply voltage, a small filament supply, and a grid voltage slightly more negative than the cathode filament. +50 volts is provided for the anode supply, and 1 volt ac for the directly-heated cathode. In order to allow the grids (either +6 or 0 volts) to become more negative than the cathodes, the 1 volt ac filament supply is made to ride at a positive 3.6 volts dc.

The supplies are shown in Figure 8-10. The +6-volt supply is used as a reference for establishing the common at +3.6 volts dc. From this point a 37-volt winding of T1 is half-wave rectified and boosted to +50 volts by the 250 μf capacitor. The 10,000 ohm resistor is a bleeder for the capacitor, and the 22 ohm resistor prevents short-circuit conditions through the capacitor when the power is first applied.



INDICATOR TUBE SUPPLIES
UNIVERSAL POWER SUPPLY

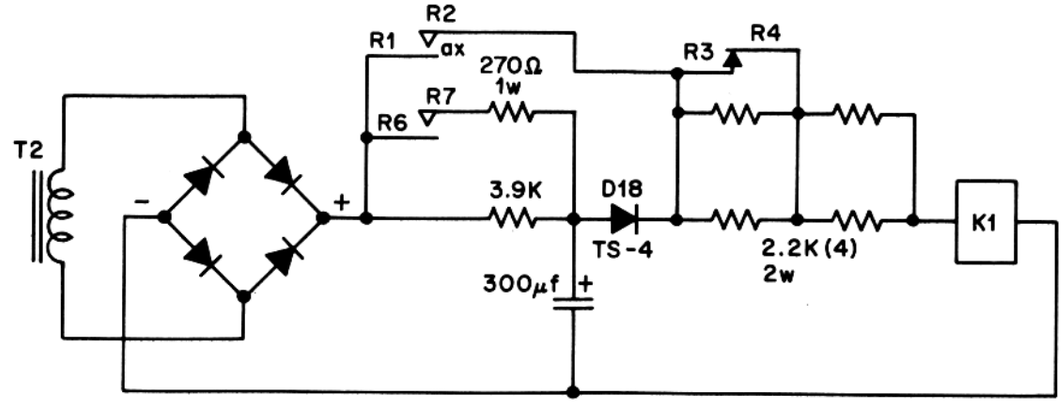
FIGURE 8-10

The filament supply is dropped to 1 volt by an adjustable 2 ohm resistor.

h. 2205 SUPPLY. The 2205 supply is simply an isolation transformer (T2) and a 4 amp fuse. However, since the primary of the transformer has three taps to allow for inputs higher or lower than 115 volts, the secondary will be held closer to 115 volts under these extreme conditions. Having the 2205 connected through this transformer also assures that it will have a ground common to the 5610.

The tape rewind motor in the desk is also connected to the secondary of the transformer.

i. K1 RELAY SUPPLY. The circuit which "picks" the K1 relay after one second of operation is also supplied from the secondary of T2. Refer to Figure 8-11. A bridge rectifier converts the 115 volts ac into pulsating dc. As soon as power is applied, current flows through the 3900 ohm resistor and the 300 μf capacitor to minus. The charge across the capacitor gradually builds up, and some current starts to flow through D18, K1 contacts R3 and R4, two of the 2200 ohm resistors in parallel, and the coil of K1. After approximately 1 second the capacitor has charged sufficiently to allow enough current to flow through the coil to "pick" the relay.



K1 RELAY CIRCUIT

FIGURE 8-11

As soon as K1 starts to pick, its *ax* contacts R1 and R2 close, providing a direct path (through R3 and R4 still closed) for the high current needed to fully energize the relay. After K1 picks fully, contacts R6 and R7 close, all but shorting out the 3900 ohm resistor. This puts the 300 μf capacitor almost directly across the supply, thus providing some degree of filtering. At the same time, contacts R3 and R4 open, inserting an extra 1100 ohms of resistance into the circuit to decrease the flow of current through the relay.

2. INTERIM POWER SUPPLY

The interim power supply is so named because it was available for use on early 5610's in the "interim", before the universal power supply was fully developed and tested.

The interim supply also employs two transformers, one for the logic supplies and an isolation transformer for the 2205 and the K1 relay. The interim transformers differ from those in the universal power supply in that they do not have primary taps for 220 volts: when a 5610 with the interim power supply is to be used on 220 volts, the entire system must be connected through a 2-to-1 stepdown transformer. The tape transport motor on machines equipped with the interim supply receives its power from the same circuit that powers the auxiliary output receptacles.

In addition to the 115 volts ac, the interim supply produces the following voltages:

+50 volts dc @ 10 ma, Unregulated
+24 volts dc @ 1.25 amp "
-24 volts dc @ .2 amp "
1 volt ac @ 1 amp "
+12 volts dc @ .5 amp, Regulated
-12 volts dc @ .2 amp "
+8 volts dc @ .75 amp "
+6 volts dc @ 4 amp "

The +8-volt supply, originally intended for use with the "piggyback" circuits, is not used. The 50-volt and 1-volt supplies are identical to those used in the universal power supply. The 6- and 12-volt circuits are quite different, and the 24-volt circuits are different because they supply the "raw" voltage to the 12-volt regulators in the interim supply.

The K1 relay circuit operates in exactly the same manner as the equivalent circuit in the universal supply: only a few component values are changed.

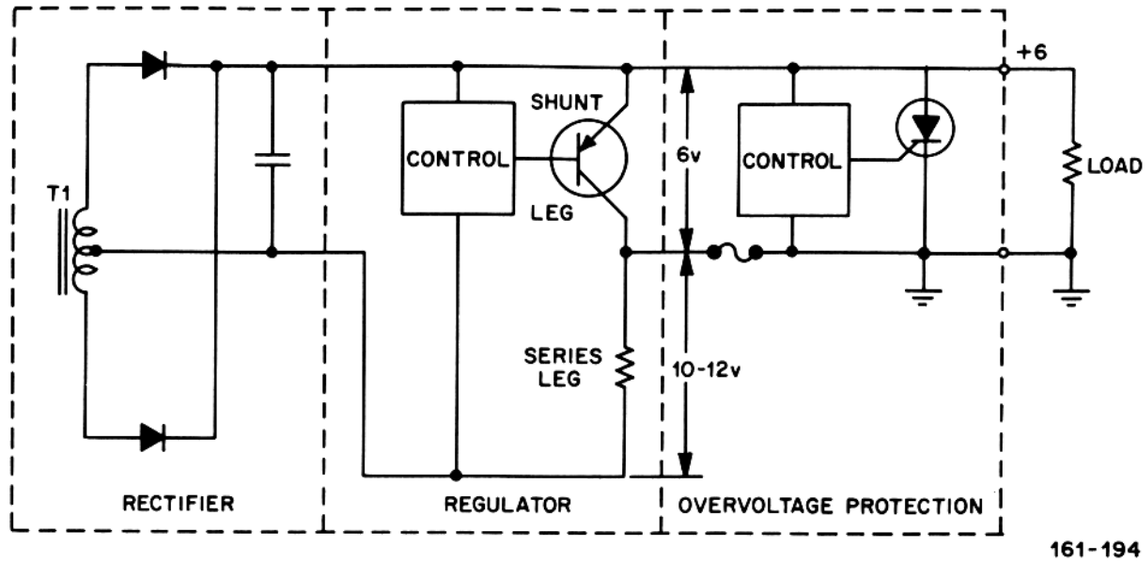
There is no battery option available for use with the interim supply.

a. +6-VOLT SUPPLY. A simplified schematic of the +6-volt portion of the interim power supply is shown in Figure 8-12. It consists mainly of a rectifier section, a regulator section, and an overvoltage protection section.

The rectifier section delivers 16-18 volts, filtered, to the regulator section. The regulator section drops 10-12 volts across its "series leg" while holding the voltage across the "shunt leg" at a constant 6 volts. The overvoltage protection section operates the same as that used in the universal power supply, turning on an SCR and blowing the fuse when the output voltage rises too far.

(1) General Operation. The rectifier section is made up of a center-tapped transformer and a full-wave rectifier with large capacitors for filters. The transformer secondary provides 13-14 volts rms, which the capacitors boost to 16-18 volts.

The regulator section consists mainly of two legs, a series leg and a shunt leg, and a circuit for controlling the shunt leg. The two legs form a voltage divider: the series leg is one part of the voltage divider and the shunt leg combines with the load to form the other part. 6 volts are dropped across the shunt/load part of the divider and the remainder of the supply is dropped across the series leg.



SIMPLIFIED SCHEMATIC
INTERIM +6-VOLT SUPPLY

FIGURE 8-12

If load current should change, shunt current changes in the opposite direction, keeping the total current the same and regulating the output voltage. If input voltage should change, shunt current again varies, changing the TOTAL current flow through the series leg, thus causing a different voltage drop across the series leg to make up for the change in input voltage.

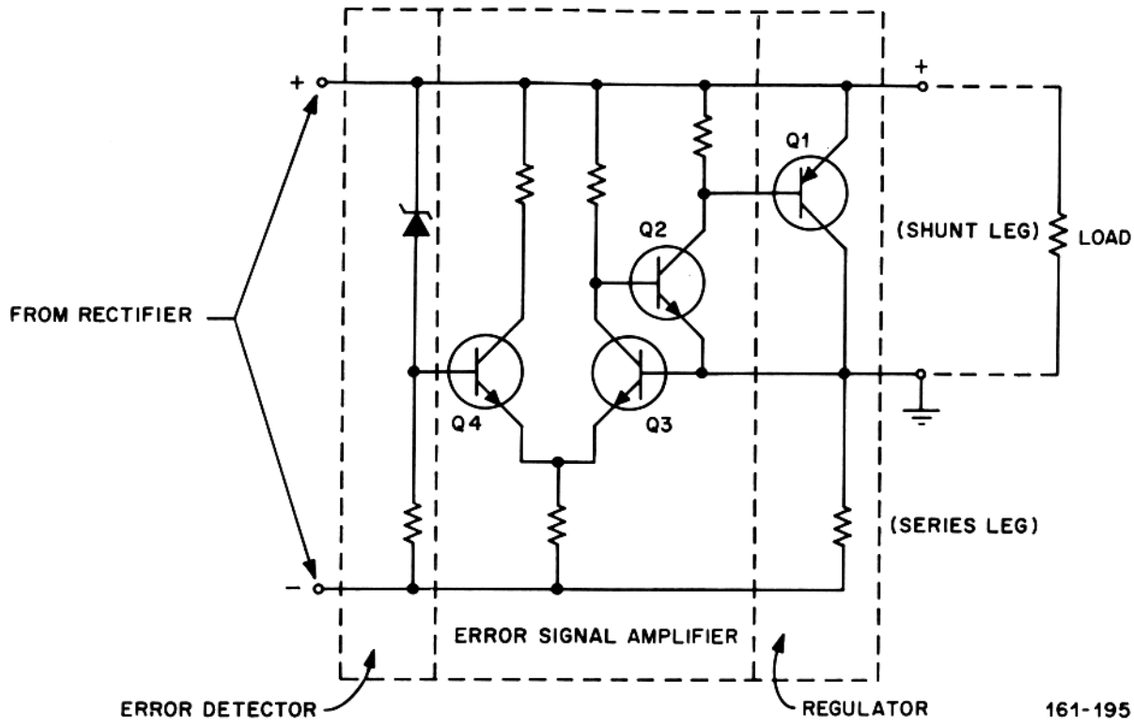
The same elements in the control circuit actually function in either of the preceding conditions (changes in load OR input). In other words, the control circuit monitors the entire 16-18 volts, and whenever a minute change occurs for any reason, it compensates by changing the current through the shunt leg.

The overvoltage protection circuit monitors the output voltage. If it should rise toward a value that might damage the components in the processor, the control circuit turns on the SCR. This effectively short circuits the output and blows the fuse. The SCR must turn on before the output voltage reaches 8.4 volts.

(2) Rectifier Section. As stated earlier, the rectifier section is simply a center-tapped transformer with a conventional full-wave rectifier and capacitor filter. This secondary winding of T1 is used ONLY for the +6-volt supply in the interim power supply, and not for a combination of supplies, as is the case in the universal supply.

(3) Regulator Section. The regulator section is made up of three parts, just as was the regulator section of the universal +6-volt supply. The error signal detector and error signal amplifier are quite similar to those used in the universal supply, but a shunt regulator is used in the interim supply, as opposed to the series regulator used in the universal supply.

A simplified schematic of the regulator section is shown in Figure 8-13. The error signal detector monitors the output and detects minute variations from +6 volts. The error signal amplifier boosts this signal to a value sufficient to control the shunt regulator, which in turn controls the total current flow in the +6-volt supply.



SIMPLIFIED SCHEMATIC
REGULATOR SECTION OF INTERIM +6-VOLT SUPPLY

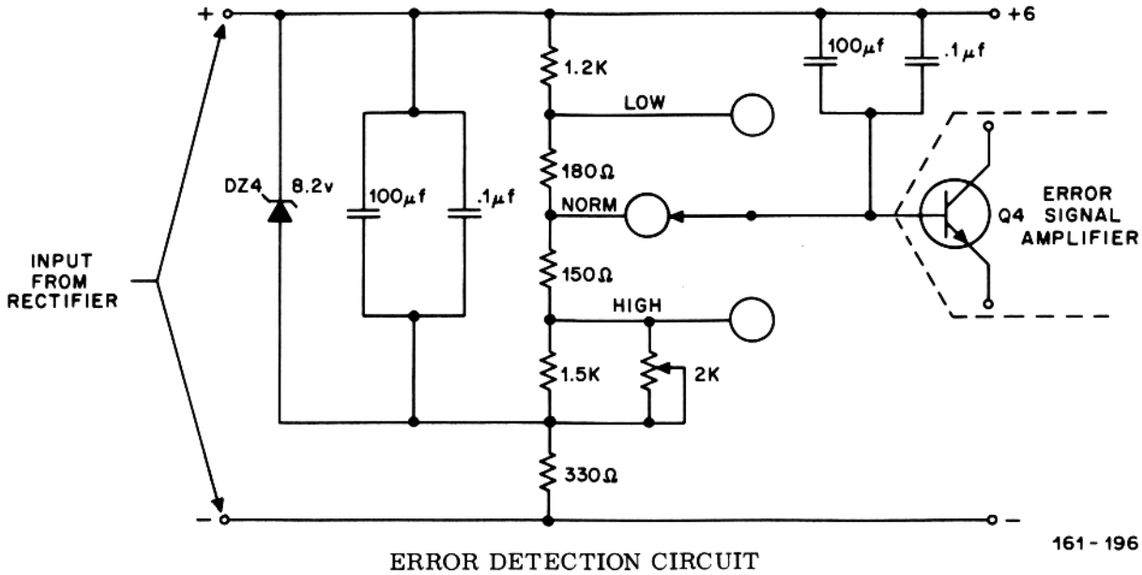
FIGURE 8-13

For example, assume the input voltage increases. Since the drop across the zener diode remains constant, the base of Q4 becomes more positive (with respect to "minus") than it was before, causing Q4 to conduct more heavily. The tendency of the Q4/Q3 emitter resistor to draw more current also tends to drive the Q3 emitter more positive, causing Q3 conduction to decrease proportionally. The Q3 collector voltage thus becomes more positive, causing Q2 to conduct more, driving the base of Q1 more negative. This turns Q1 on harder, causing more current flow through the series leg with an accompanying increase in the voltage drop across the series resistor. This increase in voltage drop is equal to the increase in supply voltage which initiated the change. The increase in series resistor current is made up entirely on the shunt leg, so load current is not affected.

An equal but opposite action occurs with a decrease in input voltage. Also, any change in load current which tends to increase or decrease the output voltage affects the circuit similarly, causing a corresponding change in Q1 current with an attendant stabilization of the output voltage. In this latter case, however, note that the change in current through Q1 does not affect the current through the series resistor, but simply compensates for the change in load current: the current through the series resistor remains constant.

In actual operation, changes in load and input can occur simultaneously, in which case the net "error" in the output is detected and amplified in the same way to cause the proper change in Q1 current.

The full schematic of the error detection circuit is shown in Figure 8-14. The junction of the 330 ohm and 1500 ohm resistors remains 8.2 volts more negative than the +6-volt output due to the action of DZ4. The three positions of the "HIGH-NORM-LOW" switch therefore remain at certain fixed levels in relation to the +6-volt output. Note, however, that any change in the +6-volt output will cause a corresponding change at the base of Q4 with respect to "minus". Thus, by monitoring the base of Q4 (with respect to minus), any "error" appearing on the +6-volt output may be immediately detected.

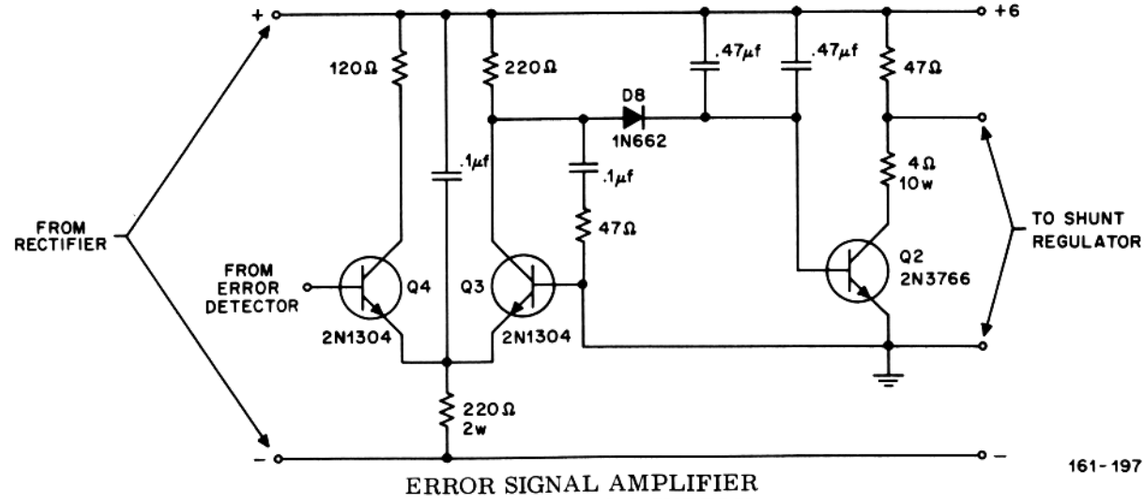


ERROR DETECTION CIRCUIT
INTERIM +6-VOLT SUPPLY

FIGURE 8-14

The 2000 ohm potentiometer varies the voltage drop across the lower leg of the voltage divider, thus providing an adjustable operating point for Q4 and Q3. This allows the output voltage to be adjusted to precisely +6 volts. The "HIGH-NORM-LOW" switch is again a service tool to be used in checking for marginal operation, as noted in the universal power supply discussion.

A complete schematic of the error signal amplifier is shown in Figure 8-15. Q4 and Q3 make up a differential amplifier which receives any error signal picked up by the error signal detector. As stated earlier, any change in Q4 current causes an opposite but proportional change in Q3 current. This varies the Q3 collector voltage, which is applied to the base of Q2. Q2 then provides the necessary amplification and phase inversion to correctly drive the shunt regulator transistor, Q1. Diode D8 is used only to provide the voltage drop necessary to properly bias Q2. The capacitors again prevent oscillation.

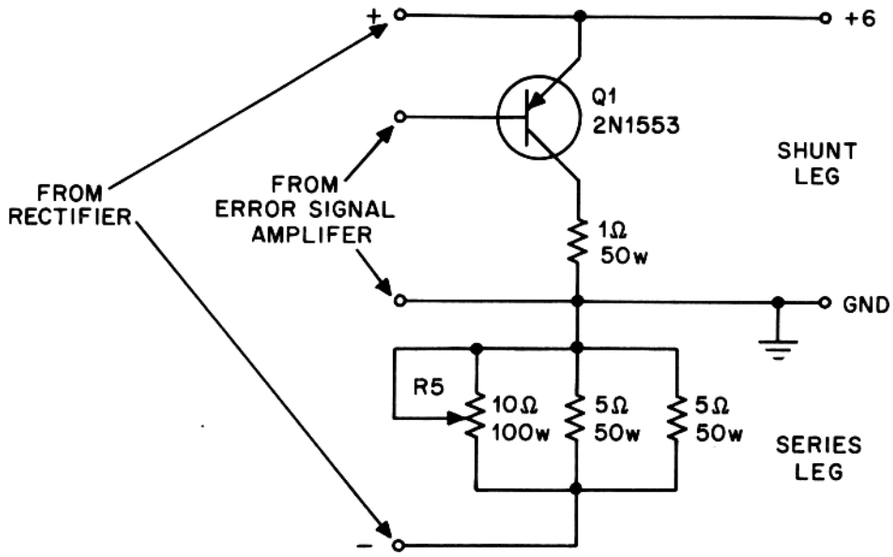


ERROR SIGNAL AMPLIFIER
INTERIM +6-VOLT SUPPLY

FIGURE 8-15

The differential amplifier is again used to minimize any tendency for drift to occur: any temperature effects will be felt by both transistors, and the effect on one will be canceled by the other.

The final regulator circuit is shown schematically in Figure 8-16. The 1 ohm resistor dissipates about half the power in the shunt leg: Q1 could handle it alone, but would then dissipate much more heat. Normal current through Q1 is about 2 amps. Initial adjustment is to apply a 4 amp load and then adjust R5 until 2 amps is flowing in the shunt circuit. The function of the regulator is, of course, to receive the amplified error signal from the previous stage and change the current flow through Q1 to compensate.



REGULATOR CIRCUIT
INTERIM +6-VOLT SUPPLY

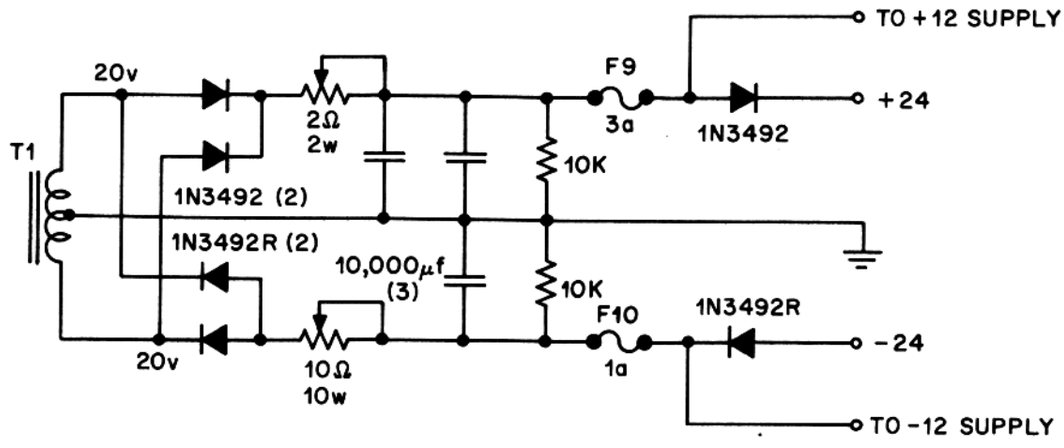
FIGURE 8-16

(4) Overvoltage Protection Circuit. The overvoltage protection used in the interim power supply is identical in concept to that used in the universal supply. If the output voltage rises too high, the SCR turns on and blows the fuse. In some power supplies the 1N662 diodes are shorted to ensure that the SCR turns on before 8.4 volts.

b. 24-VOLT SUPPLIES. The +24- and -24-volt supplies are very similar. They produce an unregulated voltage which is used both as the input to the 12-volt supplies and in the 2205 interface, and as a supply for the indicator lamps on the front of the processor.

Figure 8-17 illustrates the 24-volt supplies. A 40-volt winding on T1 with its center tap grounded provides 20 volts to each supply. Current is rectified by a conventional full-wave rectifier, and filtered by a large capacitor (two in the positive supply), which also serves to increase the voltage to 24 volts. Variable resistors allow for voltage adjustment, and diodes on the output isolate the 24-volt supplies from the 12-volt supplies. The 10,000 ohm resistors bleed off the charge on the capacitors when power is removed.

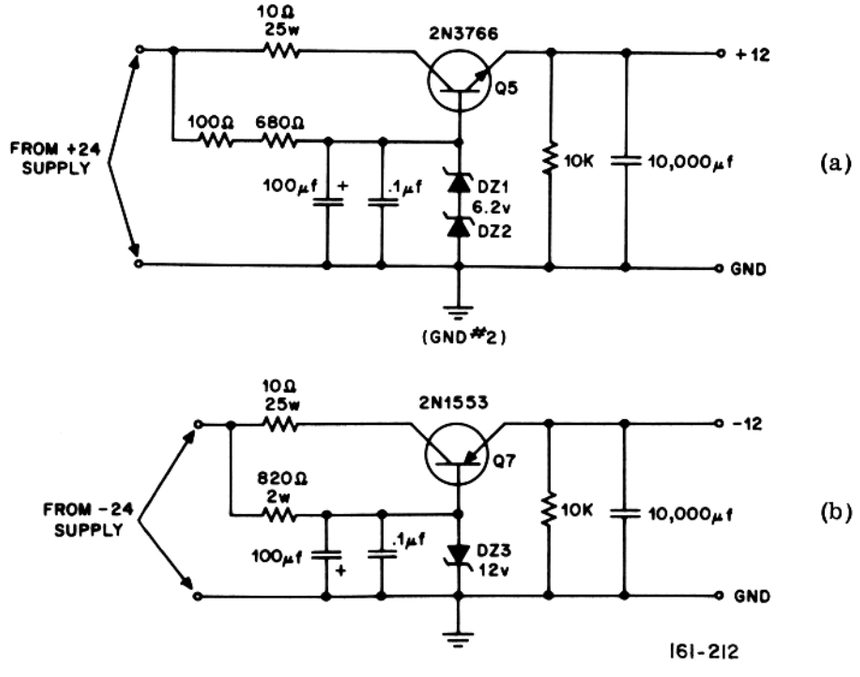
c. 12-VOLT SUPPLIES. The 12-volt supplies are also quite similar to each other. Each is fed by its respective 24-volt supply, and each employs a series regulator transistor to hold the voltage at a constant 12 volts. The 12-volt supplies are used for the delay line read/write amplifiers and other special electronic circuits.



24-VOLT SUPPLIES
INTERIM POWER SUPPLY

FIGURE 8-17

Refer to Figure 8-18 (a), which shows the +12-volt supply. The two zener diodes hold the base of Q5 at a constant 12.4 volts, regardless of changes in the load. The 10 ohm resistor serves to drop part of the 24-volt supply and to limit total current flow. All of the capacitors suppress oscillations, and the 10,000 ohm resistor serves as a bleeder for the 10,000 μf capacitor. Two 6.2-volt zener diodes are used in place of one 12-volt diode for economy.

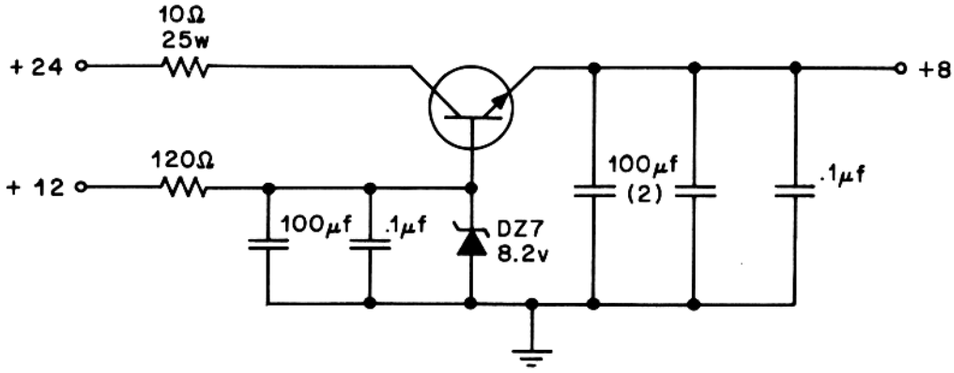


12-VOLT SUPPLIES
INTERIM POWER SUPPLY

FIGURE 8-18

Figure 8-18 (b) shows the -12-volt supply. Because of the polarity change, a PNP is used for the series regulator transistor and the single zener diode is placed in the base circuit "pointing" the opposite way. The difference in current required at the output and the fact that a different series regulator transistor is used account for the different resistance in the base circuit and the single 12-volt zener diode.

d. 8-VOLT SUPPLY. As noted earlier, the +8-volt supply, though designed to provide bias for the "piggybacks", is not used. The circuit is shown in Figure 8-19. A voltage divider from +12 volts to ground clamps the base of Q6 at +8.2 volts due to zener diode action. The +24-volt supply provides power to keep Q6 in its active region, holding the output at 8 volts. The capacitors help to absorb "spikes" and prevent oscillations.



8-VOLT SUPPLY
INTERIM POWER SUPPLY

FIGURE 8-19

e. 2205 SUPPLY AND K1 RELAY CIRCUIT. These last two circuits are all but identical to their equivalent circuits in the universal power supply. 115 volts ac is taken from the secondary of isolation transformer T2 to provide power for both circuits. A 4 amp fuse also protects both circuits. The K1 relay circuit operates exactly the same as that used in the universal supply, even though some components are different.

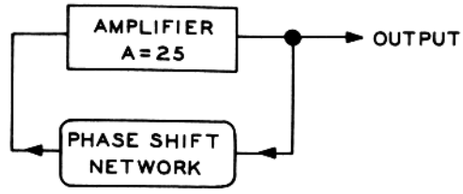
B. OSCILLATORS

An oscillator is an electronic circuit which generates a repetitive waveform output whose frequency is determined by the components. This repetitive output can be in the form of square waves, sawtooth waves, or sine waves. The waveforms can be either symmetrical or non-symmetrical.

1. BASIC OSCILLATOR

An amplifier, in the usual sense of the word, is a circuit in which a very small input signal is changed to a comparatively large output signal. This output signal is usually the same in frequency and similar in waveform, but inverted 180° with respect to the input.

An amplifier output is much larger than its input, so a portion of the output may be fed back to the input, eliminating the need for any additional input; the greater portion of the output is still available for other uses. However, for the output of a single-stage amplifier to be used as its own input, it must first be shifted by 180° to be in the proper phase.



BLOCK DIAGRAM
SIMPLE OSCILLATOR

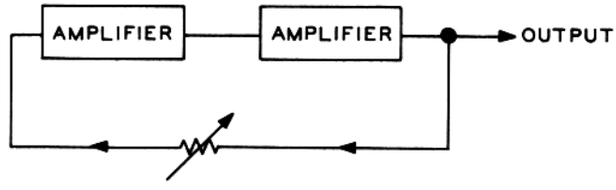
FIGURE 8-20

In the simplified oscillator circuit shown in Figure 8-20, the gain of the amplifier is 25. Only 1/25th of the output must be fed back to the input to keep the circuit active with no other input. This leaves 96% of the output still available. The phase shift network in the feedback circuit provides the 180° phase shift necessary to compensate for the inversion between the input and output of the amplifier. The phase shift network also provides the necessary feedback division to prevent overdriving the input. Excessive feedback distorts the output waveshape to an undesirable degree, and might also make the oscillator intermittent in operation.

If the phase shift network is made frequency sensitive, then only one frequency can get through the network with the proper phase and amplitude to act as an input signal. The frequency of the output, then, is determined by the characteristics of the network. If the characteristics of the network are changed in a controlled manner, the output frequency can then be controlled.

2. TWO-STAGE OSCILLATOR

If one stage of amplification gives 180° of phase shift and a second stage gives another 180°, then the total phase shift from the input of the first stage to the output of the second stage is 360°. This means that the output of the second stage is in the proper phase to act as an input to the first stage with no phase shift network needed. Only a means of controlling the amount of feedback to the first stage is necessary.



TWO-STAGE OSCILLATOR

FIGURE 8-21

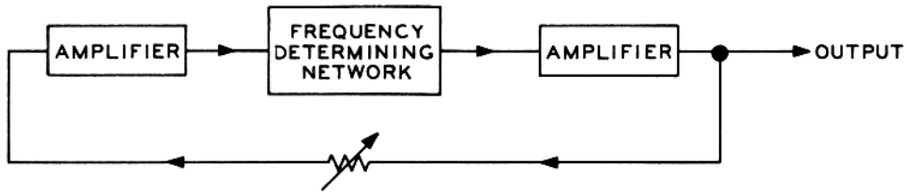
In the simplified circuit shown in Figure 8-21, two amplifiers are shown cascaded with the feedback taken from the output of the second stage. No phase shift network is needed, but the variable resistance in the feedback circuit is necessary to adjust the drive to the first stage. Since there are two stages of amplification, only a very, very small portion of the output is needed for feedback.

The circuit shown in Figure 8-21 has the disadvantage of not being frequency controllable because there is no phase shifting network whose frequency characteristics can be changed. Thus, the circuit will oscillate but at a random and somewhat variable frequency.

3. CONTROLLED TWO-STAGE OSCILLATOR

To establish and control the output frequency of a two-stage oscillator, a frequency determining network can be used as a coupling device between the two stages. Then the output frequency of the second stage is determined by the coupling, and since this is also acting as the input, the circuit assumes the frequency of the coupling network.

In the simplified cascaded two-stage amplifier shown in Figure 8-22, a frequency sensitive circuit has been inserted between the two stages. Now, as in the phase shifting network, only one frequency can get through the coupling circuit and act as the input to the second stage. This frequency, amplified by the second stage, then furnishes the feedback to the first stage, and the circuit oscillates at this frequency.



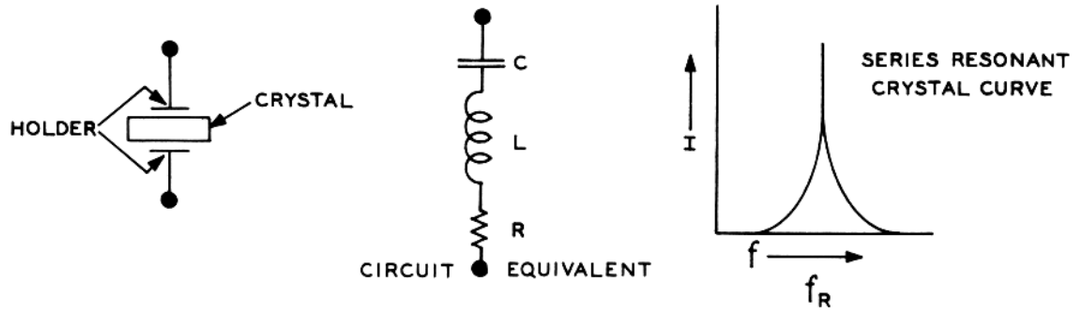
CONTROLLED-FREQUENCY OSCILLATOR

FIGURE 8-22

The coupling circuit usually is either a series-resonant or a parallel-resonant circuit. In a vacuum tube circuit, where the amplifiers are voltage controlled, a parallel resonant circuit is used. When using transistors, which are current controlled, a series-resonant circuit is employed.

4. BASIC CRYSTAL ACTION

A crystal is a piezo-electric device. This means that a voltage applied across the crystal stresses the crystal mechanically. When the voltage is removed, the release of the crystal stress creates an opposite voltage across the crystal. In effect, the crystal stores the electrical potential as a mechanical stress, then releases the mechanical stress as electrical energy. This is exactly analogous to resonant circuit action. See Figure 8-23.



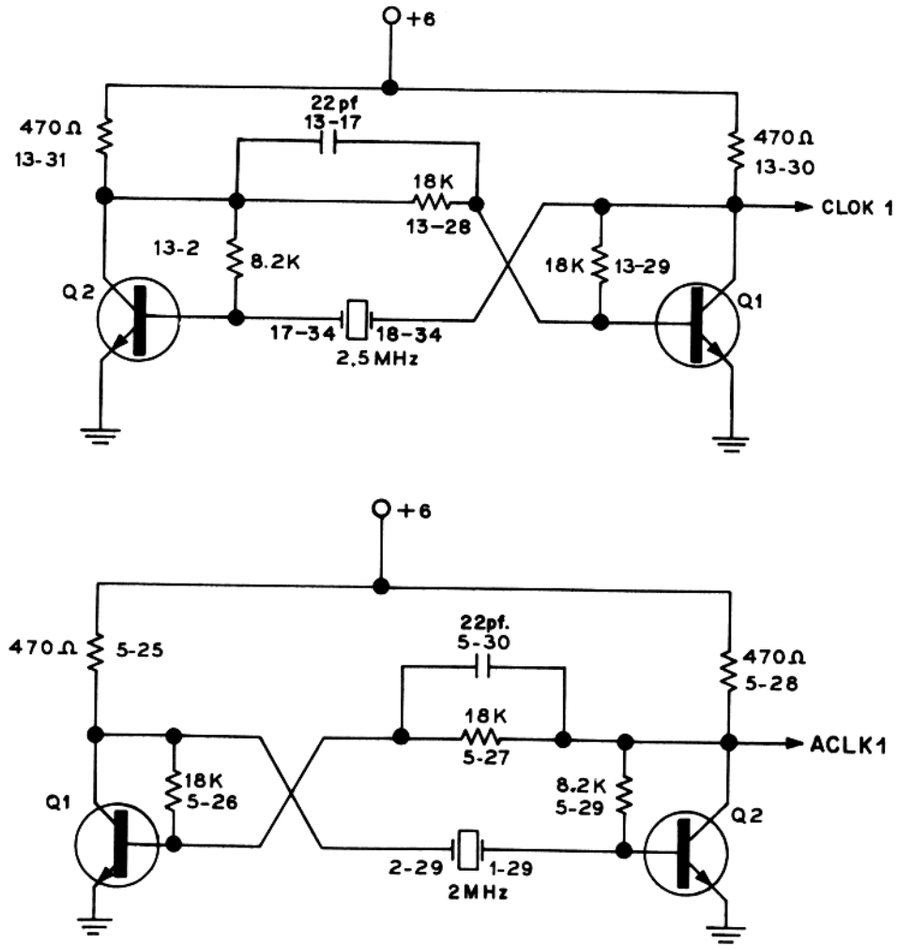
BASIC CRYSTAL ACTION

FIGURE 8-23

However, as compared to a resonant circuit composed of inductance and capacitance, a crystal is a very precise device that has only one frequency. At this frequency, and this frequency only, will the crystal resonate. This makes it very suitable for use in an oscillator circuit that is to have only one very precise output frequency.

5. 5610 CRYSTAL OSCILLATORS

The 5610 crystal oscillators operate at 2.5 MHz and 2 MHz. They are simple two-transistor amplifier circuits with the coupling between the amplifiers being precise frequency crystals. The two oscillator circuits are shown in Figure 8-24.

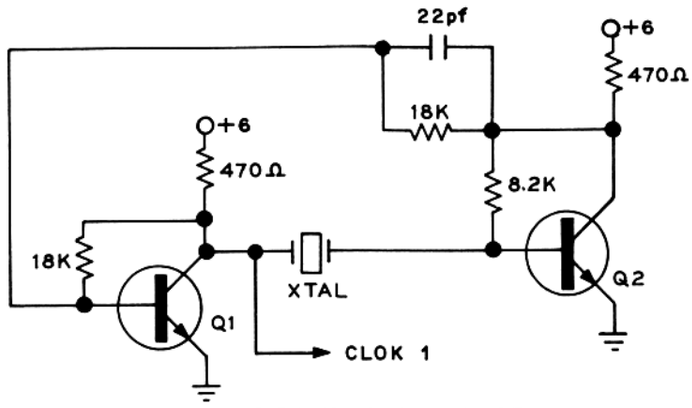


5610 OSCILLATORS

FIGURE 8-24

TP-161SL

The 2.5 MHz oscillator is redrawn in Figure 8-25 to better show the cascaded amplifier action.



2.5 MHz OSCILLATOR

FIGURE 8-25

Feedback from the output to the input is via the 18,000 ohm resistor which serves to limit the feedback current to the proper value. When the circuit is first turned on by the application of +6 volts, both transistors turn on. The voltage change across the collector resistor of Q1 appears at the crystal. A sudden change of voltage is essentially the same as the sudden rise at the leading edge of a square wave, and contains many, many frequencies. At only one of these frequencies can the crystal pass current to the base of Q2, and this is the resonant frequency of the crystal.

This frequency then becomes the input to Q2 and affects its collector current. This collector change is fed back to the input of Q1 via the 18,000 ohm resistor and, in turn, affects the collector current of Q1. This change, then, is fed back to the crystal to further affect Q2. In a very, very short time, measured in nanoseconds, the circuit oscillates at its resonant frequency. The output is taken from the collector of Q2, and is essentially a sine-wave at the frequency of the crystal, 2.5 MHz. The output resembles a sine wave because the crystal is essentially an inductance and capacitance in a series resonant circuit, and has less impedance to a sine wave than to any other waveform.

The 22 pf capacitor increases the feedback to the base of Q1, resulting in a more symmetrical output waveform.

C. HEXINVERTERS

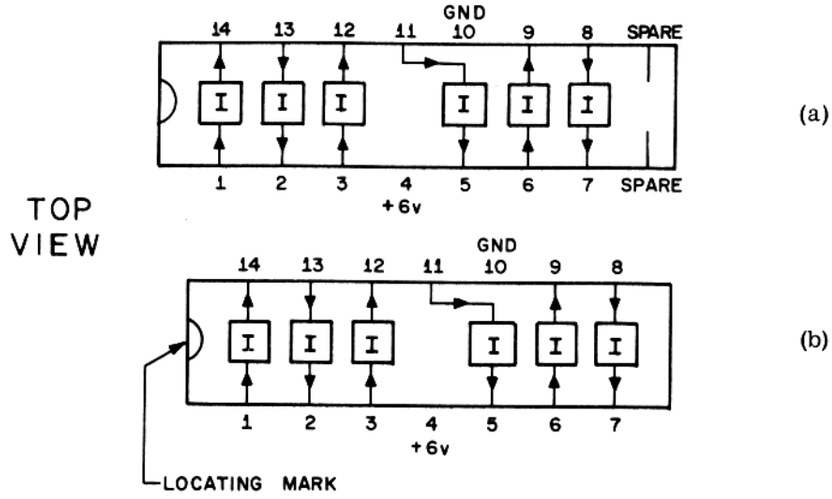
There are two kinds of hexinverters used in the 5610, the "I" type and the "H" type. These two devices are very similar in many respects, so they will first be discussed together and their differences will be pointed out later.

1. CONSTRUCTION

All hexinverters are composed of six identical inverters sealed in an epoxy package, hence the name "hexinverter", or "hex pack". All six inverters are contained in one monolithic silicon integrated circuit chip. This single chip has been fabricated to include all of the circuit elements needed for the six inverters. In the case of the "I" type hexinverter, this includes 36 resistors, 24 NPN transistors, six diodes, and the necessary interconnecting circuits. Each chip has 14 wires connected: an input and an output for each inverter, and a ground terminal and VCC terminal common to all six inverters. None of the circuit elements of any of the inverters can be seen or measured, since all are part of the same chip, and the chip is encapsulated. To the technician, this means that if any one of the elements of any one of the six inverters fails, the entire package must be replaced. This disadvantage is greatly outweighed, however, by the manufacturing and engineering advantages of these integrated

circuits: one small package can replace six entire inverter circuits, at a great saving in component space, interconnecting wiring space, and manufacturing time. These inverters also exhibit the desirable characteristics of high input impedance, low output impedance (when either on or off), and very short rise and fall times.

Figure 8-26 shows the hexinverter pin numbering. Figure 8-26 (a) illustrates the system used for hex packs contained in some earlier 5610's and Figure 8-26 (b) illustrates current usage. The hex packs represented by Figure 8-26 (a) resemble a pink rubber eraser, whereas those currently used are black in color. In any event, pin numbers 1 and 14 will be in the positions shown, closest to the locating mark.



