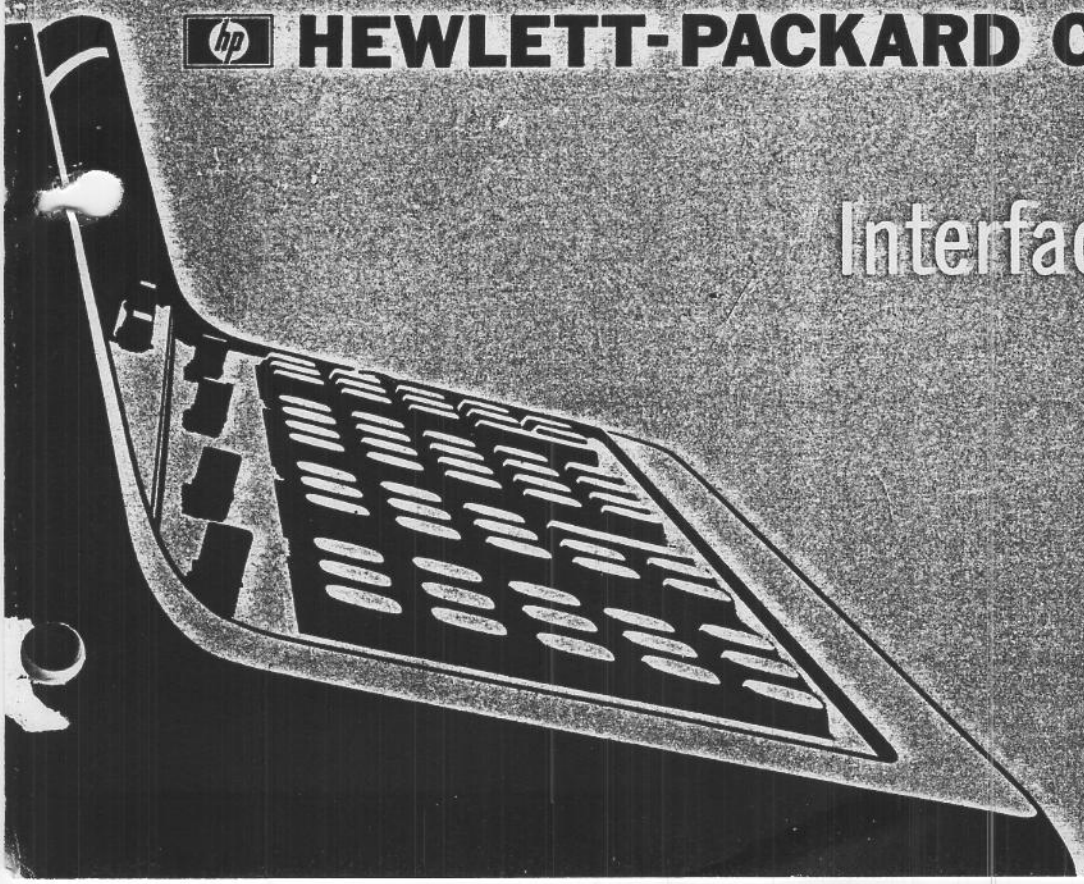
L/C. R E DAW

HEWLETT-PACKARD CALCULATOR

Model 9100A/B

Interfacing Manual



CAUTION

IT IS POSSIBLE TO SEVERELY DAMAGE THE 9100 BY IMPROPER HANDLING OF THE SIGNALS ON THE I/O TERMINALS. THE 9100 WARRANTY DOES NOT COVER THE REPAIR OF SUCH DAMAGE.

HEWLETT hp PACKARD

Interfacing Manual

HEWLETT-PACKARD CALCULATOR

Model 9100A/B

P.O. Box 301, Loveland, Colorado, 80537 U.S.A.

09100-90031

Printed: FEB 1970

TABLE OF CONTENTS

SECTIONS

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS
CONTINUED

LIST OF TABLES

SCOPE OF THE MANUAL

This manual defines the Input/Output (I/O) capability of the rear terminals of the 9100. It explains how the 9100 can be interfaced to digital devices such as voltmeters, counters, printers, or other computing mechanisms. Explanations of the standard peripherals such as the -hp- 9120A Printer, -hp- 9125A Plotter and -hp- 9150A Calculator Monitor, are in Appendix IV. These explanations illustrate the fundamental principles, and help the designer avoid compatibility problems between his interface and the standard -hp- peripherals.

Whatever the user's application, he must design interfacing hardware that meets both the requirements of his system and of the 9100. This manual explains the requirements of the 9100. In doing so this manual assumes that the user is familiar with the contents of the operating manual for his 9100.

The 9100A and the 9100B interface in the same way, except for some minor differences. The differences will be pointed out when they occur. In this manual 9100 means either the A or the B model.

Everyone who plans to construct an interface to a 9100 should read this entire manual. Although a section may give a comprehensive explanation of an I/O option, there may well be additional factors that will affect its use in the particular interface being planned. In writing this manual every effort has been made to document all of the facts of interest to the designer of an interface. Some of those facts resist organization. However, every presently known detail and exception is mentioned some place in the manual.

DEFINITIONS

A definition of terms is in order. These definitions are made for convenience in forming subsequent explanations. Figure 1 illustrates some of these definitions.

Run Mode — Run Mode is a phrase that describes the type of Calculator behavior that occurs whenever the PROGRAM/RUN switch is in the RUN position, and nothing is done on the I/O terminals to override that setting.

Program Mode — Program Mode is a phrase that describes the type of Calculator behavior that occurs whenever the PROGRAM/RUN switch is in the PROGRAM position.

Key-code — The key-code is the basic unit of communication with the Calculator. Each key on the Keyboard is represented by a key-code. At present it is sufficient to think of a key-code as just a code (with some meaning, of course) that is not necessarily associated with any particular piece of Calculator hardware. Key-codes are two-digit binary coded octal numbers. See Appendix V or your operating manual.

Example: The STOP key has key-code 41_8.

Operation — An operation is a Calculator activity that can be specified by supplying a key-code (or, in certain instances, a sequence of key-codes), which will be performed in accordance with the overall rules of Calculator behavior.

Examples of operations: + or 33_8, Format Stop or $42_8 - 41_8$, and Go To (3) (9), or $44_8 - 03_8 - 11_8$. Any single key [say, Go To ( ) ( )] could be considered an "operation".

DEFINITIONS
CONTINUED

For our purposes an operation may originate in one of three ways; two of these ways are similar and are grouped together.

Errand — The dictionary defines an errand as a short trip taken to attend to some business, especially for someone else, or, the object or purpose of such a trip. We shall say the Calculator is running (or processing) an errand if it is acting upon an operation that originated with its Keyboard or I/O terminals ("I/O terminals" includes the Magnetic Card Reader).

Errands can originate in two ways.

Keystroke — A keystroke is an errand that originated at the Calculator Keyboard.

Remote Entry — A remote entry is an errand that originated at the I/O terminals.

The third way an operation can originate is as a program step read from program memory.

Statement — A statement is an operation that originated as a step in a program that the Calculator is running. Sometimes it is convenient to call a closely related sequence of program steps a "statement".

Example: We might refer to the "Go To (3) (9)" statement.

It should be understood that all statements originate as errands, since that is the only way program steps can be stored in the Calculator memory.

We need some terms to describe what the Calculator may do as it performs an operation.

Process (verb) — Process is the general term used to denote whatever the Calculator does to accomplish the operation at hand. One of the main activities of the Calculator is to process key-codes.

Process (noun) — We will have occasion to describe four major sections of internal Calculator logic. Each will be given a name using the word "process".

Example: The Display Process

One of two things happen when an operation is processed.

Storage — An operation can result in the storage of a program step into program memory. This results in a statement. This will always be the result of an errand performed while the Calculator is in Program Mode.

Execution — The Calculator executes an operation when it accomplishes the primary meaning associated with the key-code, as opposed to storing the key-code in the program memory.

Example: To execute the operation 36_8 (X) the Calculator performs a multiplication, or completes the Format operation $42_8 - 36_8$, whichever is called for.

It should be pointed out that operations can be executed several ways. The key-code 06_8, for instance, can be executed three ways:

1) enter the digit 6 into the X Register, (normal execution).
2) as part of an address of the Calculator memory.
3) as part of the Format operation $42_8 - 06_8$, (Format 6).

Table 1. Key-codes.

KEY-CODE	KEY	KEY-CODE	KEY
00	0	41	STOP
01	1	42	FMT
02	2	43	IF FLAG
03	3	44	GO TO ( )( )
04	4	45	PRINT/ SPACE
05	5	46	END
06	6	47	CONTINUE
07	7	50	IF $x=y$
10	8	51	STEP PRGM
11	9	52	IF $x<y$
12	e	53	IF $x>y$
13	a	54	SET FLAG
14	b	55	$\|y\|$
15	f	56	π
16	c	57	PAUSE
17	d	60	ACC +
20	CLEAR	61	RCL
21	.	62	TO POLAR
22	ROLL ↑	63	ACC −
23	$x\rightarrow( )$	64	int x
24	$y\rightleftarrows( )$	65	ln x
25	↓	66	TO RECT
26	ENTER EXP	67	hyper ▾
27	↑	67**	$x\leftarrow( )$
30	$x\rightleftarrows y$	70	sin x
31	ROLL ↓	71	tan x
32	CHG SIGN	72	arc ▾
33	+	73	cos x
34	−	74	e^x
35	÷	75	log x
36	×	76	$\sqrt{x}$
37	CLEAR x	77*	▲SUB▾ / RETURN
40	$y\rightarrow( )$		

*For 9100A's key-code 77_8 causes the natural logarithm of 10.0 to appear in the X Register.