HEX PACK TERMINAL NUMBERING

FIGURE 8-26

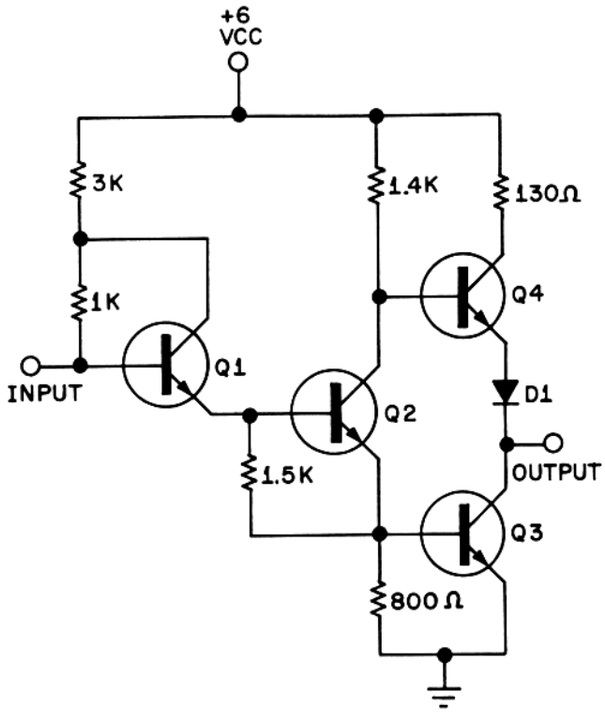
Notice also in Figure 8-26 that each of the six inverters is shown connected between two terminals. These are the input and output connections for the inverters. These numbers, along with a letter designating the hex pack of which the inverter is a part, are used to denote the location of the individual inverter on the logic drawings. For example, the location "D6-9" would represent the inverter having an input on pin 6 and an output on pin 9, located in the "D" hex pack on that particular card. (More information concerning component locations may be found in Section 1, General Information.) Pin 10 provides ground and pin 4 provides +6 volts to all six inverters.

2. OPERATION

An "I" inverter is represented schematically in Figure 8-27. Q1 acts as an emitter follower that furnishes base current for Q2. Q2 may be considered as a "phase splitter": as the emitter voltage becomes more positive, the collector voltage becomes more negative, and vice-versa. Q2 can also be thought of as a combination emitter follower-inverter. The two outputs of Q2 are applied to the bases of Q3 and Q4. Q3 acts somewhat like a conventional inverter amplifier and Q4 acts somewhat like a conventional emitter follower. The primary purpose of Q4 is to act as an active pull-up circuit to improve the rise time of the output. It also causes a low impedance to be seen by the circuits connected to its output when the inverter is in its "off", or "false" output state. D1 is necessary to establish the correct bias for Q4 when Q3 is "on". All semiconductors are of silicon, since that is the material from which the entire chip is made.

Since none of the internal connections of the inverter can be reached for measurements, the voltages at these points are of little interest. The technician need concern himself only with the input and the output. If the output reacts correctly to the input voltages, the inverter is good. If not, the entire hex pack must be replaced.

Referring to Figure 8-27, note that if the input is open, the inverter turns on, yielding a signal near ground. With a suffuciently positive voltage at the base of Q1, emitter-collector current flows and the emitter is raised to some positive potential. This places a positive potential at the base of Q2, turning it on. As the base of Q2 becomes more positive, the emitter follows it and applies a more positive potential to the base of Q3. This turns Q3 on, causing its collector (the inverter output) to go to nearly ground. At this time Q4 is virtually off, since its base and emitter are at practically the same voltage: the base voltage is determined by the voltage drops across the emitter-base junction of Q3 and the emitter-collector junction of Q2. Its emitter voltage is determined by the drops across the emitter-collector junction of Q3 and the D1 junction. These drops should theoretically be equal, and in practice they are probably close enough so that it can be safely assumed that Q4 is off.



EQUIVALENT CIRCUIT SCHEMATIC
"I" INVERTER

FIGURE 8-27

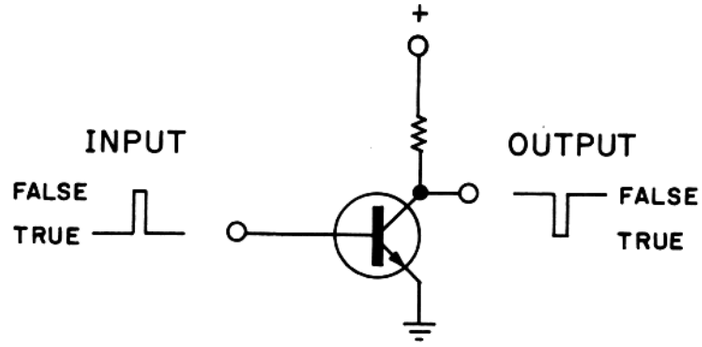
With the inverter "on" in this fashion, its input (the base of Q1) is clamped at about +2.5 volts, the exact voltage being determined by the voltage drops across the emitter-base junctions of Q1, Q2, and Q3. In the 5610, all inverter inputs are applied through isolating diodes, and when the inverter is "on" and its input is clamped to about +2.5 volts, the input diodes will be reverse biased. Thus an "on" inverter acts as though its inputs were completely open.

If the input now changes to true, approaching 0 volts, the output becomes false, rising toward +6 volts. With the inverter input near 0 volts, the base-emitter junction of Q1 is reverse biased and Q1 is cut off. With no Q1 emitter current there can be no Q2 base current, so Q2 is also cut off. With Q1 and Q2 cut off, Q3 must also be "off".

It is during this transition, while Q3 is turning off, that the real function of Q4 occurs. Since a transistor can turn on faster than it can turn off, Q4 is turned on to provide a rapid rise in output voltage while Q3 is turning off. During this transition, Q1, Q2, and Q3 are all going "on-to-off". As Q2 starts to turn off, its collector goes more positive, driving the Q4 base more positive. This starts to turn Q4 on, its emitter (at this instant) being returned to ground

through Q3. Q4 actually turns on hard, providing a high positive output, until Q3 is completely turned off and the output diode is reverse biased. When this stage is reached, Q4 emitter current is limited to that provided by leakage through Q3 and leakage from the output circuit. Thus Q4 provides a very rapid rise in the output waveform, much faster than could be obtained by merely turning off Q3.

Basically, this inverter acts the same as a regular inverter built up from individual components, even though it contains more circuit elements than a "regular" inverter. When the input is true, or at ground, the inverter is "off" and the output is false, approaching +6 volts. When the input is false, the inverter is "on" and its output is true, near ground. The integrated circuit inverter reacts to input voltages in the same way as a conventional inverter (Figure 8-28).



EQUIVALENT INVERTER

FIGURE 8-28

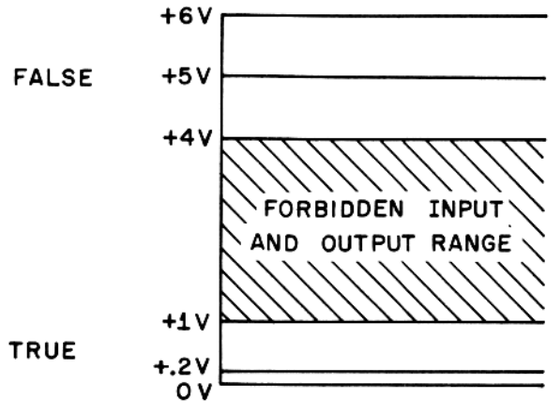
The input voltages are such that the inverters are either fully on or fully off. They are not designed to operate as linear amplifiers, and they become very unstable if the input places them in their active regions. The only time the inverters will be in their active regions is during the extremely short time of the transition from off to on or from on to off. The rise and fall times of output signals are normally on the order of 10-20 nsec (nanoseconds).

The equivalent circuit in Figure 8-28 is shown receiving a positive going input that goes from a level called "true" to a level called "false" and back again. No absolute values are given for the two levels because they may vary with different inverters. The true level in the 5610 will normally be slightly positive with respect to ground, but not enough to cause the NP emitter-base junctions of all three transistors (Q1, Q2, Q3) to be forward biased. With a true input, these junctions, shown as one emitter-base junction in the equivalent inverter of Figure 8-28, will still be reverse biased and the transistors will remain in an off condition. The false input level for the 5610 inverters will also be a positive voltage, but of sufficient amplitude to ensure that the emitter-base junctions of Q1, Q2, and Q3 are all forward-biased, causing all three transistors to be on.

When the input level to the equivalent inverter in Figure 8-28 is true, the output level will be false. The exact value of this false level can vary. It will not be +6 volts due to the voltage drop across D1, Q4, and the 130 ohm resistor in the collector circuit of Q4 (refer to Figure 8-27), which are represented by one equivalent collector resistor in Figure 8-28, This is the "pull-up" circuit mentioned previously, and even though it represents a rather low impedance, it will drop the output voltage below +6 volts when the inverter is off. The actual output voltage is dependent upon the impedance of the circuit to which the inverter output is connected, and the voltage which supplies this circuit.

When the input level is false, the output level is true. Again an exact value of voltage cannot be assigned; it can only be said that the level is near ground. The actual voltage at the output is determined primarily by the emitter-collector drop of Q3. (Q4, the pull-up transistor, will be nearly off at this time and represent a very high collector impedance for Q3.) The very low impedance of Q3 will hold the output voltage very close to ground.

While exact values of input and output levels cannot be stated, certain values may be specified that will determine whether a voltage is a valid true or false signal. In dealing with hexinverters in the 5610, a valid false signal must be positive enough to ensure that the inverter is held "off" and not allowed to move into the active region. The voltage levels presented here are considered acceptable for these inverters. Input or output voltages that do not fall within the limits specified here and shown in Figure 8-29 should be considered invalid; the cause(s) of such signals should be investigated. Invalid voltage levels of this type can easily cause errors, some of which can be intermittent.



INVERTER INPUT/OUTPUT VOLTAGES

FIGURE 8-29

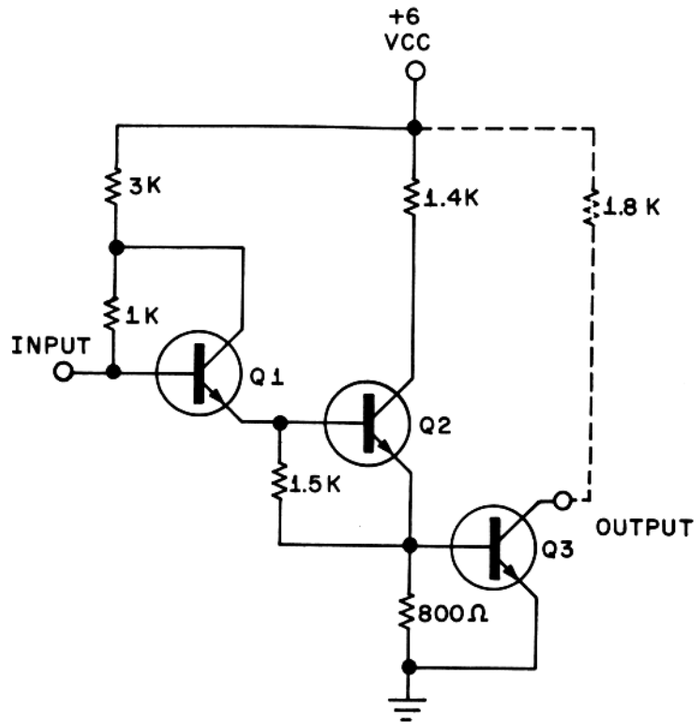
Figure 8-29 graphically depicts the more important true and false logic levels associated with 5610 hexinverters. The line at +5 volts represents what may be termed a typical false signal. Most of the inverters in the 5610 will yield an output of about +5 volts when they are turned "off" by their input. False outputs of other values between +4 volts and +6 volts are also normal and acceptable. The design minimum output of any 5610 hexinverter is +4 volts. Having all false signals at +4 volts and above provides greater noise immunity than would be available with lower voltage signals. Also "ac" gates (sequential AND gates) may introduce intermittent errors when false signals fall in the "forbidden input/output range".

The previous discussion may also be used as a basis for judging whether an input level to an inverter is valid or not. A typical inverter input signal will be in the range from +4 volts to +6 volts. Input levels below +4 volts may not be positive enough to reliably hold the emitter-base junctions of the inverter transistors forward biased under all noise conditions: noise "spikes" could turn the inverter "off". In addition, temperature variations could cause the inverter to remain "off" or go into its active region, in which case the output would be uncertain and invalid. In such cases the cause of these voltage levels should be investigated. (Note that false inputs to an inverter must be measured at the cathode of the input diode, since the anode will be clamped to a level determined by the inverter when it turns "on".)

A typical true signal is shown in Figure 8-29 as being about +.2 volt. True outputs closer to ground than this may be encountered and are normal. The design maximum true output of an inverter in the 5610 is +1 volt. If an "on" inverter has an output voltage higher than +1 volt, the cause should be investigated. If a true input to an inverter (measured at the anode of the input diode) is found at +1.5 volts or higher, the cause should be investigated.

3. "H" TYPE INVERTER

The "H" inverter used in the 5610 is very similar to the "I" type. The schematic in Figure 8-30 shows that the only difference is the elimination of the "pull-up" circuit of the "I" inverter. "H" inverters require an extermal collector load resistor (usually 1800 ohms) for Q3.

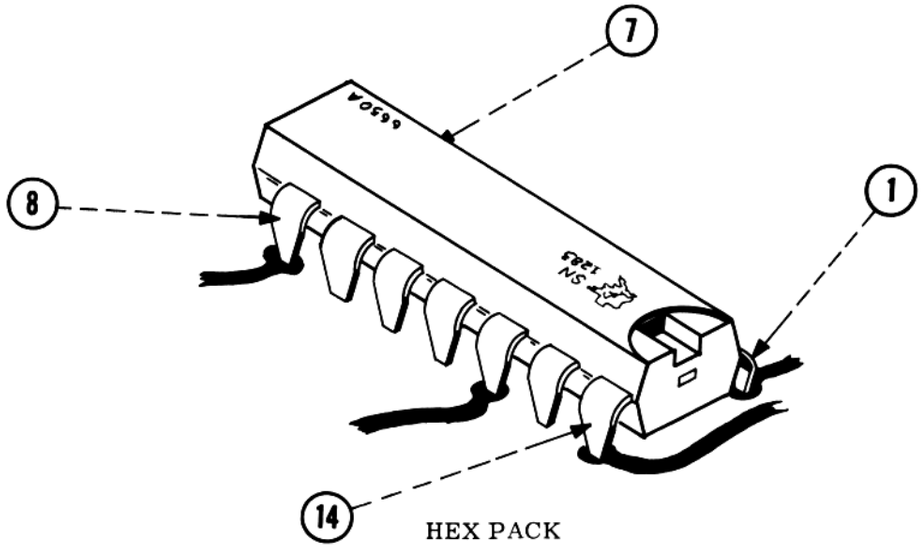


SCHEMATIC "H" TYPE INVERTER

FIGURE 8-30

Because the transistor pull-up circuit has been eliminated, the outputs of two or more "H" inverters may be joined directly together, whereas the outputs of "I" inverters must be isolated by diodes.

The physical construction of the "H" hexinverter is identical to that of the "I" hexinverter: there are six inverters in each module, and the pin numbering is exactly the same. Visually, the hex packs appear identical. They are identified by a number: "H" hex packs carry the number SN-1283, and "I" hex packs carry the number SN-1284.



HEX PACK

FIGURE 8-31

In the 5610, "H" inverters are used when the same signal is to be developed through a number of different gates, eliminating the need for additional diodes and double "I" inverters. One external load resistor is sufficient for any number of "H" inverters with their outputs connected.

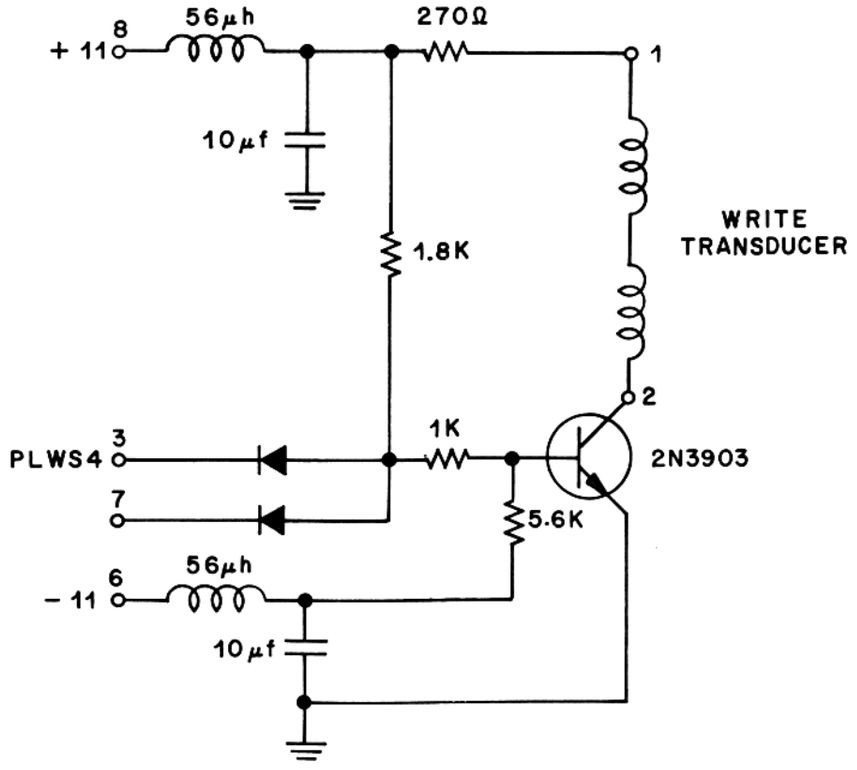
There are some cards in the 5610 with circuit requirements calling for the use of perhaps three or four "H" inverters. In this case, the other "H" inverters in the hex pack may be used in place of "I" inverters, but only when the signal being developed is "less critical", for the "H" inverter has a slightly longer propagation delay than the "I" inverter.

D. READ/WRITE AMPLIFIERS

The read and write amplifiers are powered by the + and -11-volt power supplies. As stated earlier, these two supplies are dependent upon each other so that a change in one causes a corresponding change in the other. Thus any change in either of the two supplies has a negligible effect on the read/write amplifiers.

1. WRITE AMPLIFIER

The write amplifier employs a single transistor switch to open and close the write transducer circuit. When the transistor is on, current flows through the transducer, putting a "twist" in the delay line. When the transistor shuts off, current through the transducer stops and the delay line "springs" back to normal. Refer to the schematic of the program line write amplifier in Figure 8-32. (The other write amplifiers are identical. However, the arithmetic line employs only one transducer coil.) When PLWS4 is true, the input diode is forward biased and the base of the transistor is clamped slightly negative. With the transistor off, no current flows through the transducer coil.



WRITE AMPLIFIER
FIGURE 8-32

TP-161SL

If PLWS4 should go false, applying +6 volts to the input, the anode of the input diode tends to go toward +7 volts. Before this value can be reached, however, the transistor turns on and its base becomes clamped at some slightly positive value. The voltage divider made up of the 1800 ohm and 1000 ohm resistors brings the junction of these two resistors up to around +4 volts, reverse-biasing the input diode and effectively taking the input out of the circuit. With the transistor on, current flows through the transducer (limited to a safe value by the 270 ohm resistor), thus putting a "twist" into the delay line.

When PLWS4 again goes true, the input diode again becomes forward biased, cutting off the transistor and stopping current flow through the transducer. When the current flow stops, the magnetic field built up around the coils collapses, and the resultant "inductive kick" produces a high positive "spike" at the collector of the transistor.

Either input may be used. Note, however, that if both inputs are used, both must be positive at the same time in order to write a pulse onto the line: if either one is at ground, the amplifier is disabled.

There is no adjustment to the write amplifier: if a malfunction occurs, the only method of repair is to replace the defective component(s).

2. READ AMPLIFIER

The read amplifier is very similar to the "single-ended linear pulse amplifier" described in Lesson 23 of the "FRIDEN Electronic Course". Anyone familiar with the amplifier described therein will have no trouble understanding the operation of the 5610 read amplifier, even though there are a few variations. For convenience during this discussion, a schematic of the read amplifier is included at the end of this section, in Figure 8-55.

The first three transistors receive the signal from the coil of the read transducer and amplify it linearly. The last two transistors act as a threshold detector and clipper, rejecting low amplitude signals (noise) and squaring up the output so that it may be used by the logic circuits.

Q1 is biased in the active region, with its emitter returned to +11 volts, its collector returned to -11 volts, and its base virtually at ground due to the low dc resistance of the transducer coil. This puts its collector at a value negative enough to bias Q2 in its active region. Thus the two transistors function as inverter amplifiers in series, and would exhibit a very high gain were it not for the negative feedback from the collector of Q2 to the emitter of Q1, through the 4700 ohm resistor and the 5000 ohm thermistor.

Q3 is also biased in its active region, and serves to further amplify and invert the signal. Q4 is biased off: the 10,000 ohm/27,000 ohm voltage divider puts its base at about +5 volts, and its emitter rides at about +.8 volt because Q5 is biased into saturation.

Any positive excursion at the collector of Q3 is transmitted to the base of Q5 and serves only to further bias it in the forward direction. A negative transition at the collector of Q3, however, is transmitted to the base of Q5 by the emitter follower action of Q4, and serves to cut off Q5. As Q5 cuts off, its collector, the read amplifier output, makes the transition from ground to +11 volts.

The output signal from the transducer, usually about 35-40 mv, is not normally observed on the oscilloscope because it is not readily discernable from "noise" and because the loading introduced by the scope probe affects the output signal somewhat. The same is true, although to a lesser extent, of the output of Q1 and Q2, so a test point is provided at the output of Q3. The signal at this test point is used in making all read amplifier adjustments (outlined in Service Letter #41) because this signal is still not chopped when the amplifier is properly adjusted, thus making the optimum adjustment possible.

The gain of the read amplifier is controlled by adjusting the 2000 ohm potentiometer, thus varying the unbypassed emitter resistance in the first stage (Q1). The output of the first stage (Q1 collector) is coupled directly to the input of the second stage for further amplification. The second stage output is capacitively coupled to the third stage, but a portion of the second stage output is fed back to the first stage to decrease the overall gain of both stages, thus reducing distortion and minimizing the effects of varying input signal amplitides. Note that the signal fed back from the second stage is IN PHASE with the input signal, but by applying it to the emitter of Q1 instead of the base, its effect is negative instead of positive.

The negative feedback tends to make up for differences in input signal strength: if an input signal is smaller than normal, the negative feedback is less, and attenuation of the signal is less. If the input is stronger than normal, feedback is greater and the final output signal is attenuated more. Thus varying signal strengths do appear at the input to Q3, but they are not as great in proportion as the differences at the input to Q1.

The amount of feedback also varies with temperature, due to the thermistor in the feedback circuit. As temperature rises, the resistance of the thermistor decreases, allowing the amount of negative feedback to increase. For example, assume Q1 starts to conduct more heavily due to an increase in ambient temperature. This drives the base of Q2 more positive, causing it to conduct more heavily. The collector of Q2 becomes more negative, and due to the temperature increase, a larger portion of this negative swing is seen at the emitter of Q1. A more negative level at the emitter of Q1 tends to reduce the current flow through Q1, thus canceling the effects of the temperature increase.

Q3 is biased in its active region due to the voltage divider from -11 volts to ground in its base circuit. The Q3 collector rides quiescently at +2 to +3 volts or so. Since Q3 is coupled capacitively on both its input and its output, the dc (quiescent) conditions of Q2, Q3, and Q4 are not dependent upon each other. However, any CHANGES in Q2 output will affect Q3, and any changes in Q3 output will affect Q4.

The base of Q4 is held at about +5 volts by the voltage divider, and since Q5 is in saturation, holding the emitter of Q4 at +.8 volt or so, Q4 is off. When a pulse is detected on the delay line, the .047 μf capacitor acts as a short circuit for rapid changes and the base of Q4 follows any change in the Q3 collector voltage. If the Q3 collector becomes more positive, this positive change at the Q4 base only serves to reverse bias Q4 more strongly. If the Q3 collector becomes more negative, however, the reverse bias on Q4 is decreased. If the signal at the Q3 collector swings negative enough (about +4 to +5 volts or more) to drive the base of Q4 to near ground potential, Q4 turns on. The Q4 base voltage is transmitted to the base of Q5 through emitter follower action, and Q5 turns off.

Actual data signals at the collector of Q3 swing about 10 or 11 volts negative, so the base of Q4 actually is driven 10 or 11 volts more negative than +5, where it rides quiescently. Note that signals on the collector of Q3 that swing less than 4 to 5 volts negative do not turn on Q4, thus providing the necessary amount of noise immunity.

E. END OF PROGRAM DETECTOR (EPD)

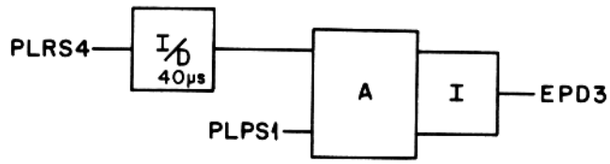
Program line home can be defined as the time from the end of the program until the sync pulse is again sensed. Since programs can vary in length from only a few characters up to 1119 characters, the program line timing chain cannot operate for a fixed amount of time: it must have the capability of operating for time periods varying between a few hundred microseconds and over 10.7 milliseconds. This is made possible by sensing the program data as it is read off the program line, and allowing the timing chain to run as long as there is still data to be read. When the end of the program is sensed and there is no more data coming off the line, the timing chain is stopped. The circuit which senses the presence or absence of data coming off the program line is the End of Program Detector, EPD.

Because there is no special signal on the line denoting the end of the program, the EPD circuit uses the absence of data to signify program line home. As long as data is sensed, the EPD output remains false, but when data stops coming off the line, EPD3 goes true. Since

it is possible to have lapses in data pulses within the program amounting to 30 μsec or so, the EPD waits until no pulses have been sensed for about 40 μsec before signaling the end of the program.

1. BASIC EPD LOGIC

Figure 8-33 illustrates the basic logic involved in the EPD circuit. The symbol containing the "I/D" represents an inverter with a built-in time delay. The AND gate is equivalent to a standard 5610 "false-AND" gate (although it is different physically): both inputs must be false in order to make EPD3 true. If PLPS1 is false and PLRS4 goes true, EPD3 will become true after a certain time delay (about 40 μsec).

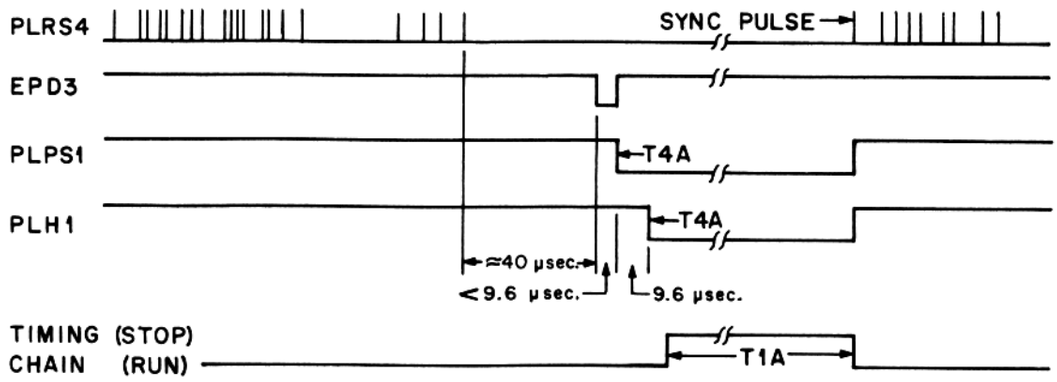


BASIC EPD LOGIC

FIGURE 8-33

In actual practice, as described in Section 3, Program Line Logic, EPD3 going true enables the set gate of PLPS, and at the next T4A time PLPS sets. This immediately drives EPD3 false again. This condition remains until the sync pulse is sensed, at which time PLPS again resets (PLPS1 goes false), enabling the lower input to the AND gate. At this time, however, PLRS4 also goes false, preventing EPD3 from becoming true.

As the program data emerges from the read amplifier, PLRS4 changes from true to false and back many times, but it never remains true for as long as 40 μsec until the end of the program is reached. After PLRS4 has been true for 40 μsec, the upper leg of the AND gate is enabled, and EPD3 goes true. The very next T4A occurs within 9.6 μsec and sets PLPS, again driving EPD3 false. Typical timing relationships are shown in Figure 8-34. The PLH1 waveform is included for reference.



EPD TIMING RELATIONSHIPS

FIGURE 8-34

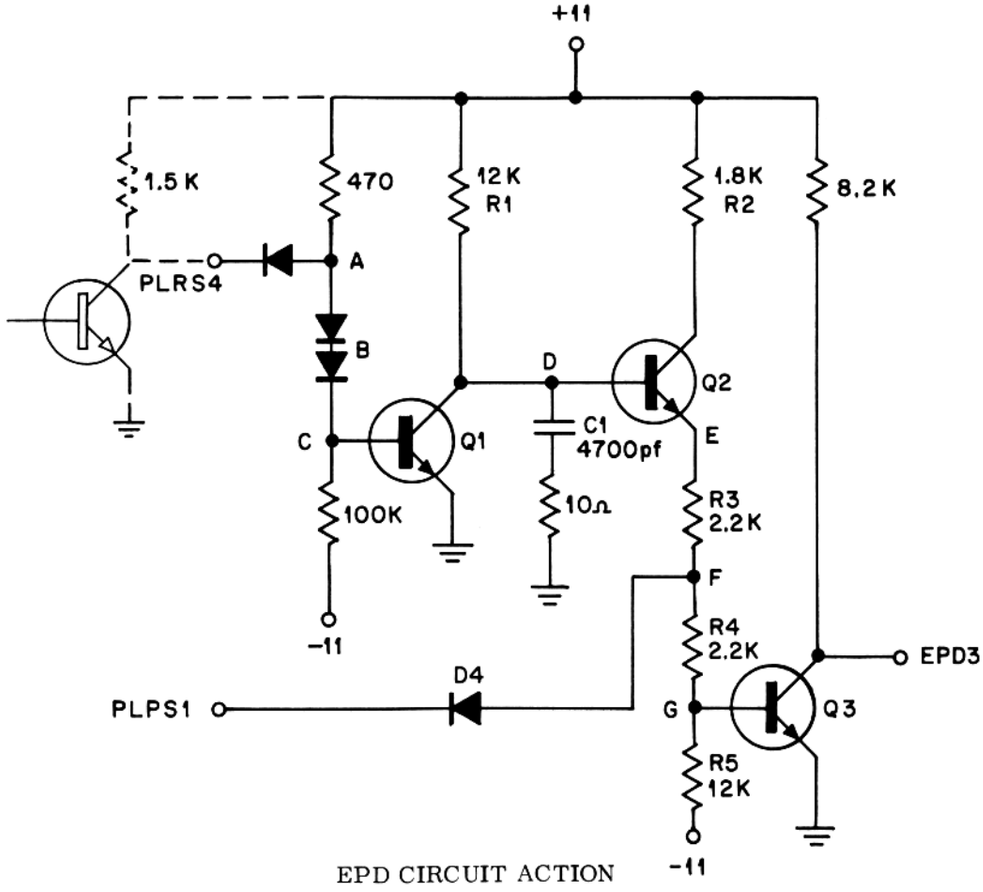
TP-161SL

2. CIRCUIT ACTION

As a starting point, assume that the end of the program has been reached, PLPS and PLH are set, and the timing chain is stopped. Nothing is changing in the EPD circuit, and it will remain in this state until the sync pulse is sensed.

Refer now to Figure 8-35 and note that with the above conditions, PLRS4 is practically at ground. This puts point A at +1 volt, point B at ground, and point C at -1 volt, reverse biasing Q1 so it is off. This would put +11 volts at point D, were it not for Q2 action.

Q2 is actually an emitter-follower; its 1800 ohm collector load resistor is necessary only to limit total current flow. Since the base of Q2 is returned to +11 volts and its emitter is returned to -11 volts, it will be on. Also, at this time PLPS1 is at ground, clamping point F to +1 volt. These biasing elements cause Q2 to be saturated at this time, and the combination of R1 and the voltage divider made up of R2 and R3 causes the base of Q2 to be held at about +7 volts. This charges C1 to about +7 volts. The voltage divider made up of R4 and R5 clamps point G to about -2 volts at this time, holding Q3 off. This makes EPD3 false, approaching +11 volts.



EPD CIRCUIT ACTION

FIGURE 8-35

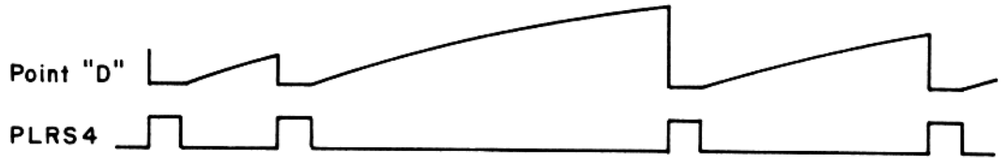
The preceding conditions remain true until the sync pulse is sensed. Then two things happen simultaneously: PLPS1 goes false, removing the clamp from point F and actually reverse biasing D4; and PLRS4 goes false, turning on Q1, discharging C1, and causing Q2 to come out of saturation. This latter action reverses the tendency of points F and G to swing positive (a tendency caused by PLPS1 going false). Instead, F and G become more negative and continue to hold Q3 off.

When PLRS4 goes false (goes toward +11 volts) the voltage at points A, B, and C rises until point C becomes positive enough to saturate Q1: when Q1 turns on it clamps A, B, and C to about +2 volts, +1 volt, and ground, respectively. Also, when Q1 turns on, C1 discharges almost instantaneously. Prior to the sync pulse C1 was charged to about +7 volts, and this voltage is shunted to ground when Q1 turns on. The 10 ohm resistor limits the current flow to protect Q1.

Q1 remains on only for as long as PLRS4 is false, nominally 400 nsec. As soon as PLRS4 returns to true, Q1 turns off and C1 starts to charge again. Note, however, that the C1 charge path is quite different from its discharge path: its charge time will be quite long in relation to its discharge time. As C1 charges, the voltage on the base of Q2 rises. The emitter of Q2 follows the base, and points E, F, and G gradually become more positive.

Before point G can become positive enough to turn on Q3, the next bit of data on the program line is sensed by the read amplifier and PLRS4 again goes false. This turns Q1 on, completely discharging C1. When PLRS4 next returns to true, C1 again starts to recharge. Note, however, that after every PLRS4 excursion, C1 is completely discharged and the subsequent charging action begins from the zero-charge state.

This procedure continues while the entire program is sensed by the read amplifier: each pulse on PLRS4 turns on Q1 to discharge C1, and when the pulse is terminated C1 again begins to recharge. A typical resultant waveform for point D is presented in Figure 8-36.



EPD TIMING CAPACITOR WAVEFORM

FIGURE 8-36

When the last program data pulse is sensed, Q1 turns on and C1 is discharged, as before. When the last pulse is terminated, C1 begins to recharge, and the voltage at points D, E, F, and G begins to rise. Since there are no more data pulses appearing on PLRS4, C1 continues to charge and the voltage continues to rise. After about 40 μsec, the voltage at point G becomes sufficiently positive to turn on Q3 to drive EPD3 true. Q3 turning on clamps point G to about +1 volt.

At this point two things happen, independently of each other: when EPD3 becomes true, it enables the set gate of the PLPS flip-flop. The next T4A time, which occurs within 9.6 μsec after EPD3 goes true, triggers PLPS, and PLPS1 goes true, clamping point F to +1 volt, and causing point G to become negative. This turns off Q3 and drives EPD3 false again. At the same time, C1 continues to charge, the voltage at point D continues to rise, and point E continues to follow point D. Because point F is clamped and point E is rising, the voltage dropped across R3 is increasing. It increases until the current flow through R2 and R3 reaches its maximum, determined by the total resistance of R2 and R3 (and the Q2 emitter-collector drop) and the supply voltage. At this point Q2 is in saturation, and it remains this way until the sync pulse is again sensed.

F. DECIMAL COMPARE

In order to complete a Decimal Print or Non Decimal Print instruction, the Decimal CoMPare signal (DCMP) must be developed. DCMP is also necessary to terminate an Enter instruction, when the right-of-decimal setting has been reached, if the instruction is not terminated earlier by a space, tab, etc.

In both cases, the left-of-decimal setting is set into the decimal register in 4M6P, and the right-of-decimal setting is set into the search register in 4M7P. In 5 mode the decimal register is counted down from the left-of-decimal setting with each digit entered or printed. When the decimal register reaches zero, the decimal point has been reached and the decimal register is automatically set to 5. It is then counted down further for each digit entered or printed. During the time the digits to the right-of-decimal are being entered or printed, the DCMP logic is at work comparing the decimal register and search register outputs.

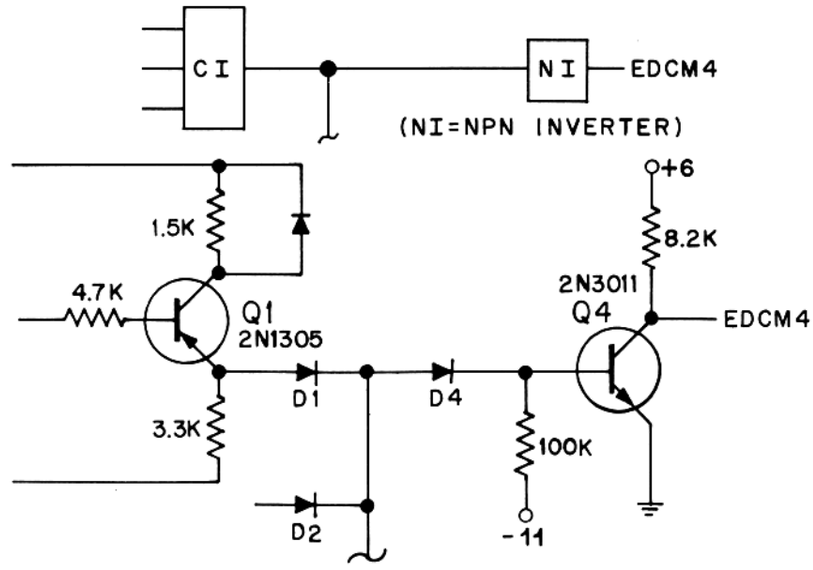
Whenever the sum of the right-of-decimal setting in the search register and the setting in the decimal register equals 5, DCMP is produced to terminate the entry or printout. Figure 8-37 compares the search register and decimal register settings which total 5 and produce DCMP.

Right of Decimal Setting	Search Register			Digits Entered or Printed	DR Setting: DR Set to 5 and Counted Down Once for each Digit Entered or Printed			
	3SR	2SR	1SR			4DR	2DR	1DR
0	0	0	0	0	5	1	0	1
1	0	0	1	1	4	1	0	0
2	0	1	0	2	3	0	1	1
3	0	1	1	3	2	0	1	0
4	1	0	0	4	1	0	0	1
5	1	0	1	5	0	0	0	0

DR AND SR SETTINGS IN PRODUCING DCMP

FIGURE 8-37

The outputs of the decimal and search registers are compared in a group of three transistor gates, called "compare inverters" (CI's). One CI is shown in Figure 8-38, along with the circuitry to which it is connected. The output of the CI, at D1, is dependent upon the combination of voltages applied to the base, emitter, and collector circuits. If the output of any of the three CI's is false, the base of Q4 is positive and Q4 is in saturation, holding Enable Decimal CoMpare (EDCM4) true. To produce DCMP, all CI outputs must be true, causing EDCM4 to be false.



BASIC COMPARE INVERTER

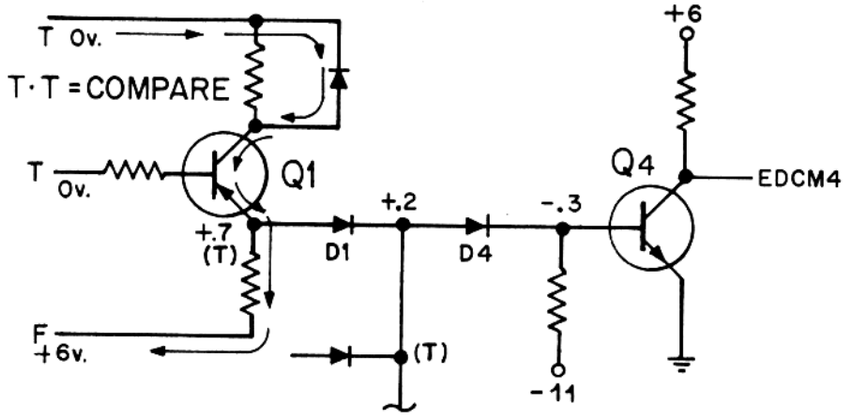
FIGURE 8-38

The output of any of the CI's will be true when the inputs on the base and collector are the same. When the collector/base inputs are different (one true and one false), the output of the gate will be false. Thus there are four possible combinations, as follows:

	Collector	Base	Output
1.	T	T	T
2.	F	F	T
3.	T	F	F
4.	F	T	F

The inputs on the collector and emitter are always opposite, being the two opposite signals from each DR flip-flop. The inputs on the bases come from the true side of the SR flip-flops.

Figure 8-39 (a) illustrates the first case, with the collector and base inputs both true. The biasing is such that the 2N1305 is in saturation; its emitter is at approximately +.7 volt. This will yield a Q4 base voltage of about -.3 volt, assuming an average drop of .5 volt across each diode. (Also assuming that the outputs of the other two CI's are true.) Any negative voltage at the base of Q4 will be sufficient to turn it off, causing EDCM4 to become false.



COMPARE INVERTER ACTION, T + T = COMPARE

FIGURE 8-39 (a)

Figure 8-39 (b) illustrates the second case, with false inputs on both the base and collector. The positive voltages on these two legs reverse bias the junctions so the transistor is cut off. This allows 0 volts (true) to be seen at the emitter.

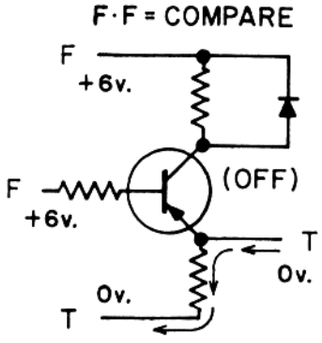


FIGURE 8-39 (b)

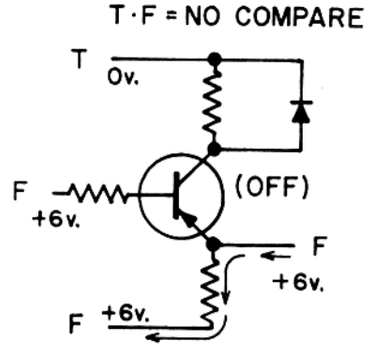


FIGURE 8-39 (c)

Figure 8-39 (c) illustrates the third case. The positive voltages again reverse bias the junctions causing the transistor to be cut off. This time, however, the positive voltage on the emitter circuit is seen at the output, causing the output to be false. The voltage at the output will not actually be +6 volts, because Q4 will turn on and clamp the emitter of Q1 at about +2 volts due to the voltage drops across the emitter-base junction of Q4 and across D1 and D4.

Figure 8-39 (d) illustrates the last case. Since the transistor operation in this situation is somewhat unusual, it will be explained in more detail.

The supply voltages tend to strongly forward bias the collector-base junction, while the emitter-base junction has virtually no bias at all. This results in the collector acting as the emitter and the transistor turns on. Note that the direction of electron flow is opposite to that found in normal PNP operation. This action would not be unusual in a symmetrical transistor, but the 2N1305 is not symmetrical: its emitter and collector are physically different sizes and they are "doped" differently. These factors result in a very low Beta when the transistor is "on" in the reverse direction. The end result is that a "false" signal appears at the emitter.