** 9100B only

A SIMPLIFIED VIEW OF THE 9100

This discussion references Figure 1. The Display Process is the closest thing to an idling state that the 9100 has. The Display Process forms one register at a time on the CRT. The Sync State changes the Display from one register to the next, and also delays the start of the X Register until a pulse derived from the powerline is received.

At the end of the formation of each register a check is made to see if there is a new errand to be done. If there is not, the 9100 returns to the Display.

If there is a new errand the 9100 enters the key-code that identifies the errand, and then processes the errand.

Processing is accomplished by determining which mode the 9100 is in and going to the appropriate major process. If the 9100 is in Run Mode the Key-Code Execution Process will execute the key-code.

At the end of the Key-Code Execution Process a check is made to see if the 9100 is running a program. (For errands the answer will always be no unless the errand was Continue.) If the answer is no the 9100 returns to the Sync State, and from there to the Display, unless another errand is ready, or sync is removed.

To process an errand in Program Mode the 9100 stores the key-code as a statement in a program. (The Step Program errand is an exception; it causes a special Program Mode Display.)

If the 9100 is running a program the Program Execution Process gets the next statement in the program and gives it to the Key-Code Execution Process, which then returns to the Program Execution Process, unless the last statement was Stop, Format, or Print. Those operations cause the 9100 to stop running the program.

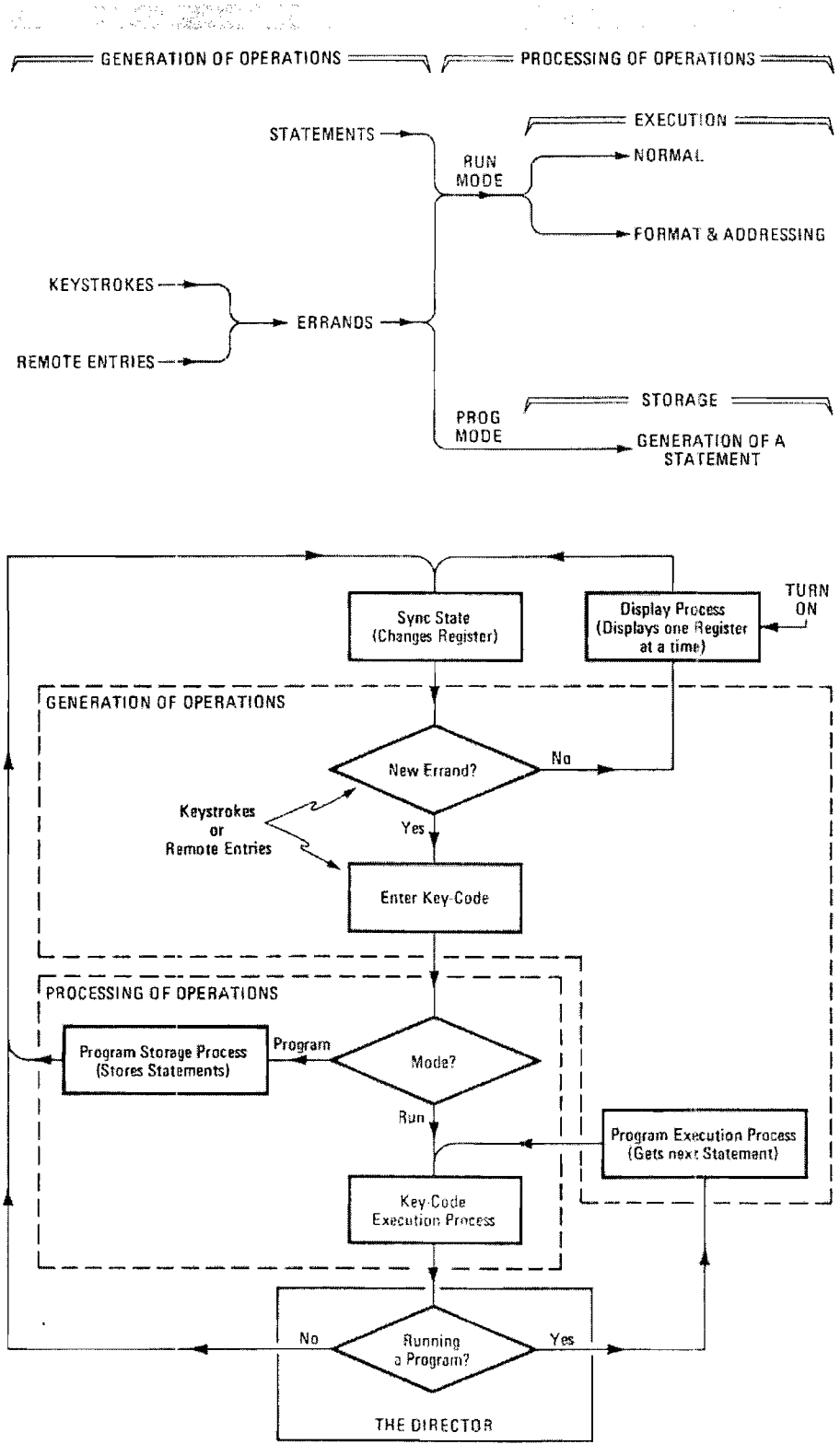


Figure 1. Forming and processing operations.

THINGS THAT CAN BE DONE

In general, the things that can be accomplished by interfacing to the 9100 are limited to the following:

1) Remote entry of any key-code;
2) Output of numerical data stored in the X, Y, or Z Registers;
3) Input and output of blocks of program steps;
4) Control of an external device.

THE OVERALL SCHEME

When devising a scheme for interfacing to the 9100 the user must be aware of the general sequence of internal events in the 9100. Figures 10 and 11 are flowcharts that depict an equivalent of these internal processes, and show the Input/Output (I/O) options associated with each process. A complete explanation of Figures 10 and 11 is given in Appendices I and II.

For external I/O of numerical data and program information the general scheme provides for two speeds of I/O operation. There are slow (limited to less than 30 characters per second) and fairly simple methods, and faster (up to 2,000 characters per second) but more complicated methods. Control of an external device can be relatively simple or quite difficult, depending upon the application, and will be treated in a separate section. Figures 10 and 11 show four major processes that characterize the 9100. In each of these processes and their interconnecting logic there is at least one I/O option.

I/O OPTIONS IN THE DISPLAY PROCESS

The Display Process has only output options (although it takes some input signals as well as output signals to accomplish this), and the output data is limited to whatever data is currently displayed in the X, Y, and Z Registers. This data may be withdrawn at a rate up to the rate at which the 9100 forms the characters in the Display (less than a sixtieth of a second for one presentation of the three registers). The actual rate is up to the user. It may be as slow as necessary.

I/O OPTIONS IN THE PROGRAM EXECUTION PROCESS

The Program Execution Process has input and output options. Output data is limited to information about the program currently stored in the core memory of the 9100. While running a program the Calculator presents each new key-code from the program to the I/O terminals prior to that key-code's execution. The input options to this process are the two ways to halt the execution of a program. If the STOP key is pressed on the Keyboard, or being properly entered on the I/O terminals, the execution of the program will be interrupted and the 9100 will return to the Display Process. A Continue errand must then be used to restart the program. The second way to interrupt the execution of a program stops the Calculator prior to the execution of a key-code, and keeps it stopped until it is released. During the time it is stopped there is no Display, or any other effective activity. For all practical purposes the 9100 is internally stopped. When action resumes it will be as though the 9100 were never stopped. Only one signal is involved in this mechanism and it is YSKIC (see Page 39).

THINGS THAT CAN BE DONE

I/O OPTIONS IN THE KEY-CODE EXECUTION PROCESS

The Key-Code Execution Process has only output options. In this instance, the information that is outputted is not numerical data, nor is it, strictly speaking, a key-code about to be executed by a program. The ouput will depend upon the key-code being executed, whether the operation is a statement or an errand, and on the input signal YAFZ. The results are summarized below.

PRINT OPERATIONS

Print errands

A signal called HOUT is given, and automatically followed by key-code 00_8. The key-code is not executed or otherwise used. It is just presented on the I/O terminals after HOUT is.

Print statements

Same as for Print errands except that the 9100 will stop the execution of the program and wait in the Display for a Continue errand to be entered. (The 9120A Printer enters a remote Continue errand after a printing cycle done in the context of a Program.)

FORMAT OPERATIONS

Format errands

1) YAFZ is not grounded:
   A signal called HFMT is given and followed automatically by key-code 00_8.
2) YAFZ is grounded:
   HFMT is given and the 9100 stops everything to wait for the next errand. Then that key-code is presented on the I/O terminals, and the 9100 returns to the Display.

Format statements

1) YAFZ is not grounded:
   Same as for Format errands except the Calculator will stop after the execution of the Format statement.
2) YAFZ is grounded:
   Here the key-code following HFMT is the one after the Format in the program. Remember — that key-code is not internally used; it is merely presented on the I/O terminals after HFMT is given. After that the 9100 returns to the Display, as the execution of the program is interrupted. A Continue must be given to restart the program.

Appendix II contains some additional information about HOUT and HFMT.

NOTE

On the 9100B there is no YAFZ on the I/O terminals. In the 9100B it is as though there were a YAFZ and that it is always grounded.

THINGS THAT CAN BE DONE

PROGRAM STORAGE PROCESS

The Program Storage Process has input and output options. They are concerned with the entry and inspection of the contents of the core memory. THESE MECHANISMS ARE INTENDED FOR USE WITH PROGRAM INFORMATION ONLY AND SHOULD NOT BE USED FOR THE ENTRY OR INSPECTION OF NUMERICAL DATA. The output mechanism can access numerical data stored in core memory; however, it would be extremely difficult to take advantage of it. See Appendix V for what happens. By the same token the input mechanism of the Program Storage Process can access registers that the user might like to store data in. The user would have to go to great lengths to insure that the data entered ended up as the intended numerical data in the register. These I/O mechanisms are quite useful, however, when dealing with program information. The rate of entry or inspection is limited to 2,000 steps per second, but can be as slow as desired.

ADDITIONAL I/O OPTIONS

The logic connecting the four major processes listed above has some I/O options of its own. One input option is a slow method for the remote processing of key-codes. If the 9100 is in Program Mode the rate of entry can be as high as 30 key-codes per second. If the 9100 is in Run Mode the execution times of the various key-codes are a limiting factor, with a practical maximum being 30 errands per second for even the shortest errands. This method parallels the way the regular 9100 Keyboard works, and is also the easiest to employ. It has a built in delay of 12 msec that allows contact bounce to end, should mechanical devices be used to generate key-codes.

Also, the Calculator presents each new errand to the I/O terminals.

A second group of input options deals with the synchronization of the Display Process. Synchronization can be ignored if desired, thus saving time, or the Display can be halted by making the 9100 wait in the Sync State. Control of the synchronization mechanism is used by several of the I/O options previously mentioned.

FORMATS FOR INPUTTING INFORMATION

It is important that the user understand the sequence of key-codes that represent the entry of information during interfacing.

NUMERICAL DATA

For entry of numerical data the sequence is exactly what it would be if those operations were entered manually from the Keyboard. Several variations are possible. The user should pick the one that best matches his application. For instance, numerical data might always be formatted as a floating point number, but a decimal point may or may not be part of the entry. Or, the input data might be formatted as a fixed point number, allowing the 9100 to keep track of the exponent. One important exception to this is when dealing with numbers whose absolute values are less than 10^{-9}. When entering a leading decimal point followed by several zeros, only the first eight zeros are effective. If a ninth zero is entered and then followed by a non-zero digit, the result is simply decimal point - digit. The best way to see this is to try to enter such a number manually from the Keyboard while the Display is set for floating point. Note that the exponent does not increment from -9 to -10, but from -9 to -0.

REMOTE PROGRAMMING

The sequence of key-codes used to enter a program from the I/O terminals would be identical to that used for manual entry from the Keyboard. The starting address must be set prior to the actual entry of program information.

INPUT SIGNALS

Among the I/O terminals are six lines, NK20 - NK25, called Key Lines. They are used to encode key-codes. A key-code is a binary coded octal number. Its six line code is obtained, complemented, and then applied to the Key Lines. See Appendix V. Note that a Key Line is either left alone or grounded. How the 9100 is made to accept the key-code depends upon which method is being used to enter the key-code, and will be covered in the section titled INTERFACING INWARDS.

There are some input signals that are not Key Lines. See page 24.

FORMATS FOR OUTPUTTING INFORMATION

PROGRAM DATA

Output data may be of two types: program steps and numerical data. Each program step will appear as a binary coded octal number across six lines. The user must select the starting address prior to beginning the actual inspection process. After that, the address will increment in a straightforward manner, as the Step Program operation is used to change the address.

NUMERICAL DATA

For any data to be obtained from the Display, be it program information or numerical data, the characters will be available in an order dependent upon the Display itself. Strictly speaking, the Display is available as output data, and not the contents of the X, Y or Z Registers*. What is available is this: starting with the X Register, the rightmost character followed by those to the left until the leftmost has been presented. Then the Y Register is presented in the same way, followed by the Z Register.

Restricting our attention to presentation of individual characters, we find that they may be presented in each of two ways. The first way is as a binary coded decimal number across six lines. The second way is peculiar to the 9100 and was provided to interface with the 9120A Printer. In the second way each character except the decimal point is given two halves, and a code that corresponds to that particular half. These codes are used to unblank the beam of the CRT as it moves through the Display. See Appendix I for complete details.

OUTPUT SIGNALS

The six lines involved in the outputting of codes are called the Buffered Bits, and are named F20B through F25B. Normally, they are all at ground and are strobed to reveal a code by a signal called ITBB. For the next 500 nsec a code is presented on the Buffered Bits.

When numerical data is transmitted from the 9100, the user must be able to tell which register the data came from. Two lines called the Buffered Words (F41B and F40B) make this possible. In contrast with the Buffered Bits, the Buffered Words are continuously available to the user. The indications are of a DC nature, except when internal transitions are in progress, but are valid only while the 9100 is involved in the Display Process. See Table 4.

There are other output signals available on the I/O terminals. See page 24.

*The displayed registers can contain "blanked non-zero digits". See Appendix I, Blank Characters, and Appendix V, Core Memory Conventions.

NATURE OF THE ELECTRICAL SIGNALS

TIMING

Some of the outputs from the 9100 will be as short as 500 nsec while others will be essentially DC. While the inputs to the 9100 will range from DC to the millisecond range, they will be subject to 500 nsec sampling. Hence, the user should pay close attention to the transition times of the signals he supplies to the 9100: they should be as short as possible.

CABLE LENGTH

Because of those factors, problems in maintaining adequate logic levels at the I/O terminals or at the interface device can arise if the two are separated by a distance of more than approximately three feet. Generally, these problems can be solved by the use of suitable buffers. However, the use of long wires means additional crosstalk. In order to insure maximum logic levels, one should use coaxial cable with suitable driver circuitry.

> CAUTION
> **IT IS POSSIBLE TO SEVERELY DAMAGE THE 9100 BY IMPROPER HANDLING OF THE SIGNALS ON THE I/O TERMINALS. THE 9100 WARRANTY DOES NOT COVER THE REPAIR OF SUCH DAMAGE.**

RECOMMENDED CIRCUITS

It is highly recommended that the user employ the suggested input circuit for each input he wishes to use. See Appendix III. In no case may the user allow a positive DC voltage to come in contact with an input to the Calculator.

The output circuits are less delicate than the input circuits. Still, no external voltage should be applied to any of the outputs, except as shown in the recommended level sensing circuit. Appendix III contains a recommended level sensing buffer circuit.

For noise immunity it is strongly suggested that any output level sensing circuits be referenced to −2V rather than ground. Because the outputs are often generated by internal gating, there is a minimum impedance that can be put from an output to ground, before the internal clamping action is unable to divert current from the output. Thus, too low an input impedance on the sensing device will result in false outputs. This problem disappears and the sensing device can become a true current sink as long as it is refrenced to −2V rather than ground.

SIGNAL LEVELS

The 9100 uses negative-true logic. If an output terminal is more positive than −2V that signal is a logical 0. If the terminal is more negative than −5V (for level sensing by voltage) or supplying current (for current sink type sensors), the output is a logical 1. If an input terminal is grounded the 9100 treats that signal as a logical 0. If an input terminal is more negative than −8V or a high impedance to ground the 9100 treats that signal as a logical 1.

NATURE OF THE ELECTRICAL SIGNALS

SIGNAL TIMING

All output signals except the Buffered Words and ISTP are pulses. The pulse width is about 500 nsec (they are not very sharp and their measurement is open to interpretation). Many signals (ICHF, IADV, IOPF, etc.) occur at irregular intervals, depending upon what the 9100 is doing.

However, there are three output options that have a similar timing nature. They are YCHA, YCOD and YGNPS output options. While they are being used pulses on ITBB and the Buffered Bits will occur repeatedly as shown by Figure 2. This is also the highest repetition rate that is ever found when outputting codes.

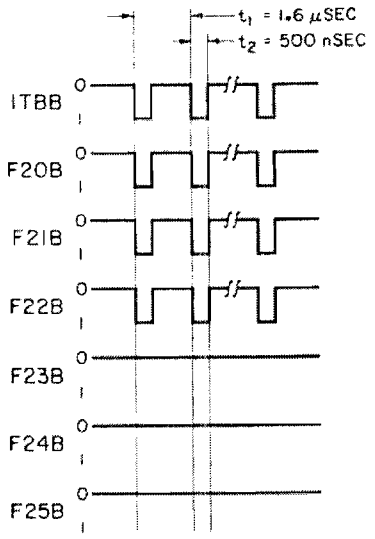


Figure 2. Maximum repetition rate for ITBB and the Buffered Bits.

REMOTE PROCESSING OF KEY-CODES

The following discussion references Figures 10 and 11.

SLOW RATE PROCESSING

Slow rate processing of remote entries is accomplished by applying key-codes to the Key Lines (NK25 - NK20) in accordance with the scheme outlined in Appendix V. In order to make the 9100 operate on an applied key-code all that is necessary is to make YKDN go negative while the 9100 is displaying. Then the key-code will be seized and processed. The 9100 will then come back to the Display Process. This sequence of events is exactly the one used by the Keyboard. In fact, the 9100 cannot tell the difference and almost all rules of Calculator operation are in effect. However, entering the Pause key-code (57_8) will not halt execution of a program when that key-code is held on the Key Lines during a Pause in a program.

In order to cause a key-code to be processed again, or to process another key-code, set to the desired key-code and release YKDN (in either order) for at least 10 msec, and then let YKDN go negative again. Making YKDN go negative will cause ONE use of the key-code. After the key-code is processed YKDN will no longer cause the New Errand Qualifier to be met until YKDN is released for about 10 msec and one pass through the Sync State has been completed.