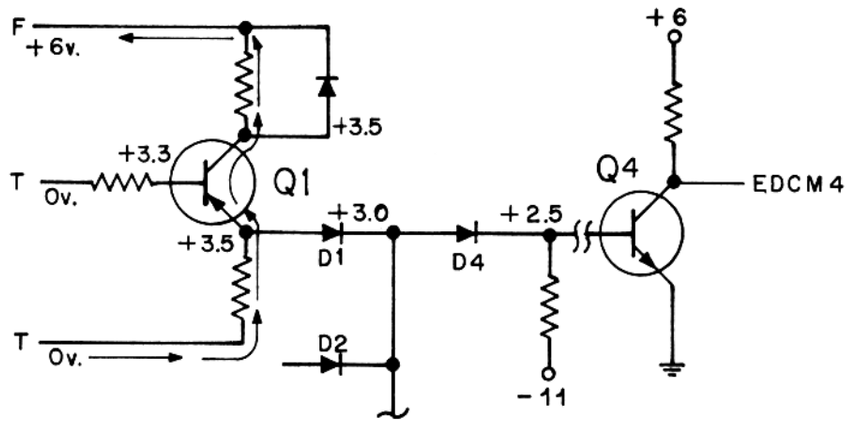


FIGURE 8-39 (d)

Referring again to Figure 8-39 (d), note that with Q4 removed from the circuit D1 and D4 are forward biased through the 100,000 ohm resistor and Q1 is in saturation, having emitter and base voltages practically equal. A high positive potential is available to apply to the base of Q4. Now observe Figure 8-39 (e) and note that when Q4 is added to the circuit it turns on, causing EDCM4 to be true. Also, by the action of Q4 turning on, the emitter of Q1 is again clamped firmly at about +2 volts.

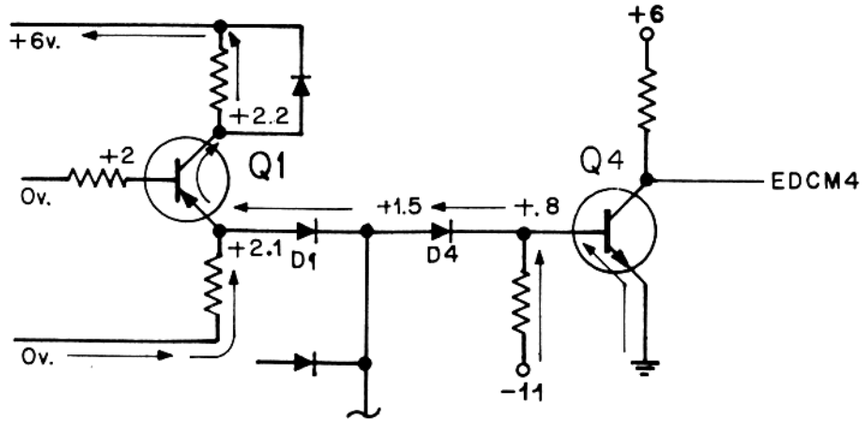
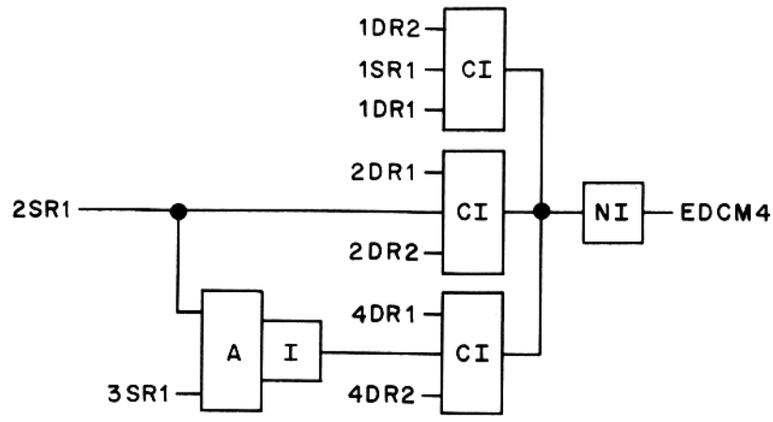


FIGURE 8-39 (e)

The CI's explained here are identical to those discussed in Section 5B, Storage Location Correct.



EDCM LOGIC

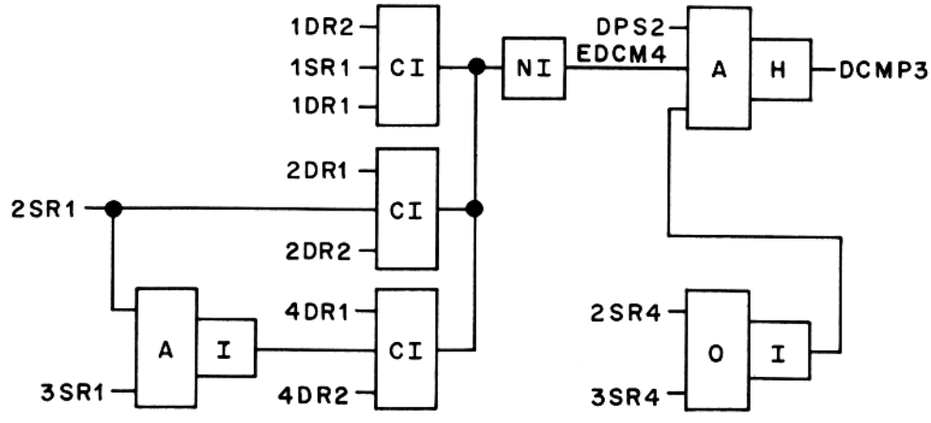
FIGURE 8-40

Figure 8-40 illustrates all three CI's used in the DCMP logic. In order to make EDCM4 false, the following conditions must be met:

1. 1SR and 1DR must be in the opposite logic state.
2. 2SR and 2DR must be in the same logic state.
3. If either 2SR or 3SR are set, 4DR must be reset; if 2SR and 3SR are both reset, 4DR must be set.

Compare these rules and the logic in Figure 8-40 to the table shown in Figure 8-37.

The complete DCMP logic is shown in Figure 8-41. In addition to EDCM4, two other signals must be false in order to make DCMP3 true. DPS2 prevents decimal compare from occurring when digits to the left-of-decimal are being entered or printed. (DPS becomes set when the decimal point is reached.) The last qualification to DCMP3 is necessary to allow for the entry of the algebraic sign: if the programmer wishes to allow for the entry of five digits right-of-decimal, plus the sign, he must call for an LRD pattern of 6 to-the-right. With a 6 in the search register, both 2SR and 3SR are set, thus holding DCMP3 false. Since DCMP is never produced with a 6-right LRD, the entry MUST be terminated with a space, minus, tab, etc. Also, since DCMP is necessary in order to terminate a print instruction, no LRD greater than 5-right can be used for printing.



COMPLETE DCMP LOGIC

FIGURE 8-41

G. INTERFACE

The two main parts of any interface are the input and the output circuits. In the 5610 the output circuits are further subdivided into "translator drivers" and "control drivers". In addition, circuits for "Busy", for "reader run", and for energizing the KCL relay are also included in the 5610 interface.

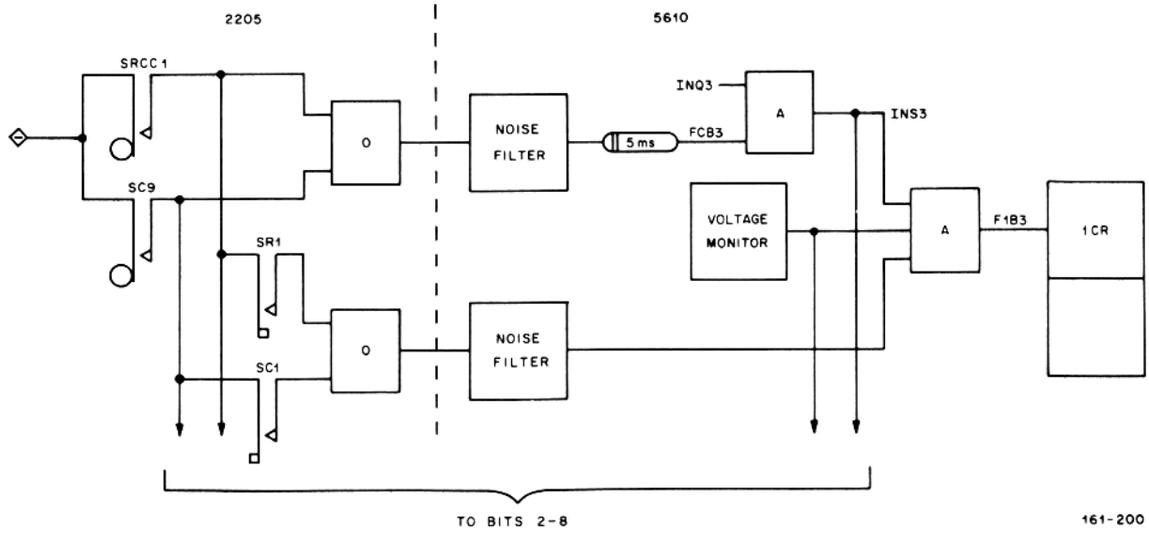
1. INPUT CIRCUITS

A simplified block diagram of the input circuitry is presented in Figure 8-42. Note that this is only a BLOCK diagram, and that the actual circuitry differs in many respects, mainly in that there are many more components than shown.

a. GENERAL OPERATION. Beginning in the 2205 portion of Figure 8-42, note that an entry can be made from either the reader (reader contacts) or the keyboard (selector contacts). (When the individual circuits in the block diagram are broken down in detail, it will be noted that these two entry sources are interlocked so that only one of them can provide an entry at any one time.) Note also that it is necessary for the common contacts to close before the respective code contact can be made "hot". Since the code contacts in the 2205 transfer before the common contacts, any contact bounce in the code contacts will have no effect on the circuit. Common contact bounce is compensated for by employing a 5 ms delay in the Flexowriter Common Bit (FCB3) circuit.

Notice that INput Signal, INS3, cannot be developed until FCB3 and INput Qualification, INQ3, are both true. Thus the 2205 may be operated, but no input will be made to the 5610 processor unless INQ3 is true.

By the time INS3 is developed, the signal from the code contacts is already present at the input to the large AND gate (in Figure 8-42). If the voltage monitor circuit has detected no significant change in the +24-volt supply, F1B3 through F8B3 will be developed, depending upon the code entered, and the corresponding common register flip-flops will become set.

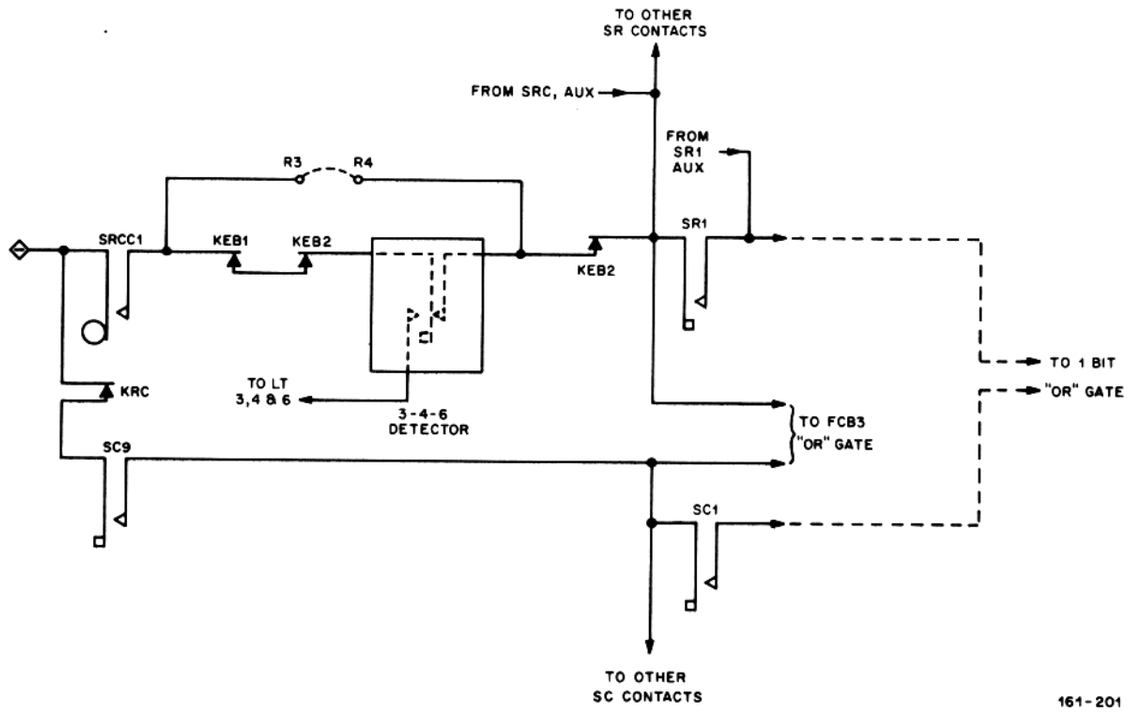


BLOCK DIAGRAM
INPUT CIRCUITRY

FIGURE 8-42

b. 2205 CONTACTS. As noted earlier, entries can be made from either the reader or the keyboard of the 2205. Reader entries are taken directly from the reader contacts, rather than the associated selector contacts, so that entries can be made in non print, or with an 8 bit blocking the print. See Figure 8-43. When a reader entry is made, the selector contact circuit is opened up by a set of KRC contacts: this prevents a double entry of the same code when its selector slide operates as a result of reading the code.

Figure 8-43 also shows that the 2205 common circuit is opened by the two KEB (Entry Block) relays during a "medium speed" entry from the auxiliary reader.



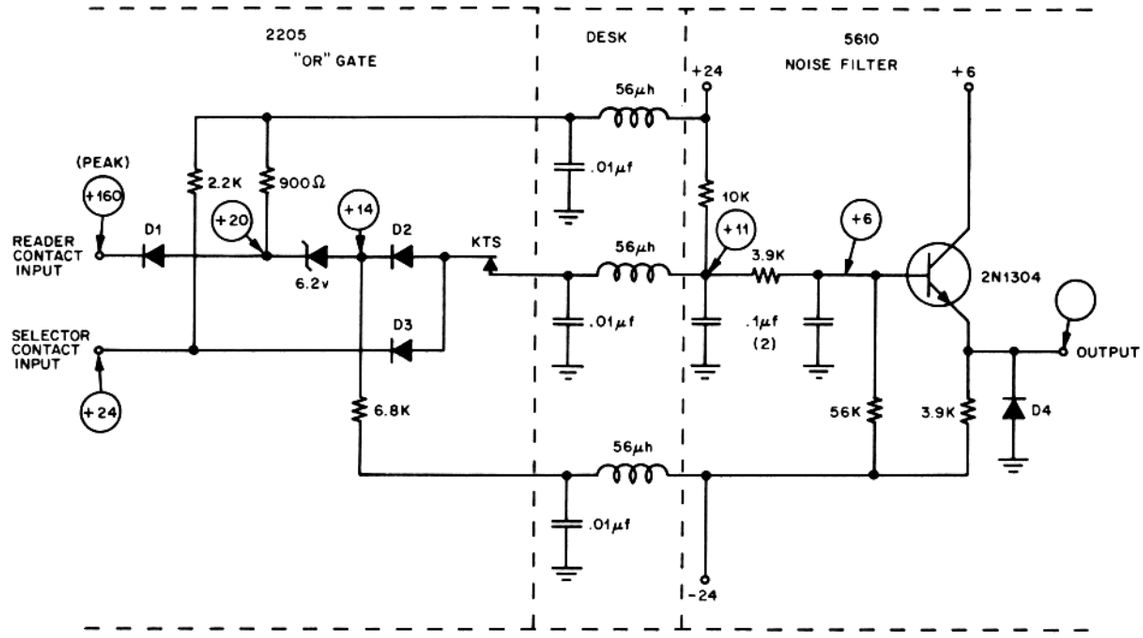
2205 CONTACTS IN ENTRY CIRCUITS

FIGURE 8-43

Still referring to Figure 8-43, note that a special matrix of reader contacts is used to detect the F3 (3-4-6) code. This circuit is designed to allow the programmer the option of having the F3 code non-reproducing when it is read in non print. If it is desired to reproduce the F3 code, a patch is installed in the 2205 patch panel from R3 to R4. Note, however, that if this patch is not installed, the F3 code will not produce FCB3, and it will be impossible to enter the F3 code into the common register. Note also that the F3 code must reproduce (R3 dioded to R4) in order for the SWIFT translator to function properly.

Figure 8-43 also shows the point at which the auxiliary reader connects to the input circuitry. The auxiliary reader common circuit also supplies the auxiliary code contacts and feeds the FCB3 "OR" gate (in Figure 8-44 (a)). The auxiliary reader code contacts connect directly to the 2205 code contacts for input to the eight "OR" gates.

c. "OR" GATE AND NOISE FILTER. The "OR" gate and the noise filter must be studied together because the bias in each circuit determines to some extent the voltage readings present in the other. Figure 8-44 (a) shows the schematic of the two circuits with typical voltages given for no-input conditions. Bear in mind that these values are only approximate (they are rounded off to the nearest volt) and that the actual readings obtained on a particular machine may vary somewhat.



"OR" GATE/NOISE FILTER CIRCUITRY

FIGURE 8-44 (a)

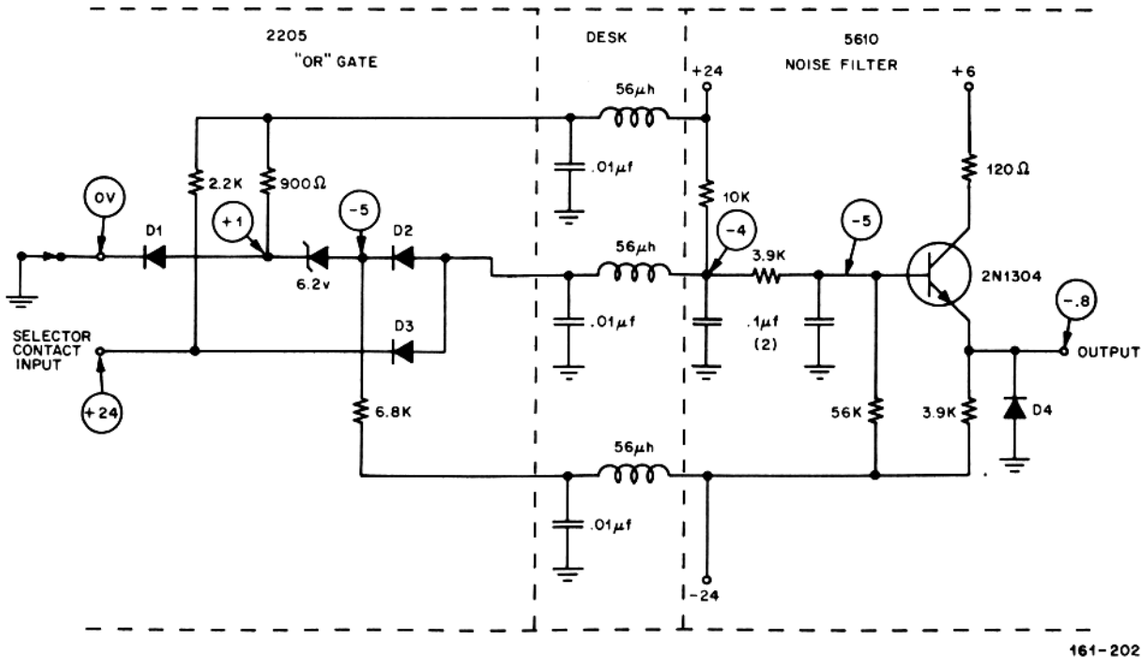


FIGURE 8-44 (b)

With no inputs present, diodes D1, D2, and D3 are all reverse biased, the 2N1304 is heavily forward biased, and the output is false (+6).

(1) Reader Entry. Now see Figure 8-44 (b). Here a reader entry is symbolically represented, pulling the anode of D1 to ground, forward biasing it. The zener diode drops 6 volts, leaving -5 volts at the cathode of D2, thus forward biasing D2 also. D2 drops about 1 volt, placing -4 volts at its anode. The voltage divider made up of the 3900 and 56,000 ohm resistors from this point to -24 volts establishes the base of the transistor negative in respect to its emitter, which can go no more negative than about -.8 volt because of the diode to ground (D4). Thus the transistor turns off and the output becomes true (close to 0 volts).

Note that in this case, diode D3 remains reverse biased. The zener diode is employed to make sure the transistor base voltage will be sufficiently negative to hold the transistor off. For example, since the path to ground from D1 goes through many reader contacts (and some relay contacts), it is possible to encounter a small amount of resistance between D1 and ground. This would tend to allow the voltage at the anode of D1 to rise slightly, with an attendant increase in voltage at other points in the circuit. The zener diode drops enough voltage to insure that a small amount of resistance in the reader contacts will not allow the base of the transistor to rise high enough to enable the transistor to turn on.

(2) Keyboard Entry. Typical voltages to be encountered during a keyboard entry are given in Figure 8-44 (c). When a selector contact closes, completing the circuit to ground from the cathode of D3, D3 becomes forward biased, placing +1 volt at its anode. The voltage divider establishes -1 volt at the base of the transistor, and since the emitter cannot go less negative, the transistor turns off and the output becomes "true".

In this case note that D1 and D2 are reverse biased. No zener diode is needed in this instance because the ground circuit passes through only three contacts; a selector code contact, a selector common, and a KRC relay contact.

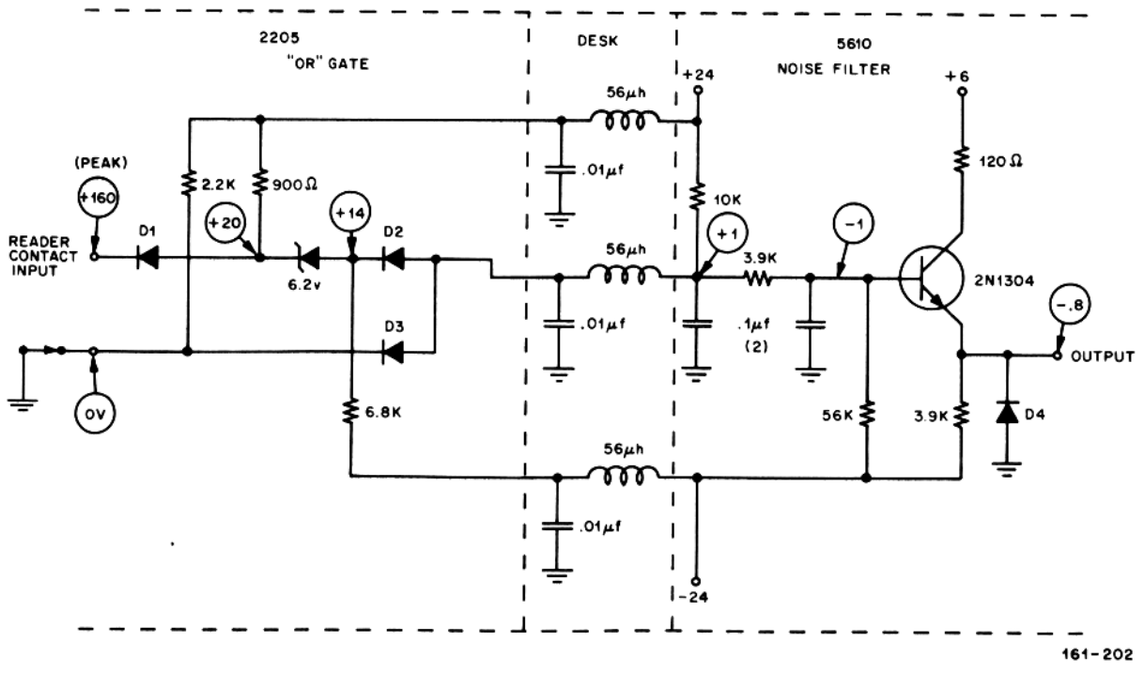


FIGURE 8-44 (c)

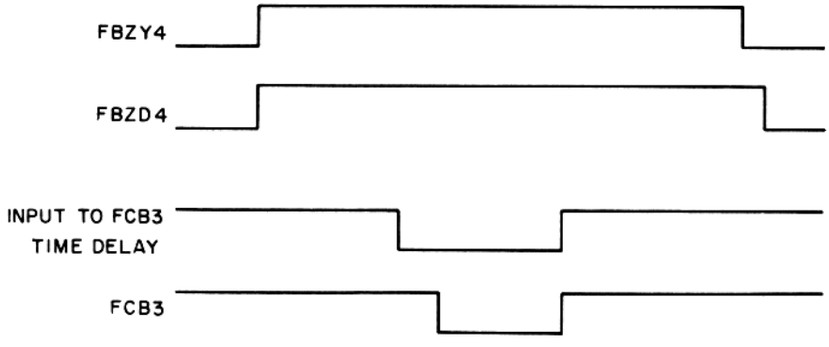
The preceding examples illustrate a typical "bit" noise filter. The common noise filter operates essentially the same way, the only difference being that larger capacitors (.47 μf) are used at the base of the transistor for more filtering action. Also, the collector load resistor is eliminated in the common noise filter, mainly because it next feeds the time delay circuit instead of the final "AND" gate shown earlier.

Again, it should be noted that the voltage values given in these examples are only TYPICAL values, and that the actual values measured on a machine may be quite different.

Note also in Figure 8-44 (a) that a set of KTS contacts prevents the development of FCB3 when the 2205 is in a tape skip condition. KTS contacts are not necessary in the code circuits because no "bits" can be developed without a common bit being developed first.

The inductors and capacitors in all lines between the 5610 and the 2205 are designed to block high frequency noise spikes from entering the processor. All input lines to the processor are also twisted into pairs with a ground wire to form a partial shield, thus lessening the tendency for these wires to collect stray signals.

d. TIME DELAY. The time delay circuit in the FCB3 line is exactly the same as that used in the "Busy" circuit, as discussed in Section 6, Flexowriter Commands. The time delay circuit is not an inverter, so a positive input will yield a positive output. When the input goes negative (toward ground), the output does not follow immediately, but waits approximately 5 milliseconds before dropping. When the input returns to positive, however, the output follows almost immediately. This is the same electronic operation that occurs in the "Busy" time delay, but the opposite logic operation. This is due to the fact that in the busy operation, a "false" signal, FBZY4, is used as the input, and in this case a "true" signal (having no signal name of its own, mainly because it doesn't leave the "SS" card) is used as the input. A comparison of the waveforms is made in Figure 8-45.

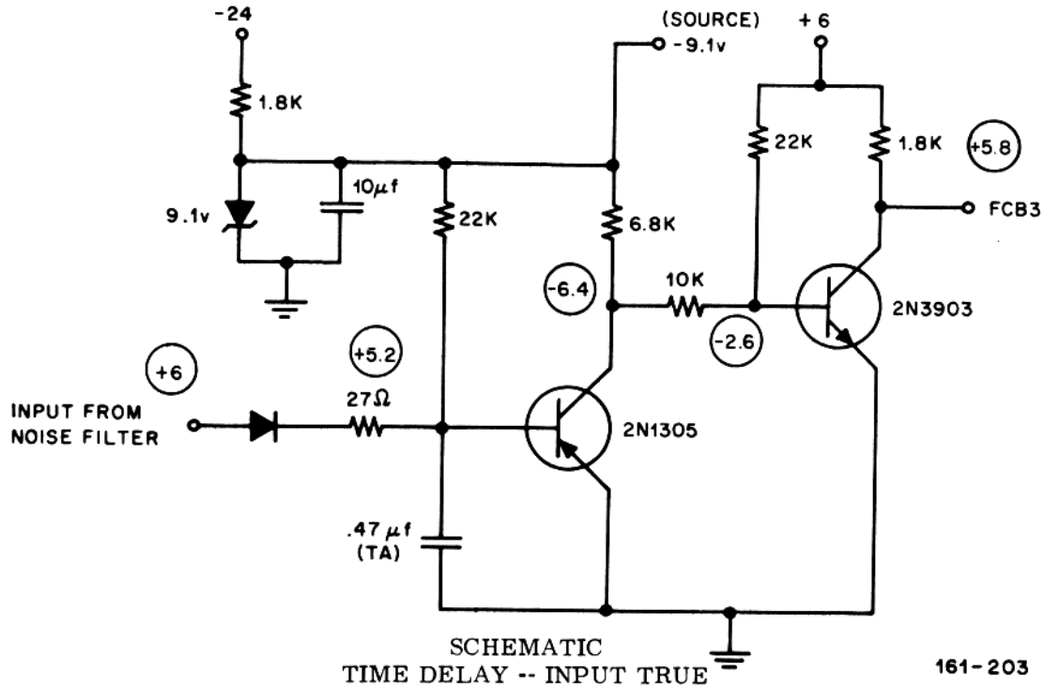


TIMING CHART
COMPARISON OF "BUSY" AND COMMON BIT WAVEFORMS
(Not in same time scale)

FIGURE 8-45

The schematic of the time delay circuit is shown in Figure 8-46. Figure 8-46 (a) also presents the approximate voltage values at different points in the circuit with the input false (high). If the input has been false for a sufficient period of time, the .47 μf tantalum capacitor is fully charged to about 5.2 volts, and the output is also false.

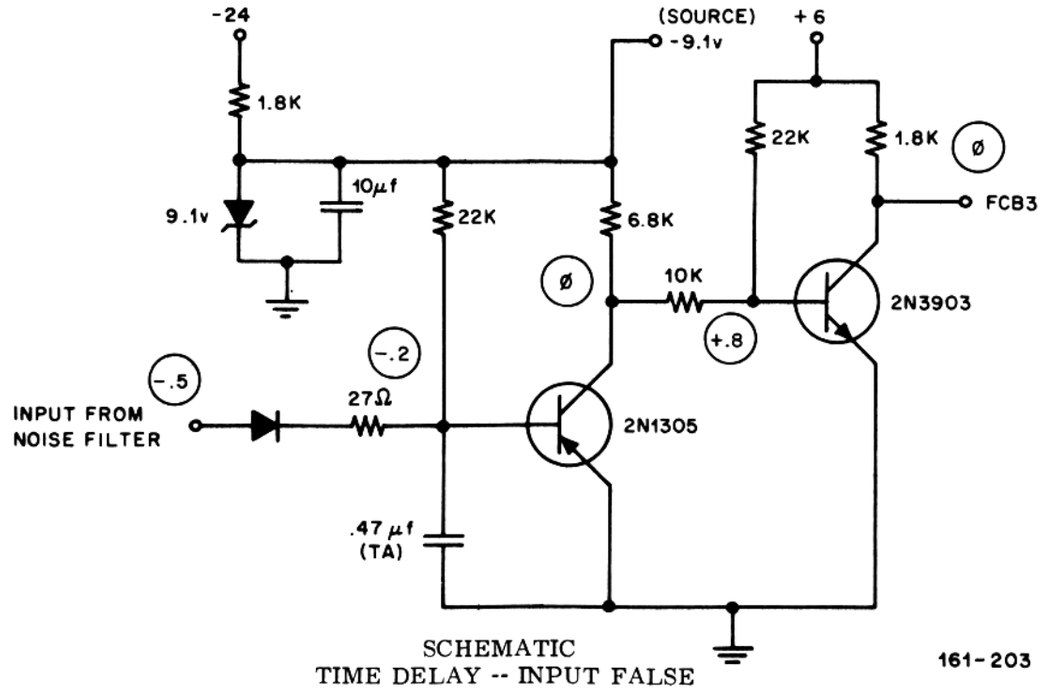
If the input should now become true (about -.5 to -.8 volt), the input diode immediately becomes reverse biased and the capacitor begins to discharge through the 22,000 ohm resistor and the -9.1-volt power supply. After about 5 ms, the base of the 2N1305 becomes negative enough to turn the transistor on, and this clamps the base at about -.2 volt. Its collector becomes grounded, which places the base of the 2N3903 at about +.8 volt, thus turning the second transistor on and driving the output to 0 volts (true). This last condition is reflected in the voltage values presented in Figure 8-46 (b).



SCHEMATIC
TIME DELAY -- INPUT TRUE

FIGURE 8-46 (a)

When the input returns to false the input diode is again forward biased and the capacitor charges almost immediately, through the 27 ohm resistor. Thus there is a 5 ms delay between a false-to-true change at the input and the corresponding change at the output, but there is no significant delay between a true-to-false change at the input and the equivalent change at the output. The 27 ohm resistor is necessary to limit the initial capacitor charging current, thus protecting the emitter follower in the noise filter.



SCHEMATIC
TIME DELAY -- INPUT FALSE

FIGURE 8-46 (b)

TP-161SL

The time delay circuit used to produce Flexowriter Busy Delayed, FBZD4, (discussed in Section 6, Flexowriter Commands) is identical to this circuit, except for the fact that the three components making up the -9.1-volt supply are not duplicated: the -9.1-volt level produced on the "SS" card for the FCB3 circuit is wired to the FBZD4 time delay circuit on the "PP" card.

e. INS DEVELOPMENT. The logic used to produce INS when FCB3 becomes true was covered previously in Section 3F, Program Load: since this part of the input circuitry is strictly logic, and no unusual electronic circuits are involved, it will not be repeated here.

f. "AND" GATE. The large "AND" gate symbolically represented previously in Figure 8-42 is schematically shown in Figure 8-47, along with the circuitry coupling the INS signal to it, and the voltage monitor circuit. This illustration actually shows 8 AND gates, one for each bit. In order for F1B3 through F8B3 to become true, the outputs of the two 2N1304's AND the output of the voltage monitor AND the output of the respective noise filter must all be true. If any one of the three is false, the output signal will be false and the corresponding common register flip-flop will not become set.

(1) INS Circuit. The voltages noted in Figure 8-47 are typical of those to be found when INS4 is true (FCB3 not developed). The 2N3903 is off, and both 2N1304's are on, supplying a positive voltage to the eight bit signals, holding them false. When a common bit is developed and INS4 goes false, the 2N3903 turns on, applying a slightly negative voltage to the base of the 2N1304's, causing them to turn off. In this state, they have no control over the signals appearing at the eight outputs, so any true signals from the noise filters will be allowed to appear at F1B3 through F8B3.

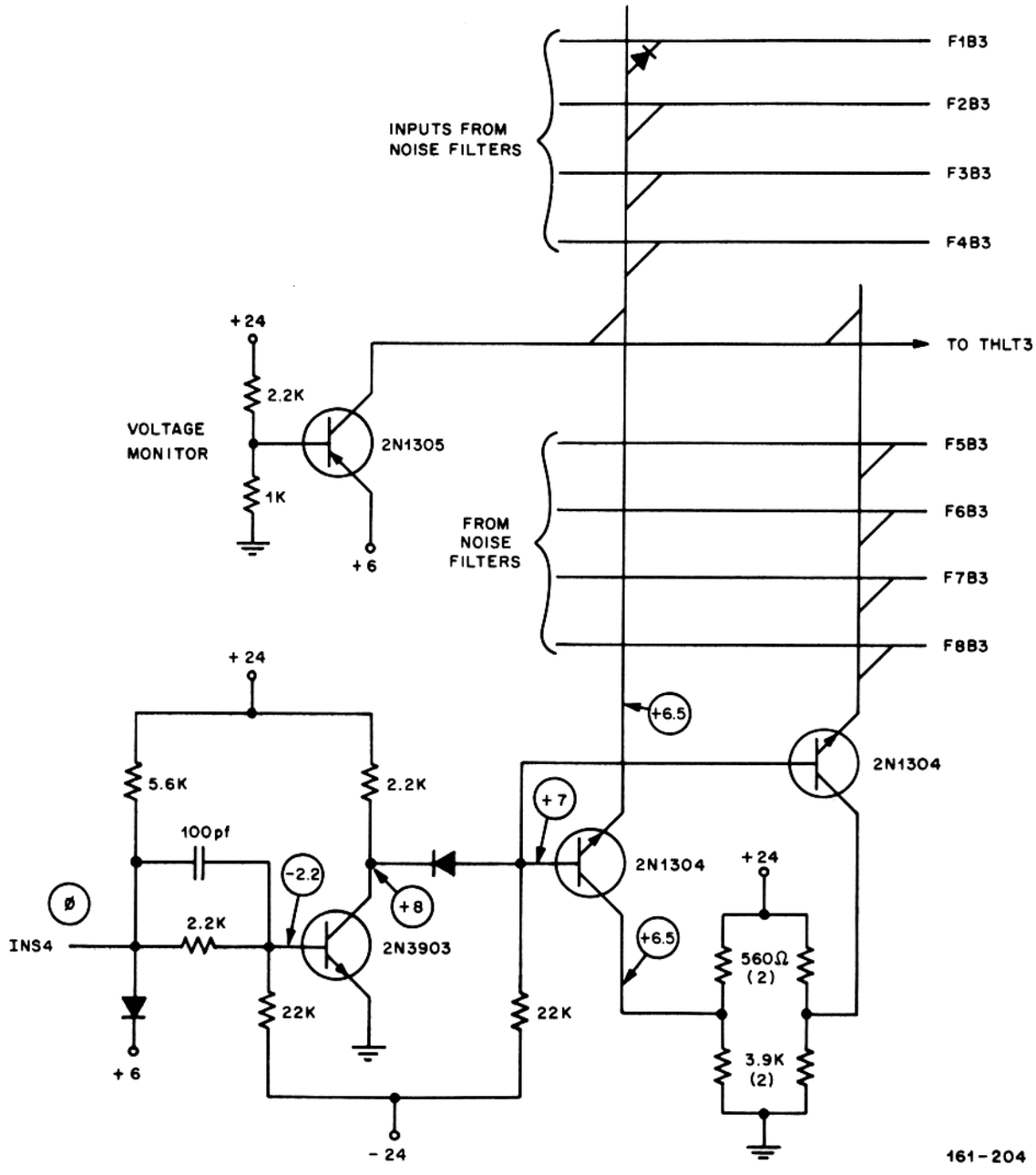
(2) Voltage Monitor Circuit. The purpose of the voltage monitor circuit is to prevent any inputs from occurring when the +24 supply voltage drops appreciably. This occurs when the line voltage drops out and the system switches over to battery operation: no inputs or outputs can occur during operation by battery power. Still referring to Figure 8-47, note that a large drop in the +24 supply will forward bias the 2N1305, placing +6 volts on F1B3 through F8B3.

2. TRANSLATOR DRIVERS

There are nine identical translator drivers, one for each of the eight bits and one for the translator clutch. The Common OUTput signal, COUT3, enables the clutch driver, and the Enable Driver signals, E1D3, E2D3, etc., "turn on" the bit drivers. These drivers are used during all outputs other than Special Flexowriter Functions: this includes both of the print instructions, the Type instruction, and miscellaneous functional instructions, such as Carriage Return, Tab, Upper Case, etc.

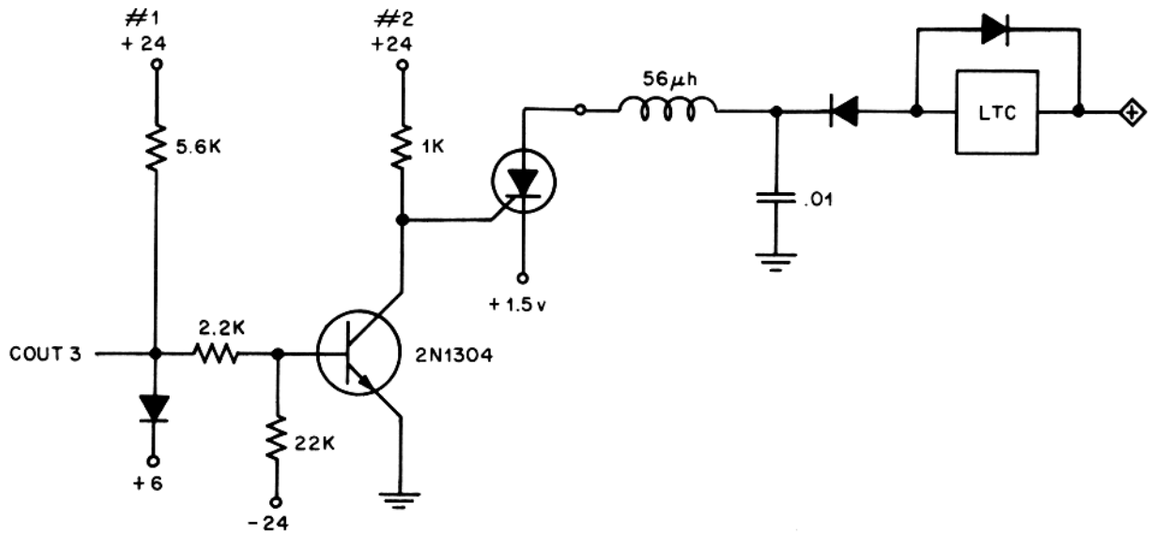
Each driver employs a Silicon Controlled Rectifier (SCR) to open and close the circuit to the individual translator magnets. In order to turn on, an SCR must have its gate become more positive than its cathode. Then, the only way to turn off the SCR is to have its anode circuit opened. In the 5610 the dc current used to energize the magnets is of the pulsating (not filtered) type. This means that the current flow returns to zero each half cycle and the SCR cuts off. It is then turned on again by the voltage at the gate still being more positive than that on the cathode. When the gate voltage later returns to the more negative value, current flow stops at the next half cycle and does not restart. Thus the pulsating voltage is used to "open" the anode circuit, turning off the SCR.

The translator clutch driver is shown in Figure 8-48. (Note the similarity between this circuit and the INS circuit in Figure 8-47.) Quiescently COUT3 is false, forward biasing the transistor and holding ground on the gate of the SCR. When COUT3 goes true it drives the base of the transistor to some negative value, cutting off the transistor and driving the SCR gate positive. This turns on the SCR, energizing the magnet for as long as COUT3 is true (until the translator reaches 20 degrees, BZY is developed, and the phase counter can step - refer to Section 4G, Enter, and Section 6, Flexowriter Commands, for more information).



SCHEMATIC
"AND" GATE, INS CIRCUITRY, AND VOLTAGE MONITOR

FIGURE 8-47

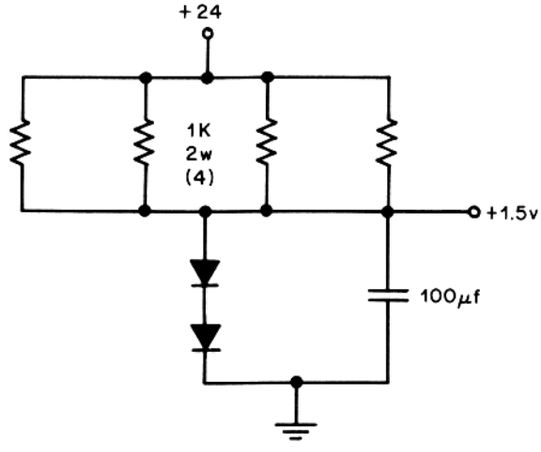


SCHEMATIC
TRANSLATOR DRIVER

FIGURE 8-48

The bit drivers operate exactly the same as the clutch driver, and they are turned on simultaneously with the clutch driver. (The E1D3, E2D3, etc., signals are enabled by COUT3.) Thus the translator clutch and all bit magnets for the code being processed are energized for the same length of time.

The +1.5-volt supply, pictured in Figure 8-49, is simply a group of dropping resistors and two diodes providing a voltage more positive than ground on the SCR cathode to insure a reliable shut-off of the SCR's when the output pulse is to be terminated. Note that all current required to energize the translator clutch and a maximum of five bit magnets must pass through these resistors: hence the higher wattage.



+1.5-VOLT POWER SUPPLY

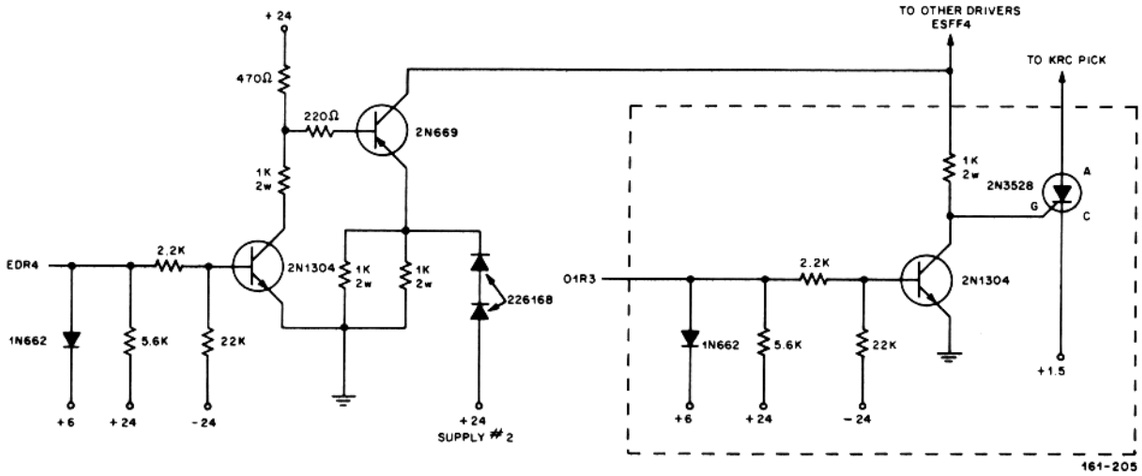
FIGURE 8-49

The "#2" +24-volt supply is controlled by a K1 relay contact so that no control drivers can be energized while the power supplies are building up after machine turn-on.

3. CONTROL DRIVERS

There are eight control drivers, one for each of the Special Flexowriter Functions outlined in Section 6. These drivers use the same +1.5-volt supply and are identical to the translator drivers in all respects other than the collector supply voltage: whereas the translator driver collectors are returned directly to +24 volts, the control driver collectors are returned to a special circuit which produces a signal called Enable Special Flexowriter Functions, ESFF4. This circuit supplies all eight control drivers, and it is controlled, in turn, by the Enable DRivers signal, EDR4.

This special circuit is illustrated in Figure 8-50. Quiescently EDR4 is true (ground), reverse biasing the 2N1304. The two diodes in the emitter circuit of the 2N669 drop approximately 2 volts, so the emitter of the 2N669 rests at about +22 volts. With the 2N1304 off, the 2N669 is also reverse biased, opening the circuit to the collectors of all eight drivers.



SCHEMATIC
CONTROL DRIVER

FIGURE 8-50

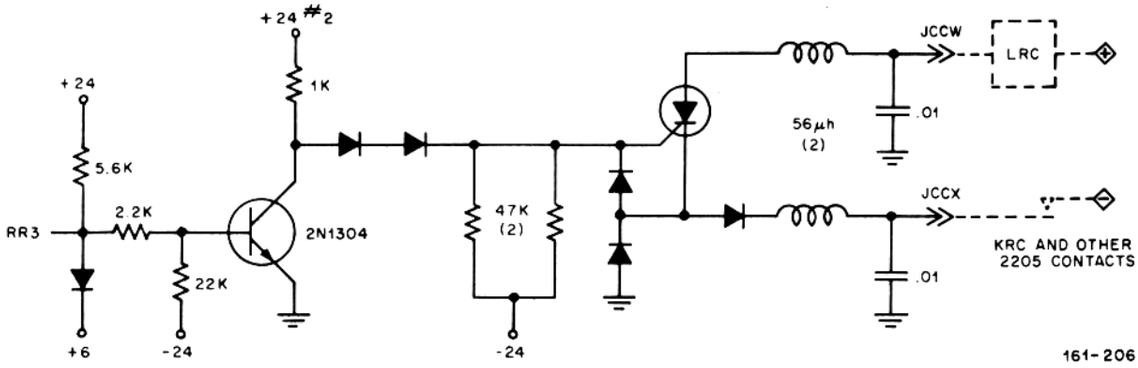
When EDR4 goes false, the 2N1304 turns on, providing about +16 volts at the junction of the 470 ohm, 1000 ohm, and 220 ohm resistors. This is enough to turn on the 2N669, which supplies a level of about +22 volts to the collectors of the eight control drivers. One of the drivers will have its input signal true, and the transistor will turn off, providing a positive voltage at the SCR gate, allowing it to turn on. The SCR will then remain on until EDR4 returns to true, removing the collector supply from the driver, or until the 2205 relay circuitry opens the SCR anode circuit.

The +24-volt #2 supply is again used to prevent any outputs from being generated during the initial machine start-up.

4. LRC CIRCUIT

The circuit used to energize the reader clutch magnet (LRC) is represented logically by an AND gate: the Reader Run signal, RR3, must be true, and the KRC relay must be energized. (The usual 2205 interlocks are also included in this last qualification. This includes the normal condition of the Start Read, Tape Skip, and Non Print switches, the SDC contacts, the KPE relay, the SCRT contacts, etc.) In other words, to all of the 2205 qualifications necessary to energize the reader clutch, the RR3 signal is ANDed.

The electronic circuitry involved is pictured in Figure 8-51, along with a symbolic representation of the 2205 qualifications. If RR3 is true, the 2N1304 is off and the resistor-diode-resistor voltage divider between +24 and -24 volts places a positive voltage at the SCR gate. If all 2205 qualifications are satisfied, the SCR anode and cathode circuits will be complete and the SCR will turn on, energizing the reader clutch magnet.



SCHEMATIC
READER RUN CIRCUITRY

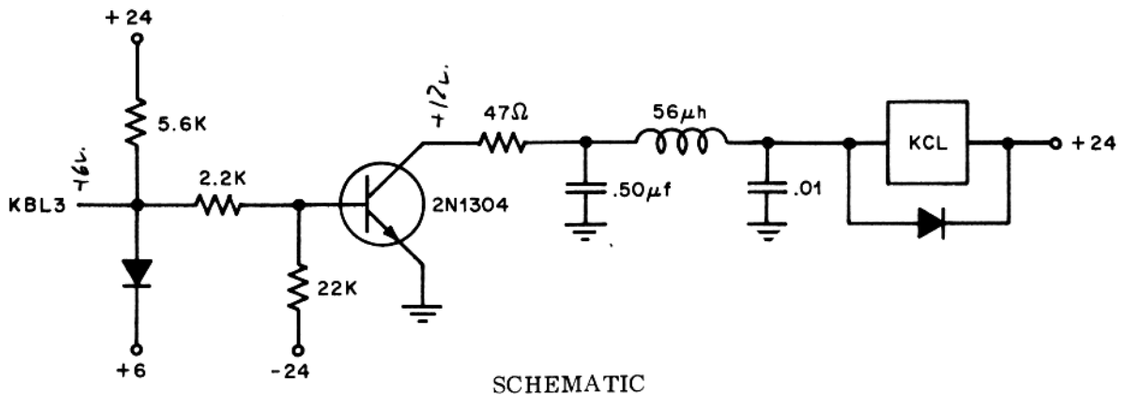
FIGURE 8-51

If any of the 2205 contacts in the circuit open, such as SCRT, SBS, KDC, SRT, etc., the SCR turns off and the clutch magnet de-energizes. On the other hand, if RR3 should go false, the 2N1304 turns on, placing about -2 volts on the SCR gate. The SCR then shuts off as its current goes to zero on the next half cycle, and it cannot re-fire until after RR3 again goes true. Thus both qualifications must be met in order to energize the reader clutch: RR3 must be true and all of the regular 2205 conditions must be in order.

5. KCL CIRCUIT

The KCL relay must be energized in order to "pick" the KKL relay, which unlocks the 2205 keyboard. Whenever a parity error or some other type of error occurs, the KeyBoard Lock signal, KBL3, is developed, which causes KCL to drop out. This, in turn, drops KKL, locking the keyboard.

The circuit used to drive the KCL relay is very similar to the LRC driver and the other drivers. It is shown in Figure 8-52. Quiescently, KBL3 is positive (false), holding the transistor on, thus supplying ground to the relay, which is energized. When KBL3 becomes true, the transistor turns off, opening the relay circuit and allowing the relay to drop out.



SCHEMATIC
KCL RELAY DRIVER

FIGURE 8-52

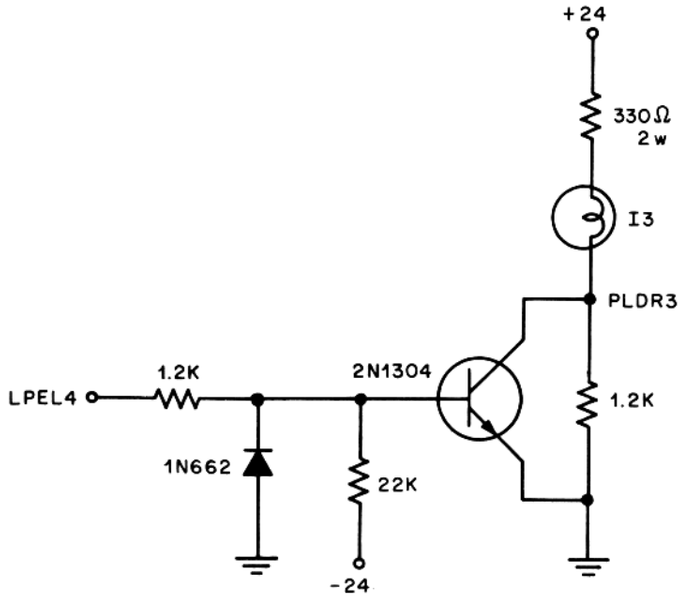
6. "BUSY" CIRCUIT

The noise filter and 5 ms delay circuits used in the "Busy" circuit are exactly the same as those used in the development of FCB3. Refer to Figures 8-43 through 8-46 for review of their electronic operation. The logic operation of the "Busy" circuit is covered in Section 6, Flexowriter Commands.

H. LAMP DRIVER

There are four lamp drivers, two each on the "PP" and "RR" cards. Three of these are used to light the "Sequence", "Load", and "Error" lamps on the front of the processor: the fourth is a spare. The other indicator lights on the processor ("Translate", "Type Only", and "Power On") are controlled directly by the respective mechanical switches: "Power On" is illuminated whenever power is applied to the system and the -24-volt power supply is activated. "Translate" and "Type Only" are illuminated whenever the corresponding switch is operated. Refer to the "CP" drawing for details.

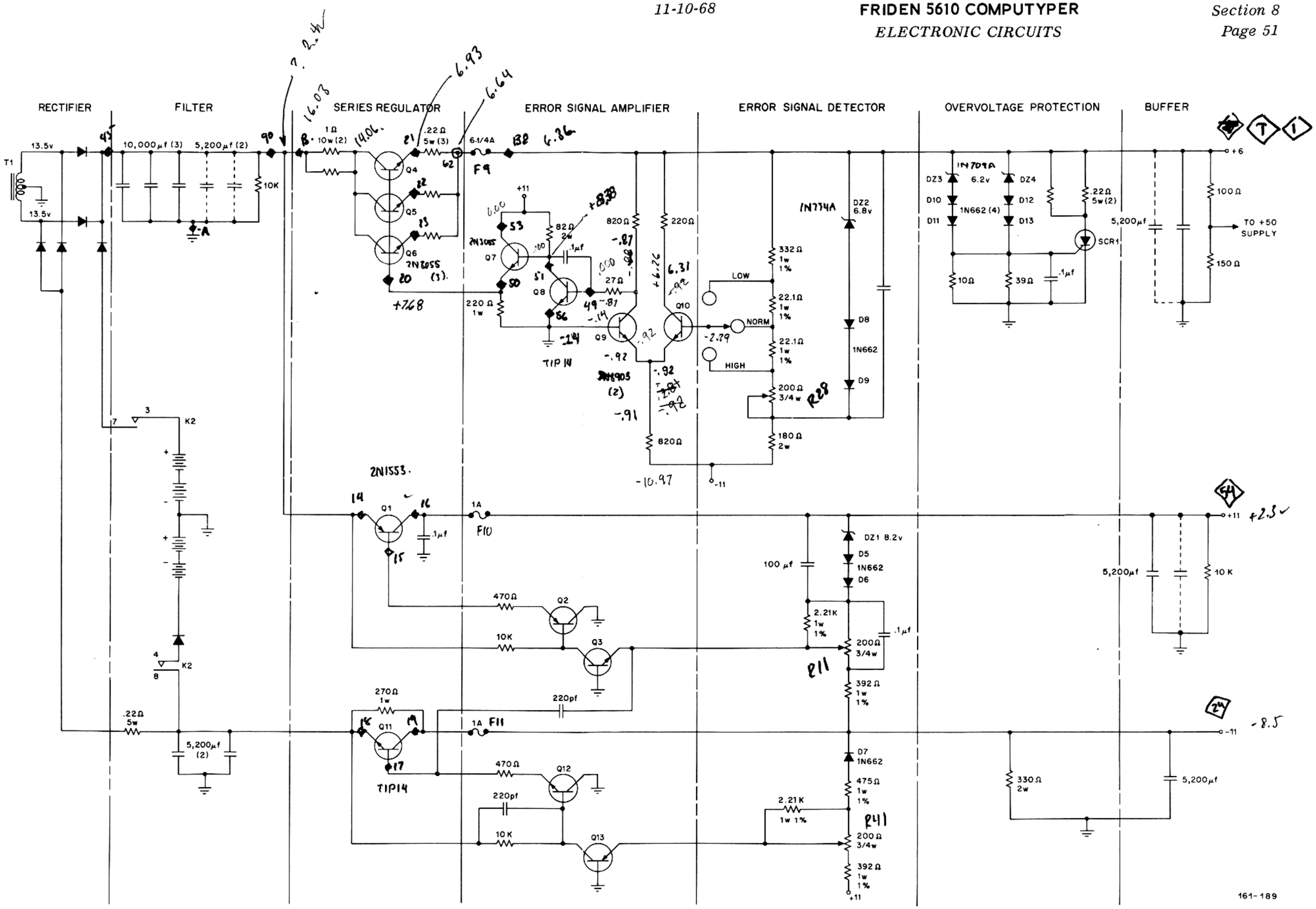
A typical lamp driver is shown in Figure 8-53. Note that it is very similar to the various circuits used in the interface. Quiescently the Light Parity Error Lamp signal, LPEL4, is true, allowing the 1N662 diode to clamp the base of the 2N1304 at about -.7 volt, thus holding the transistor off. This allows only a very small current to flow from -24 volts, through the 330 ohm resistor, the indicator lamp, and the 1200 ohm resistor, to ground: this is not enough current to illuminate the bulb. When an error condition occurs, LPEL4 goes false, toward +6 volts. This drives the base of the transistor positive enough to turn it on, shorting out the 1200 ohm resistor and allowing enough current to flow through the lamp to turn it on.



SCHEMATIC
LAMP DRIVER

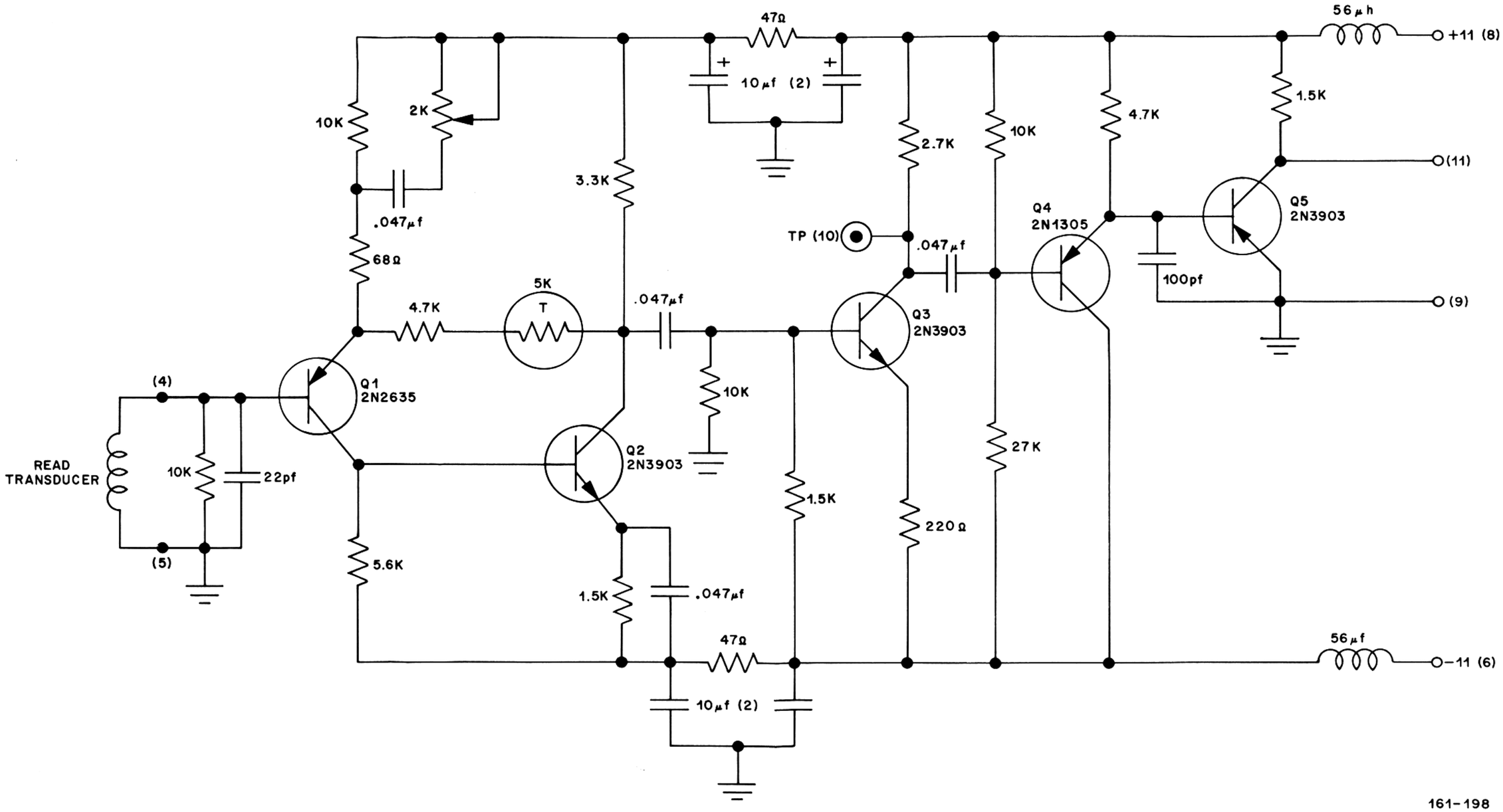
FIGURE 8-53

The small current through the 1200 ohm resistor when the transistor is off keeps the lamp filament warm, thus maintaining a higher resistance in the filament. This prevents a surge of current through a cold filament when the transistor first turns on, which could destroy the 2N1304. It also prevents a full 24 volts from being impressed across the transistor, and increases the life of the lamp.



6-VOLT AND 11-VOLT SUPPLIES
UNIVERSAL POWER SUPPLY

FIGURE 8-54



161-198

SCHEMATIC
READ AMPLIFIER

FIGURE 8-55

TP-161SL

TABLE OF CONTENTS

	Page
SECTION 9, SPECIAL/OPTIONAL FEATURES	
PROCESSOR SPECIAL FEATURES	9-1
Optional Decimal Positions	9-1
Check Protection	9-7
Address Modifier	9-8
Battery Option	9-12
2205 SPECIAL FEATURES	9-12
Miscellaneous	9-12
Magnetic Color Shift	9-12
Intermediate Margin Control	9-13
Front-Feed Posting Carriage	9-14
Automatic Vertical Spacer	9-14
Code Delete/Blank Tape Feed	9-14
Individual Tab Card Reader	9-15
AUXILIARY EQUIPMENT	9-16
2312-1 Auxiliary Tape Reader	9-16
2313-1 Auxiliary Card Reader	9-18
2314-1 "SELECTADATA" Selective Reader	9-20
2315-1 Auxiliary Tape Punch	9-23
2316-1 Card Punch Control	9-24
2323-1 Printing Card Punch	9-25
4600 AUXILIARY TEN-KEY KEYBOARD	9-29
(See separate Table of Contents)	9-29

SECTION 9, SPECIAL/OPTIONAL FEATURES

There are basically two groups of features covered in this section. The optional features are, as the name implies, available, but not necessarily included in all machines. Other features are included in all machines now being built, but are not available for some older machines. Still others are included in all 5610 auxiliary units, but they are considered "special" features because they are unique to the 5610 units and are not included in the "standard" auxiliary units.

The many special features are most easily classified into three groups: processor special features, 2205 special features, and auxiliary equipment.

The following list provides a quick reference to parts and drawings associated with the various features.

FEATURE NAME	KIT OF PARTS	INSTRUCTION DRAWING OR SERVICE LETTER #
PROCESSOR FEATURES:		
Check Protection	No Kit needed	Service Letter #2A
Optional Decimal Positions	No Kit needed	7000748 (SL)
Address Modifier	7000759 (SL)	7000756 (SL)
Battery Option	356140 (SL)	Service Letter #54
2205 FEATURES:		
Magnetic Color Shift	2023967 (R)	2000675 (R)
Intermediate Margin Control	7000762 (SL)	7000761 (SL) & 2023802 (R)
Front-Feed Posting Carriage	Factory Installed Only	See TP-700 RD
Automatic Vertical Spacer	2012816 (R)	2012876 (R)
Code Delete/Blank Tape Feed	Standard Feature	Service Letter #50A
Single-feed Tab Card Reader	2007667 (R)	2007668 (R)
AUXILIARY EQUIPMENT		
Auxiliary Ten Key Keyboard	7301014 (SL)	7301059 (SL)

A. PROCESSOR SPECIAL FEATURES

Each of the features in this group is contained wholly inside the processor cabinet and is not visually apparent. Two of these, the Optional Decimal Positions and Check Protection, are applicable to all machines. The Address Modifier and the Battery Option apply only to later machines.

1. OPTIONAL DECIMAL POSITIONS

Most 5610's are wired for the standard capacity of 8 digits left-of-decimal and 5 digits right-of-decimal. With minor modifications (changing diodes and jumper wires on a few cards), the decimal position can be shifted to provide LRD maximums of either 10/3 or 7/6. Instructions listing the specific diode and jumper changes required can be found on drawing no. 7000748, available from San Leandro.

Variations in LRD maximums affect machine operation in only a few instructions: Constants, Enter, Decimal Print, Non Decimal Print, Multiply, and Divide. During the two print instructions, the maximum left-of-decimal capacity affects the logic used to locate the first digit to be printed. Termination of the Non Decimal Print instruction with 0 right-of-decimal is altered by a change in the fixed decimal position. During all of these instructions, the "set-right-of-decimal" logic ("multiply-right-of-decimal" during Multiply) is affected by a change in the decimal position, causing the decimal register to be set to a different value once the DPS

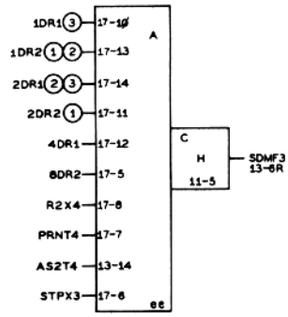
flip-flop becomes set. During Multiply and Divide the proper setting of the decimal register is necessary to obtain correct alignment of the answer. During Enter and the two print instructions, this setting is needed to allow the development of "decimal compare", so that the instruction can be terminated. The decimal compare logic is also altered by a change in the decimal position. The different setting of the decimal register also affects the termination of an Enter instruction through entry of a decimal point after the decimal position has been reached.

a. LOCATING FIRST DIGIT OF PRINTOUT. When a print instruction is retrieved, the number of digits to be printed left of the decimal point is set into the decimal register. The decimal register is then counted down in sync with the arithmetic column counter, and at the proper time a "binary ten" (the "non number") is written onto the arithmetic line in the column one greater than the first digit to be printed. The "proper time" is determined by a gate on the "EE" card, and is affected by the maximum left-of-decimal setting used. The Set DEMP Flip-flop signal, SDMF3, is developed at the proper time to initiate the shift left which inserts the "non number" into R1 at the correct position. Refer to Section 4H, Print, for review.

With the "standard" LRD maximum of 8/5, the decimal register is at 9 when SDMF3 is developed. If the 10/3 variation is used, SDMF3 is developed when the decimal register reaches 11. With the 7/6 option, SDMF3 is produced when DR=8.

The gate used to develop SDMF3 is shown in Figure 9-1. A combination of two of the four diodes noted is inserted to produce SDMF3 at the correct time.

MACHINE LEFT OF DECIMAL CHARACTERISTICS	DIODES TO BE USED
10	1
8	2
7	3

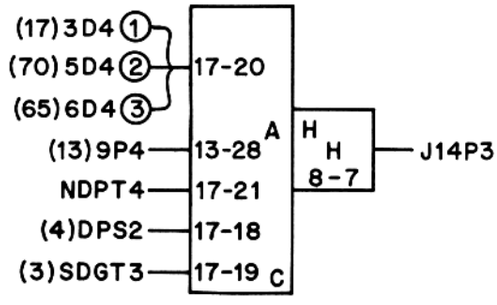


SETTING DEMP TO SHIFT "BINARY TEN" ONTO MATH LINE

FIGURE 9-1

b. TERMINATING NON DECIMAL PRINT WITH 0 RIGHT-OF-DECIMAL. When a Non Decimal Print instruction is retrieved which specifies 0 right-of-decimal, a jump occurs from 9 phase to 14 phase instead of a step through phases 10, 11, 12, and 13. This situation is covered in detail in Section 3H, Print, near the end of Part 8: refer to that location for more information.

If either the 10/3 or 7/6 option is used, the decimal register is set to 3 or 6, respectively, instead of 5, so the 3D4 or 6D4 signals must be used instead of 5D4. The gate affected is shown in Figure 9-2 for convenience.

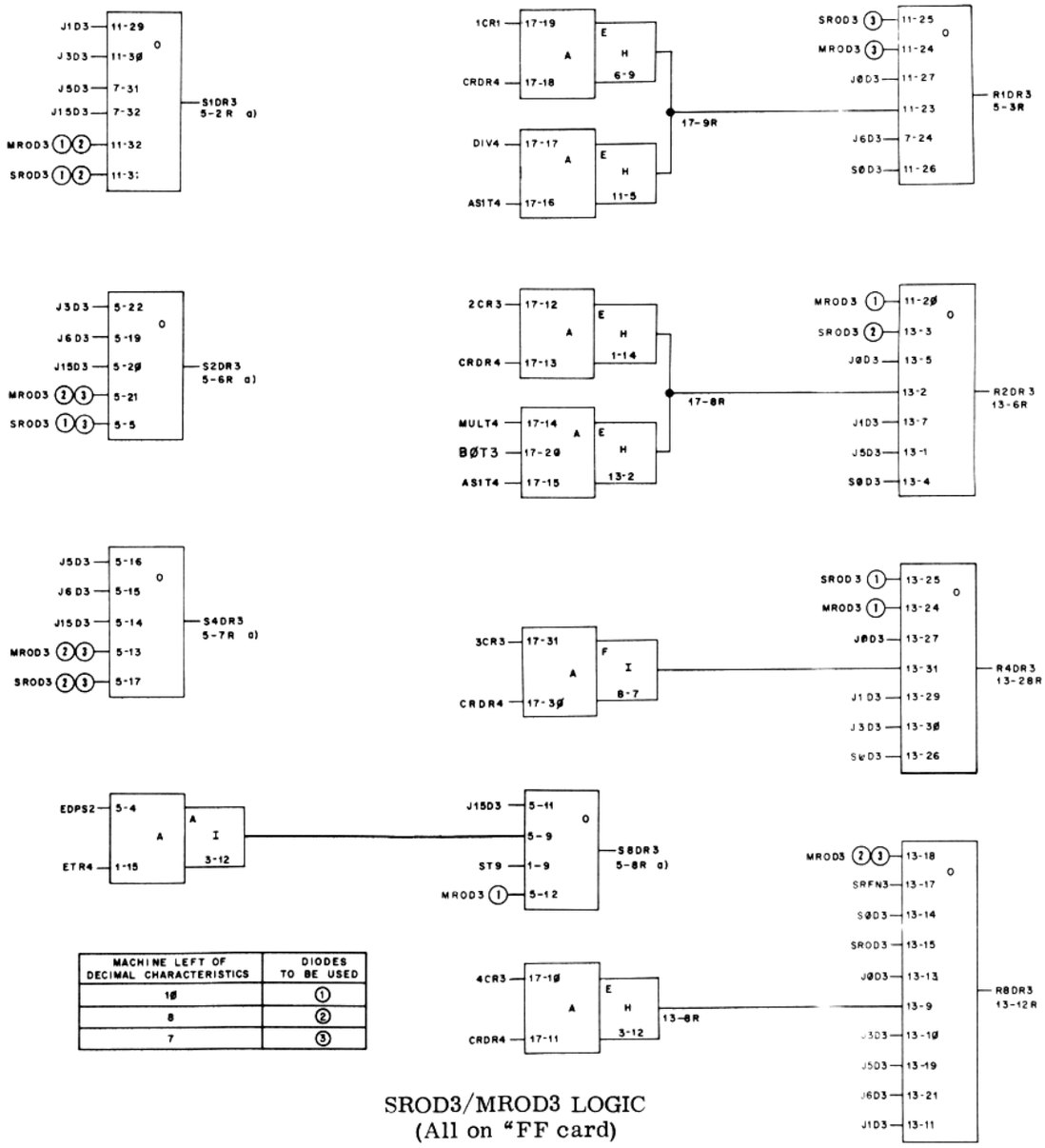


AFFECT OF DECIMAL POINT CHANGE ON NDP

FIGURE 9-2

c. SET RIGHT-OF-DECIMAL. As noted in the discussions of the Constant, Enter, and print instructions in Section 4, the decimal register is counted down (from the left-of-decimal setting specified by the address of the instruction) once for each digit entered or printed left-of-decimal. When the decimal register is counted down to zero, or when a decimal point is entered, the Set Right Of Decimal signal, SROD3, is produced, which sets the decimal register to the maximum allowable right-of-decimal setting. The decimal register is then counted down further until DCMP3 is developed or until the entry is terminated in some other way.

In the "standard" LRD maximum of 8/5, SROD3 sets the decimal register to 5. In the 10/3 or 7/6 maximums, SROD3 sets the decimal register to either 3 or 6 respectively. The changes from one system to another in this regard are very simple, as shown in Figure 9-3. It is only necessary to add or remove the proper diodes on the "FF" card, as illustrated.



MACHINE LEFT OF DECIMAL CHARACTERISTICS	DIODES TO BE USED
1ø	①
8	②
7	③

SROD3/MROD3 LOGIC
(All on "FF card)

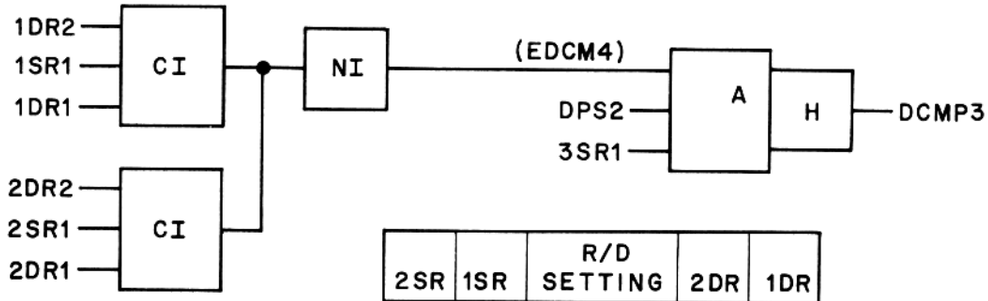
FIGURE 9-3

TP-161SL

d. MULTIPLY RIGHT-OF-DECIMAL. Figure 9-3 also shows the changes necessary when "moving" the decimal point to accomplish the correct decimal alignment of the product of a multiplication. In the "standard" 8/5 configuration, MROD3 sets the decimal register to 7. When the diodes are changed to either the 10/3 or 7/6 configuration, MROD3 sets the decimal register to either 9 or 6, respectively. Further information on the function of MROD3 can be found in Section 4L, Multiply or Divide. (Division uses SROD3 for alignment, so the changes to this signal automatically provide the proper alignment of the quotient.)

e. DECIMAL COMPARE. The basic purpose of the decimal compare logic is to signal when the proper number of digits have been entered or printed right-of-decimal. The proper number is that number specified by the address of the instruction. Further details concerning the function of the Decimal CoMPare signal, DCMP3, can be found in Section 4G, Enter, and Section 4H, Print. The "standard" decimal compare logic used with the 8/5 maximum LRD is explained in Section 8F, Decimal Compare, along with the electronic circuitry involved. The discussion here will be confined to those aspects of decimal compare associated with the 10/3 and 7/6 LRD maximums.

(1) 10/3 Maximum LRD. All of the logic necessary for the development of DCMP3 with a maximum of 3 digits right-of-decimal is shown in Figure 9-4. 1SR and 1DR must be in the opposite logic state, as must 2SR and 2DR. This is reflected in the truth table. Note also that if 3SR is set, calling out a right-of-decimal setting of 4, it indicates that the possibility of entering a sign exists. In this case, DCMP3 cannot be developed, and the entry MUST be terminated with a space, dash, or some other non-numeric character.



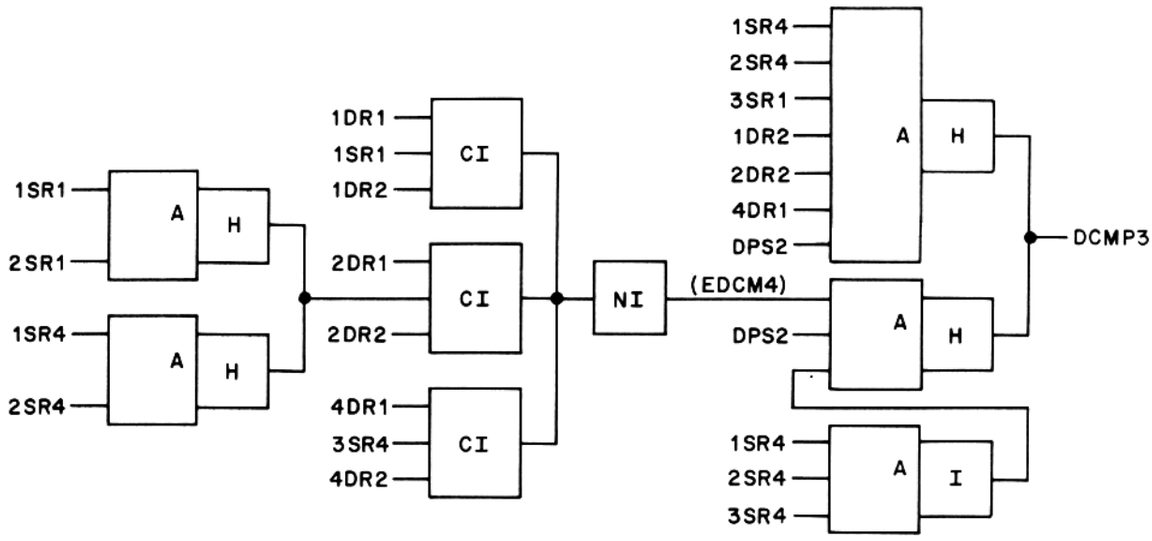
2SR	1SR	R/D SETTING	2DR	1DR
0	0	0	1	1
0	1	1	1	0
1	0	2	0	1
1	1	3	0	0

DEVELOPMENT OF DCMP3
(R/D = 3 Maximum)
(All on "EE" card)

FIGURE 9-4

(2) 7/6 Maximum LRD. The logic required to develop DCMP3 with a maximum of 6 digits right-of-decimal is slightly more complicated, as shown in Figure 9-5. The upper CI dictates that 1SR and 1DR must be in the same logic state. The second CI requires that 2DR be set if 1SR and 2SR are in the same logic state, and 2DR be reset if 1SR and 2SR are different. The third CI requires 3SR and 4DR to be in the opposite logic state.

The three CI's cover all possible combinations between zero and 6 digits right-of-decimal with the exception of 3: note that with 3 digits right-of-decimal, both 3SR and 4DR are reset. This leaves the output of the lower CI false. To provide DCMP3 in this case, the large AND gate was added. If both the search register and the decimal register contain a 3, the output of the AND gate will provide DCMP3.



3SR	2SR	1SR	R/D SETTING	4DR	2DR	1DR
0	0	0	0	1	1	0
0	0	1	1	1	0	1
0	1	0	2	1	0	0
0	1	1	3	0	1	1
1	0	0	4	0	1	0
1	0	1	5	0	0	1
1	1	0	6	0	0	0

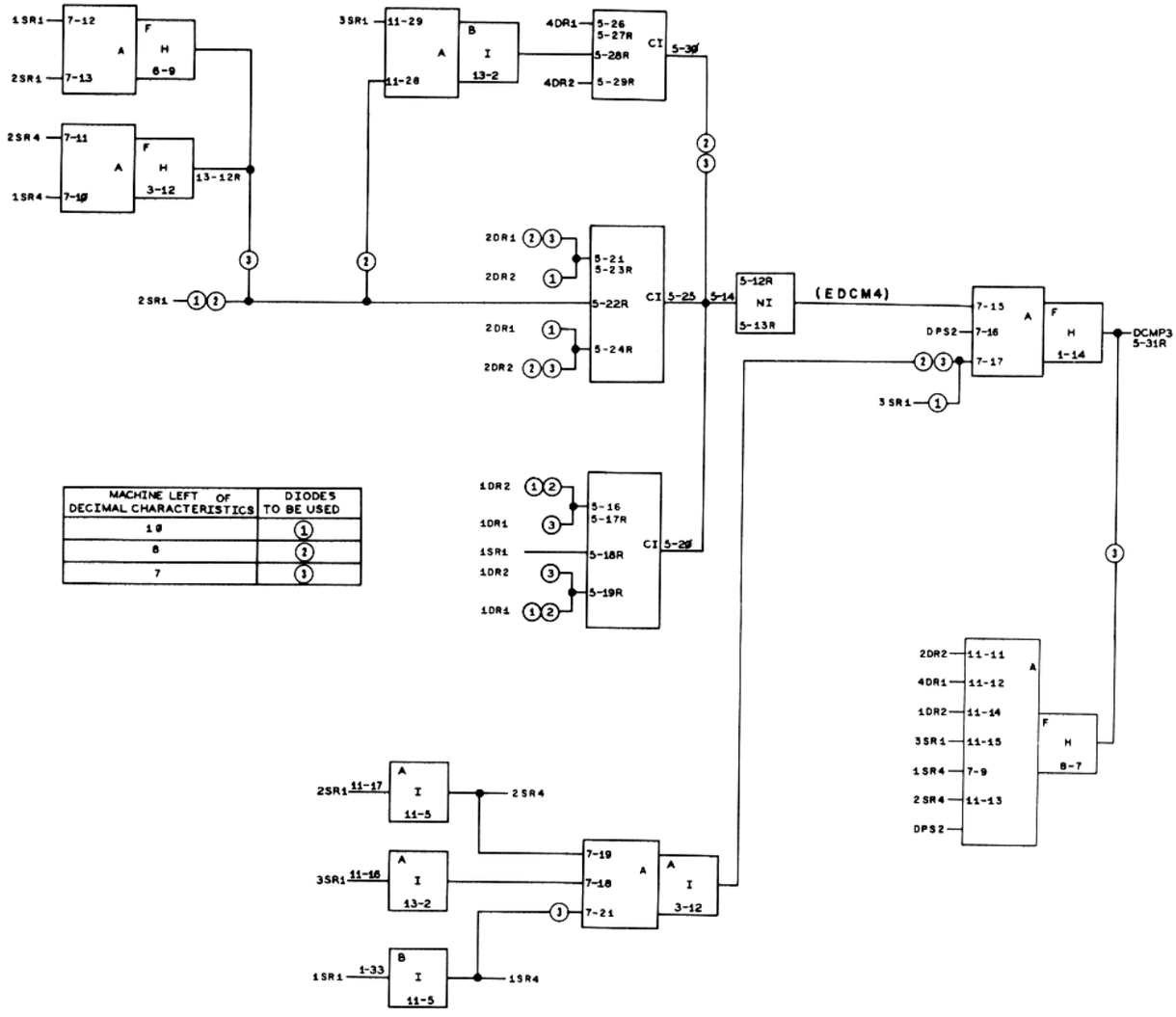
161-209

DEVELOPMENT OF DCMP3
(R/D = 6 Maximum)
(All on "EE" card)

FIGURE 9-5

Last, note that DCMP3 is inhibited if the search register contains a 7, allowing for the entry of a sign.

(3) Organization. All of the logic required for the development of DCMP3 for the three different right-of-decimal maximums is illustrated in Figure 9-6. When changing from one system to another, diodes and jumper wires must be removed from and added to the "EE" card according to the numbered circles shown.

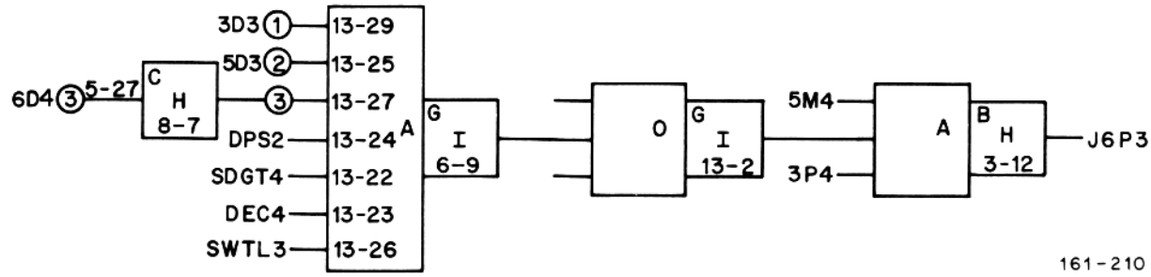


COMPLETE DCMP3 LOGIC
(All on "EE" card)

FIGURE 9-6

f. TERMINATING ENTRY WITH DECIMAL. It was noted in Section 4G, Enter, that entry of a non-numeric, including a decimal, after the decimal position had been reached would cause the entry to be terminated. If the decimal is entered right AT the decimal position, however, it has no effect: this is to allow typing of a decimal point after the decimal position has been reached by filling the left-of-decimal requirement with digits.

The fact that the decimal position has been reached is indicated by the fact that the DPS flip-flop is set. Therefore, if a decimal point is entered with DPS set, it causes termination of the entry, unless the decimal register is still at its maximum right-of-decimal setting. With the "standard" 8/5 maximum LRD, 5D3 is used to block the gate giving the jump to 6 phase which terminates the entry. If the 10/3 LRD is used, 3D3 must block this gate instead, and if the 7/6 LRD maximum is used, 6D3 is used. This is illustrated in Figure 9-7.