Making YKDN go negative does not in itself cause the New Errand Qualifier to be met. Twelve msec after YKDN goes negative an internal signal is generated which actually causes the qualifier to be met. The 12 msec is provided to allow contact bounce of the Keyboard switches to end. Hence, mechanical devices can be used to form key-codes, if the above method with YKDN is used.

Roughly speaking, the maximum processing rate is determined in the following way: 12 msec delay for YKDN (spent in the Display or possibly the Sync State) plus a maximum of 5 msec to get back to the New Errand Qualifier (once in the Display) plus processing time of the key-code (Run or Program Mode makes a difference) plus maximum of 16 msec waiting for sync plus a minimum of 10 msec to reset YKDN.

Several variations are possible, depending upon how long it takes to process the key-code and at what point in time YKDN goes negative. For instance, you could get back to the Sync State and have to wait almost the entire 16 msec. Use that time to recycle YKDN and set up the new key-code. But on the other hand, you might get back to the Sync State just before the sync pulse and have to recycle YKDN while in the Display. Another variable is the actual time spent forming the registers, since the New Errand Qualifier is asked after every register. If the Display is being formed in floating point the time per register is fixed at about 5 msec. If, however, the Display is in fixed point, then the time spent per register can be anywhere from less than 0.5 msec to about 5 msec. So it is very difficult to state precisely how fast operations may be processed with the slow method. A rule of thumb for digit entries and other operations with short processing times is that a maximum of around 30 operations per second is practical. If the 9100 is in Program Mode, then all the key-code processing times are the same and short enough to allow the 30 per second limit.

There are things that could be done to enhance this method, but in every case it is done by borrowing one or more ideas from the faster method.

REMOTE PROCESSING OF KEY-CODES

FAST RATE PROCESSING

The main difference between slow rate processing and fast rate processing is that in the fast case all ambiguity is removed about the various delays in the Display and Sync State. In addition, there is no longer a 12 msec wait for contact bounce to end. The limiting factor becomes essentially only the time for the processing of the individual key-codes.

It is necessary to use YDSP if there is a series of key-codes to be processed in a remote manner in the shortest possible time. If YDSP is used to cause the New Errand Qualifier to be met, the qualifier will be met every time it is asked, with no restrictions. If the same key-code were left on the Key Lines for a long time while YDSP was at ground, that key-code would be processed repeatedly. Thus, the user is responsible for changing the key-code on the Key Lines during the processing of the key-code, if he is to maintain the sequence he desires. Now he might find this hard to do for several reasons, some of which may have to do with his source of key-codes. The thing to do in this case is ground NHLD. Then when the 9100 is finished processing the key-code, it will come back to the Sync State and wait there until NHLD is released (5 to 10 μsec would be plenty)* or NDGO is grounded for a few μsec. In this way the user may have as much time as necessary to formulate the next key-code, and yet have it processed immediately when it is ready. See Appendix V for an important remark about NHLD.

REMARKS ABOUT EITHER METHOD

If the 9100 is in Run Mode there is a way to tell right away that the key-code has been accepted, whether YKDN or YDSP was used to enter it. The first IOPF that occurs after the onset of YKDN or YDSP indicates that the key-code has been accepted. This method won't work for remote entry of program information because there are subsequent events that require the key-code to remain present on the Key Lines.

There is yet another way that the acceptance of a key-code can be determined, and it works for most remote processing situations. The exceptions are a Continue errand, a Pause errand and the first two key-codes of a Go To ( ) ( ) errand in the 9100A or (when YAFZ is grounded) the first key-code of a Format errand. The method is to use ICHF to indicate that the operation has been processed, and that the 9100 is headed back towards the Display Process.

If the 9100 is running a program ICHF will not be given after most key-code executions, not even in the case of a Pause. ICHF will be given only when the 9100 goes back to the Display Process to be available for use on errands. This includes when a Stop operation is used to interrupt the execution of a program, and the execution of the Format and Print statements. ICHF will be given after every completed errand except as noted in the previous paragraph.

The disadvantage to using this method is that if control of the Sync State is not employed the amount of time between ICHF and the next asking of the New Errand Qualifier can vary between about 5 μsec and 16 msec, depending upon the sync pulse. The problem is that there might not be much time to get the next key-code set up if maximum speed remote processing is desired. If YDSP is continually grounded during remote processing then the next key-code **must** be set up in that time or there must be control of the Sync State to guarantee time to set up the next key-code. If YKDN is being used, the 9100 will go back

*Releasing NHLD has its drawbacks. See the section titled "A Comment on NHLD", Appendix V.

REMOTE PROCESSING OF KEY-CODES

to the Display Process anyway, and as much time as needed can be taken to set up the next key-code.

So far the idea of remote processing of key-codes has assumed that the 9100 was not running a program. If it is, then the 9100 never ever looks for new errands unless there is a Pause in the program, and even then the New Errand Qualifier is disabled insofar as YKDN is concerned. It is **NOT** disabled for YDSP. YDSP should not be allowed to go to ground during the execution of a program.

This does not mean that a Stop errand cannot break in on the execution of a program. The mechanism for that is in the Program Execution Process, and the next ICHF after key-code 41$_8$ is put onto the Key Lines indicates that the Stop errand has been accepted. All that is necessary is to put key-code 41$_8$ on the Key Lines. It is not necessary that YKDN go negative. The execution of the program will be interrupted, unless the key-code about to be executed by the 9100 is 47$_8$ (Continue - it undoes the effect of the Stop)**. This is so since the 9100 actually does the next step in the program before it finally stops.

The remote execution of the Go To and Format errands is subject to some restrictions. See Appendix II.

**A Print would also result in a Continue.

BLOCK ENTRY OF PROGRAM INFORMATION

PROGRAMMING WITH YSKIC

There is a method using YSKIC that allows storage of entire blocks of program steps at a rate up to 2,000 key-codes per second. The significant difference between this method and the other methods of entry is that with the other methods all entries affect the contents of the X Register, while the method with YSKIC does not. Another difference is that YSKIC can be used to set the rate of information flow, rather than as in any of the other methods.

The method is to ground YRUN, NDGO, YDSP, and YSKIC. After a minimum of 250 μsec after the New Errand Qualifier is asked YSKIC is made true for a maximum of 150 μsec. Rate control is achieved by taking more time before letting YSKIC go true.

The method involving YSKIC enters the key-code three times. (See Figures 10 and 11.) The entry immediately following the New Errand Qualifier does not count, and the key-code that is entered there is not used as long as it is not 51_8 (Step Program—don't try to store it as part of a program). The desired key-code must be set up prior to the second "Enter key-code", and last for at least 5 μsec after YSKIC becomes true. After that the key-code can begin its re-cycling process towards the next key-code.

YRUN, NDGO, and YSKIC should already be grounded (in any order) not later than when YDSP reaches ground or all four may be grounded at once by very fast low impedance clamps. This should be done while the 9100 is in the Display Process, or locked in the Sync State.

YDSP should remain at ground until the Program Storage Process has been entered with the last key-code to be stored. YDSP must then be made true before the New Errand Qualifier is asked again. NDGO can be made true at the same time as YDSP, or later if desired. After YSKIC is made true to enter the last key-code in the sequence, it can stay true. YRUN must remain at ground for at least 5 μsec after YSKIC goes true for the last key-code.

Prior to using this method for the block storage of program information, the desired starting address of the program must be entered with one of the remote methods, or from the Keyboard. This would be done with an End or Go To errand.

Strictly speaking, it is possible (but difficult) to use the block entry method to enter numerical data directly into a register (0 through 9 and a through d). See Appendix V for the necessary re-ordering of the digits of the number prior to the entry, and an explanation of the hazards involved.

It is not absolutely necessary to use YDSP to make the New Errand Qualifier be met. It could be done with YKDN. By making YKDN be logical opposites of both YRUN and YSKIC, a hybrid remote method can be developed for the block storage of key-codes. Inside the Program Storage Process the path is exactly the same. The major difference is that entries can be made (at a much slower rate) with less sophisticated logic. There is no worry about multiple entries if the next key-code is not set up in time. YKDN must be recycled each time a new entry is to be made. In between times the 9100 just displays the unaltered X Register.

OUTPUT OF DISPLAYED DATA

Appendix I contains a complete explanation of character and register formation in the Display Process. Data available for output is limited to the characters visible in the Display, and a few other pieces of information, such as the register currently being displayed. Note that characters can be blanks. However, not all blank spaces in the Display are formed from blank characters. See Appendix I. There are no provisions for remote setting of the Decimal Wheel, should fixed point Display be required. Any output scheme that takes data from a fixed point Display must have provisions for a Display that reverts to floating point due to too large a number in the register.

It is more difficult to get numerical data out of the 9100 than to get it in. The biggest problem is that the digits of the Display are presented digit-serial, bit-parallel, from right to left as they are viewed. This is exactly the opposite order that would be necessary to drive a teletype or an electric typewriter. If cost is not a factor the easiest solution to many output problems will be the -hp- 2570A Coupler.

OUTPUT OPTIONS

Grounding YCHA will interrupt the Display Process and cause it to repeatedly output the BCD code for the character being formed. Grounding YCOD interrupts the Display Process and causes it to repeatedly output the unblanking code about to be implemented. These options are provided to allow the user to get information from the Display, and to allow the Calculator to be slaved to an output device that is slower than it is. See Figure 2.

The actual output data takes the form of six line codes on the Buffered Bits. It can be seen from a flowchart of the Display Process (see Figures 10 and 11) that the BCD codes for the characters formed in the Display are available twice per character, whether they are used or not. The user must "ask" for the unblanking codes by grounding YCOD.