MACHINE LEFT OF DECIMAL CHARACTERISTICS	DIODES TO BE USED
10	1
8	2
7	1

TERMINATING ENTRY WITH DECIMAL
(All on "N" card)

FIGURE 9-7

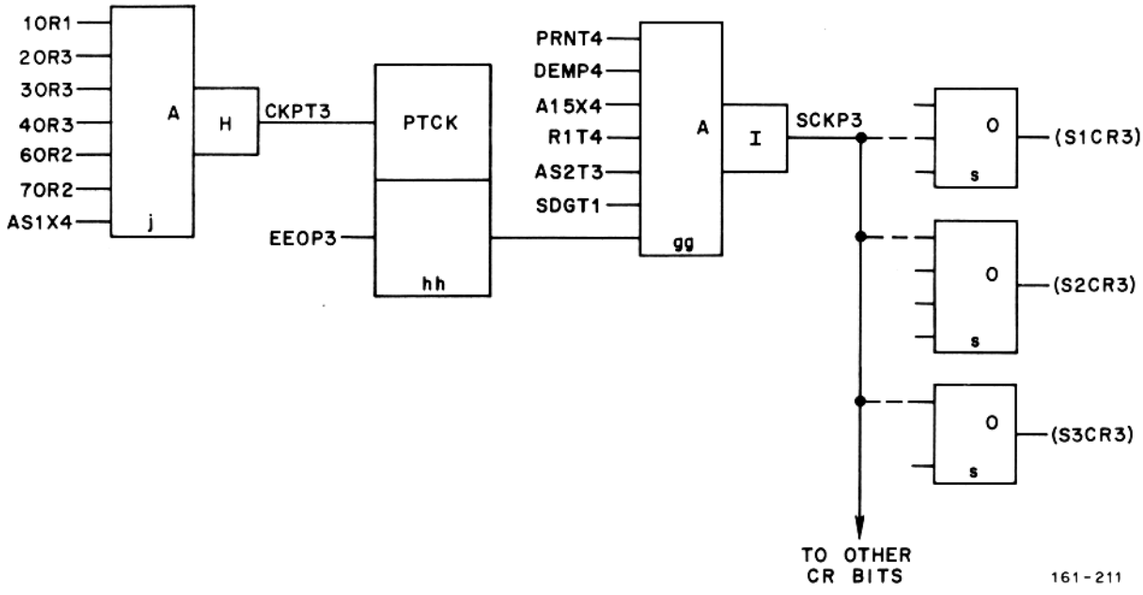
2. CHECK PROTECTION

During execution of either of the two print instructions, the Check Protection option allows a preselected character to be typed in place of the spaces representing insignificant zeros. Most of the hardware required for this option is built into all 5610's: it is only necessary to install diodes in the proper positions on the "S" card to select the protection character desired. Refer to 5610 Service Letter #2A for details.

a. GENERAL OPERATION. In order to "set up" the check protection function, one of the two print instructions must be preceded by a Check Protection instruction. Retrieval of the Check Protection instruction sets the ProtecT CheckK flip-flop, PTCK. During the subsequent execution of the print instruction, the fact that PTCK is set causes the check protection character to be printed instead of a space until the first significant digit or the decimal point is printed. The termination of the print instruction automatically resets PTCK, so a separate Check Protection instruction must precede each print instruction during which check protection is desired. If the print is not preceded by a Check Protection instruction, operation will be normal, with insignificant zeros causing spaces to print.

b. CHECK PROTECTION INSTRUCTION. Retrieval of the Check Protection command (ampersand, 5-6-7 code) does not develop any Op Decoder outputs immediately. Since no special signals at all are developed, the processor follows the course for normal 0-address commands and jumps to 5 mode 15 phase. Since no signals have been developed to block the reset of DOST, the action step counter unlocks and steps to 1. As soon as action step 1 is reached, the CheckK ProtecT signal, CKPT3, is developed, which sets PTCK. See Figure 9-8. The action step counter then steps to 2 and returns to 0. The phase counter follows its usual path for "arithmetic" instructions, jumping to 6 phase after DOST resets, and stepping to 7 phase after DOST again sets. At program line home the jump to 3M6,4 initiates the search for the print instruction.

c. PRINT INSTRUCTION. All aspects of the Decimal Print or Non Decimal Print instructions are the same as covered previously in Section 4H, Print, with the addition of the logic shown in Figure 9-8. Each time a zero is retrieved from the arithmetic line in action step 3, Set CheckK Protection, SCKP3, is developed to set the desired protection character into the common register. The DEMP flip-flop is set during the shift left that retrieves the character from the arithmetic line, and if SDGT is reset at the end of this shift-left line pass (A15 time), SCKP3 is developed. If some digit other than zero is retrieved from the arithmetic line, SDGT sets at A15R0 time, blocking the development of SCKP3. Once SDGT sets, it remains set until the instruction is terminated, thus preventing the output of any more check protection characters.



CHECK PROTECTION LOGIC

FIGURE 9-8

SCKP3 is wired to the set inputs of all common register flip-flops, and the code to be printed is selected by inserting diodes into the proper positions on the "S" card, as explained in 5610 Service Letter #2A. Note that if no diodes are inserted, a space will still occur for each insignificant zero, since the 5-bit is set in 10 phase during the print instruction. Note also that left-justified printouts can be obtained by selecting a non-escaping code for the check protection character.

When the print instruction is terminated, the End of Entry or OutPut signal, EEOP3, resets PTCK (and SDGT), requiring another Check Protection instruction before the check protection feature can operate again.

3. ADDRESS MODIFIER

The address modifier feature of the 5610 enables the programmer to address storage registers indirectly: the address of a Store, Recall, or Accumulate instruction is obtained from the operating stack instead of from the program line. All machines above serial number 1940 have the address modifier feature installed, and the feature is optional on machines between 1020 and 1940. It cannot be installed on machines prior to serial number 1020.

a. PROGRAMMING CONSIDERATIONS. When the programmer wishes to perform a storage instruction utilizing an indirect address, he merely loads the instruction on the program line with an address of 99. Later, when this instruction is retrieved, the address modifier takes over and replaces the 99 with the two digits immediately left-of-decimal in R1 (or R2, depending upon the instruction).

Since this address is located in the operating stack, it can be modified by the programmer at will. For example, the following program could be used to clear all storage registers.

```
SEQuence          C
CONstant          1
DUPlicate
DUPlicate
TRUncate          F8
DUPlicate
RESume            A
STOre             99
ADD
DUPlicate
TRUncate          F8
Positive BRanch   A
```

A hardware check circuit insures that the indirect address will not be higher than 59: in the sample program above, when the indirect address is incremented to 60 and then retrieved from the arithmetic line, the check circuit detects this fact and causes removal of the marker and termination of the sequence.

During Recall the indirect address is retrieved from R1. Then the data from the storage line is shifted into R0, and finally the stack is shifted up. During Store and Accumulate, the data to be stored or accumulated is in R1, so the indirect address is obtained from R2 instead. Note that the indirect address is in the opposite register upon completion of the instruction: after a Recall, the indirect address is in R2; after a Store or Accumulate it is in R1.

Note that the register containing the indirect address can contain data other than the address, but only the two digits immediately left-of-decimal are checked when retrieving the indirect address.

b. GENERAL OPERATION. The instruction and its "99" address are retrieved from the program line in 4 mode. As soon as the "99" is retrieved, it causes a jump to 6 mode. Here the indirect address "tens" digit is retrieved from the arithmetic line (shifted into the common register) and then transferred to the decimal register. The indirect "units" digit is shifted from the arithmetic line into the common register, and from there it is transferred to the search register. (The "units" digit is actually retrieved first, since the arithmetic line looks at the LSD first.) This process utilizes phases 9, 10, and 11 in 6 mode. A jump occurs to 5 mode and then to 15 phase, and the balance of the operation is the same as for a regular Store, Recall, or Accumulate instruction.

c. COMMAND AND DIRECT ADDRESS RETRIEVAL (Refer to Figure 9-9). The original retrieval of the instruction from the program line is the same as retrieval of a regular Store, Recall, or Accumulate instruction up until the second address digit is retrieved in 4 mode 7 phase. The "tens" digit is retrieved in 6 phase and set into the decimal register. Then, in 7 phase, if the decimal register contains a 9, a jump to 6 mode 9 phase is enabled. Once in 6 mode 9 phase, the PLCR, CRPL line routing needed to maintain the program line data on the line is continued until program line home, when a step to 10 phase takes place. Note that ECTR3 for this step to 10 phase is obtained through the standard 6 mode and PLH gates (K card) used in Special Flexowriter Functions.

d. INDIRECT ADDRESS RETRIEVAL. All of the address modifier action in 10 phase is governed by the Address Modifier Counter, a scale-of-eight counter that operates as a "sub-phase" counter within 10 phase. During one arithmetic line pass, this counter is triggered each R0 time, so that its settings are related to the arithmetic column times A2 through A9. Referring again to Figure 9-9, note that the Single Cycle Modifier Flip-flop, SCMF, is set before 6 mode is reached. At 6 mode, 10 phase, A2R2 time, the Address Modifier "A" flip-flop, AMA, sets, and at each succeeding R0 time, Pulse Address Modifier Counter, PAMC3, is developed. The AMC is then stepped once for each arithmetic column time, beginning at A3R0. When the AMC returns to its original state from its "A9" state, it resets AMA so no further pulsing of the AMC can occur. AMA cannot set again because the first pulse on PAMC3 resets SCMF, insuring that only a "single cycle" will occur.

When AMA first resets, it allows the development of the Modified CRSR line routing signal, MCRSR3. This signal enables the transfer of data from the common register to the search register. This transfer cannot occur, however, until Modified SHift signals, MSH4, are provided (via inverter B8-7) to trigger the actual shifting of data. This does not occur until arithmetic columns 7 and 8, at which time PDLG3 is developed (at the proper R-time according to the instruction in the operation register).

With PDLG3 true during R1 or R2, A7 and A8, the indirect address is shifted from the arithmetic line into the common register, LSD first. Since the common register was clear to begin (via ARCR3, inverter D13-2), the pulses on MSH4 that occur during A7 time shift the clear condition of the common register into the search register. In A8 time, as the MSD of the indirect address is shifted into the common register, the LSD is shifted from the common register into the search register (MCRSR3). (Note that MDW3 is true throughout 10 phase so that all arithmetic line data is recirculated direct, with no changes, even though PDLG3 is used to shift data into the common register. MDW3 drives WRTD4 false on the "JJ" card.)

PDLG3 goes false after A8 time, so no further shifting of data occurs from this source. Note, however, that after the four shifts in A8, the LSD digit is in 7SR, 6SR, 4SR, and 3SR, and two more shift pulses are necessary in order to align the digit properly in the search register. These shift pulses are produced by inverter F3-12 during A9R0B1&2 and A9R1B1&2, and since MCRSR3 is still true (AMA is still set), the LSD digit is aligned. With PDLG3 now false, the CRSP flip-flop does not set and the common register is not shifted.

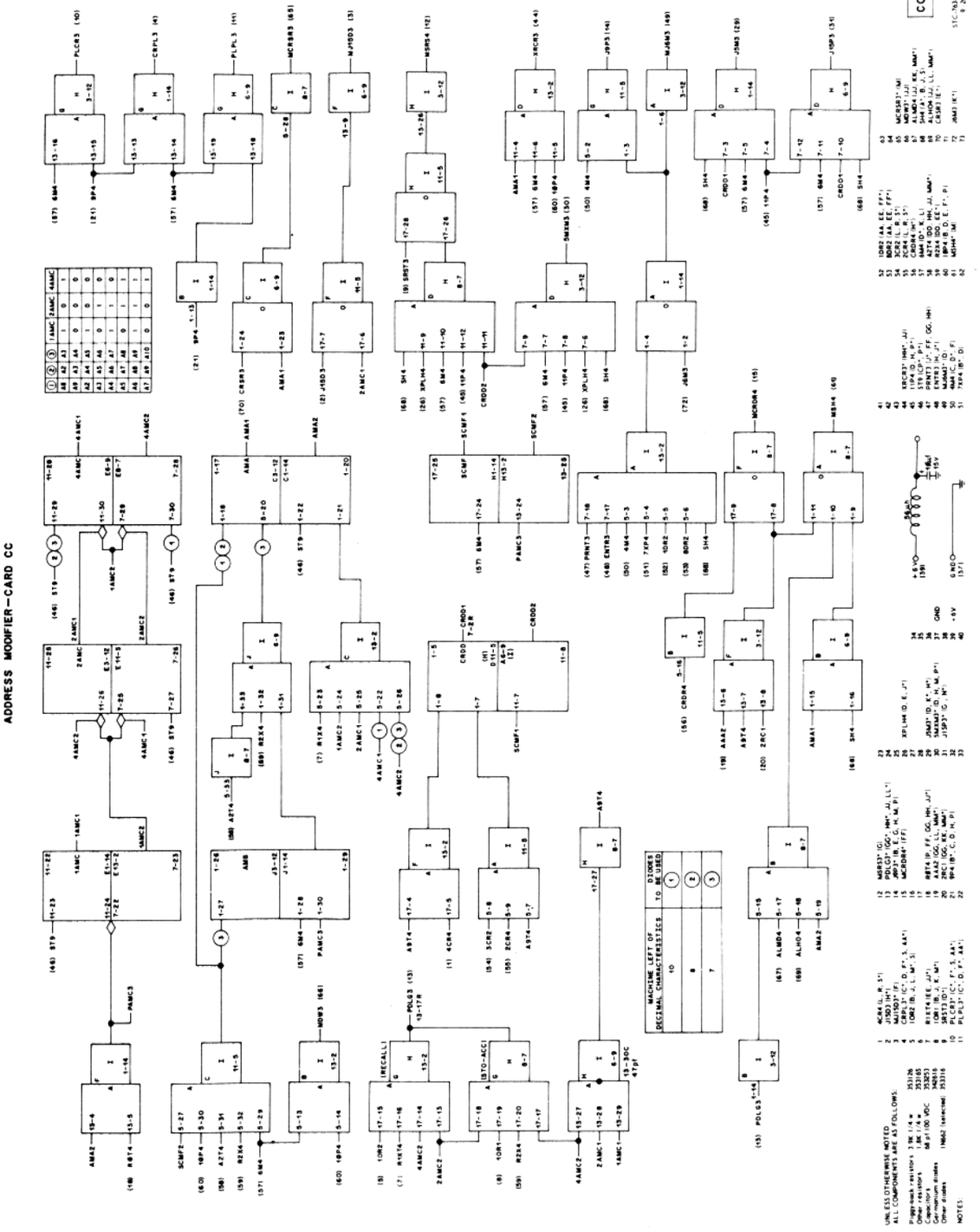
The same gate (F3-12) that provides the last two shift pulses to the search register, also provides two pulses on MCRDR4 (only one is needed), which allows the MSD digit to be strobed from the common register into the decimal register. (The decimal register was set to 15 by MJ15D3 during A5, A6, A7, and A8.)

One other operation occurs in A9 time: the check to detect an MSD that is > 5. Should such a condition be sensed, the Common Register Detect Digit flip-flop, CRDD, sets. The condition of CRDD is checked later in 11 phase to determine whether to go on with the storage instruction or not. Inverter A11-5 sets CRDD if the common register contains a 6 or a 7 and inverter F13-2 sets CRDD if the common register contains an 8 or a 9.

As the arithmetic timing chain steps to A10, R0 triggers the AMC, which returns to its original ("A2") state. This resets AMA, and with SCMF set, no further arithmetic line action occurs. Nothing else happens until program line home, when ECTR3 is produced and the phase counter steps to 11.

e. RETURN TO REGULAR OPERATION. 11 phase remains true for another program line pass, during which time the jump to 5 mode occurs and the jump to 15 phase is set up. At the end of this program line pass, ECTR3 is again produced for the jump to 15 phase, and the balance of the storage instruction occurs the same as a directly-addressed instruction. Note, however, that if CRDD is set, the jumps to 5 mode and 15 phase are blocked and 5MXM3 is developed instead, which produces jumps to 3 mode (D card), 6 decimal (H card), and 4 phase (H card), develops ECTR3 (P card), and resets the search register (M card). MSRS3 is also developed, which sets the RST flip-flop. These two factors then join to search out and destroy the marker, resulting in the 5610's dropping out of program.

f. OPTIONAL DECIMAL POSITIONS. The truth table in Figure 9-9 illustrates the state of the AMC in each arithmetic column for each of the three optional decimal point positions. In the "standard" decimal position, with an 8/5 maximum LRD pattern, 2AMC and 4AMC are set during arithmetic columns 7 and 8, providing PDLG3 so that the indirect address digits can be retrieved from the arithmetic line in these columns. With the 10/3 configuration (option #1), 4AMC starts out reset, causing the AMC to start out in the 100 state. This drives PDLG3 true during columns 5 and 6.



ADDRESS MODIFIER LOGIC

FIGURE 9-9

In the #3 option, that of 7/6, the counter starts out in the same state as in the 8/5 "standard" configuration, but the first counting pulse on PAMC3 is delayed until A3 time, providing a count one column "behind" the standard. This is accomplished by inserting diodes 1-27 and 5-20, thus introducing the AMB flip-flop into the circuit. At A2R2 time, AMB sets instead of AMA, and at the following R2 time (A3R2) AMA sets. This provides the first PAMC3 pulse at A4R0, which resets both SCMF and AMB, in addition to incrementing the AMC. The AMC continues to count once each R0 time until it returns to its 101 configuration, which occurs at A11R0. In this way, both 2AMC and 4AMC are set in A8 and A9 time, developing PDLG3 to retrieve the indirect address digits from these two columns.

g. LONGER DELAY LINES. The longer delay lines (6.05 ms, ± 150 μsec each) must be used for the program line when the address modifier is used, to insure that program line home will be long enough to reliably accomplish the retrieval of the indirect address from the arithmetic line before the step to 11 phase takes place. The step from 6 mode 9 phase to 10 phase normally occurs at program line home. ECTR3 is immediately developed for the step to 11 phase, which occurs at the first T6B after the timing chain restarts, and the retrieval of the indirect address must be completed before the step to 11 phase. Machines below serial number 1875 have shorter (5.5 ms) delay lines, so if the address modifier is added to these machines, the longer delay lines must also be added. With a long program loaded on the shorter lines, program line home is not long enough to allow complete retrieval of the indirect address.

4. BATTERY OPTION

The battery option can be installed in any 5610 beginning with serial number 1909. It allows input voltage dropouts to occur for a minimum of six minutes out of every 8-hour operating period without affecting delay line data. Although six minutes is the maximum guaranteed time, under optimum conditions it is possible for data to remain intact for a longer period. Note, however, that the 2205 does not operate and that neither input nor output functions can be performed during these power dropouts. Only the 6-volt and the two 11-volt power supplies are maintained during the dropouts, so any function dependent upon the other power supplies is disabled.

The battery option consists basically of four 6-volt batteries and a "DC" printed circuit card. Two batteries are used for each of the 11-volt supplies. The "DC" card is installed in the Universal Power Supply in place of the "AC" card. It contains the battery chargers and the circuitry necessary to switch the batteries in and out of the circuit.

The operation of the battery option is covered in Section 8A, Part 1, the Universal Power Supply. Instructions for field installation are found in 5610 Service Letter #54.

B. 2205 SPECIAL FEATURES

Most special features available on the 2201 FLEXOWRITER writing machine are also available on the 2205. Many of these, such as type slug changes, pin feed platens, etc., require no special considerations when used on the 2205. Others, however, such as the Intermediate Margin Control, require arc suppression changes and/or different program panel patching.

The 2205 special features are not described in detail here. Instead, only those aspects are discussed which pertain to the 5610.

1. MISCELLANEOUS

Most miscellaneous 2201 optional features and accessories, such as type slug changes, Flexofeeds, forms stands, etc., are also available on the 2205. The more simple of these are no different when used on the 2205.

2. MAGNETIC COLOR SHIFT

The magnetic color shift used on the 2205 is identical in operation to that used on the standard 2201 FLEXOWRITER writing machine. Field installation is simplified, however, because all

mounting holes are predrilled in the manufacture of the 2205. Arc suppression (a "back-to-back" rectifier) is also included in the 2205 version of this feature. Kit #2023967 contains all necessary parts, and instructions are included in drawing #2000675. Both are available from Rochester.

The color shift solenoid is wired to patch point R20. It must be held energized during the printing of all "red" characters, so a selector relay is employed in the programming. Normally, the selector relay is programmed to energize upon operation of a 2205 function key (sometimes with other qualifying conditions), so a Type F-- instruction is required to initiate the color shift. Any number of various 2205 functions can be employed to drop the selector relay.

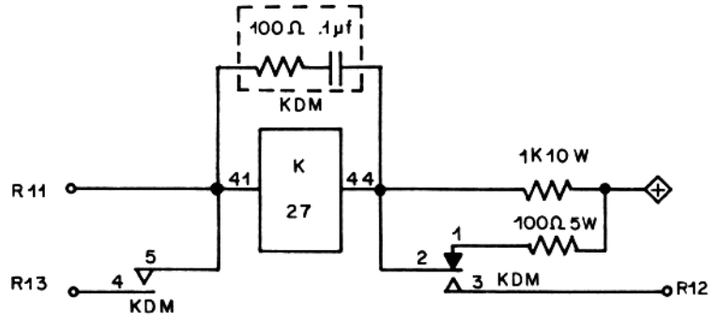
A typical example of color shift programming can be found in Volume DP11.1 of the Branch Office Library, Data Processing Section, on pages 1-14-1 through 1-14-5. This is Utility Subroutine no. 1-14, "Color Shift on Credit Printouts", prepared by Systems Engineering.

Mechanical adjustments for the 2205 color shift are the same as for the standard 2201 color shift. Complete adjustments will be found in the 2200/2300 Series Theory, Operation, and Adjustment Manual, TP-700 RD.

3. INTERMEDIATE MARGIN CONTROL

The intermediate margin control feature, sometimes referred to as the "double margin control" or "partial carriage return", decreases the time required to prepare written documents by providing a second left-hand margin. When the intermediate margin control relay is energized, a carriage return function returns the carriage to the intermediate margin stop, located on the margin rack to the right of the standard margin stop.

The mechanical operation of the 2205 intermediate margin control is identical to that of the 2201 version, but the electrical installation and programming is slightly different, as shown in Figure 9-10. The Double Margin relay, KDM, can be energized only by applying minus to patch hub R11. It can then be HELD energized by holding minus on either R11 or R13: the normal procedure is to use R11 as the "pick" and R13 as the "hold". With KDM energized and no minus applied to R11, the relay can be de-energized either by removing the minus at R13 or by applying minus to R12, to "knock down" the relay.



KDM RELAY WIRING

FIGURE 9-10

Mechanical installation instructions and adjustments are contained on Rochester drawing no. 2023802. All parts are included in kit no. 7000762, available from San Leandro, and wiring instructions are listed on San Leandro drawing no. 7000761.

4. FRONT-FEED POSTING CARRIAGE

The front-feed posting carriage option makes it possible to use cut or continuous journals, snap-out forms, ledgers, statements, and other forms combined and inserted in a variety of ways. It is available only as a factory-installed option, and cannot be installed in the field. It can be used with a Flexofeed, but cannot accept the Automatic Vertical Spacer.

Basically, the front-feed posting carriage allows for insertion of ledger cards or other forms from the front of the platen, on either the right or left side. A split platen allows for independent positioning of two different forms on the two sides, but the two platen sections can be locked together when desired. A Flexofeed can also be attached to accurately position continuous pin feed forms.

This feature is identical to that available on the standard 2201 FLEXOWRITER writing machine. Further information and adjustment/lubrication information will be found in the special features section of the 2200/2300 Series Theory, Operation, and Adjustment Manual, TP-700 RD.

5. AUTOMATIC VERTICAL SPACER

The Automatic Vertical Spacer (AVS) is a forms feeding attachment for the 2205 writing machine. It provides a more rapid means of feeding continuous multiple forms through the writing machine, while holding them in proper registration. The AVS cannot be installed on the Front-Feed Posting Carriage.

The AVS can be termed an "electric Flexofeed", because its basic operating principles are the same as those employed in the standard Flexofeed, yet it is driven by an electric motor. It can be installed in the field, but requires a special wiring harness, new carriage end covers, a new pressure release lever, and other parts.

Programming information is contained in Data Processing Release #8283, dated 11-5-68. A special crimping tool (T18474) is required to program the format control. This tool should be ordered from Rochester.

Installation, operation, and service information on the AVS is contained in the 2200/2300 Series Theory, Operation, and Adjustment Manual, TP-700 RD. The installation custom kit, #2012816, should be ordered from Rochester for AVS installations that are not part of the original machine sale. The AVS itself, #2012812, is not part of the kit, and must be provided by the sales department.

6. CODE DELETE/BLANK TAPE FEED

This is not an optional feature, but is standard on all later 5610's. Beginning with serial number 207536, all 2205 writing machines contain the "Code Delete/Blank Tape Feed" feature. This includes 5610's beginning with serial number 2460, except for a few units either side of this number, which may or may not contain the feature. Prior units have the dual-function 7-channel tape feed/code delete.

This feature allows the programmer to choose between 7-channel tape feed and blank (feed hole only) tape feed. If no special patching is installed, depression of the Tape Feed switch will cause punching of the feed hole only. Note, however, that a prior punching error removes minus from the Tape Feed switch, and the KPE relay must be dropped before tape feed operation can occur.

When using standard tape feed, Program Modification Switch No. 4 is used to add the 8 bit to any code punched. When using blank tape feed, PMS4 is used to punch a single 7-channel delete code, and PMS3 is used for the "add 8" feature.

All subject machines should use program panels patched to one of the following lists:

TP-161SL

a. Standard Tape Feed.

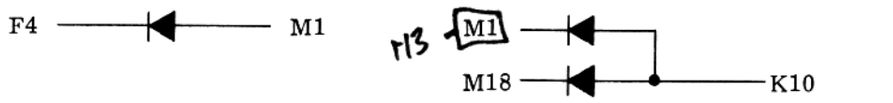
PMS4 used for "add 8".
Tape feed equals bits 1 through 7.
Tape feed initiated by Tape Feed switch.

Install:		
	A1-D4	M9-M10
	D4-M20	M11-M12
	F4-K10	M13-L20

b. Code Delete/Blank Tape Feed.

PMS4 used for code delete.
Code delete equals bits 1 through 7.
PMS3 used for "add 8".
Blank tape feed initiated by Tape Feed switch.
KSL-1 required.

Install:			
	A1-D3 ✓	F3-K10 ✓	M11-M12 ○
	D3-N4 ✓ N14	L20-M13 don.	M16-Q4 ✓ Q14 ✓
	D4-M15 ✓	M5-M14 ✓	M18-Q5 ✓ ~~Q15~~ Q11 ✓
	E4-N5 ✓ ~~N15~~ N11 ✓	M9-Q3 ✓ Q13	M19-~~N3~~ ✓ N13 ✓



If a patch panel wired to either of the preceding lists is installed in a 2205 that does not have the new feature, tape feed operation will be normal, i.e., depression of the Tape Feed switch will cause punching of 7-channel tape feed and will reset the punch error condition. If a panel wired for a 2205 that does not contain this feature is installed in one of the newer machines, tape feed will be blank and will occur only if a punch error condition does not exist. (The patches in the former list must be installed for "standard" operation.) In either case, no other functions will be affected, and no detrimental action will occur.

The wiring changes incorporating this feature are included in schematic #4503990, beginning with revision D.

7. INDIVIDUAL TAB CARD READER

The 2205 edge card reader can be replaced with a single-feed tab card reader. This reader allows tab cards to be inserted one at a time for subsequent reading. It is normally used in conjunction with an auxiliary tape reader and a 2316-1 Card Punch Control. It is the same reader as that used on the 2201 or 2301 FLEXOWRITER writing machines. Note, however, that when installed on the 2205, the reader must be wired to revision 1 (or later) of schematic #4503290. Readers wired to revision 0 do not contain the arc suppression necessary for 5610 operation. The revision 1 reader can be identified by a "3TCR-02" stamped on the casting: revision 0 units are stamped "3TCR-01".

For field installation, custom kit #2007667 should be ordered from Rochester. This kit contains reader assembly #2020711 and kit of parts #2024297.

C. AUXILIARY EQUIPMENT

There are six auxiliary units that may be cable-connected to the 5610 system in order to provide greater input/output capabilities. These are as follows:

1. 2312-1 Auxiliary Tape Reader (tape only)
2. 2313-1 Auxiliary Card Reader (tab cards only)
3. 2314-1 SELECTADATA selective reader (tape only)
4. 2315-1 Auxiliary Tape Punch
5. 2316-1 Card Punch Control
6. 2323-1 Printing Card Punch (tape or edge-punched cards)

All but the 2314-1 may also be used with a standard 2200 or 2300 series writing machine with no major changes. However, in some cases patch wire changes must be made in the auxiliary unit when coverting from one type of writing machine to another. Consult the individual schematics for details.

Following is a list of drawing numbers, available from Rochester, representing the schematic diagrams for each of the auxiliary I/O units. No revision numbers are listed for the first five, because in these cases the revision should be used that represents the particular machine being serviced.

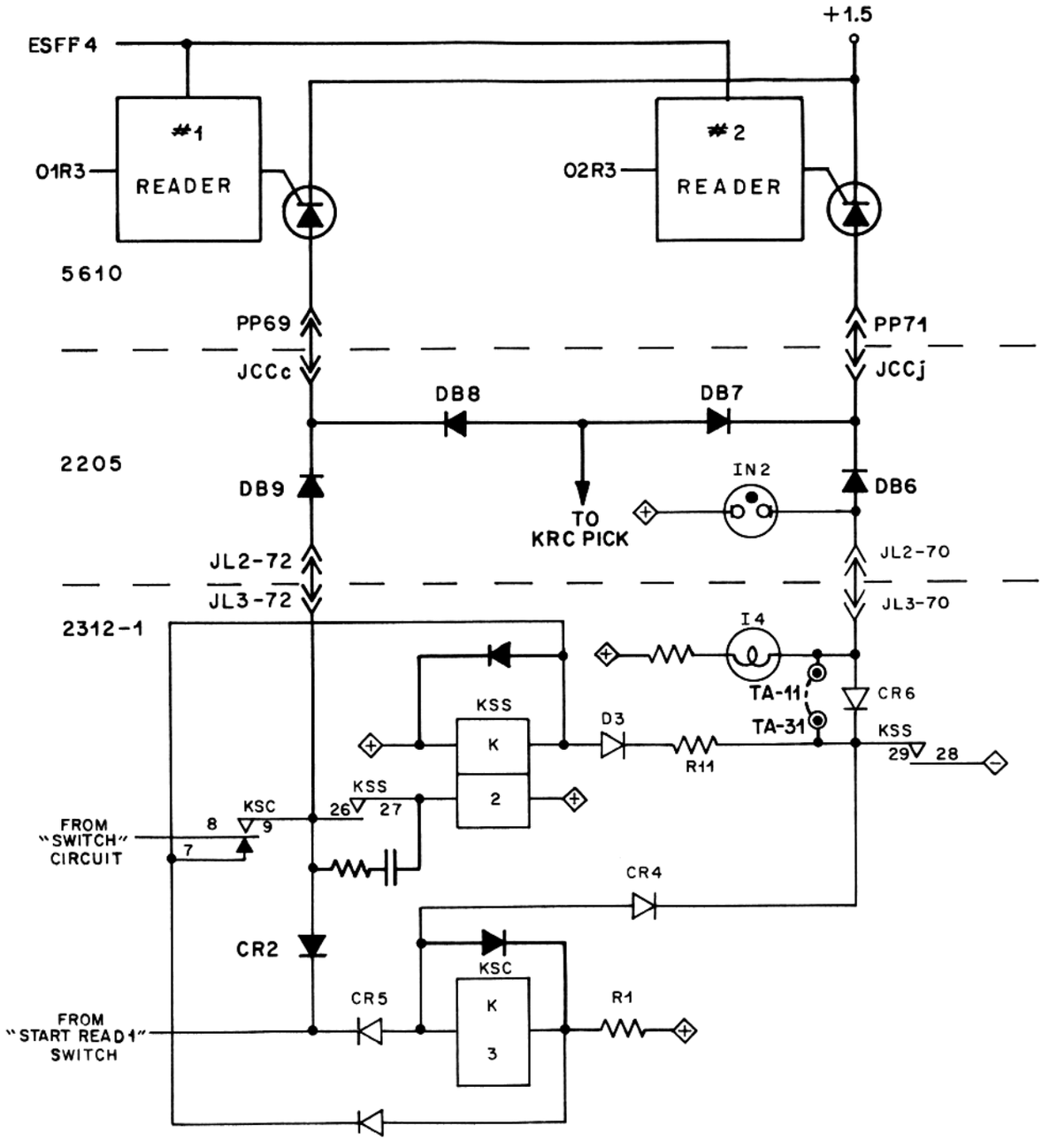
1. 2313-1: #4503593
2. 2313-1: #4503760
3. 2314-1: #4503585
4. 2315-1: #4503665
5. 2316-1: #4503675
6. 2323-1: #4504243, rev. D or later

In general, the 5610 auxiliary units have more arc suppression than the standard units. In most cases, relay coils in the 5610 units will have a diode or an RC network added around each relay coil, and an RC network added across each set of cam-operated contacts. The tape rewind motors in the 5610 units have a capacitor added, and often times the suppression across the clutch coil is different.

All 5610 auxiliary unit mechanical adjustments are the same as the corresponding adjustment in a "standard" auxiliary unit, and should be performed as outlined in the 2200/2300 Series Theory, Operation, and Adjustment Manual (TP-700 RD), published by the Rochester Division. All mechanical adjustments are the same, regardless of the type of machine on which the unit is used. (However, as stated earlier, certain patch changes are sometimes required when switching from the 2205 to the 2201 or 2301.) Those auxiliary units not yet covered in TP-700 RD will be added in the future.

1. 2312-1 AUXILIARY TAPE READER

Most of the 2312-1 wiring changes necessitated by the 5610 are shown in Figure 9-11. In addition to the diodes and arc suppressor shown, a diode is added across LRC and arc suppressors are added across SRCC1 and SRCC2. Diode CR2 is added to prevent the Reader 1 On instruction from holding up KSC, and the jumper wire between TA11 and TA31 is required to allow the Reader 2 On pulse from the 5610 to bypass CR6 and energize KSC and KSS.



Standard wiring and components

Added wiring and components for 5610 use

2312-1 PECULIARITIES

FIGURE 9-11

TP-161SL

2. 2313-1 AUXILIARY CARD READER

Aside from the addition of arc suppression diodes or RC networks around most relay coils and across various contacts, there are only minor differences between the standard 2313 Automatic Card Reader and the 2313-1 card reader used with the 5610. Minor wiring changes are required due to the "direct pick" outputs from the 5610.

Most of the wiring changes necessary for the 2313-1 are shown in Figure 9-12. First, diode D8 was added to prevent a back circuit from holding up KSC when the Reader 1 On instruction is executed. In this case, execution of the R1O instruction simply de-energizes ("knocks down") KSS, and KSC then drops as a result.

Second, the JOBK jumper must be patched to JOCC instead of to JOBU. This is to allow the Reader 2 On direct pick pulse from the 5610 to bypass CR1 and reach KSS and KSC. This is also shown in Figure 9-12.

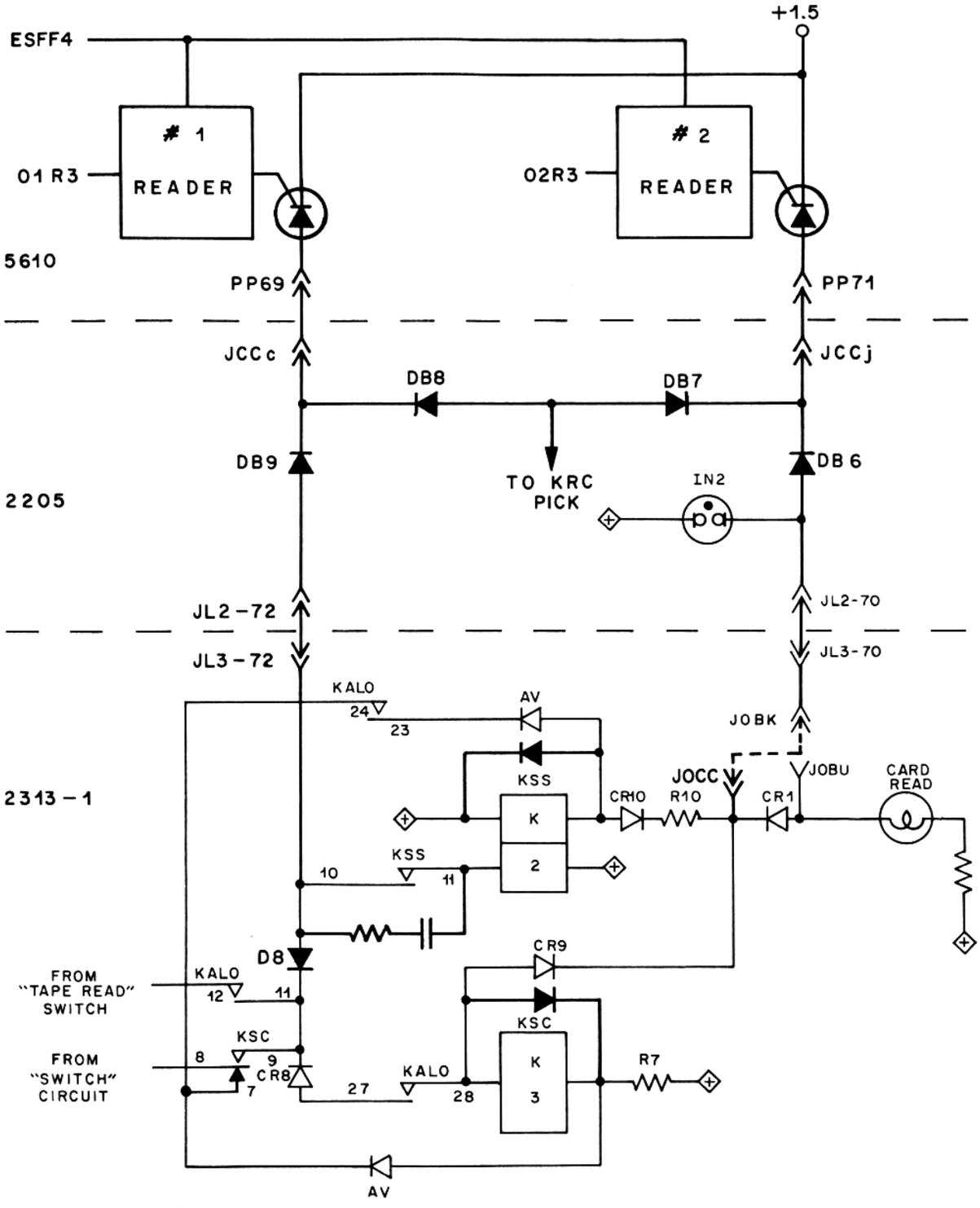

+1.5
ESFF4
1
READER
01R3
2
READER
02R3
5610
PP69
PP71
JCCc
JCCj
DB8
DB7
DB9
DB6
TO KRC PICK
IN2
2205
JL2-72
JL2-70
JL3-72
JL3-70
KALO
24
23
AV
JOBK
JOCC
JOBU
CARD READ
KSS
K
2
CR10
R10
CR1
2313-1
10
11
D8
CR9
FROM "TAPE READ" SWITCH
KALO
12
11
KSC
FROM "SWITCH" CIRCUIT
8
9
CR8
7
27
KALO
28
K
3
R7
AV
Standard wiring and components
Added wiring and components for 5610 use
2313-1 PECULIARITIES

FIGURE 9-12

3. 2314-1 "SELECTADATA" SELECTIVE READER

The 2314-1 SELECTADATA selective reader is quite different from the "standard" 2314 SELECTADATA reader. In addition to the usual 5610 requirement of more arc suppression, the 2314-1 also has the capability of entering data into the 5610 at the rate of 25 codes per second (referred to as "medium speed read"), and of searching for codes specified directly by the 5610 processor (as opposed to codes specified by the 2205). All other SELECTADATA reader functions are the same as in the standard 2314, although minor circuit variations and different relay contact numbers may be used: the basic principles of the other circuits are the same. The 2314-1 can NOT be used with a standard 2201 or 2300 series FLEXOWRITER writing machine.

a. "MEDIUM SPEED READ". Medium speed read gets its name from the fact that it is faster than normal reading speed, but slower than normal search speed. It is faster than normal reading speed because 2205 printing is disabled, and the reader does not have to "wait" for the translator. It is slower than normal search speed because each code must be read, instead of every second code. Printing is prevented by disabling the translator, and not by placing the 2205 in non print. (In fact, the 2205 must NOT be in non print when medium speed read is initiated: if this occurs, a circuit is established to the punch clutch, but not to any punch bits, and a parity error results.) Search speed is cut in half, even though the 2314-1 drive shaft rotates at full speed, because the Double Feed coil (LDF) is disabled. This allows every code to be read.

(1) Uses. Medium speed read can be used in any application requiring a large number of entries from tape. It is also used quite extensively during program load.

(2) Programming. Medium speed read can be initiated in either of two ways: by reading a 3-4-6-8 code in the 2314-1 or by pulsing patch point R17 in the 2205. In either case, the 5610 is still in control, and the Reader Run signal, RR3, must be true in order to complete the circuit to the Reader Clutch magnet, LRC. The reader can be started and stopped via RR3 while the medium speed read condition remains in effect. The medium speed read condition can be terminated either by reading an F1 (stop) code or by depressing the Stop Read switch.

Additionally, 2205 program board patching can prevent the F1 code from terminating the condition if it is so desired: in this way, any number of F1 codes may be used in the program being loaded without disabling the medium speed read condition. The usual procedure in programming is to have a selector relay energized along with the initiation of the medium speed read condition. Minus is then applied, through the selector relay contacts, to patch hub R15, which prevents the F1 code from resetting the medium speed read condition. This method, however, requires that depression of the Stop Read switch de-energize the selector relay to remove the minus from R15.

(3) Circuitry. There are six relays involved in the medium speed read function. Two of them, the Entry Block relays, KEB1 & 2, are located in the 2205, and serve to open all circuits to the translator clutch and bit magnets. The 2314-1 contains the other four relays, the DElay relay, KDE, two Entry relays, KE1 & 2, and the STop relay, KST. KDE is the first relay energized upon initiation of the medium speed read condition, and it serves to delay the reader until medium speed control has been established. KE1 & 2 are the main control relays for medium speed read. They provide the circuits to energize KEB1 & 2, drop KDE, and energize KST when the entry is completed. KST terminates the medium speed read condition by dropping KE1 & 2 and KEB1 & 2.

All of the circuitry involved in medium speed read is shown in Figure 9-16 at the end of this section. Refer to this figure for the following discussion.

(a) Initiating the Condition. All that is required to initiate the medium speed read condition is to energize the KDE relay. This is accomplished most easily by pulsing patch hub R17 in the 2205. The method used to do this is determined by the individual program. The circuit is as follows: patch hub R17, line 7 (JL2-7 in the 2205, through the desk, to JL3-7 in the 2314-1), DB55, to KDE coil.

KDE can also be energized by reading the 3-4-6-8 code in the 2314-1: minus on line 3, through SRCC2-2 & 3, KCO 32 & 31, KSC 5 & 6, SR1-4 & 5, SR2-4 & 5, SR5-4 & 5, SR4-4 & 3, SR6-4 & 3, SR7-4 & 5, SR3-4 & 3, SR8-3 & 4, DB30, TA19, jumper to TA18, to KDE coil.

KDE holds from minus on KE1 24 & 23 and KE2 22 & 21, through KDE 5 & 6, to KDE coil.

As soon as KDE comes up, its contacts 3 & 4 open the reader clutch circuit.

KDE *ax* contacts 1 & 2 close to provide a pick circuit to KE1 & 2: minus standing on KCO 30 & 29, through DB28, KDE 1 & 2, to KE1 & 2 coils.

KE1 & 2 seal through a set of normally closed KST contacts: minus at KSS 28 & 29, through KST 23 & 24, KE1 10 & 9, to KE1 & 2 coils. Note at this point that if KSS had not been energized previously (2205 had been in control), both KSS and KSC would pick through the KE1 & 2 pick and hold circuits: minus on KCO 30 & 29, through DB28, KDE 1 & 2, KE1 9 & 10, KST 24 & 23, through R2 and DB5 to pick KSS, and through DB1 to pick KSC. Subsequent KSS and KSC operation is "standard".

KDE drops as KE1 24 & 23 and KE2 22 & 21 open. KDE 3 & 4 remake, establishing the LRC circuit. At the same time, KE1 30 & 31 are closed, bypassing SRCC1.

In the case of reading the 3-4-6-8 code, KEB1 & 2 pick when SRCC2 restores. If patch hub R17 is pulsed, KEB1 & 2 pick as soon as KE1 comes up: minus on SRCC2-2 & 1, through DB13, KE1 28 & 29, line 5, to KEB1 & 2 coils in 2205.

If the 3-4-6-8 code is being read, KEB1 & 2 have a temporary hold circuit from minus on SRC 1 & 2, through DB14, KE1 28 & 29, line 5, to KEB1 & 2 coil. The other hold circuit is from KPE minus in the 2205, line 64, through the 2314-1 Stop Read switch, line 14, the 2205 Stop Read switch, KRC 32 & 31, CR10, line 11, DB14, KE1 28 & 29, line 5, to KEB1 & 2 coils.

(b) Operating in Medium Speed Read. With KE1 & 2 and KEB1 & 2 energized, the Reader Run signal from the 5610 allows the reader shaft to rotate at full speed until the entry condition is terminated. Since the Search relay, KS, in the 2314-1 is not energized, the Double Feed coil, LDF, is not energized, and the reader senses every code in the tape. KEB1 & 2 open the translator clutch circuit, preventing STC3 from interrupting the 2314-1 LRC circuit, so LRC de-energizes only when SRC contacts open. The Code Output relay, KCO, remains energized, providing the output from the SELECTADATA reader to the 5610 processor.

The 2314-1 reader clutch circuit is the same during medium speed read as it is during normal reading except for the KE1 contacts 30 & 31 used to short out SRCC1. Without these contacts, LRC would de-energize between 20° and 308°, allowing the clutch to latch up momentarily at home position. The normal translator clutch circuit on line 13 is opened by KEB1 22 & 21. Another potential circuit to LTC, through the R3-R4 patch (if installed) is opened by KEB2 1 & 2. With the translator clutch disabled, STC3-1 & 2 remain closed. This allows the reader shaft to run "free", with LRC being de-energized only when SRC-3 & 4 open. With this being the only qualification, the reader runs at 1500 codes per minute (25 codes per second), until a stop code is read, the Stop Read switch is operated, or the Reader Run signal is driven false.

With each code read, the FCB3 circuit (line 53) is enabled directly from the normal LTC circuit by KE2 8 & 7. The reader code contacts are also connected directly to line 53, so every code read is transferred to the processor via the logic on the 2205 interface card.

KEB1 contacts 23 & 24 prevent LT 3, 4, & 6 from becoming energized should an F2 (Print Restore) code be read while in medium speed read. (Do not confuse this with the 3-4-6-8 code that initiated the medium speed read.)

(c) Terminating Medium Speed Read. Medium speed read can be terminated either by reading a stop code or by depressing the Stop Read switch. It can be interrupted by satisfying the 5610 entry conditions, but in this case, RR3 only stops the reader clutch: the 2314-1 is still in the medium speed read condition. When RR3 is again established, codes are again entered at the 25/second rate.

With each code read while in the medium speed read condition, minus is applied to the coil of the STop relay, KST, through SRCC2-2 & 3, KCO 32 & 31, DB18, and KE2 9 & 10. The relay does not pick, however, because minus is also applied to the other side of its coil through a matrix of reader contacts which sense the 1-2-4 Stop (F1) code. Part of this matrix, also fed by SRCC2 and KCO 32 & 31, is through DB56, normally closed SR1, SR2, and SR4 contacts, and DB50. The other part senses bits in addition to the 1-2-4, using normally open SR3, SR5, SR6, and SR7 contacts, with a diode following each.

Thus each code read applies minus to one side of the KST coil, and every code read except the F1 code applies minus to the other side of the coil. KST energizes, therefore, only when an F1 code is read.

As soon as KST picks, it holds through 2205 minus on KEB1 29 & 28 and KEB2 29 & 28, line 4, DB35, KST 3 & 4, to the KST coil.

Also, when KST energizes, its contacts 23 & 24 open to drop KE1 & 2, and its contacts 25 & 26 close to knock down KRC in the 2205. KE1 28 & 29 open to interrupt one circuit to KEB1 & 2 coils. Another circuit is complete, however, through DB15 and KST 5 & 6, as long as SRCC2 is operated. When the SRCC2 contacts restore, KEB1 & 2 drop out, and this allows KST to drop out. This puts the 2314-1 back into normal operation with the KRC relay not energized. Some further action must then be taken in order to initiate reading.

During program load it is often necessary to load the F1 code into the 5610 without allowing it to terminate the medium speed read condition. When this is the case, it is common to use the Tape Skip switch to initiate medium speed read by patching it to hub R17. A selector relay is also picked by the Tape Skip switch, and its contacts are patched to provide a constant minus at patch hub R15. This constant minus prevents KST from picking even when the F1 code is read.

The only way to terminate medium speed read under these conditions is through depression of the Stop Read switch, which, of course, must also knock down the selector relay to remove the minus from hub R15. This entire process is usually selected through depression of a program modification switch on the 2205.

b. INITIATING SEARCH FROM 5610. It is possible for the 5610 processor to initiate a search in the 2314-1 for an address code specified by the 5610. The address code can come from either the program line or from R1, depending upon the type of output employed. The search is initiated as dictated by the 2205 program panel patching, usually by "typing" an F12 code. This energizes the Auto Search relay, KAS, in the 2314-1, thus setting up the conditions for the next code to be set into the code storage relays. The next code delivered by the 5610 thus becomes the address code for which the search is conducted. A typical program could be either of the following:

F11	Reader 1 On		F11	Reader 1 On
F12	AID		F12	AID
½	Type		T10	Non Decimal Print 1/0
X	Address			

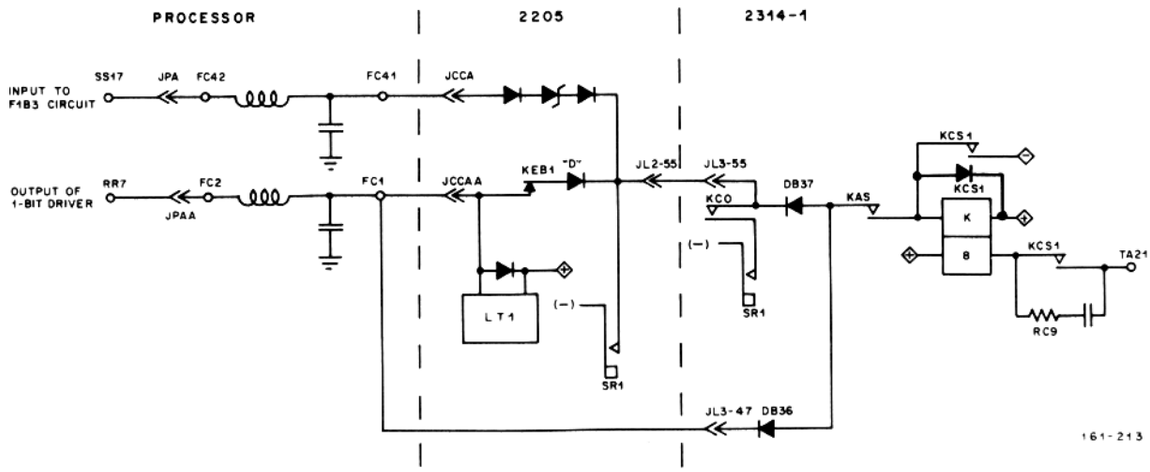
The F12 code (or other code designated by the 2205 patching) provides a pulse at JL3-35 which energizes KAID. As soon as this pulse is removed, KAID1 & 2 provide a circuit to energize KAS and to hold KAID.

KAS31 & 32 close to complete the circuit energizing LT3, LT4, and LT6 in the 2205. (This is shown in Figure 9-14.)

The 2314-1 then waits for the next code (the Address) to be sent from the 5610. This code performs three main functions:

1. Sets the corresponding Code Storage relays (KCS1 through KCS7) in the 2314-1.
2. Energizes the KS relay and drops the KCO relay in the 2314-1.
3. Causes the 8 bit to be set into the 2205 translator. stepping Fwd.

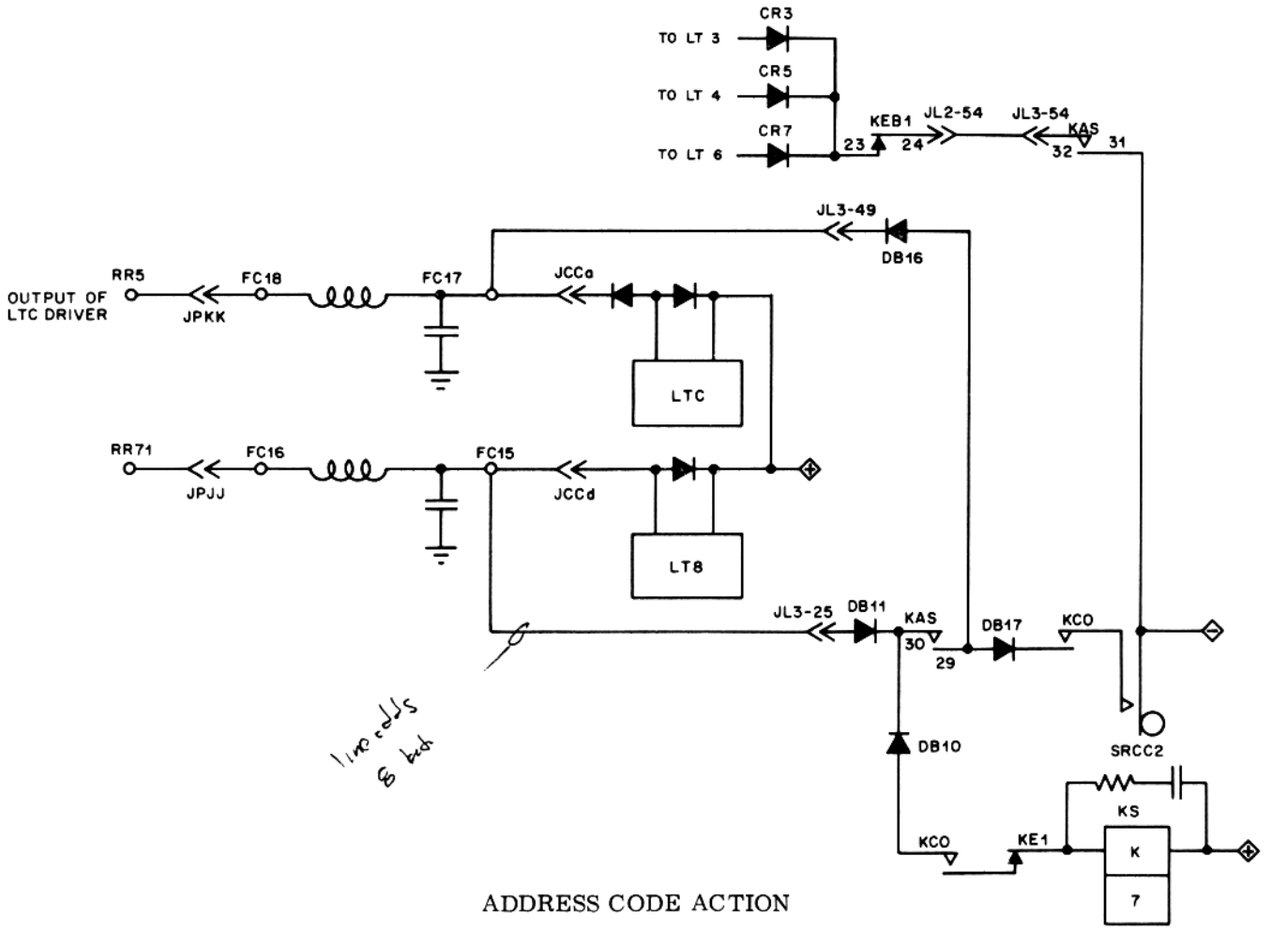
A diode has been added to each of the KCS relay circuits to allow the SELECTADATA reader to be used with the 5610. The circuit for LT1 and KCS1 is illustrated in Figure 9-13. A similar circuit exists for each of the other six magnets and relays. When the individual bits are driven by the 5610 (during the Type instruction or the Non Decimal Print instruction), this added diode provides a path to energize the corresponding code storage relay directly. The diode marked "D" in Figure 9-13 is needed to prevent 5610 outputs from being fed back into the processor as inputs. This same diode, however, blocks the 5610 output from ever getting into the 2314-1. For this reason, a separate line is run from FC1, through JL3-47 and DB36, to energize KCS1 after KAS has been picked. Diode DB37 prevents this signal from coming back into the 5610 as an input.



ENERGIZING KCS RELAYS FROM 5610

FIGURE 9-13

The KS relay in the 2314-1 is energized via a special wire, added to the Filter Card at FC17, which leads directly to JL3-49 (see Figure 9-14). When the translator clutch is driven by the 5610 (during the Type instruction or the Non Decimal Print instruction), a pulse is delivered through DB16 and DB10 to energize KS. This opens KS21 & 22, dropping out KCO.



ADDRESS CODE ACTION

161-214

FIGURE 9-14

This same translator clutch pulse also adds the 8 bit to the translator via a special line connected between JL3-25 and FC15. See Figure 9-14.

Enter ÷ /

In addition, this same pulse knocks down KAID (while holding up KAS temporarily). When this pulse is finally removed, KAS drops, and its contacts 1 & 2 close to energize the Double Feed magnet, LDF. This energizes the Reader Clutch magnet, LRC, through KCO22 & 21 and KS5 & 4.

So F11 = R10
then F12 = AID
then Address

R10 = Recode 1-ON

only for program operation F9

Note that an R10 instruction must precede the F12 (AID) instruction. Otherwise, the FFF flip-flop would set on the address-code translator cycle, and since no selector slides are operated, FCB3 could not be developed to reset FFF. This would cause 5610 program operation to cease. By having KRC energized, FFF cannot set. Note also that if an R2O instruction precedes the F12, the operation performed will be a repeat search.

R2O = F?13

So if F13 + F12 = Repeat Search for existing code
if * F11 + F12 = Search for new code.
+address

c. ARC SUPPRESSION. Each relay in the 2314-1 has a diode and/or an RC network wired across its coil terminals. RC networks are also added across SRCC1 and SRCC2 contacts, and across the Double Feed coil. The capacitor normally across the LRC coil is replaced with a diode. Details may be found on the schematic, drawing #4503585.

IE to Search New address
* F11 + F12 + address (8bit? + keys?)
Repeat Search F13 + F12 (takes address out of Selected data Storage)
to Initiate READ press START READ

4. 2315-1 AUXILIARY TAPE PUNCH

* Only from program.

The circuitry of the 2315-1 Auxiliary Tape Punch is identical to that of the standard 2315 Auxiliary Punch. The only difference between the two lies in the additional arc suppression installed in the 2315-1. The only form of arc suppression present in the standard 2315 is a diode around the clutch coil. The 2315-1 also has a diode around each of the bit magnets, and either a diode, a "back-to-back" diode contact protector, or an RC network around each relay coil. The RC network for the "knockdown" coil of KPS is installed across KPS contacts 1 & 2, instead of around the coil itself. Additional details may be found on the schematic.

F 6 = Reader Off

Could also be F1 (Reader Off)
Program F11 + F12 + address + F1
Reader off?

Print Type Restore = F 3

5. 2316-1 CARD PUNCH CONTROL

Since the standard 2316 Card Punch Control has rather extensive arc suppression, there are only minor differences between it and the 2316-1 in this respect. The 2316-1 has two additional relays, however, necessitated by peculiarities of the 5610. These are the Drop SCR relay, KDS, and the Auto Delay relay, KAD.

a. KDS RELAY. The Drop SCR relay, KDS, uses eight sets of form "B" contacts inserted into the circuits between the processor output and the eight code relays in the 2316-1. The relay is energized each time the Master Select relay, KMS2, is energized, or each time the 5610 common line output goes true. Thus, the sequence of events is as follows:

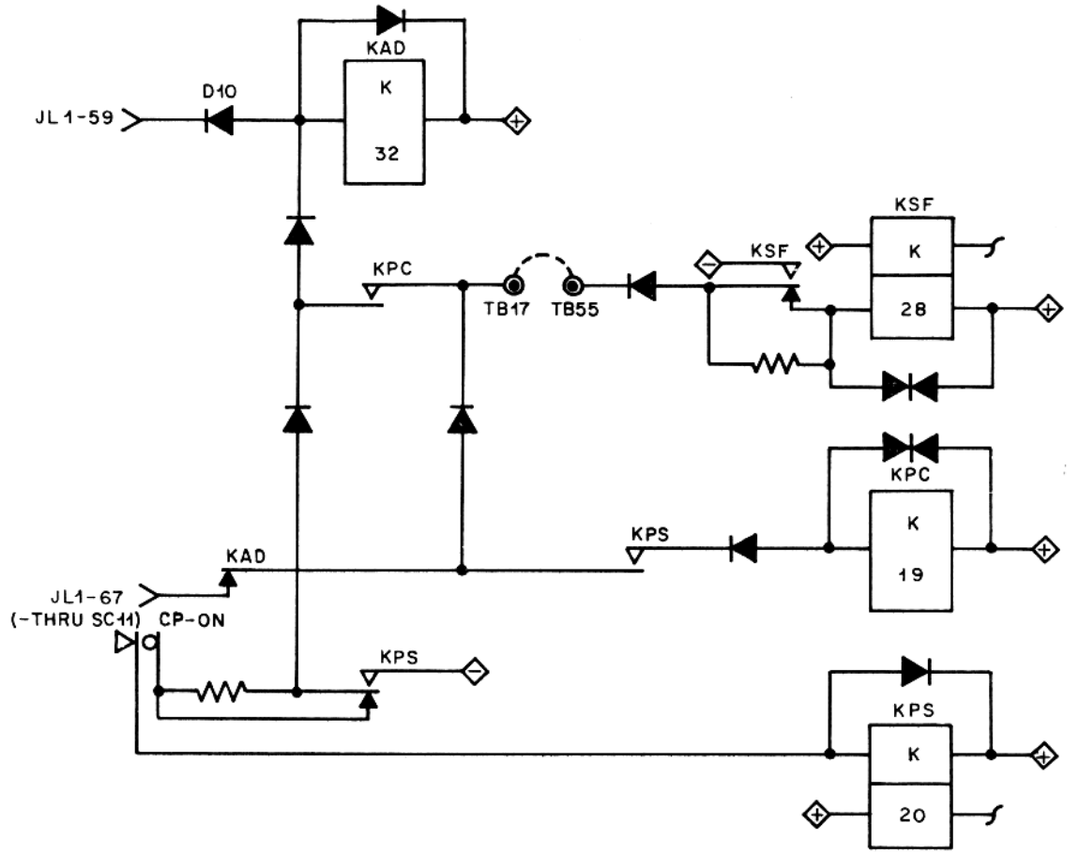
1. Energize 2316-1 lines 73 (common) and 4 through 13 (code), thus energizing KMS2 and KC1 through 8, depending upon code sent from 5610.
2. KMS2 contacts 2 & 3 transfer, energizing KDS.
3. KDS contacts transfer, opening the circuits to KC1 through 8 and allowing the SCR's in the 5610 to turn off.
4. Normal card punch action follows, and the keyboard restore pulse energizes KAR. KAR contacts 1 & 2 open, dropping out KDS.

The net result of having KDS in the circuit is to provide a more constant pulse length to the KC1 through 8 relays and to insure the turn-off of the SCR's if (1) KAR should fail to pick (due to the card punch being turned off or due to some malfunction), or (2) there should be leakage through the suppressor around KMS2 2 & 3.

b. KAD RELAY. The Auto Delay relay, KAD, was added to compensate for the "direct pick" action of the 5610 utilized in the performance of the Special Flexowriter Function, P2O, which turns on the no. 2 punch (in this case, the 2316-1). The circuitry is shown in Figure 9-15. Standard 2316 action, with no KAD relay in the circuit, is for an "F" key to operate, providing minus at JL1-59. With the CP-ON switch operated, this energizes KPS. Since all keylevers on the 2205 auxiliary keyboard operate SC11, there is no minus available (via JL1-67) to energize KPC until after the "F" key selector slide restores, and with KPC not energized, no code can punch. Thus the first "ON-2" code will not punch. The second "ON-2" can be made to punch by patching it to energize KSF, Shunt Filter, which allows all codes to punch by bypassing the code filter network.

When the On-2 comes from the 5610 via direct pick action, SC11 is not operated, so minus would be available to energize KPC as soon as KPS comes up, were it not for KAD. In this case, KAD energizes along with KPS, opening the pick circuit to KPC. At the end of the Special Flexowriter Function KAD drops out, allowing KPC to energize. A second P2O instruction in the 5610 will energize KSF if the patch is in between TB17 and TB55, so the programmer should compensate for this if necessary. Note, however, that none of the Special Flexowriter Functions cause any codes to be punched because they do not provide minus on JL1-73, the common output line, so they cannot energize KMS1 & 2.

Note that in the 2316-1, KAD operates whenever minus is applied to JL1-59, regardless of whether the minus is the result of a keylever action, a "direct pick" function, or a field switch operation. KAD is not necessary for keylever operation, but it is allowed to operate because it does no harm. "Direct picks" are preferred in place of field control in programming the 2316-1, because KAD remains energized as long as the field switch is operated and KPC cannot pick. Unless some interlock is incorporated in the 2205 program panel patching, the code that causes the carriage to move off of the field switch may or may not punch, and an error could result. Therefore, field control of the 2316-1 is not recommended.



KAD RELAY OPERATION

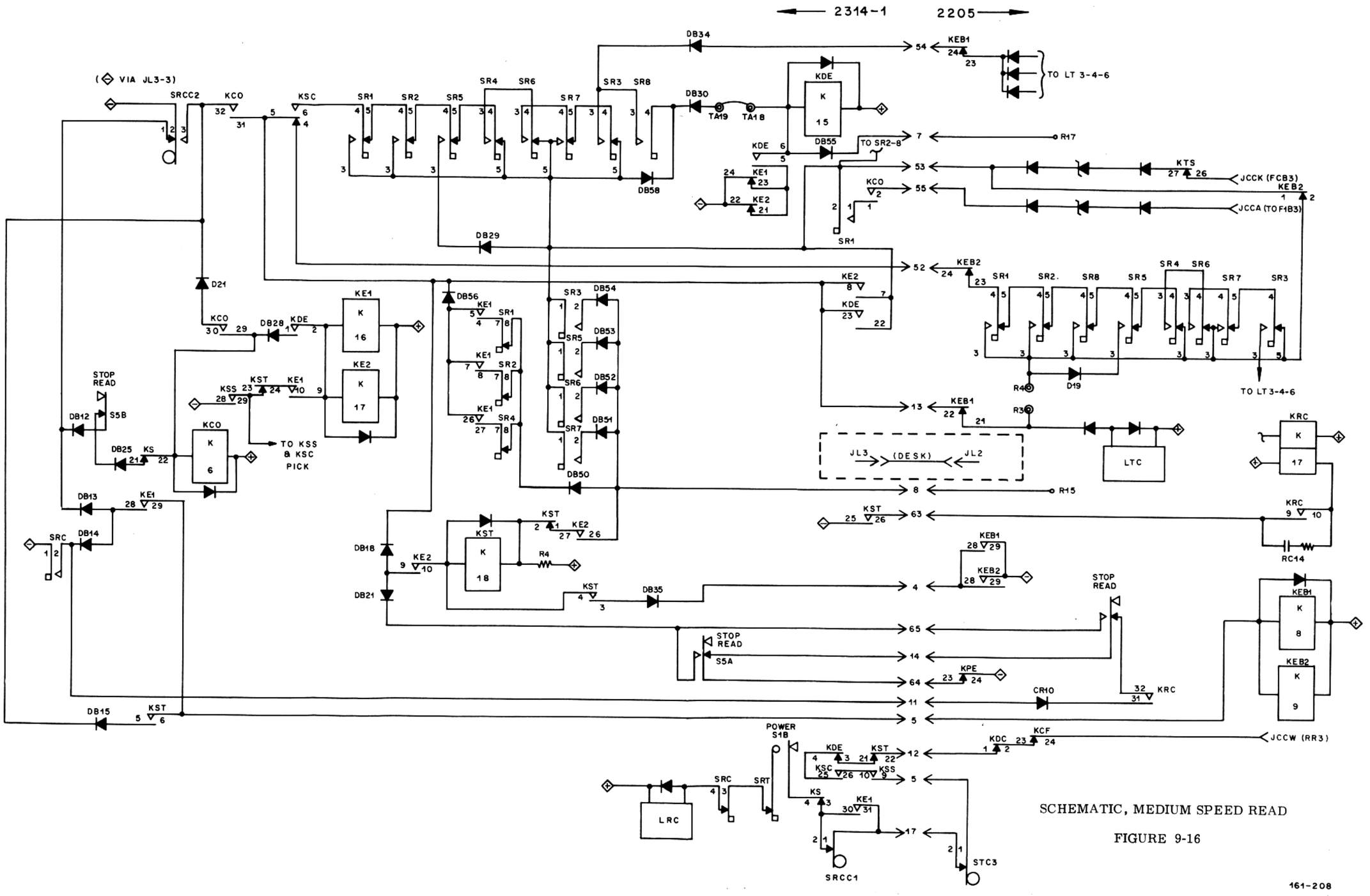
FIGURE 9-15

6. 2323-1 PRINTING CARD PUNCH

The 2323-1 Printing Card Punch makes edge-punched card punching and printing possible on the 5610. A standard 2323 can be connected to the 5610, providing the punch is wired to schematic 4504243, rev. D (or later). Machines prior to this revision do not have adequate arc suppression.

The same model can be used on other FRIDEN systems equipment having auxiliary output capabilities.

The 2323-1 is connected to the 5610 by plugging its cable into the AUX PUNCH receptacle (JL1) in the desk, the same as any other output unit.



SCHEMATIC, MEDIUM SPEED READ

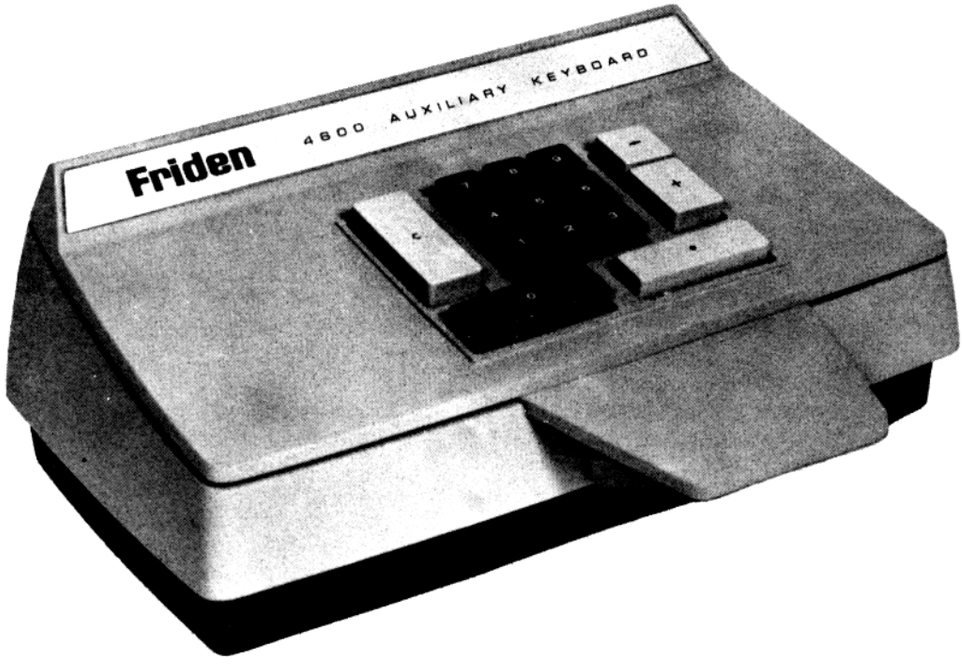
FIGURE 9-16

161-208

4600 TEN-KEY KEYBOARD

TABLE OF CONTENTS

	Page
SECTION 9, SPECIAL/OPTIONAL FEATURES (Cont.)	
BASIC INFORMATION	9-31
GENERAL OPERATION	9-31
INSTALLATION	9-32
LOGIC	9-32
Symbols	9-32
Signal Glossary	9-34
Clear Keyboard	9-36
Pre-Shifting	9-38
Digit Entry	9-40
Terminate Keyboard Entry	9-42
AKO Instruction Retrieval	9-44
Enter Instruction	9-44
MAINTENANCE	9-47
Lubrication	9-47
Component Locations	9-47
ADJUSTMENTS	9-47
Interlock Bar Assembly	9-47
Key Lock Bail	9-48
Contacts	9-48
LOGIC SCHEMATIC, TEN-KEY KEYBOARD	9-49



4600 AUXILIARY KEYBOARD

D. 4600 AUXILIARY TEN-KEY KEYBOARD

The 4600 Auxiliary Ten-Key Keyboard is a standard optional feature that can be added to any 5610 starting with serial number 1940. It may be used with any other optional features. Numeric data (up to ten digits and a sign) can be entered and stored in the keyboard, and then transferred to the 5610 upon execution of a subsequent Enter instruction. The 2205 does not print during this transfer.

1. BASIC INFORMATION

The 4600 consists basically of two parts: the keyboard unit and the "BB" printed circuit card. The keyboard unit is identical to that used with the 5005 COMPUTYPER electronic billing machine. It houses a printed circuit card containing a 10-digit buffer and all of the electronics necessary to enter a number into it. The "BB" card contains the additional circuitry needed to transfer each digit from the buffer to the common register in the 5610, and to control the shifting of the buffer as digits are removed. It also contains the circuitry needed for decimal alignment when more than two digits are entered right-of-decimal, for adding parity bits to the codes before they are set into the common register, and for decoding the sign information sent from the keyboard.

The keyboard has a fixed 4-foot cable with a 24-pin plug, which is plugged into a 24-pin receptacle on the processor. It has ten digit keys, 1 through 9 and 0, and four function keys; decimal, plus, minus, and clear. Their functions are as follows:

Digit Keys: Depression of any of these keys causes entry of the corresponding digit into the buffer. All digits previously entered are shifted to the next higher order digit position.

Decimal Key: Terminates the left-of-decimal portion of the keyboard entry the first time it is operated in each entry: subsequent key depressions have no effect.

Plus Key: Terminates keyboard entry and designates the sign of the data in the buffer as being positive.

Minus Key: Terminates keyboard entry and designates the sign of the data in the buffer as being negative.

Clear Key: Clears all digits out of the buffer and conditions it for a new entry.

A ball interlock prevents depression of more than one key at a time (except the Clear key). A key LOCK magnet is also provided, which becomes energized after a keyboard entry is terminated. This magnet holds the Plus or Minus key locked down until after the number in the buffer is fully transferred to the 5610. Thus, if a keyboard entry is completed and the 5610 is not yet ready to accept it, further entries are prevented by the LOCK magnet.

2. GENERAL OPERATION

A number can be entered on the keyboard and stored in the buffer register until the 5610 calls for the entry, or the 5610 may call for the entry first, and wait for the 4600 entry to be completed before the buffer contents can be transferred to R1.

Prior to the execution of the Enter instruction, the 5610 must execute an Auxiliary Keyboard On (AKO) instruction (F10, 1-2-3-4-5 code). This instruction simply preconditions the 5610 to accept the next entry from the 4600 buffer register. Then a subsequent Enter instruction transfers the number in the buffer to R1. A separate AKO instruction is required for each 4600 entry to the 5610.

The AKO instruction blocks the reader, but does not disable normal 2205 keyboard entries. If no prior entry is made into the 4600, an AKO instruction followed by an Enter will interrupt the reader and cause the 5610 to wait for the Ten-key entry. On the other hand, if the 2205 reader is "off", the same situation will allow entry from EITHER the 2205 keyboard or the 4600. If desired, a Non PRint instruction may be used prior to the Enter instruction to lock the 2205 keyboard.

The maximum number of digits that can be stored in the 4600 buffer is ten. The maximum number of digits that can be entered left-of-decimal is 8 and right-of-decimal is 4. Naturally, if 8 digits are entered left-of-decimal, only two more may be entered after the decimal. Thus the MAXIMUM LRD entries possible on the 4600 are 8/2, 7/3, and 6/4. One LRD pattern, 8/5, used as the address of the 5610 Enter instruction, allows for all of these possibilities.

Since the 5610 entry is terminated by the sign signal sent from the 4600, the LRD pattern need not be filled (it COULD NOT be filled by the 4600). Decimal alignment and insertion of insignificant zeros are automatic.

There are certain error conditions which may occur in the 4600 entry, as follows:

1. Entry of more than 8 digits left-of-decimal
2. Entry of more than 4 digits right-of-decimal
3. Entry of more than 10 digits

When any of these conditions occurs, the keyboard is disabled (not locked) and transfer of data to the 5610 is prevented. The Clear key must be operated and the correct digits must be re-entered.

3. INSTALLATION

The wiring and other changes necessary for installing the auxiliary keyboard on 5610s with serial numbers from 1940 to 2767 are covered in detail in the "Field Installation Kit", part number 7301014. This kit contains the "BB" card and all of the necessary parts and instructions for installing the keyboard cable and wiring changes, but does NOT include the keyboard itself, which is a separate item (part number 3200632). Using these instructions, normal installation time is about 8 hours for machines prior to serial number 2767. Above that number, the mother board wiring is already complete, so installation time is reduced to about 6 hours.

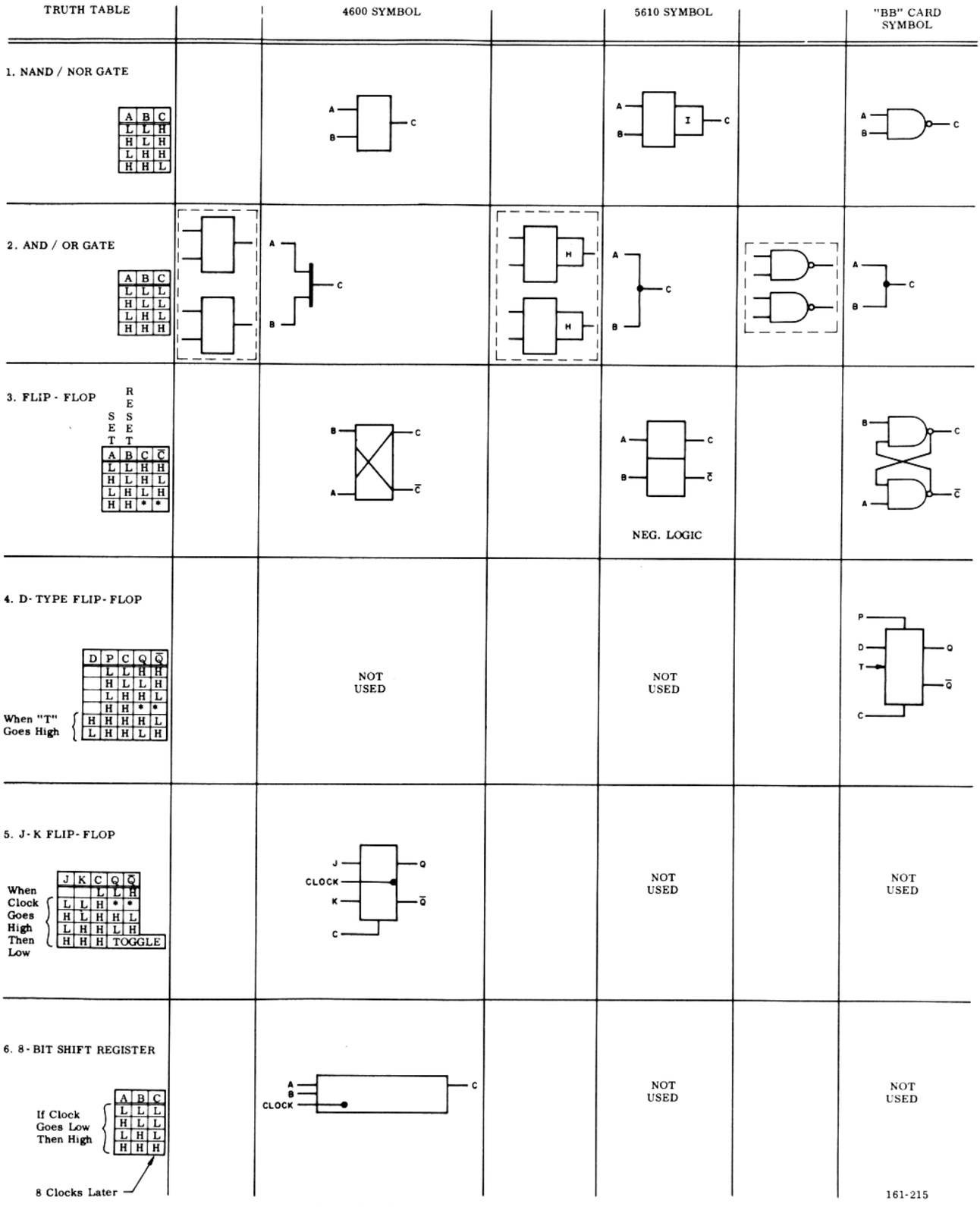
Caution should be observed when inserting the "BB" printed circuit card into its socket. On some cards, the printed circuit that is connected to plug pin #1 may short out on the metal card guides in the card rack. If this possibility exists, insulate the printed circuit with electrical tape.

Note that once the keyboard is modified as per these instructions it can not be used with a 5005 COMPUTYPER without first removing the jumper wire INSIDE the keyboard between keyboard-to-contact-assy. terminals 6 and 20. Failure to remove this jumper could result in burned LOCK magnets when the keyboard is used with a 5005.

4. LOGIC

The 4600 and the "BB" card were designed using "positive logic" (logic 1 = higher voltage level than logic 0), whereas the 5610 uses negative logic. This could lead to a great deal of confusion, so in discussing the 4600 keyboard, the terms "high" and "low" are used instead of "true" and "false". Bear in mind that when a flip-flop in the 4600 or on the "BB" card is set, its "upper", or "true" output is "high".

a. SYMBOLS. There are many logic symbols on the 4600 and "BB" card schematics that have not been used previously with the 5610. A brief comparison of each is shown in Figure 9-17 (further details may be found in the FRIDEN integrated-Circuit Handbook, TPA-259SL, and in Lesson 24 of the "FRIDEN Electronic Course".



COMPARISON OF LOGIC SYMBOLS

* NO CHANGE: REMAINS IN PREVIOUS STATE

FIGURE 9-17

TP-161SL

b. SIGNAL GLOSSARY. The following signal glossary explains the more important signal names and their functions.

A	Enables special circuits for use with the 5005. In 5610 operation, A is ALWAYS low, A* is ALWAYS high.
ABRa-d	"A" Buffer Register: first 8 digits of buffer register.
BBRa-d	"B" Buffer Register: ninth digit of buffer register.
BBR0	B Buffer Register clear: BBR0* goes low when BBR = 0.
BFF	Buffer Full Flip-flop: sets on 11th digit entered or on 5th digit entered right-of-decimal. Resets VEF, prevents further buffer shifts and MOC increments.
CBRa-d	"C" Buffer Register: tenth digit of buffer register.
CBRs	C Buffer Register set: CBRs* goes low when CBR = binary 15.
CBR0	C Buffer Register clear: CBR0* goes low when CBR = 0.
CC	Clear Common: clears entire keyboard when Clear key on keyboard is depressed or when ST9 or SWRS9 in 5610 goes low.
CL	Clear Local: clears all of keyboard except BFF. Produced by CC and by RC, at termination of the Enter instruction in the 5610.
CLF	CLear Function: initiates CC; developed by ST9 and SWRS9 in 5610.
CPF	Count Pulse Flip-flop: controls buffer shifts as digits are shifted into the 5610 (after pre-data shifts have been completed).
DPF	Decimal Point Flip-flop: sets on depression of Decimal key in keyboard (Kd), enables MOC increments.
ET	Entry Terminated: developed after Plus (Kp) or Minus (Km) key in keyboard is operated; enables pre-buffer shifts, blocks clearing action, enables S1.
FXXK	F10 Ten-Key flip-flop: set in 5 mode when F10 instruction is retrieved. Enables setting of CPF and TKEN when Enter instruction is retrieved.
Ia, b	Information signals: control setting of PDA & PDB (PDZ Counter) according to number of digits entered right-of-decimal (status of MOC counter). Developed by decoding MOC outputs, gated by S1.
ID	Input Digit: control signal used to shift each digit into buffer as digit is entered. Produced by KOK one-shot. Also used to gate MOC increment, BFF set, VEF set/reset.
K	Key switches: K1-K9, K0: Digit switches Kc Clear Kd Decimal Km Minus Kp Plus
KCB	Keyboard Common Bit: equivalent to Flexowriter Common Bit for keyboard use; developed on BB card in 5M1P to shift code from keyboard into CR and to cause step to 2P.
KOK	Keyboard OK: 3 ms. one-shot fires each time a digit key is operated; "OK" means no digit key is operated.