BLOCK INSPECTION OF PROGRAM INFORMATION

EDITING WITH YGNPS

There is a method using YGNPS that allows inspection of the program memory of the 9100 at a rate of up to 2,000 key-codes per second. This method does affect the contents of the X Register. YGNPS can also control the rate of information flow.

The method is to ground YRUN, NDGO, YDSP, and YGNPS, while key-code 51_8 (Step Program) is on the Key Lines. About 30 μsec after the New Errand Qualifier is asked, ITBB will begin to repeatedly strobe the Buffered Bits, which will contain the program step in the current program address. Rate control is achieved by taking more or less time to let YGNPS go true after the key-code appears on the Buffered Bits. For all but the last key-code, YGNPS can be made true for at least 5 μsec but less than 150 μsec. After the last key-code is obtained YGNPS can remain true.

This method is based on the properties of the Step Program operation. The only difference between this method and the processing of Step Program is that with YGNPS grounded the 9100 will hang up in an output state until YGNPS is released. YDSP and NDGO are grounded only to allow a higher rate of information flow. YRUN is grounded to force Program Mode.

YRUN, NDGO, and YGNPS should already be grounded (in any order) not later than when YDSP reaches ground, or all four may be grounded at once by a very fast low impedance clamp. Key-code 51_8 must be on the Key Lines at the time YDSP is grounded. This should be done while the 9100 is in the Display Process, or locked in the Sync State.

YDSP should remain at ground until the Program Storage Process has been entered to inspect the last address in the sequence. YDSP must then be made true before the New Errand Qualifier is asked again. NDGO can be made true at the same time as YDSP, or later if desired. After YGNPS is made true following the last key-code it can stay true. YRUN must remain at ground for at least 250 μsec after YGNPS goes true for the last time.

Prior to using this method for the inspection of program information, the desired starting address of inspection must be entered with one of the remote methods, or from the Keyboard. This would be done with an End or Go To errand.

Strictly speaking, it is possible (but difficult) to use YGNPS to read numerical data out of a register that can be addressed through the use of the Program Address. (0 through 9 and a through d). See Appendix V for the ordering of the digits.

The control of an external device will involve all or some of the methods covered to this point. Though not all interfaces are the same, it is probable that: 1) the primary communications path from the 9100 to the device will be Format (XX_s), and 2) the primary communications path from the device to the 9100 will be the Key Lines.

Figure 4 shows all outputs from the 9100A except ISTP, as the 9100A runs the program segment shown in Figure 3.

STEP	KEY	KEY CODE	DISPLAY X	DISPLAY Y
00	GO TO ( )( )	44		
01	1	01		
02	0	00		
10	CLEAR x	37	0	
11	3	03	3	
12	•	21	3	
13	1	01	3.1	
14	4	04	3.14	
15	$x\rightarrow$( )	23		
16	9	11		
17	↑	27		3.14
18	FMT	42		
19	$\sqrt{x}$	76		
1a	PRINT	45		
1b	GO TO ( )( )	44		
1c	0	00		
1d	0	00		

Figure 3. Program segment for Figure 4.

The outputs shown on Figure 4 span the time from just before the first Go To until just after the execution of the Print. After the Format $\sqrt{x}$ and Print statements the 9100A went back to the Display Process. The method used in Figure 4 allowed the X register to be displayed one time, with the Decimal Wheel set to 0, before the New Errand Qualifier was met and key-code 47_8 (Continue) was entered. The things of interest in Figure 4 are listed below:

1) For all operations except the Continue (in the 146th cm) F41B is a 1 when IOPF is given, indicating that the upcoming operation is in the context of a program, ie., it is a statement. The Continue is an errand.

2) The first ITBB after an IOPF reveals a code on the Buffered Bits which is the next key-code to be executed.

3) During each operation there is some additional number of ITBB's. The number varies from operation to operation, and the codes revealed by these ITBB's are of no interest.

4) During each operation there are some ITBB's and IADV's. They convey the classification of the operation under the rules of the Print-After-Printer. See Appendix V.

5) After each operation there is an IADV followed by four ITBB's.

6) The Go To (1) (0) statement results only in one IOPF and one ITBB (which reveals 44_8 - Go To) for the entire statement. The (1) (0) information is not presented. The 9100B does present the address following the Go To. See page 37.

7) The execution of the Format $\sqrt{x}$ statement (90th cm) results in an IOPF followed by an ITBB that reveals 42_8 on the Buffered Bits. Then HFMT is given and the next ITBB reveals 76_8 ($\sqrt{x}$) which is the next step in the program after the Format. (YAFZ is grounded.)

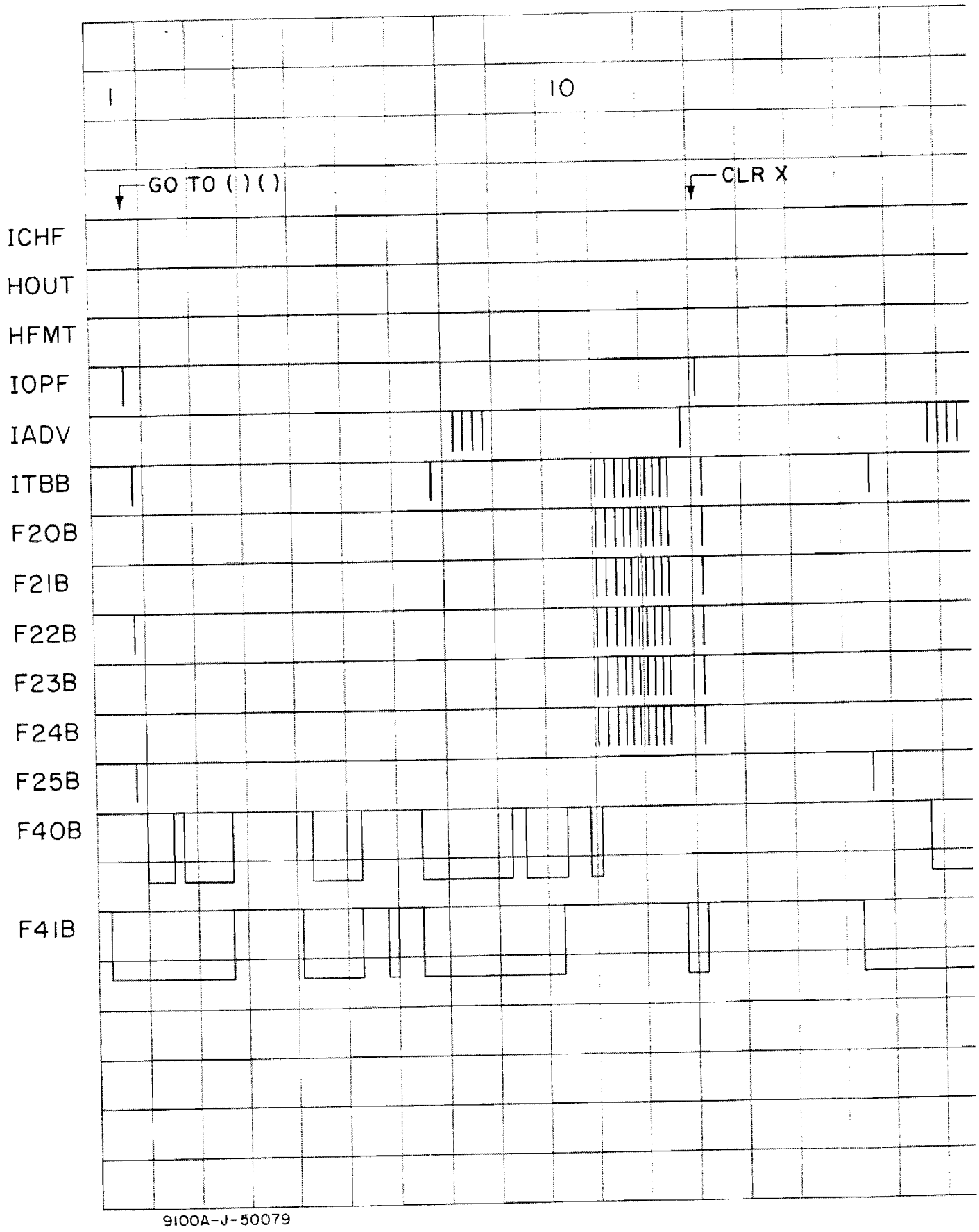
8) After the execution of the Format $\sqrt{x}$, there is an ICHF (95th cm) followed by an IADV prior to the return to the Display Process.