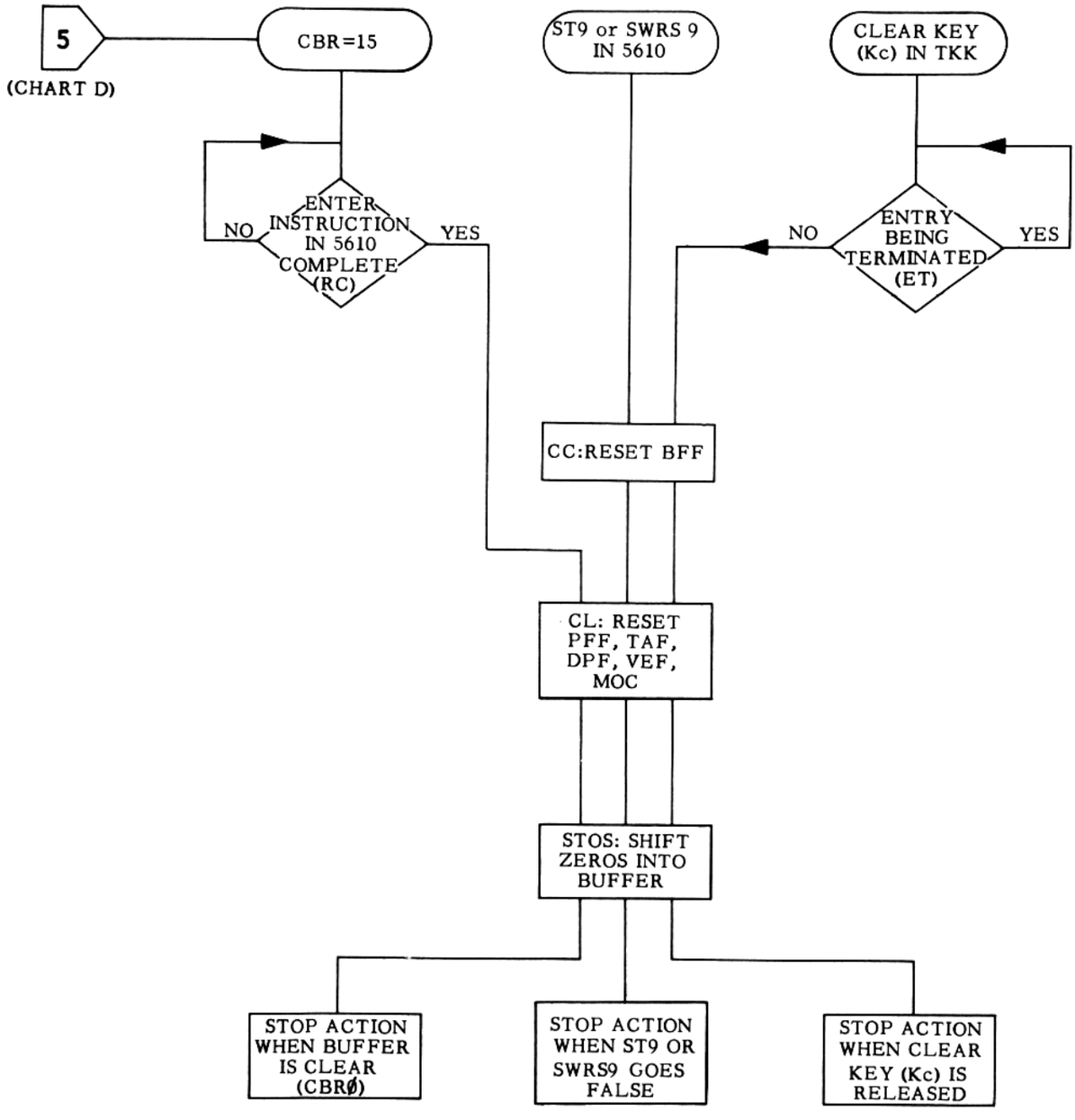
LO	Left Overflow: product of DPF reset and BBR ≠ 0. Developed when 9th digit left-of-decimal is shifted into buffer. Prevents further buffer shifts, resets VEF, and requires Clear key operation to correct this condition.
LOCK	LOCK: Holds Plus or Minus key locked down after either is depressed, prevents depression of other keys. Valid as long as PFF or TAF is set.
MOCa-c	MOde Counter: three-flip-flop counter used to count digits entered right-of-decimal. Incremented by each ID after DPF is set. Outputs decoded into Ia & Ib.
MOC0,1	MOde Counter = 0 or 1: denotes that either 0 or 1 digits have been entered right of decimal; used to cause pre-buffer shifts to automatically insert two zeros right-of-decimal.
OSC	OSCillator: output normally low, oscillates (high for 6 μs, low for 10 μs) when turned on by STOS. Used to shift zeros into buffer.
PDA, PDB	Flip-flops in PDZ counter.
PDZ	Pre-Data Zero counter: two-flip-flop counter used to control pre-data shifting. Initial state set in by Ia & Ib. PDZ is pulsed each time 2P is reached until both of its flip-flops are reset, thus allows 1 or 2 pre-data shifts.
PFF	Plus Flip-Flop: set by depression of Plus or Minus keys (Kp, Km), develops ET, LOCK.
RC	Request Clear: developed whenever TKEN ff is reset, enables clearing of buffer, along with CBRs. Also develops CL to reset MOC, VEF, PFF, TAF, thus unlocking Kp & Km.
(Send) SIGN	Sends Minus sign from keyboard to BB card: developed whenever TAF is set, sets lone 7 bit into CR when gated by CBR = 15 after all digits have been transferred to R1.
STOS	STart OScillator: developed by CL during clearing of buffer or by MOC0,1 and ET during pre-buffer shifts. Provides bias to turn on oscillator and low signal to shift zeros into buffer.
S1	Signal 1: provides binary 15s at input to buffer, enables TK when 5610 calls (tells 5610 that keyboard is ready to transmit); developed by ET, CBRs*, and MOC0,1* when keyboard entry is terminated and pre-buffer shifts are complete.
TAF	Take Away Flip-flop: set by Minus key, develops Send Sign, denoting negative entry, sets PFF, develops LOCK.
TK	Ten Key: allows shifting of buffer after each digit is transferred to 5610; enabled by S1 and gated by TKK-CALL.
TKK-CALL	Ten Key Keyboard CALL: "calls" keyboard, developed when 5610 reaches 5M1P with FXXK set. Stays "low" until pre-data shifts are finished, then goes "high" each time 2P is reached. When low, primes buffer shift (TK), when high, shifts buffer.
TKKa-d	Ten Key Keyboard buffer outputs to 5610.
TKEN	Ten Key ENtry ff: set in 5M1P if FXXK is set, enables TKK-CALL, enables setting of TKRF, blocks RC.

TKRF — Ten Key Reset Function ff: set in 5M6P when TKEN is set. Also set by ST9 or SWRS9. When reset, allows zeros to be shifted into CR for pre-data shifts, enables development of KCB, and allows Ia & Ib to set PDZ counter flip-flops.

VEF — Valid Entry Flip-flop: set by ID when BFF is reset, reset by ID when BFF is set. Also reset by LO and CL. Must be set to allow setting of PFF or TAF to terminate keyboard entry.

c. CLEAR KEYBOARD. The 4600 is cleared fully when the 5610 power is turned on, when the Reset switch is operated, and when the Clear key is depressed. A partial clear, clearing everything except the Buffer Full Flip-flop, occurs at the end of the 5610 Enter instruction.

Clear key action is controlled by the Entry Terminated signal, ET. An incorrect entry can be cleared at any time prior to termination of the entry. After the 4600 entry is completed (by depression of the Plus or Minus key), ET remains high until the 5610 Enter instruction is finished. During this time, the Clear key can be depressed, but it will not function: once a 4600 entry is complete, the contents of the buffer MUST be transferred to the 5610. If the 4600 entry is complete, and the 5610 is not yet ready to accept it, the Clear key will not function.



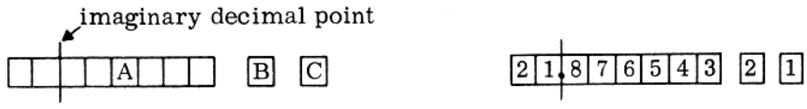
FLOW CHART A
CLEAR TEN-KEY KEYBOARD

FIGURE 9-18

TP-161SL

d. PRE-SHIFTING. There are two types of pre-shifts that may occur: pre-buffer shifts and pre-data shifts. Pre-buffer shifts occur in the keyboard when less than two digits are entered right-of-decimal. Pre-data shifts occur in the 5610 when more than two digits have been entered right-of-decimal. If exactly two digits are entered right-of-decimal, no pre-shifting occurs.

(1) Pre-Buffer Shift. Assume an imaginary decimal point exists in the keyboard buffer, as shown in Figure 9-19 (a). If the decimal key is struck, two more digits are entered, and then the keyboard entry is terminated, the buffer contains an 8/2 entry properly aligned around this imaginary decimal. See Figure 9-19 (b). The fact that there are 8 digits "left-of-decimal" (even though some or all of them may be insignificant zeros) causes the number to be decimally aligned correctly when it is transferred to the 5610.



BUFFER PRE-SHIFTING

FIGURE 9-19

If less than two digits are entered right-of-decimal, pre-buffer shifts must be performed to allow only eight digits to be transferred to the 5610 left-of-decimal. This is shown in Figure 9-19 (c).



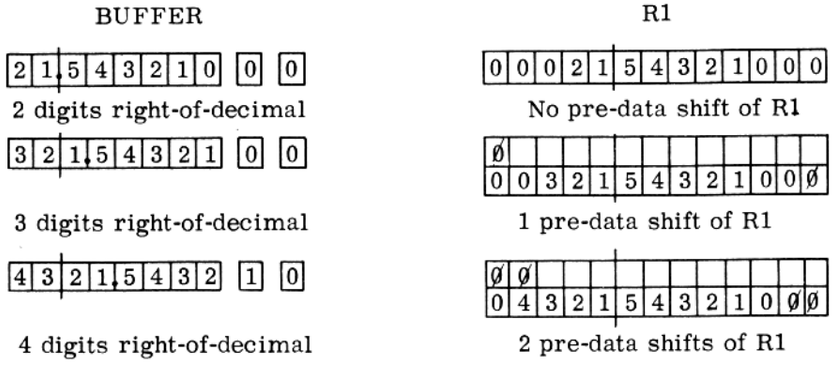
1 digit right-of-decimal
1 pre-buffer shift

No digits right-of-decimal
2 pre-buffer shifts

FIGURE 9-19 (c)

The mechanism used to control pre-buffer shifting is the MOde Counter, MOC. If, at the termination of the keyboard entry, the MOC counter is equal to 0 or 1, pre-buffer shifts occur, shifting zeros into the buffer and incrementing MOC, until the MOC counter equals 2.

(2) Pre-Data Shift. If more than two digits are entered following entry of the decimal, it means that less than 8 digits are available to transfer to the 5610 left-of-decimal. In this case, R1 in the 5610 must be shifted left (and the decimal register counted down) prior to accepting the first digit from the keyboard. Examples are shown in Figure 9-20.



PRE-DATA SHIFTS

FIGURE 9-20

The MOC counter is again used, since it has count of the digits entered right-of-decimal. Its outputs are decoded into Information signals, Ia and Ib, which control the setting of the PDZ counter flip-flops, PDA and PDB, as shown in Figure 9-21.

Pre-buffer shifts	2	1	0		
MOC	0	1	2	3	4
Ia	H	H	L	H	L
Ib	H	H	L	L	H
PDA			R	S	R
PDB			R	S	S
Pre-data shifts			0	1	2

MOC EFFECT ON PRE-SHIFTS

FIGURE 9-21

If three or four digits are entered right-of-decimal, a zero is shifted into R1 (R1 is shifted left) and the decimal register is counted down once for each state of the PDZ counter: after the zero is set into the common register, the PDZ counter is pulsed, and this action repeats until PDA and PDB are both reset. This process is detailed in the flow chart in Figure 9-26. A truth table for the PDZ counter appears in Figure 9-22.

	PDA	PDB	
	0	0	
2 x SR1L	0	1	Set zero to CR
1 x SR1L	1	1	Set zero to CR
	1	0	Set buffer to CR
0 x SR1L	0	0	

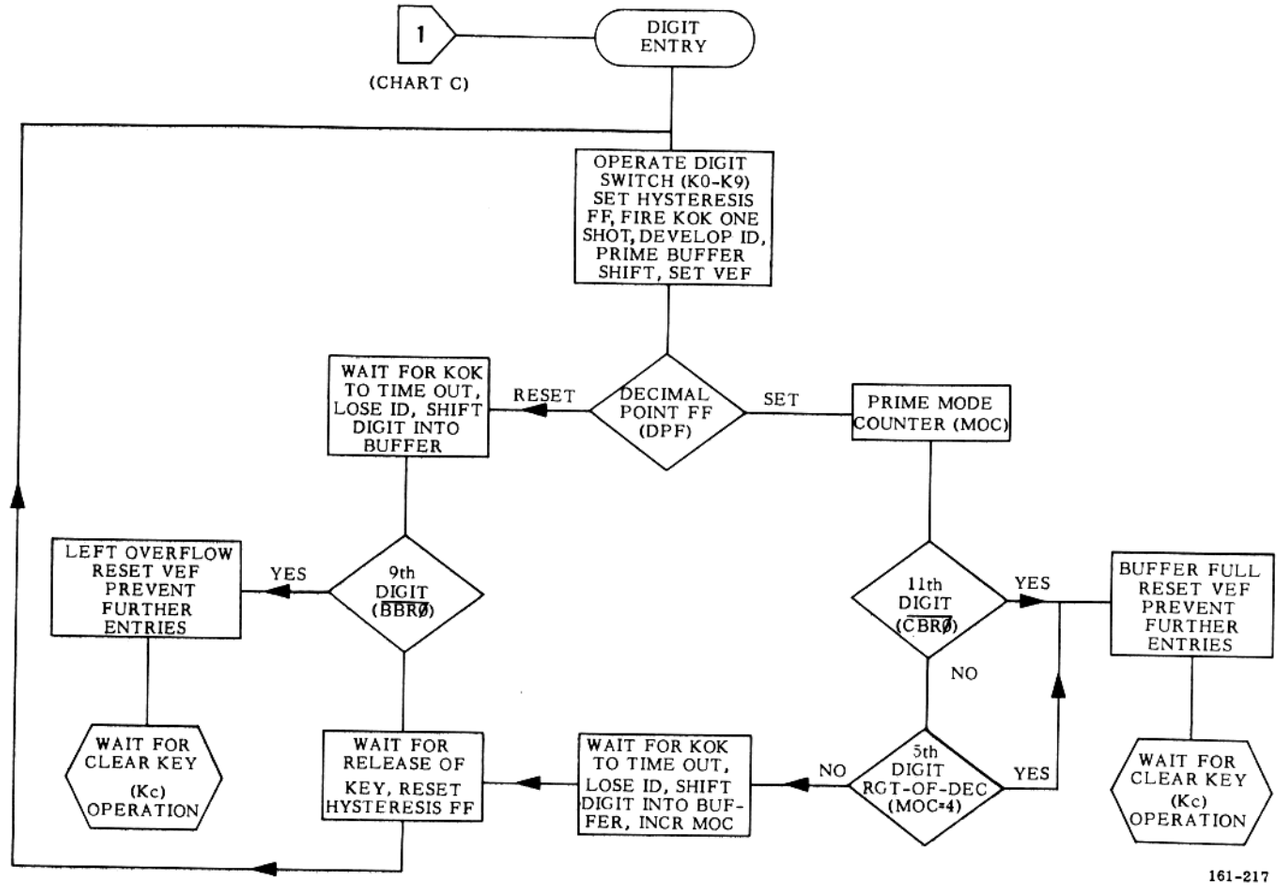
TRUTH TABLE - PDZ COUNTER

FIGURE 9-22

e. DIGIT ENTRY. Each digit key depressed fires the KOK one shot, developing ID to shift the digit into the buffer. The first digit entered sets VEF, and it remains set until CL at the end of the Enter instruction (as long as no error occurs).

When an error occurs, VEF is reset immediately, preventing the setting of PFF and TAF, and LO + BFF prevents any further entries. The Clear key must be operated.

The hysteresis flip-flop prevents re-triggering of KOK, so any digit key must be released before the next digit is entered.

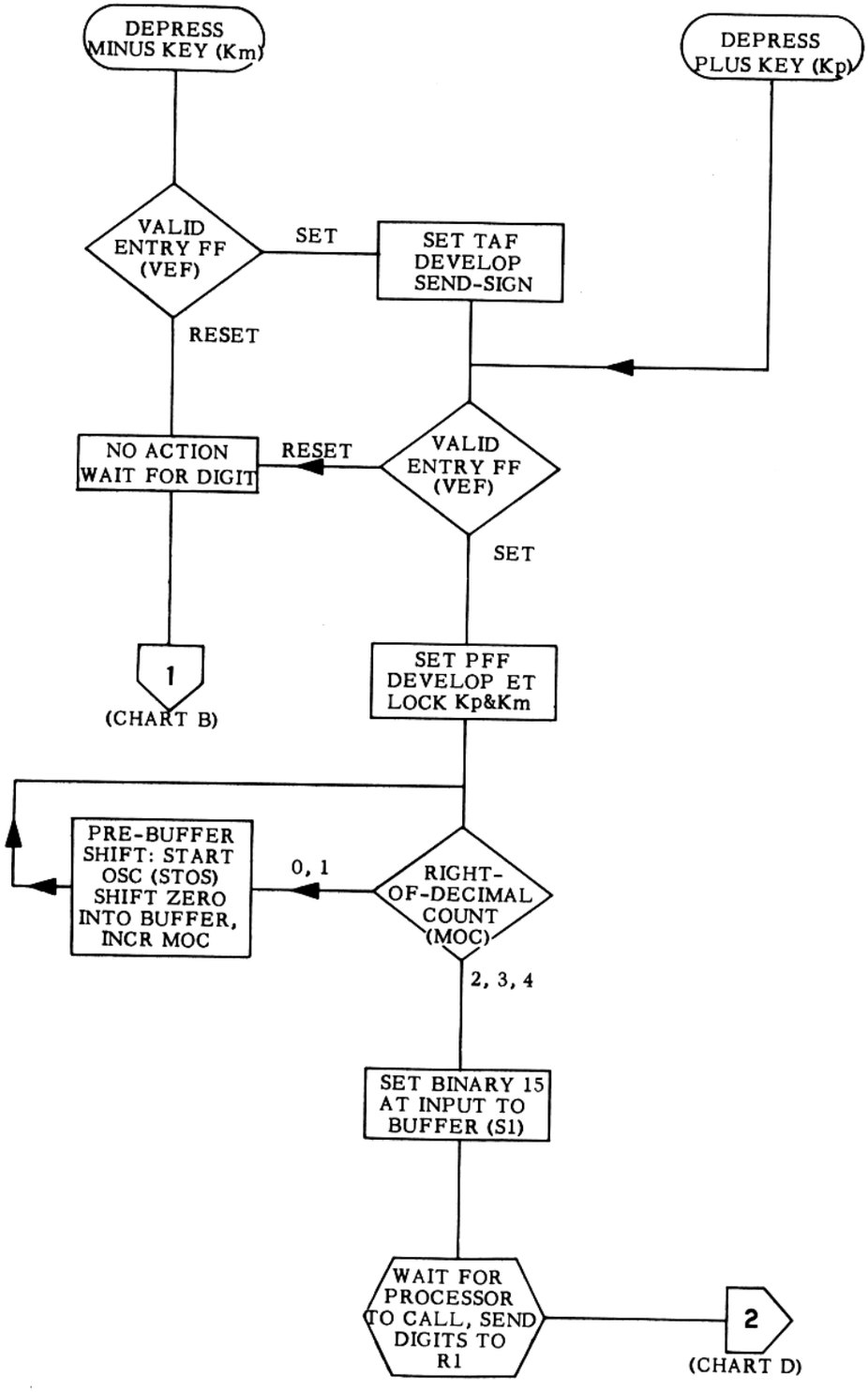


FLOW CHART B
DIGIT ENTRY

FIGURE 9-23

f. TERMINATE KEYBOARD ENTRY. A keyboard entry is terminated by depression of either the Plus key or the Minus key after a digit has been entered. If no digit has been entered, VEF is reset, and the Plus and Minus keys have no effect. If an entry of zero is to be made, at least one 0 must be entered in order to set VEF.

As soon as the Plus or Minus key is depressed, any necessary pre-buffer shifts are performed, and the MOde Counter, MOC, is decoded into Ia and Ib, which will be used to set up the pre-data shifts when the 5610 calls for the buffer data.



FLOW CHART C
TERMINATE KEYBOARD ENTRY

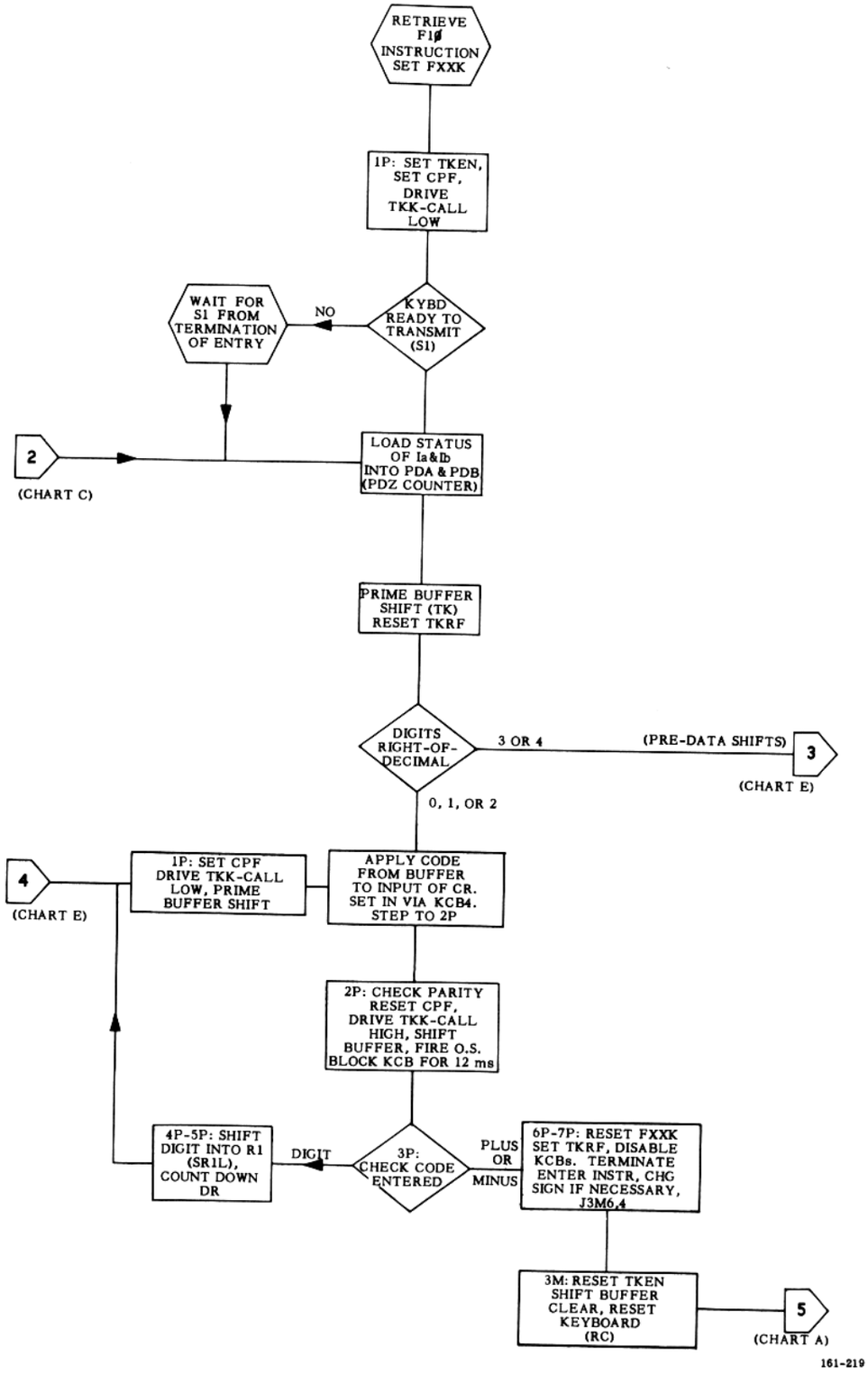
FIGURE 9-24

g. AKO INSTRUCTION RETRIEVAL. Retrieval of the AKO instruction (F10 code) causes operation quite similar to that of a Special Flexowriter Function. Normal instruction retrieval causes a jump from 4M3P to 5M15P because F10 is a 0-address code. In 5 mode, 5F103 is developed (BB), which sets the F10 Ten Key flip-flop, FXXK (BB), and develops SFF3 (K). This latter signal causes a jump to 6 mode, and subsequent operation is the same as for Special Flexowriter Functions (covered in Section 6F), except that no drivers are "turned on" and no "direct pick" occurs. The setting of FXXK is the only function of the AKO instruction: the routing through 6 mode is used only to disable any 5 mode output of the F10 code. (It is still possible to output an F10 code by using it as the address of a Type instruction.)

h. ENTER INSTRUCTION. The 5610 waits in 5M1P if the 4600 is not ready to transmit. When the 4600 is ready, any pre-data shifts required are first performed. Then each digit is set into the common register in 5M1P, and the 5610 operation from that point (through 2-3-4-5 phases, or through 6-7 phase upon termination) is the same as for a regular Enter instruction.

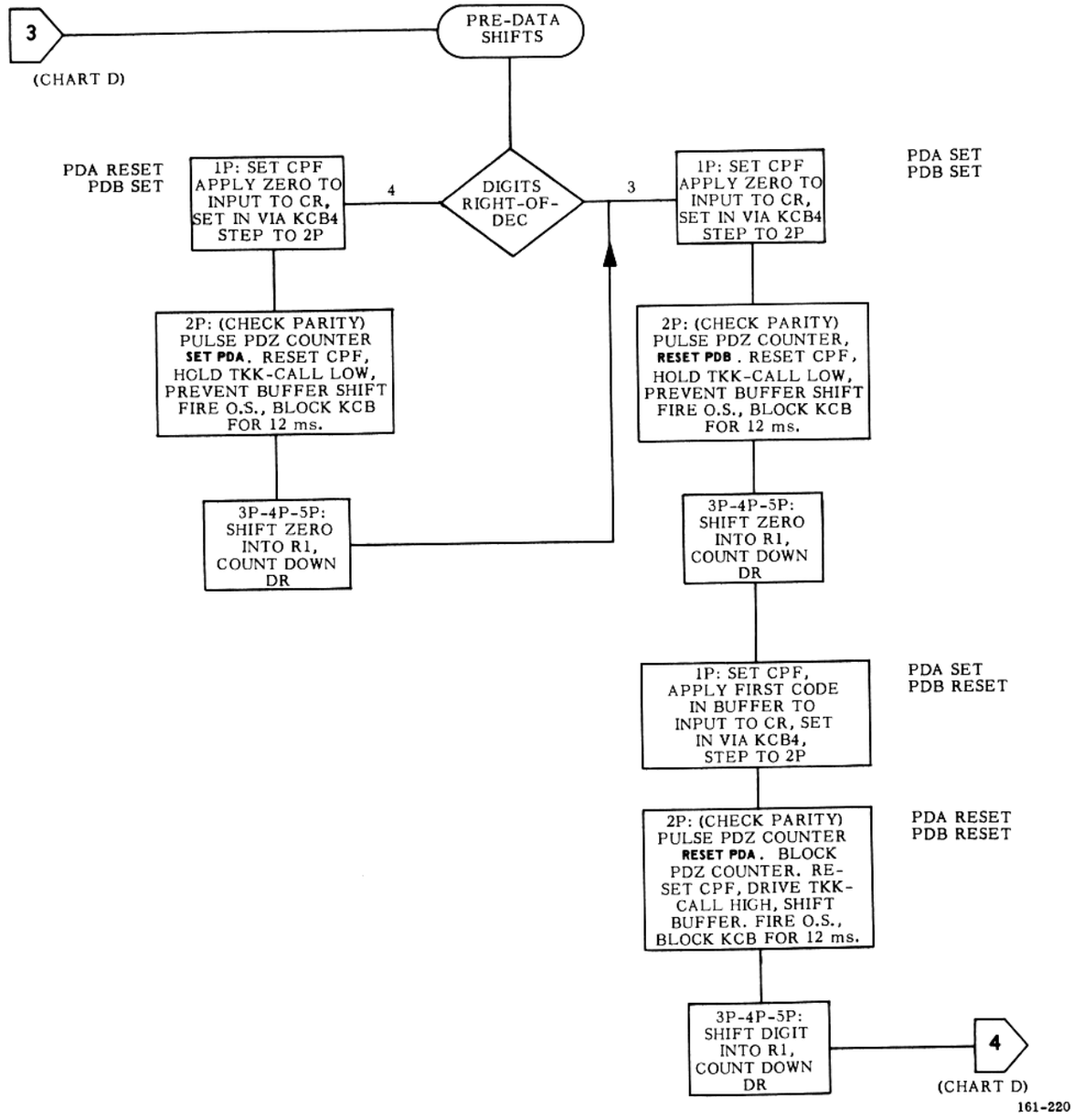
The signal S1, developed when the 4600 is ready to transmit, applies a binary 15 to the input to the buffer. Then, as each digit is shifted out of the buffer into the 5610, a 15 is shifted in. When all digits have been sent to the 5610, the first 15 is in CBR. This initiates action on the "BB" card which sets the sign code into the common register, ultimately terminating the Enter instruction. (Note that by using the 8/5 LRD pattern for the 5610 Enter instruction, the instruction is always terminated by entry of the sign code into the common register, and not by developing DCMP through filling the LRD.)

The 15 in CBR also develops CBRs, which combines with RC when the 5610 leaves 5 mode, to develop CL. This clears the buffer, and resets MOC, VEF, PFF, and TAF, which unlocks Km and Kp, making ready for the next keyboard entry.



FLOW CHART D
ENTRY INSTRUCTION, 5 MODE, TKK

FIGURE 9-25



FLOW CHART E
PRE-DATA SHIFTS

FIGURE 9-26

5. MAINTENANCE

There is no specific maintenance inspection procedure for the 4600 keyboard. The only preventive maintenance possible is lubrication.

a. LUBRICATION. The mechanical portions of the keyboard should be inspected, both visually and operationally, at every 5610 maintenance inspection, to determine if lubrication is necessary. Lubricate as follows:

(1) Interlock Bar Assembly. Lubricate sparingly with powdered graphite (CC #7, T18199). Use care to keep the graphite away from the electrical contacts. "Split" the key assembly and remove the interlock bar assembly before applying the graphite. Do not attempt to apply the graphite powder while the interlock is installed in the key assembly.

(2) Key Lock Bail. Apply a light coat of medium grease to both surfaces of both angled slots, and to the end bearing surfaces.

(3) Key Stems. Apply a light coat of medium grease to all bearing surfaces.

b. COMPONENT LOCATIONS. Locating the individual components is different on the 4600 printed circuit card and the "BB" card, and both are different from other 5610 circuit cards.

(1) "BB" Card. The only locations given on the "BB" card schematic (#7301043) are for the integrated circuit chips. The large number inside the symbol refers to the chip package number, and the small numbers outside the symbol refer to the package pin numbers. The package number also appears physically on the printed circuit card adjacent to the package, and this numbering runs generally from left to right (component side), and bottom (plug end) to top (ejector end). "BB" card chips use the same pin numbering system used by the hex pack chips.

Other components can be located only by tracing out the individual circuits.

(2) Keyboard. All components are given location references on the "AD" card (4600 printed circuit card) schematic (#3001380). A coordinate system is used. As in the 5610, each component is given a two-coordinate location designator, but in this case, a letter and a number are used. Letters run from left to right, starting with "A" to "Z", going through "$\overline{A}$" to "$\overline{Z}$", and finally "a" to "e". Numbers run from 1 (bottom) up to 28 (top). Unlike the 5610, however, the 4600 coordinates designate a point near the center of the component, and not one of the ends.

As on the "BB" schematic, the 4600 schematic designates chip locations within the symbol and pin numbers adjacent to the symbol. Pin numbering is again the same as for the hex pack chips.

6. ADJUSTMENTS

There are no electronic adjustments associated with the 4600 keyboard. The only electrical adjustments are those associated with the contacts. The only mechanical adjustments pertain to the interlock bar assembly and the key lock bail assembly.

The following adjustments are given IN THE CORRECT SEQUENCE, and when an adjustment affects a prior adjustment, it is so noted. If all adjustments are not made in sequence, care should be taken to see that all necessary adjustments are carried out.

Prior to making these adjustments, remove the two screws from the bottom of the keyboard and "split" the keyboard. All adjustments are made on the key assembly, which is fastened to the upper cover.

a. INTERLOCK BAR ASSEMBLY. Adjust the end stops to provide .060" clearance between any two interlock balls: remove the four mounting screws from the lower plate (on which the contacts are mounted), and remove the plate. Insert a .060" set of feeler gauges into the centermost slot of the interlock bar, and remove excess clearance by positioning the end

stops snug against the interlock balls. Tighten the end stops in this position. Reassemble the lower plate to the key assembly. (The key stems can be positioned into their respective guide slots by wiggling the key tops.)

b. KEY LOCK BAIL. Depress either the Plus or Minus key. The lock bail bracket (the "armature") should just contact the cores of the coils as the key bottoms out. If not, remove the key assembly from the top cover (four shoulder screws in rubber mounts), and move the magnet mounting bracket until the proper contact is obtained. Do not replace the top cover until after the contact closing point is checked (adjustment c (3)).

With neither the Plus nor the Minus key operated, the tops of the angled slots in the key lock bail should be positioned just opposite the key stems of both keys. Adjust the screw and lock nut in the lock bail bracket ("armature") to obtain the correct position.

c. CONTACTS. Contact gap is relatively unimportant. In general, normally open contacts should have sufficient gap to prevent their bouncing together when they are not operated. On the other hand, too much gap requires excessive flexing of the contacts, which leads to premature contact breakage. A suitable compromise should be obtained.

(1) Check that all contact assemblies are stacked straight and are properly centered in relation to the key stem. Loosen the mounting screws and adjust, if necessary.

(2) Check to see that all stationary contact springs have sufficient tension to hold them firmly against their stiffener. If not, remove the spring from the stack, re-tension it, and re-install.

(3) With the key assembly removed from the top cover, place a .180" gauge between each key top, in turn, and the upper mounting plate. All normally open contacts should just close as the key top touches the gauge. Form the stationary contact stiffener to adjust. After adjusting, recheck the stationary contact tension as in (2) above.

NOTE: In place of a .180" gauge, a No. 15 drill may be used. A suitable alternate can be any round object 3/16" in diameter, such as the shank of the yellow handled screwdriver (T18106) found in most tool kits.

(4) With the stationary contact adjusted as in the preceding step, check to see that each movable contact has enough tension to provide AT LEAST .050" gap for all normally open contacts. Form the movable contact spring to adjust.

(5) Depress the Plus key. Hold it locked down by manually operating the lock bail, and make sure its contacts remain closed. If not, form the stationary contact stiffener. (Be certain lock bail adjustments are correct before adjusting contacts.)

(6) Depress the Minus key. Make certain that the normally open contacts touch positively. Hold the lock bail operated and make sure that the normally closed contacts are NOT touching. The normally open contacts need not remain closed with the key locked down.

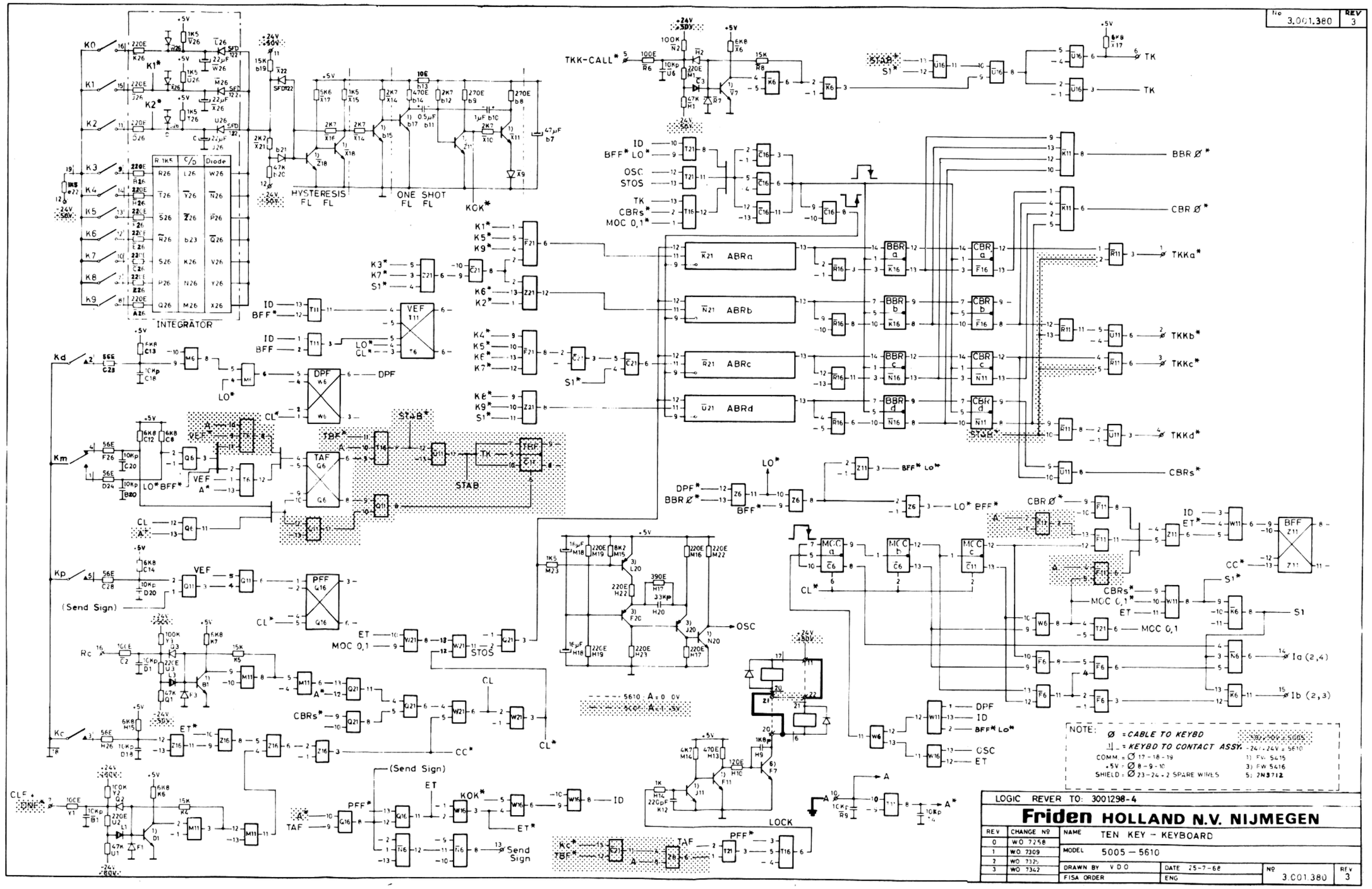

No 3.001.380
REV 3
INTEGRATOR
HYSTERESIS FL FL
ONE SHOT FL FL
KOK*
TKK-CALL*
STAB*
TK
BBR Ø*
CBR Ø*
TKKa*
TKKb*
TKKc*
TKKd*
CBRs*
S1
Ia (2,4)
Ib (2,3)
ABRa
ABRb
ABRc
ABRd
OSC
STOS
VEF
PFF
DPF
STAB
LOCK
Send Sign
NOTE:
Ø = CABLE TO KEYBD
= KEYBD TO CONTACT ASSY.
COMM. = Ø 17 - 18 - 19
+5V - Ø 8 - 9 - 10
SHIELD : Ø 23 - 24 - 2 SPARE WIRES
LOGIC REVER TO: 3001298-4
Friden HOLLAND N.V. NIJMEGEN
NAME TEN KEY - KEYBOARD
MODEL 5005 — 5610
DRAWN BY V.D.O
DATE 25-7-68
FISA ORDER
ENG
Nº 3.001.380
REV 3

TABLE OF CONTENTS

	Page
SECTION 10, MAINTENANCE	
PREVENTIVE MAINTENANCE	10-1
Writing Machine	10-1
Processor	10-1
Miscellaneous	10-2
Checkout	10-2
Auxiliary Units	10-3
CORRECTIVE MAINTENANCE	10-3
Locating the Malfunction	10-3
Correcting the Malfunction	10-3
TEST/DISPLAY PANEL	10-4
General Description	10-4
Operation	10-9
Logic Description	10-13

SECTION 10, MAINTENANCE

Maintenance has traditionally been divided into two areas, preventive maintenance and corrective maintenance. In actual practice, these two areas cannot be completely separated, however, because a degree of preventive maintenance always follows any corrective actions performed. Also, the experienced CSRep knows that he is invariably called upon to carry out at least minor corrective measures whenever a preventive maintenance inspection is performed: one of the purposes of the maintenance inspection is to recognize minor deficiencies and correct them before they cause a breakdown.

Therefore, it should be realized that even though some of the items in this section are discussed under either "Corrective" or "Preventive" maintenance, they are not necessarily confined to that area. They will, on occasion, be performed in conjunction with steps in the other group.

A. PREVENTIVE MAINTENANCE

A regular maintenance inspection should be performed on each 5610 twice a year. In addition to the items covered in the following list, the CSRep should make an attempt to determine what problems, if any, have been occurring recently, and then keep these problems in mind as he goes through the steps of the inspection. If practical, he should review the Service Letters beforehand, in regards to the machine being inspected, so that any necessary parts may be procured ahead of time.

The "453" oscilloscope is necessary in making the delay line read amplifier adjustments. It should also be used when checking power supply outputs, but care should be taken when reading ac voltages, since the oscilloscope shows peak voltages, and not rms values.

The maintenance inspection procedure follows:

1. WRITING MACHINE

   (1) Perform standard 2201 FLEXOWRITER writing machine maintenance inspection.

   (2) During or after this inspection, install any NECESSARY changes as dictated by the FLEXOWRITER-JUSTOWRITER Service Letters or the 5610 COMPUTYPER Service Letters. Pay particular attention to FLEXOWRITER-JUSTOWRITER Service Letter #312, pertaining to the carriage return pulleys. (Refer also to 5610 Service Letter #33.)

2. PROCESSOR

   (1) If applicable, replace batteries in battery option (once every two years).

   (2) Tighten all ground connections.

   (3) Inspect the nylon screws holding the heat sink/PS card assembly to the rear of the power supply. Replace if necessary. (Ref. Service Letter #33.)

   (4) Check power supply outputs and adjust, if necessary, to conform to the following: (Refer to Service Letter #47 for adjustment procedures.)

TEST POINT	VOLTAGE	TOLERANCE
TS1-7 & 8	115 volts rms	+11.5 volts
TS2-5	+6.0 "	.05 volt
TS2-4	+11 "	.05 "
TS2-6	-11 "	.05 "
TS2-3	+24 "	.8 "
TS2-8	-24 "	.8 "
TS2-9	+50* "	.5 "
TS2-11	1 volt rms	.1 "

*Not measured to GND, but to Lamp Common at TS2-10. (Be certain to disconnect scope from wall outlet ground or 5610 ground before connecting it to TS2-10.)

(5) Perform the "fine" adjustment on all delay line read amplifiers. The "coarse" adjustment will not normally be necessary. Refer to Service Letter #41.

3. MISCELLANEOUS

(1) Inspect all cables and plugs to make sure all cable jackets are securely held in the plug cable clamps.

(2) Remove the tape handling spools and clean all chad and dust from the tape transport area. If necessary, an extremely small amount of oil can be applied to the plastic rewind bearings, but care should be exercised because excessive oil will drip onto the card cage. A drop of oil should be applied to the bronze bearing supporting the punch tape supply reel.

The clutches in the two rewind assemblies may be disassembled, cleaned, and relubricated if they have insufficient drag to wind the tape tight enough. If, after relubrication, these clutches still have insufficient drag it may be necessary to replace the brass wear plate (#358056) or even the tape reel drive (#358007). On the other hand, if relubrication provides too much tension on the tape, the clutch will have to be disassembled again and some of the grease removed. (It is possible, with a clutch having excessive drag and a reader or punch having light detent action, to skip a code or to tear a tape now and then.)

(3) Glue any laminated desk surfaces that may have separated.

(4) Clean the window of the sliding desk top. Clean the rest of the desk and the exterior of the 2205.

(5) Replace any broken tape guides.

(6) Carefully review the list of service letters and install those that apply.

4. CHECKOUT

(1) Turn on system power. Depress the Type Only switch and check the signal on JCCK as various keylevers are operated. If necessary, adjust SC9-1 & 2 to provide a pulse of 20 ms, -5 ms, +10 ms, on JCCK. Adjust the selector code contacts 3 & 4 (each bit) to close before and open after SC9-1 & 2. This can be checked by observing the waveforms on JCCA through JCCJ, and making sure that none of the pulses seen there are SHORTER than those seen on JCCK.

NOTE: The JCC connector is designated as JC on older 2205 schematics: JCCK is shown as JCK, JCCA as JCA, etc.

(2) Make the same check with reader entries: if the pulse on JCCK is not within the tolerance previously specified, adjust SRCC1-1 & 2 to correct. Be certain that the reader code contacts 1 & 2 (each bit) close before and open after SRCC1-1 & 2. This can be checked by observing the waveforms on JCCA through JCCK as described above.

NOTE: All input lines should be standing at +15 volts, ±2 volts, quiescently. When the entry is made, they should drop to +.7 volt, +.3 volt, -.7 volt, and then rise back to +15 volts after the time limit specified. If these tolerances are not met, the source of the discrepancy should be located and corrected.

(3) Monitor the input lines, JCCA through JCCK, with the 2205 in tape skip. All input lines should remain $\geq$ +10 volts.

(4) Load a program that will provide repetitive output instructions. (This can be Test #5 of the "Branch Acceptance Test", the "Production 5610 Flexowriter Test", or some similar program.) Initiate the type out test and monitor the input lines (JCCA through JCCK) to make sure they are maintained at +15 volts, ±2 volts, during execution of the output instructions.

5. AUXILIARY UNITS

Any auxiliary units used with the system should receive the same type of inspection that the "standard" counterpart unit receives. When this is done and after all of the above checks have been completed, any auxiliary input unit should be used to repeat the preceding test 4 (2). Adjust the auxiliary unit's reader contacts as necessary.

B. CORRECTIVE MAINTENANCE

Corrective maintenance encompasses two major areas: locating the malfunction and correcting the malfunction. The 5610 test/display panel is oftentimes helpful in locating a malfunction, but it is treated separately here because it is a separate piece of hardware.

1. LOCATING THE MALFUNCTION

Since the number of different malfunctions is nearly infinitesimal, no method of troubleshooting can be defined which will be equally successful in all cases. The following material is presented simply as a guide, and should not be considered complete. This is not a PROCEDURE, but simply a list of items to be considered when troubleshooting.