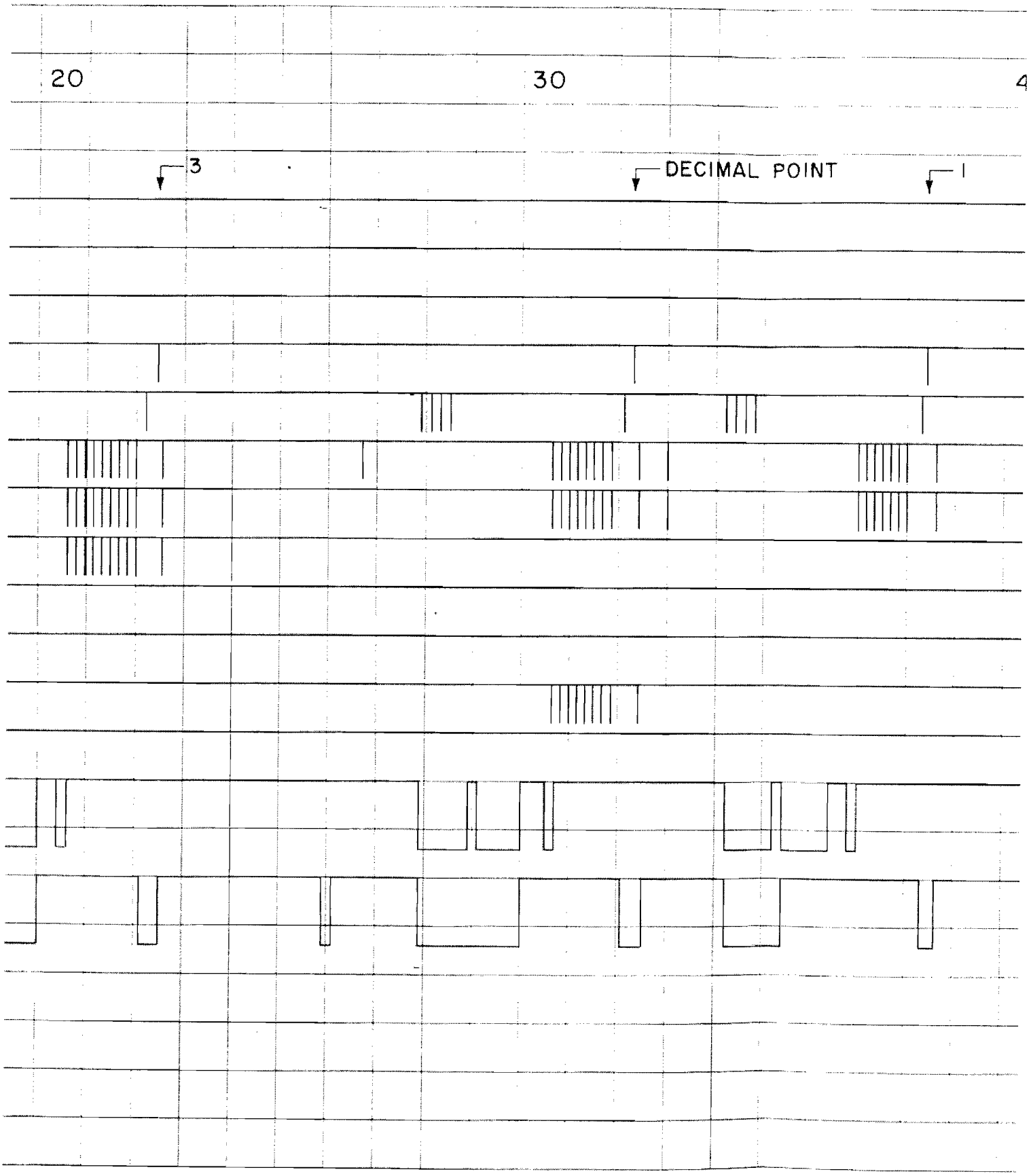
9) During the Display (96th cm through 145th cm) the outputs are in accordance with the scheme outlined in Figure 8. Since the Decimal Wheel is set to 0, all that is displayed is (+) 3.

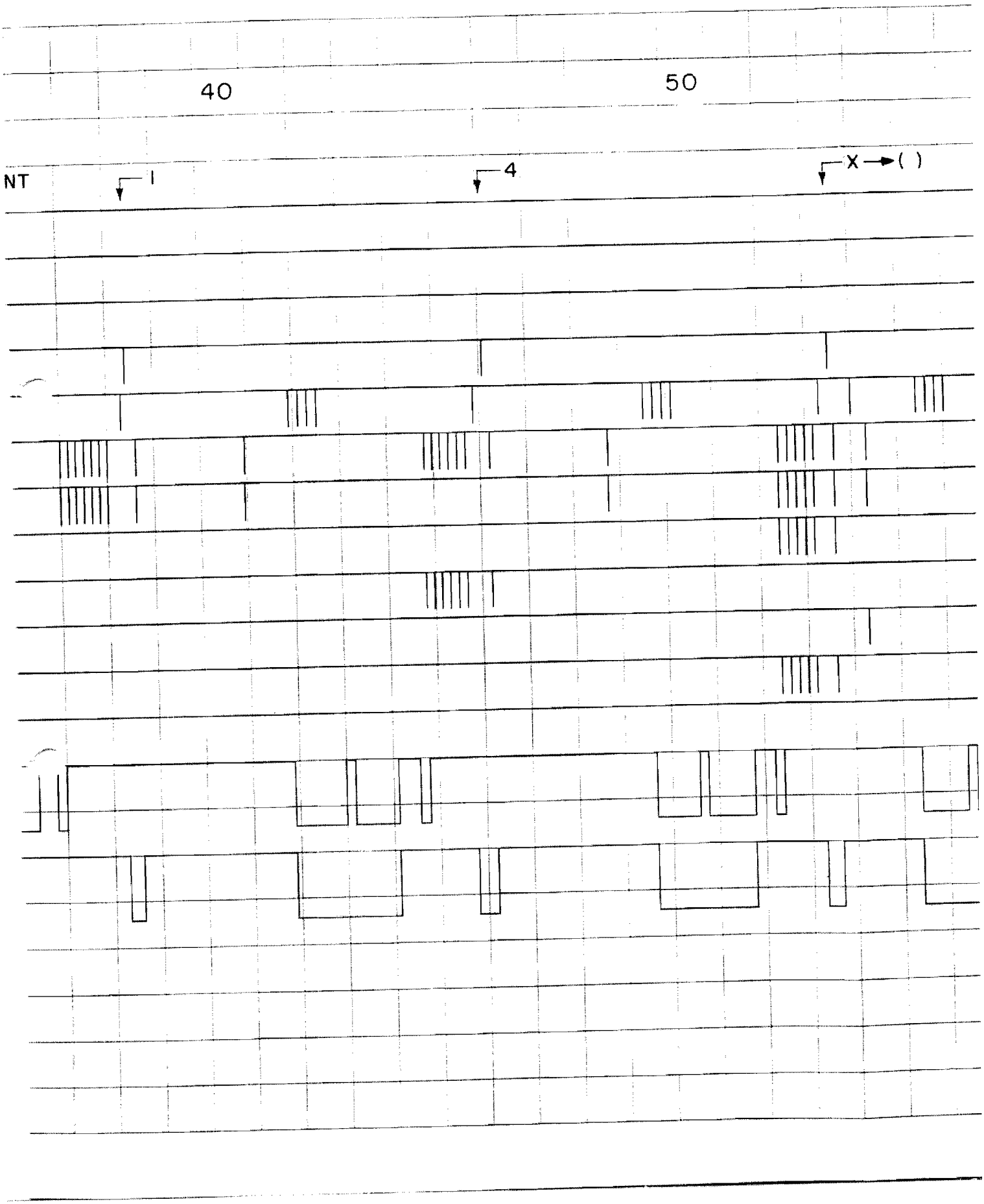
10) The execution of the Print statement results in an HOUT with a glitch on HFMT. See Appendix II.

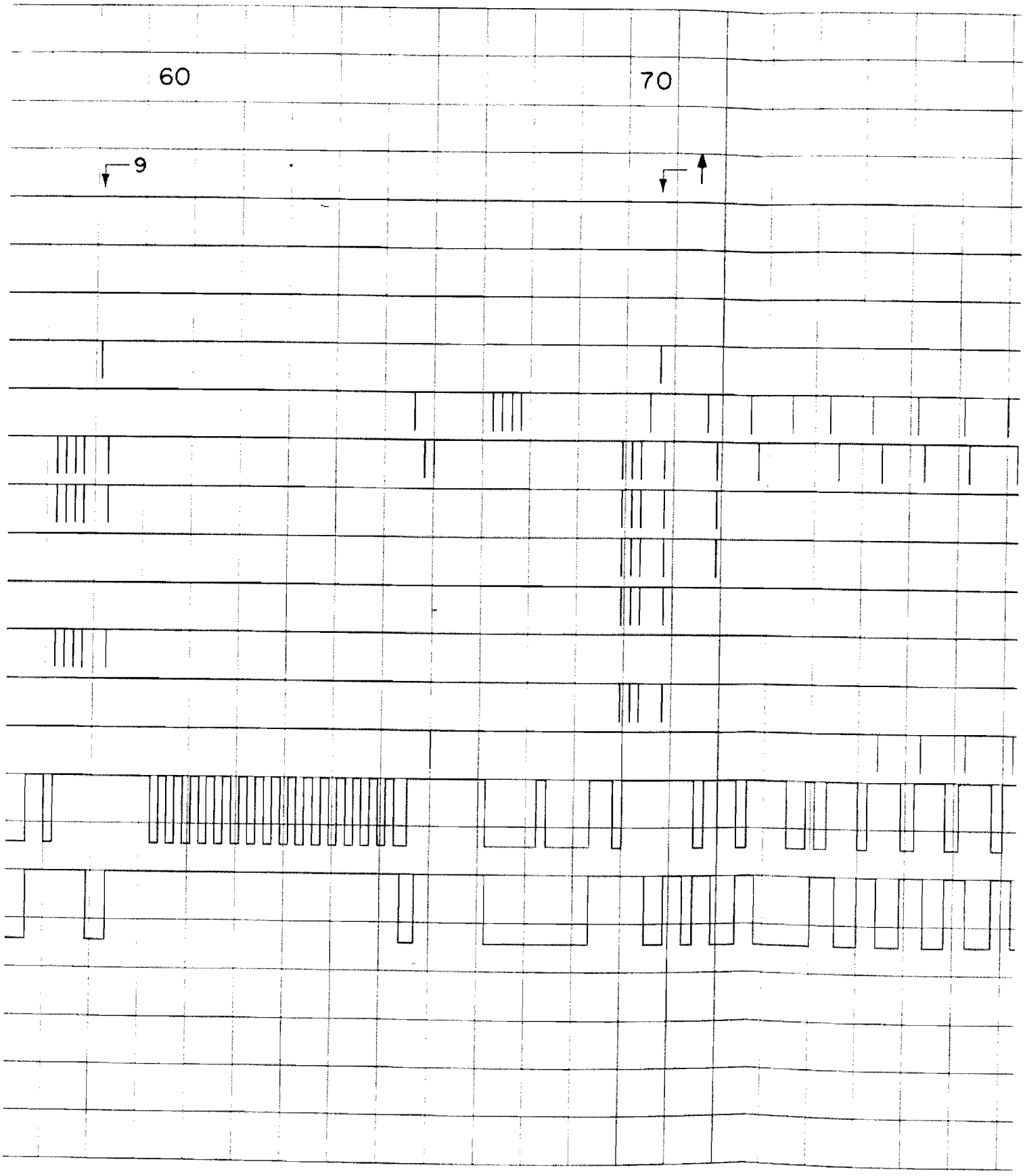
11) After the execution of the Print statement there is an ICHF followed by an IADV as the 9100A returns to the Display Process.

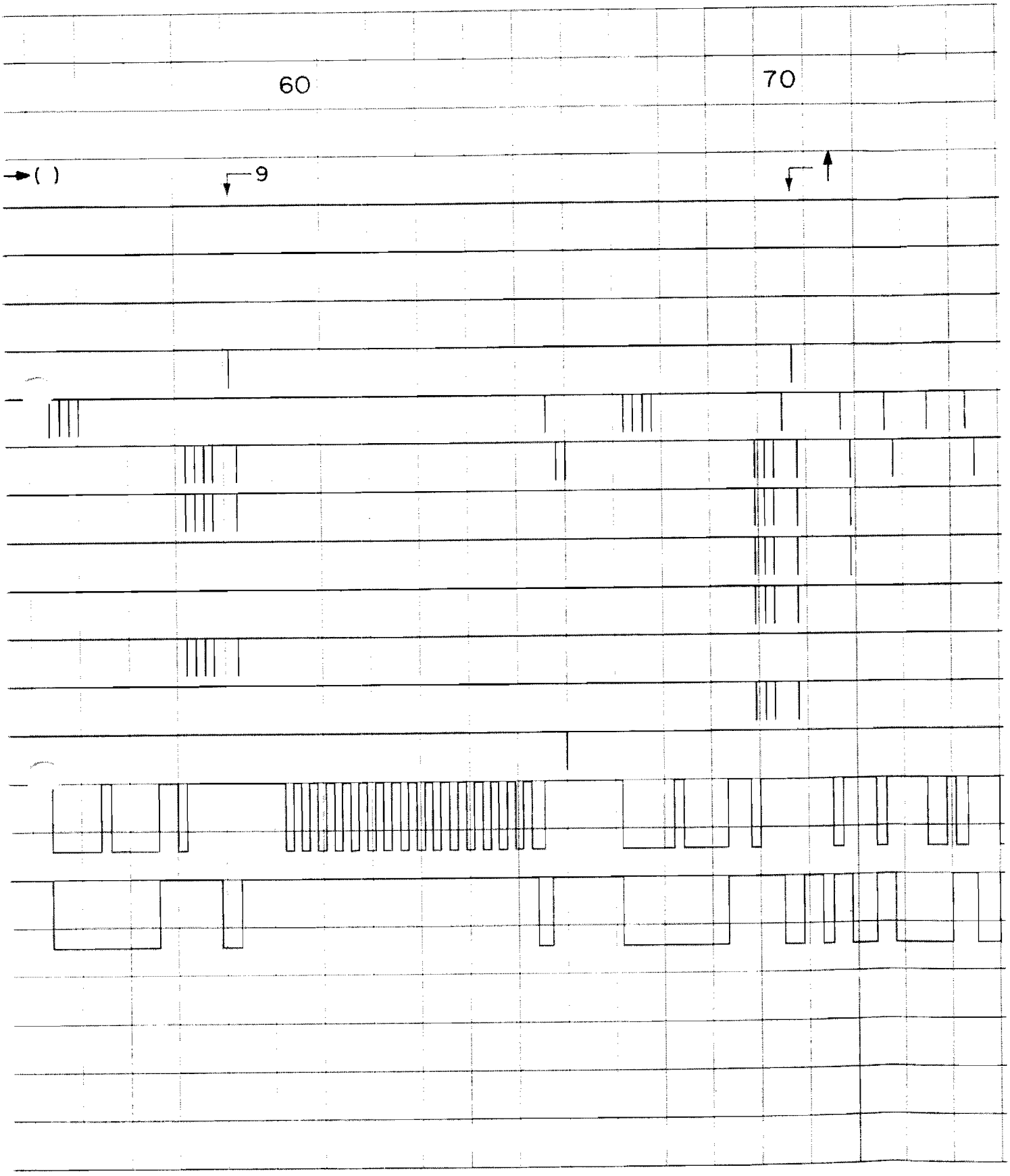
FIGURE 4. ➡

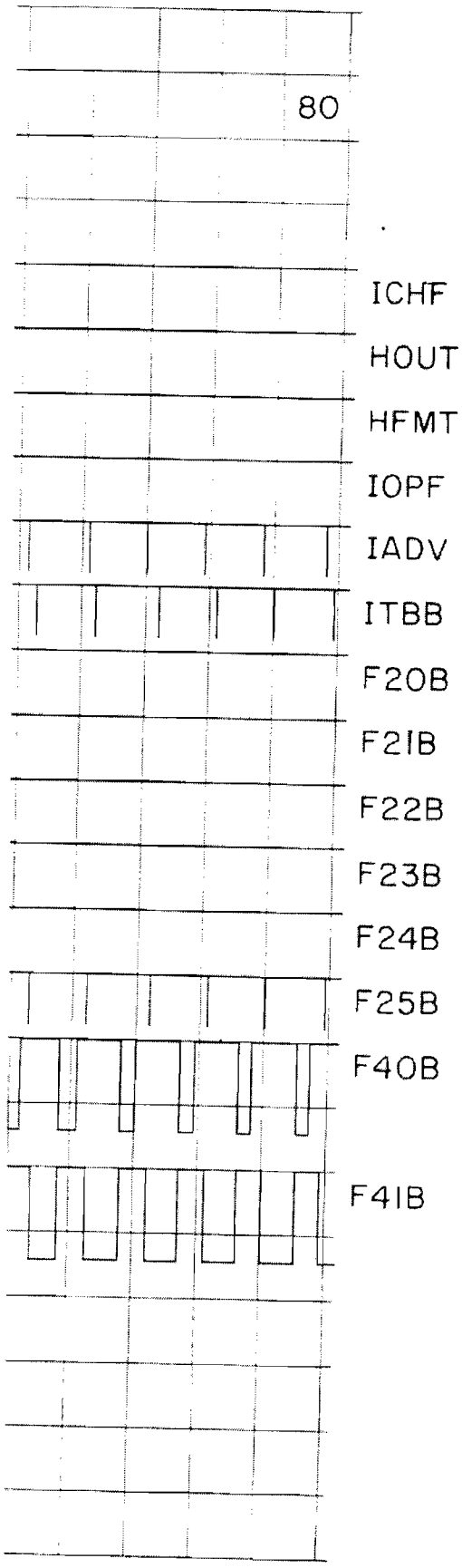

I
IO
GO TO ( ) ( )
CLR X
ICHF
HOUT
HFMT
IOPF
IADV
ITBB
F20B
F21B
F22B
F23B
F24B
F25B
F40B
F41B