(1) Determine the extent to which the problem is related to the 2205. The great majority of 5610 malfunctions are centered in the 2205.

(2) Check suspected 2205 failure by operating it in Type Only or through use of a short test program.

(3) Eliminate the possibility of auxiliary unit problems.

(4) Be certain that all switches, both on the processor and on the 2205, are in the correct position. Be certain the correct 2205 program panel is properly installed.

(5) Check the possibility of operator error.

(6) Isolate the problem--narrow it down as far as possible. What appears to be an error in Multiply could be caused by an error in Entry. This allows it to be repeated more easily through optimization of the program for troubleshooting.

(7) Check delay line and power supply adjustments. Make certain that delay line data is in correct time relationship.

(8) Observe the following points in relation to signals and scope use:

(a) Are there too many or too few pulses?

(b) Is the operation occurring in the correct phase and mode?

(c) Will it help to sketch signal relationships as seen on the scope, for comparison with the scope trigger?

(d) Is the best trigger signal being used? Do not use the suspected "bad" signal for synchronization: use a known "good" signal.

(e) How SHOULD the machine work? What SHOULD be seen? Is the scope being used properly?

2. CORRECTING THE MALFUNCTION

Once the source of the error has been located, it is usually a relatively simple matter to correct it. Correction usually takes on one of two forms: either replacing defective components, or making some type of adjustment. Concerning the former, the usual precautions about using

heat sinks, rosin core solder, etc., will not be elaborated upon here because all CSReps should be aware of these aspects of servicing. However, it is highly recommended that either the "Soldapullt" desoldering tool (T-18620) or some "Wik-it" (T-18624) be used to remove solder when extracting defective components. These aids make possible easy removal of the old part without damage to the circuit card, and leave clean holes for insertion of the new part. Use of one of these aids precludes the need for clipping out an old component and soldering the new component to the old leads: this practice leads to cold solder joints and should be avoided.

As far as adjustments are concerned, the 2205 and auxiliary unit adjustments are covered in the 2200/2300 Series Theory, Operation, and Adjustment Manual, TP-700 RD. 5610 adjustments are covered in the Service Letters. The following list briefly summarizes the 5610 Service Letters covering the different adjustments.

Service Letter #	Adjustment(s)
16	Interim Power Supply
41	Delay Line Read Amplifiers
47	Universal Power Supply

C. TEST/DISPLAY PANEL

The primary purpose of the test/display panel is to allow the CSRep to stop 5610 operation at some predetermined point in order to observe the contents of various registers and the conditions of certain signals at that time. The panel also makes it more convenient to observe certain signals during 5610 operation.

In the case of arithmetic line operation, action can be stopped and restarted after each action step (and, with a modification installed, after each line pass). In the case of program line operation, however, once the 5610 has been stopped, it is not possible (in most cases) to restart operation correctly.

The following description of the test/display panel is broken up into three main parts: General Description, Operation, and Logic Description. The former serves mainly as an introduction to the device in general. The second part explains the ways in which the panel may be used. The latter part, although not required for the CSRep to be able to use the panel, provides insight into the operation of the panel, and should make possible wiser and more efficient use of the panel.

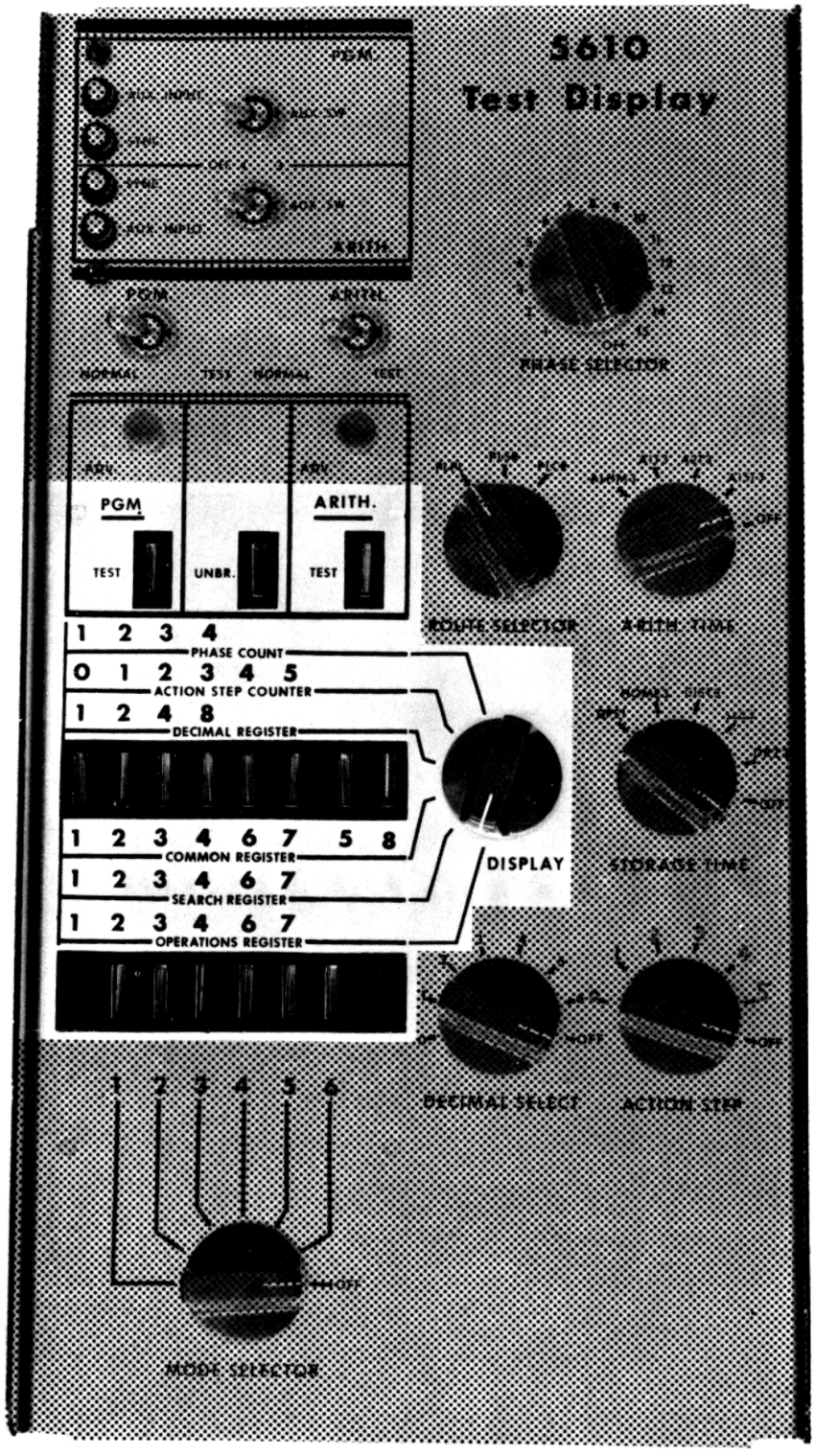
1. GENERAL DESCRIPTION

The test/display panel consists of (1) the display panel, (2) two printed circuit cards (the "L" and "AA" cards), and (3) the cables connecting these two elements to each other and to the processor.

a. DISPLAY PANEL. The display panel can be broken up into three main areas: the DISPLAY portion, the PROGRAM LINE portion, and the ARITHMETIC LINE portion.

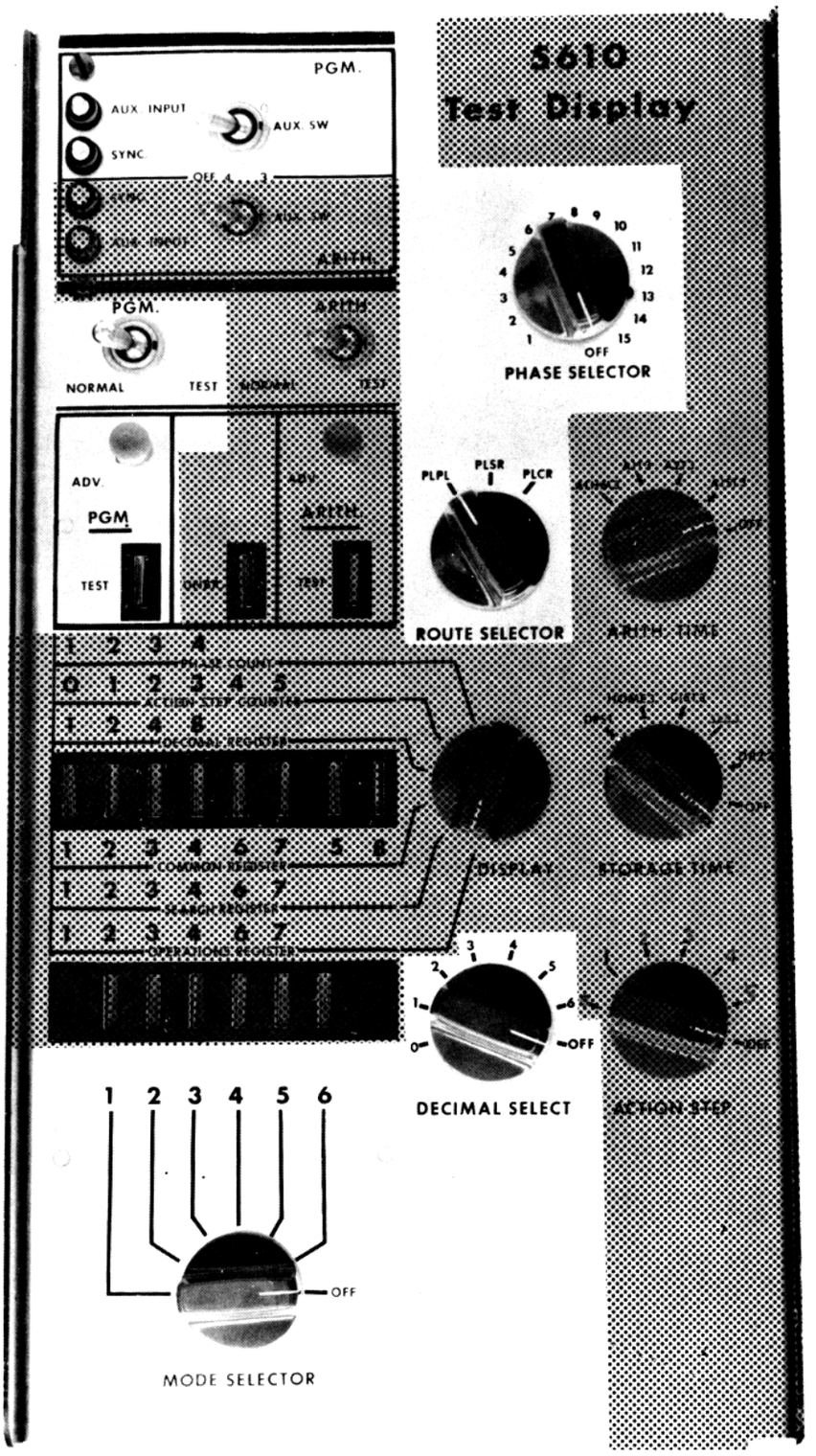
(1) Display. (Refer to Figure 10-1.) This portion of the display panel includes all of the lamps (indicator tubes) and the Display rotary switch. The lamps are arranged in groups: mode, status, UBR, and test.

(a) Mode Indicators. The mode indicators are located in the lower left portion of the display panel. One of the six tubes will be illuminated, designating the mode that is currently "true" in the machine.



DISPLAY PORTION
TEST/DISPLAY PANEL

FIGURE 10-1



PROGRAM PORTION
TEST/DISPLAY PANEL

FIGURE 10-2

(b) Status Indicators. There are eight status indicator lamps, located in the left-center of the display panel, just to the left of the Display rotary switch. From left to right, these lamps are numbered 1-2-3-4-6-7-5-8. The lamps display the contents of the register selected by the Display rotary switch.

(c) Display Switch. This rotary switch is used only in conjunction with the status indicators. It selects the register whose contents are to be displayed. That is, if the Display switch is set to "Operation Register", a combination of the status indicator lamps will light, representing the code in the operation register at that time. The "5" and "8" lamps are associated only with the "Common Register", since this is the only register having eight flip-flops.

When the Display switch is in the "Phase Counter" position, the lamps show the condition of the Phase Register flip-flops: a copy of the phase counter truth table is pasted to the side of the display panel to facilitate decoding of the display into one of the 15 phases.

When the Display switch is set to "Action Step Counter", the status indicators do not show the condition of the ASC flip-flops, but instead light only one lamp, showing which of the six action steps is true at the time.

(d) UBR. The UBR lamp illuminates whenever the Unconditional BRanch flip-flop in the 5610 is set.

(e) Test Lamps. The PGM and ARITH test lamps are illuminated whenever machine operation has been stopped through operation of other test/display panel switches. This will be expanded upon shortly.

(2) Program Line. (See Figure 10-2.) The program line section of the display panel consists of the Mode Selector, Decimal Selector, Route Selector, and Phase Selector rotary switches, the PGM TEST and program AUX toggle switches, the PGM SYNC and AUX INPUT terminals, and the PGM ADV pushbutton.

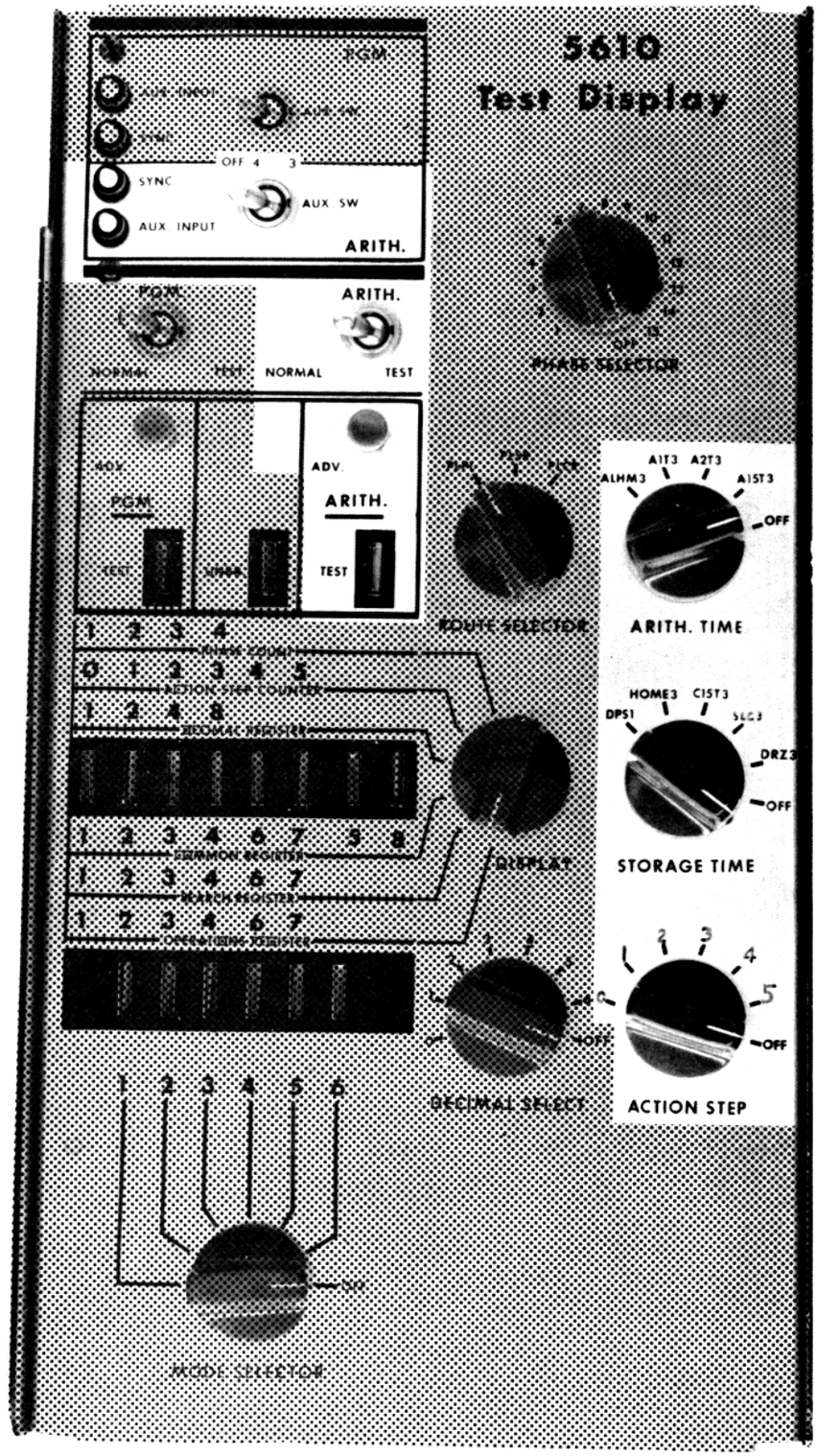
The 5610 can be stopped at any point in the program line operation of an instruction by presetting the mode, decimal, and phase switches to the desired point and transferring the PGM TEST toggle switch to "Test". When the 5610 reaches the point selected by these switches, it stops all further action until the ADV pushbutton is depressed. The Route Selector must be set to the line routing desired: this is usually the routing normally in effect at the mode/decimal/phase setting at which operation stops.

The AUX INPUT and AUX toggle switch are used together to add one more qualification to the test, as described above. For example, the CSRep may want to stop machine operation in 4 mode 3 phase only when a Halt instruction is retrieved. He would then jumper OQ4 from H9 or K7 on the mother board to the AUX INPUT. If a false signal is available, the AUX toggle switch remains in the "4" position: if only a true signal were available (OQ3), the AUX toggle switch would have to be thrown to the "3" position.

The PGM SYNC terminal provides a point from which an oscilloscope may be triggered. The time of triggering is determined by the setting of the mode/decimal/phase rotary switches and by the signal applied to the AUX INPUT. In other words, the scope trigger will appear at the same point in the program as the machine would stop, had the PGM TEST switch been operated. (Although it is possible, the SYNC will not normally be used to trigger the scope when machine operation is being stopped through operation of the PGM TEST switch.) The trigger pulse is a positive-going signal.

(3) Arithmetic Line. (Refer to Figure 10-3.) This portion of the display panel is made up of the Arith. Time, Storage Time, and Action Step rotary switches, the ARITH TEST and AUX toggle switches, the ARITH SYNC and AUX terminals, and the ARITH ADV pushbutton.

The operation of these various switches is very similar to the operation of the program line portion of the display panel: a scope trigger pulse will appear at the SYNC terminal each time all of the qualifications are met, as dictated by the settings of the three rotary switches and any auxiliary input used. (Note that with nothing connected to the AUX INPUT, action is the same as if a false signal were connected.)



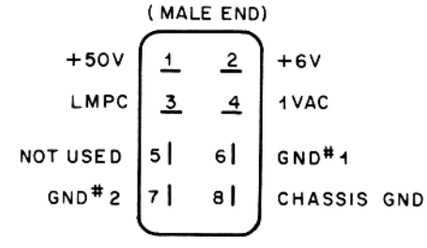
ARITHMETIC LINE PORTION
TEST/DISPLAY PANEL

FIGURE 10-3

Arithmetic TEST operation is somewhat different from the program line TEST operation. To begin, only one rotary switch, the Action Step switch, controls the point at which operation will stop. In addition, when the ADV pushbutton is operated, normal operation resumes from the point at which action was stopped.

b. PRINTED CIRCUIT CARDS. The "L" and "AA" cards are similar to the other 5610 printed circuit cards in that they are inserted into the card cage in the same way and the components on them are of the same type. They are different, however, in that they each have a number of cable wires soldered to them at the end opposite the plug pins. Thus connections are made to the 5610 through the plug pins, and to the display panel through the soldered connections and the cables.

c. CABLES. There are two cables from each of the printed circuit cards to the display panel, and one additional cable from the display panel to the processor. This latter cable, designated JTP, contains seven wires, which provide the different power and ground lines for the display panel. See Figure 10-4.



JTP CONNECTIONS

FIGURE 10-4

2. OPERATION

There are basically two functions which the test/display panel is capable of performing: providing signals for use on an oscilloscope, and stopping machine operation at a predetermined point. These two divisions can be further segregated into "program line" operation and "arithmetic line" operation.

a. INSTALLING THE TEST/DISPLAY PANEL. The 5610 must always be turned off when the test/display panel is being installed. Not only is there some danger of damaging some components, but the test flip-flops on the two printed circuit cards must be reset (by ST9) when power is first applied, so normal operation would not occur if the cards were inserted with the power on. In addition to inserting the "L" and "AA" cards in their respective sockets, it is necessary to mount the display panel in the two slotted holes in the side of the card cage and to insert the JTP plug into its socket near the bottom of the delay line carrier assembly.

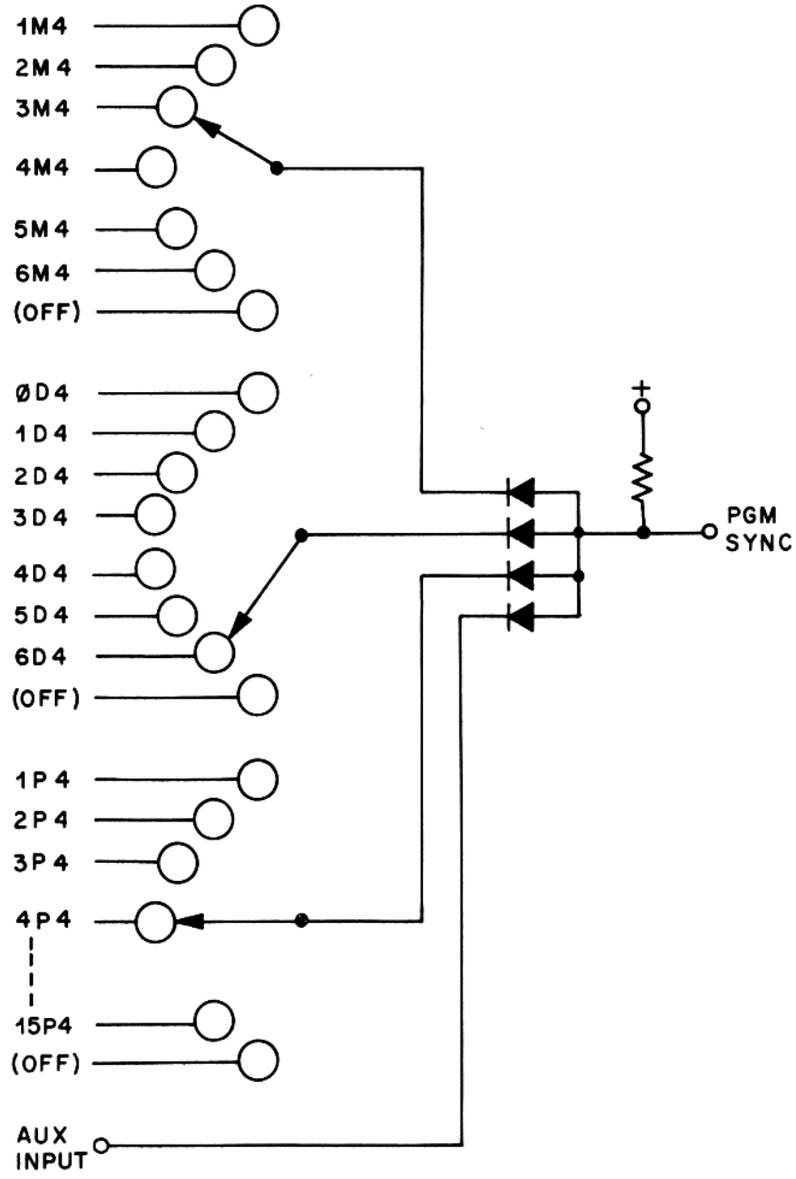
b. PROGRAM LINE OPERATION. The program line portion of the test/display panel is more useful as a source of scope signals than as a means of stopping 5610 operation at a given point. This is mainly due to the fact that it is difficult to resume correct operation once the machine has been stopped.

(1) Sync. The SYNC terminal is useful not only as a source of trigger signals for the scope, but it makes the observation of many different signals easier by providing these signals at the display panel, thus eliminating the need to hunt for them on the mother board.

(a) Scope Trigger. The SYNC terminal is normally at ground potential, and goes to +6 volts when the processor, in the course of executing an instruction, reaches the mode, decimal, and phase settings indicated by the position of the respective rotary switches. The SYNC terminal is the output of a false AND gate, requiring all inputs to be false before the output can become

false. As long as any input is true, the output will be true: as the last input goes false, the output goes false. For example, if the three rotary switches are set at 3 mode, 6 decimal, and 4 phase, 3 mode first becomes true, followed by 6 decimal (or both go true together). Finally, 4 phase becomes true, and this allows the output at SYNC to go false. As soon as the processor leaves 4 phase, the output again becomes true.

A simplified schematic of the SYNC circuit is shown in Figure 10-5 (a). In the example shown (3M6,4), the SYNC output will remain true until 3M4, 6D4, and 4P4 all go false. If another signal is applied to the AUX input, it will also control the SYNC output.

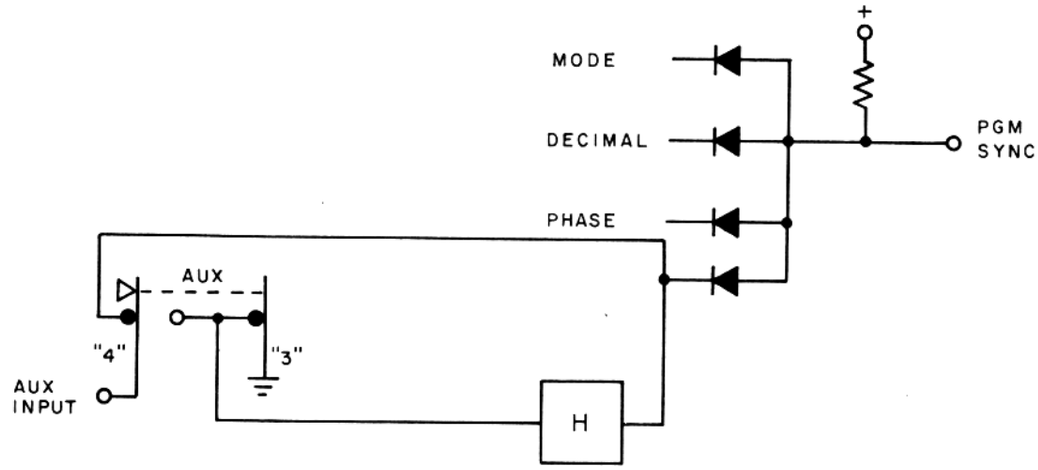


SIMPLIFIED SCHEMATIC - TEST PANEL
PROGRAM SYNC CIRCUIT

FIGURE 10-5 (a)

Note also in Figure 10-5 (a) that each rotary switch also has an "Off" position. This allows the qualification of that switch to be removed. For example, the scope can be triggered every time 6 phase is reached by setting the Phase Selector to "6" and positioning the Mode and Decimal Selectors to "Off". Thus the SYNC signal goes false whenever 6P4 goes false, regardless of the mode and decimal signals true at the time.

Figure 10-5 (b) shows the wiring of the AUX INPUT circuit in more detail. With the AUX switch in the "4" position, any auxiliary input used must be a normally true, or "4" signal. Such a signal will go false when the desired function occurs, allowing the SYNC output to follow. If such a signal is not available, a "3" signal may be used, but the AUX switch must then be transferred to the "3" position. This applies the auxiliary input to the gate through an inverter, providing the same operation as would occur had a "4" signal been available.



AUX INPUT CIRCUIT
PROGRAM SYNC

FIGURE 10-5 (b)

(b) Signal Source. The SYNC terminal is also quite useful as a source of signals to be observed on the scope. For example, the Mode and Decimal Selectors can be positioned to "Off", and the SYNC output then becomes the wiper of the Phase Selector. By attaching the scope probe (not the scope trigger input) to the SYNC terminal, any of the phase signals may be observed simply by positioning the Phase Selector switch to the desired signal. It follows that the decimal settings may be observed by positioning the Phase and Mode Selectors to "Off" and rotating the Decimal Selector to the desired setting. Thus, switching from one signal to another is merely a matter of turning the switch, instead of removing the probe from one pin and fastening it to another.

For example, assume there is some problem in searching (3 mode). The following program could be loaded (without loading an error sequence), and any keylever except the "4" depressed.

L 1 2 3 H 4 ½ 5 6

The 5610, if operating anywhere near normally, will stay in 3 mode searching for the error sequence. While it is in this condition, each of the phases may be observed by attaching the scope probe to the SYNC terminal and turning the Phase Selector to each phase, in turn (Mode and Decimal Selectors "Off"). If the scope is triggered at program line home, the entire line pass can be seen on the scope screen, and the relationship of each phase to the others can be seen, simply by rotating the Phase Selector knob.

The above is merely an example: other similar techniques can be developed as the situation arises.

(2) Program Test. As mentioned earlier, the 5610 operation can be stopped, at a point dictated by the settings of the Mode, Decimal, and Phase Selector switches and any AUX INPUT added, by transferring the PGM TEST toggle switch to the "Test" position. Once the 5610 stops in this manner, the PGM TEST lamp illuminates, and the 5610 "idles" until the ADV pushbutton is operated. As in "sync" operation, any combination of the Mode, Decimal, and Phase Selectors may be set to "Off" in order to remove them as a qualification.

(a) Precautions. There are certain precautions which must be observed if proper operation of the program test portion of the panel is to be achieved.

The Route Selector switch must be set to provide the line routing normally occurring at the mode/decimal/phase setting under test. When the 5610 stops under test conditions, all phase signals (1P3, 2P3, 3P3, etc.) are driven false, and since a phase signal is needed for every program line routing, all normal program line routings are disabled. The Route Selector switch only becomes "live" during the time the 5610 is stopped under test. When more than one line routing normally occurs in a given mode/decimal/phase setting, only the one that recirculates program line data can be maintained after the 5610 is stopped.

Once 5610 operation has been stopped by the test panel, it can be restarted only at program line home time. If it is stopped at some other time, certain conditions may be set up which would not normally occur during program line home. If the ADV pushbutton is operated under these conditions, 5610 operation resumes with incorrect conditions, and most of the time an "endless loop" results. Therefore, if the 5610 is stopped at some time other than program line home, program reloading should be expected. Even if an "endless loop" did not immediately result, subsequent operation would be in error.

(b) Uses. Since correct resumption of 5610 operation is impossible except after stopping at program line home, the usefulness of the test/display panel in this function is limited. Of course, any conditions occurring at program line home can be checked easily, and 5610 operation can then be resumed correctly. This method can be used to advantage by setting the panel to stop at 3M6,4. Then, each time the ADV pushbutton is operated, the 5610 will perform a single instruction and then stop. The Route Selector must be set to PLSR.

Other uses of the program test portion of the panel are limited to testing to see whether or not a given mode/decimal/phase setting is ever reached. For example, if an unconditional branch instruction is not working correctly, the panel could be set up to stop the 5610 in 3 mode 4 decimal. If the operation reaches this point and stops, it can be assumed that everything up to that point is working properly: OCBS3 has been developed and the correct sequence has been located. If operation does not stop, the panel can be set for 3 mode 5 decimal.

If the 5610 stops under these conditions, it can be assumed that OCBS3 has been developed and that the problem is somewhere in the sequence search. Other operations can be tested similarly, but it may be necessary to reload the program many times before achieving any meaningful results: it is wise to use a short program when utilizing this portion of the test/display panel.

c. ARITHMETIC LINE OPERATION. As in the program line section, there are basically two parts to the arithmetic line portion of the test/display panel: sync and test.

(1) Sync. The basic operation of the arithmetic line sync circuit is quite similar to that of the program line sync circuit, so it will not be covered in detail. Note, however, that a negative-going pulse, instead of a positive-going pulse, will appear at the ARITH SYNC terminal when all of the conditions specified by the Arith Time, Storage Time, and Action Step rotary switches are met. An AUX INPUT terminal and a "3-4" (normal-invert) toggle switch are also available for qualifying the signal even further when desired.

As in the program line section, each rotary switch has an "Off" position to remove it as a qualification, and by positioning two of the switches to "Off", the wiper of the other one is available at the SYNC terminal for either triggering the scope or monitoring the waveform.

This portion of the test/display panel is not as useful for observing waveforms as is the program line section because there are fewer signals available and those that are available are either not as useful or are readily accessible on the mother board anyway.

(2) Test. Arithmetic line test operation is different from program line test operation in that only one of the three arithmetic rotary switches is used in the test function: when the ARITH TEST toggle is thrown to "Test", 5610 operation will stop when it reaches the action step specified by the Action Step rotary switch. The Arithmetic Time and Storage Time switches control only the SYNC output terminal.

When the ARITH ADV pushbutton is depressed, operation resumes at arithmetic line home time, and since all action steps begin and end at this time, operation resumes normally. By positioning the Action Step switch to "Off", it is possible to step through an instruction one action step at a time.

By modifying the test/display panel as per Service Letter #52 it is possible to step through instructions one line pass at a time. This is valuable in troubleshooting arithmetic instructions when incorrect answers are resulting.

3. LOGIC DESCRIPTION

A complete logic drawing of the entire test/display panel is shown in Figure 10-7 at the end of this section. Refer to that drawing for the following discussion.

a. DISPLAY. All of the display lamps utilize the same type of circuitry. Each "lamp" is actually a miniature triode electron tube. A positive 50 volts is applied to the anode, and 1 volt ac, riding at a positive 3.6 volts, provides the heater current for the directly-heated cathode. The grid is connected, through the proper rotary switch, to the controlling signal. When this signal is at +6 volts, the tube is conducting and it is "lit". When the controlling signal is true, the grid is at ground, and the tube is biased "off".

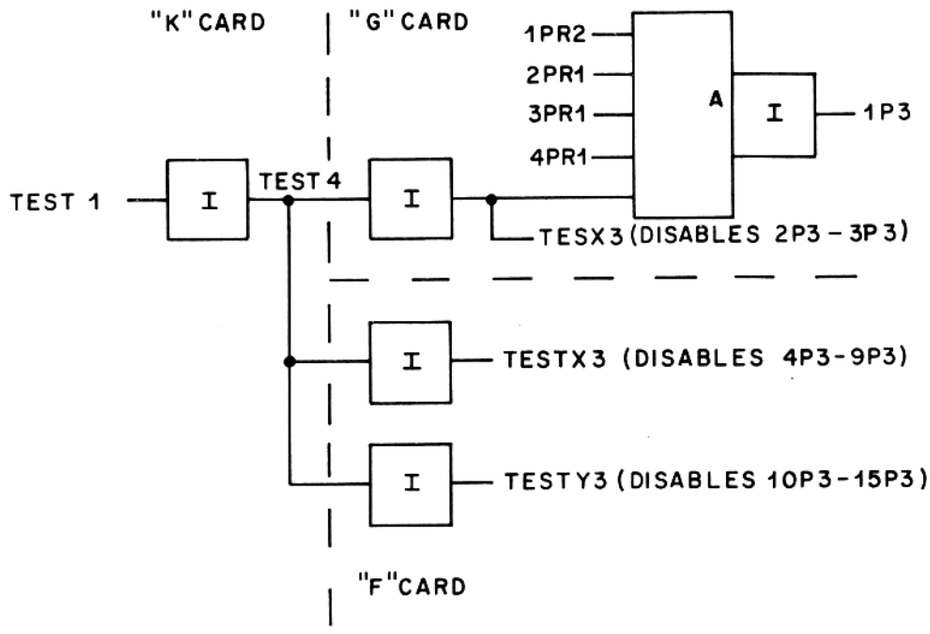
There are six levels to the 6-position Display Selector switch, each of which controls one of the indicators, I1, I2, I3, I4, I6, or I7. Each level is connected to the corresponding flip-flop signal from the phase counter, operation register, search register, common register, decimal register, and to the decoded action step counter signals. When the rotary switch is turned, all levels are switched simultaneously. Note that the unused positions of levels E and F are grounded so that I6 and I7 will remain off when the decimal register or phase register is selected. Note also that the grid circuits to I5 and I8 are not switched, and that they will ALWAYS reflect the condition of the 5CR and 8CR flip-flops, respectively.

The Mode indicators and the UBR indicator are also wired directly, and will always indicate the state of their respective signals, regardless of the positions of the switches. The grids of the two test indicators are driven positive and the tubes "light" when the corresponding test flip-flops set. This will be covered shortly.

b. PROGRAM LINE. Figure 10-7 shows that the Set TeST signal, STST4, is the output of another AND gate, having the same inputs as the SYNC gate. This output is held grounded, however, until the PGM TEST toggle is operated. When this occurs, the TEST flip-flop sets when all of the qualifications, as determined by the three rotary switches and the AUX INPUT, are met. When TEST sets, TEST2 goes false, allowing I15 to turn "on".

The TEST flip-flop setting also drives all phase signals false, as shown in Figure 10-6. This makes it necessary to provide artificial line routings through the Route Selector switch. Note that when the PLSR or PLCR routings are selected, the "B" level of S8 provides the necessary SRPL or CRPL routing.

In order to resume operation once it has been stopped, the PGM ADV pushbutton must be depressed. This sets the Test ADVance flip-flop, TADV. After TADV sets and the pushbutton is released, the following program line home resets TEST and allows operation to resume. TEST2 going true resets TADV.



TEST FLIP-FLOP
ACTION ON PHASE SIGNALS

FIGURE 10-6

c. ARITHMETIC LINE. There are few differences between the arithmetic portion logic and the program portion logic. The major difference in the sync logic is that the SYNC output in the arithmetic portion is inverted. The major difference in the test logic is that only one switch (Action Step) controls the stopping of operation in the arithmetic portion, whereas three switches and the AUX INPUT control the program test circuit.

As in the program line portion, the arithmetic SYNC output changes when all inputs become false. Since there is an inverter on the output of the AND gate, however, the SYNC output will go from false to true: if this signal is used for triggering the scope, the scope must be adjusted to compensate for the different slope of the signal. Note also that since the arithmetic and storage line timing chains are not synchronized to each other, one of the two corresponding switches will usually be used in the "Off" position.

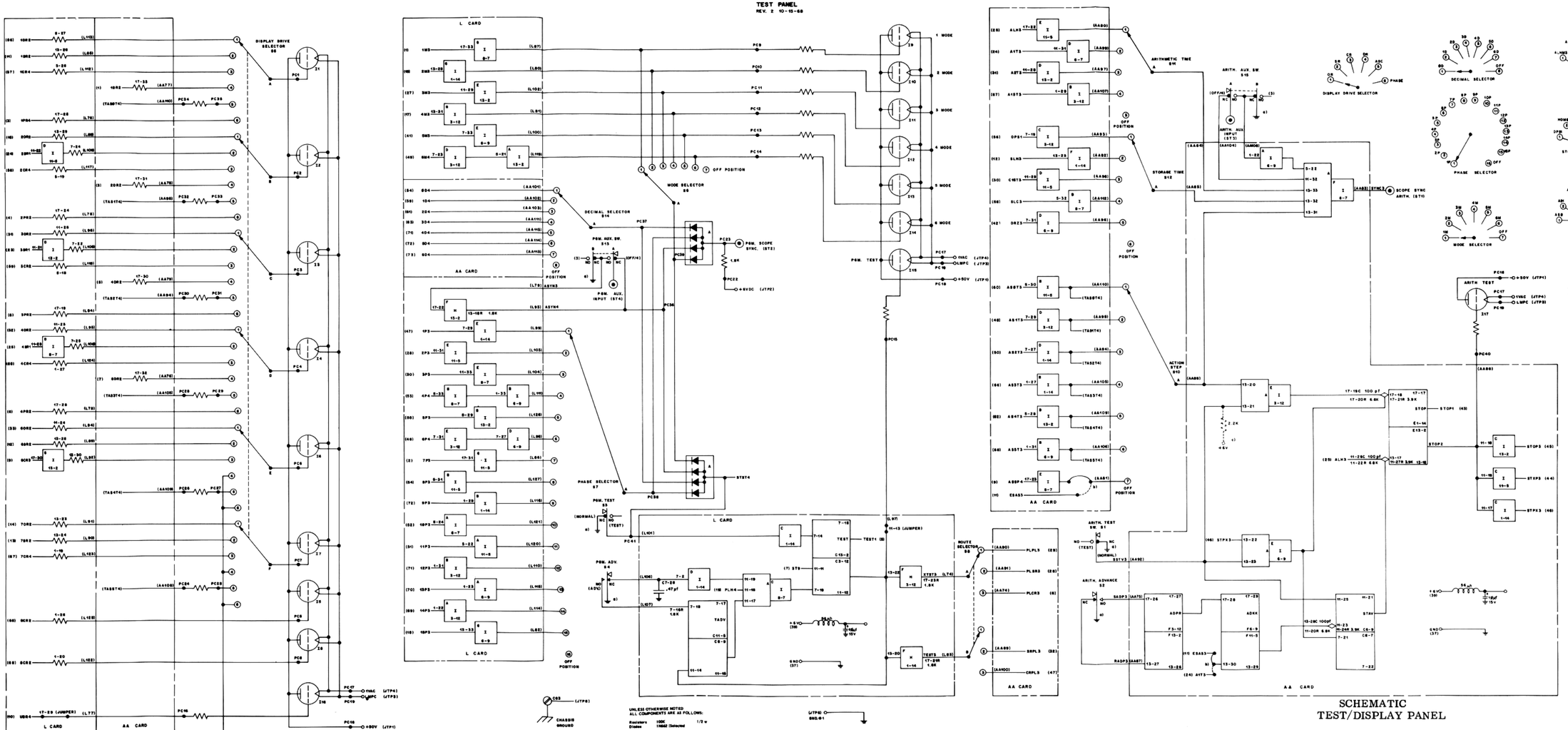
The Action Step switch alone controls the arithmetic test circuit: when the ARITH TEST toggle is thrown to the "Test" position, a pulse to set the STOP flip-flop is developed each time the selected action step is reached. Note that if the Action Step switch is set to "Off", the STOP flip-flop will set each time the action step is changed.

When STOP sets, virtually all arithmetic line action ceases, except for the recirculation of data on the delay line. The STOP signal, and its variations, are applied to many gates throughout the machine, preventing the development of such signals as S1MD, XS1L, AS1L, SROD, SEDP, ROED, BASC, SDMF, S6CR, ETR, etc. In short, once the STOP flip-flop sets, all mathematical action ceases and data is simply recirculated ALAL.

The logic associated with resuming arithmetic operation is slightly more involved. Depressing the ARITH ADV pushbutton sets the ADvance PRime flip-flop, ADPR, which also serves to "debounce" S2. ADPR immediately sets ADKK. The pushbutton can now be released, and ADPR will reset. ADKK does not reset, however, until the following A15R4 time, when ESAS3 goes true. As ADKK2 goes true, it sets the STop AdVance flip-flop, STAV, which "permisses" the reset gate of STOP, allowing it to reset at the next arithmetic line home. When STOP resets,

it unclamps all of the arithmetic logic signals throughout the machine and allows normal operation to resume. As STOP resets, STPX3 goes false, "permissing" the STOP flip-flop's own set gate and resetting STAV. Normal machine operation then continues until STOP sets again by virtue of the proper action step's being reached again.

Service Letter #52 explains a modification that can be made to the test/display panel that will allow stepping through an arithmetic instruction one line pass at a time instead of simply one action step at a time. ASSP3 at the "Off" position of the Action Step switch is replaced with ESAS3, so that a pulse is available to set STOP at every A15R4 time, instead of only when the action step changes. Also, ESAS3 at the reset to ADKK is replaced with A1T3 so that a set and a reset to STOP will not both be developed at the same time. Refer to Service Letter #52 for more details.



SCHEMATIC
TEST/DISPLAY PANEL

FIGURE 10-7

TP

STC-762-36 Rev. 2
10-15-68

TP-161SL