9100A-J-50079

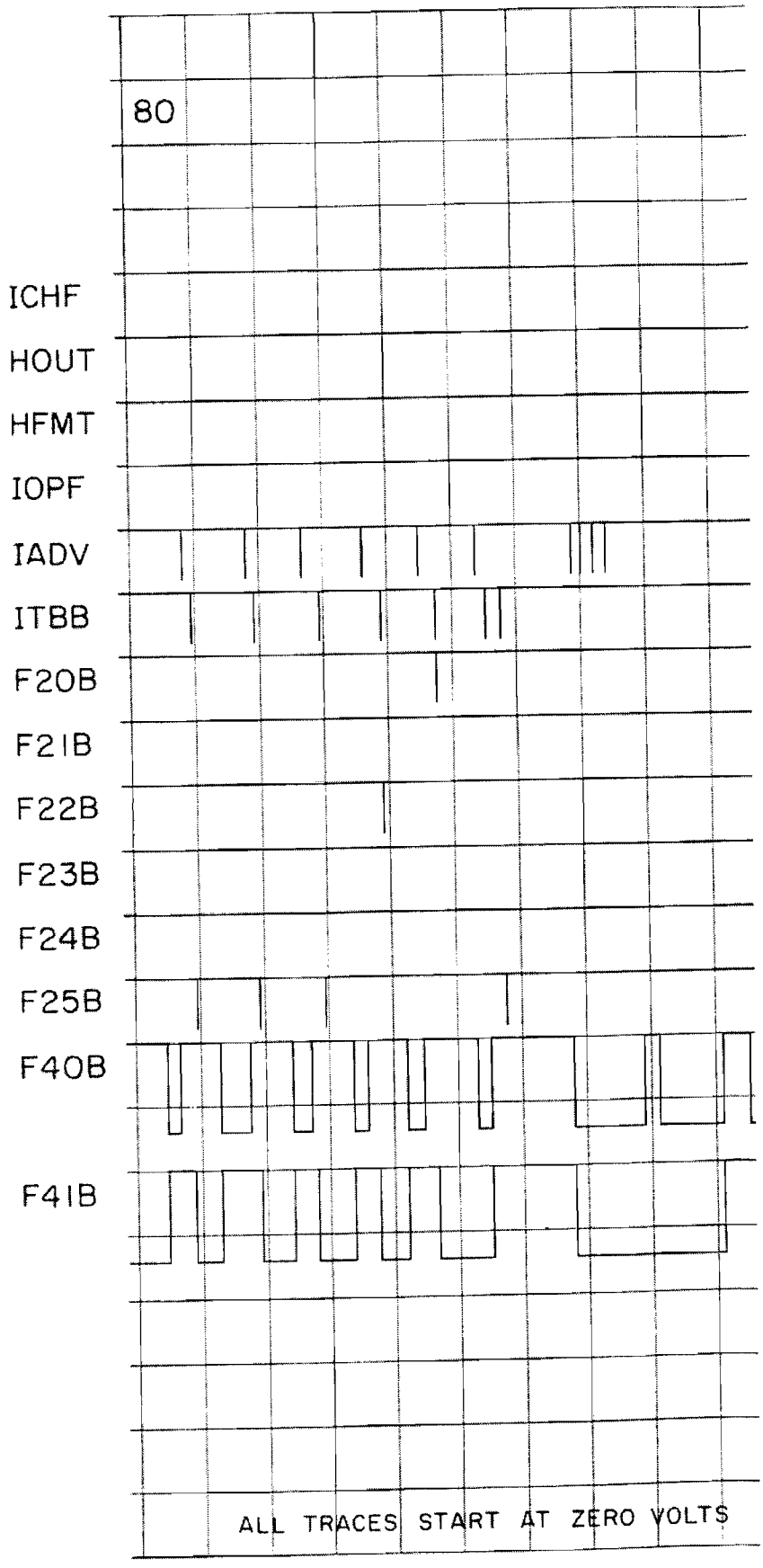

20
30
4
3
DECIMAL POINT
1

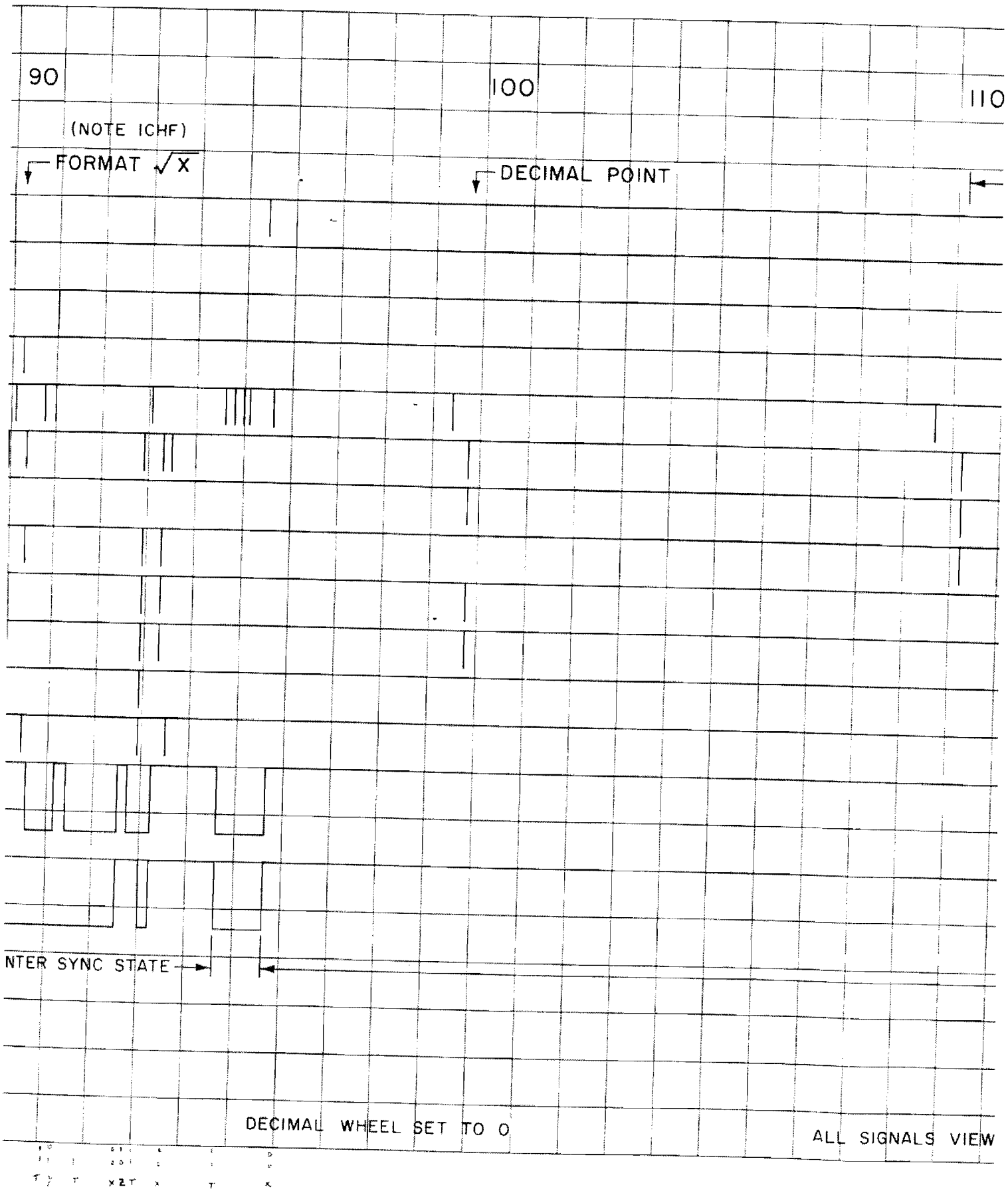

40
50
NT
1
4
X→( )

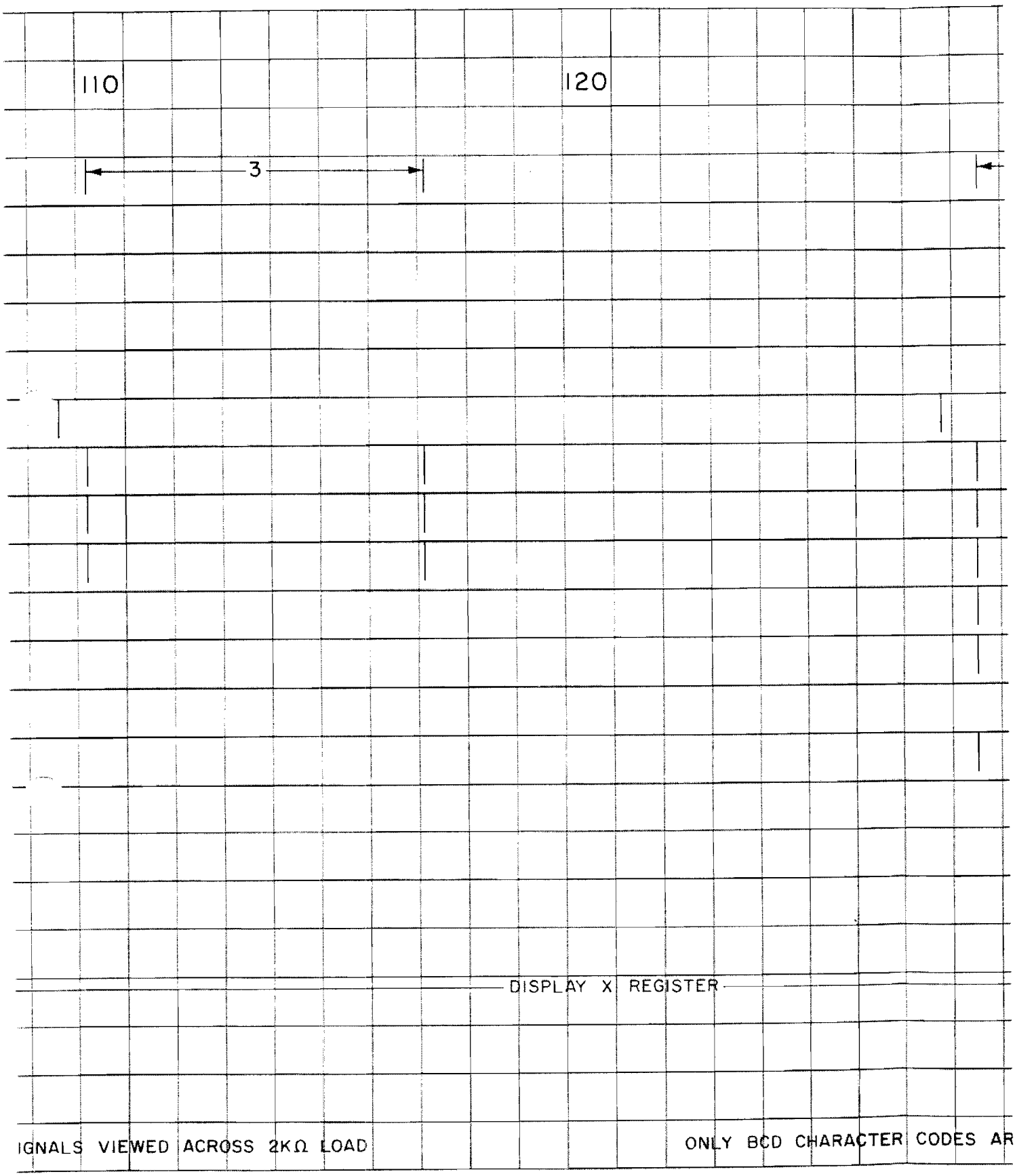

60
70
9

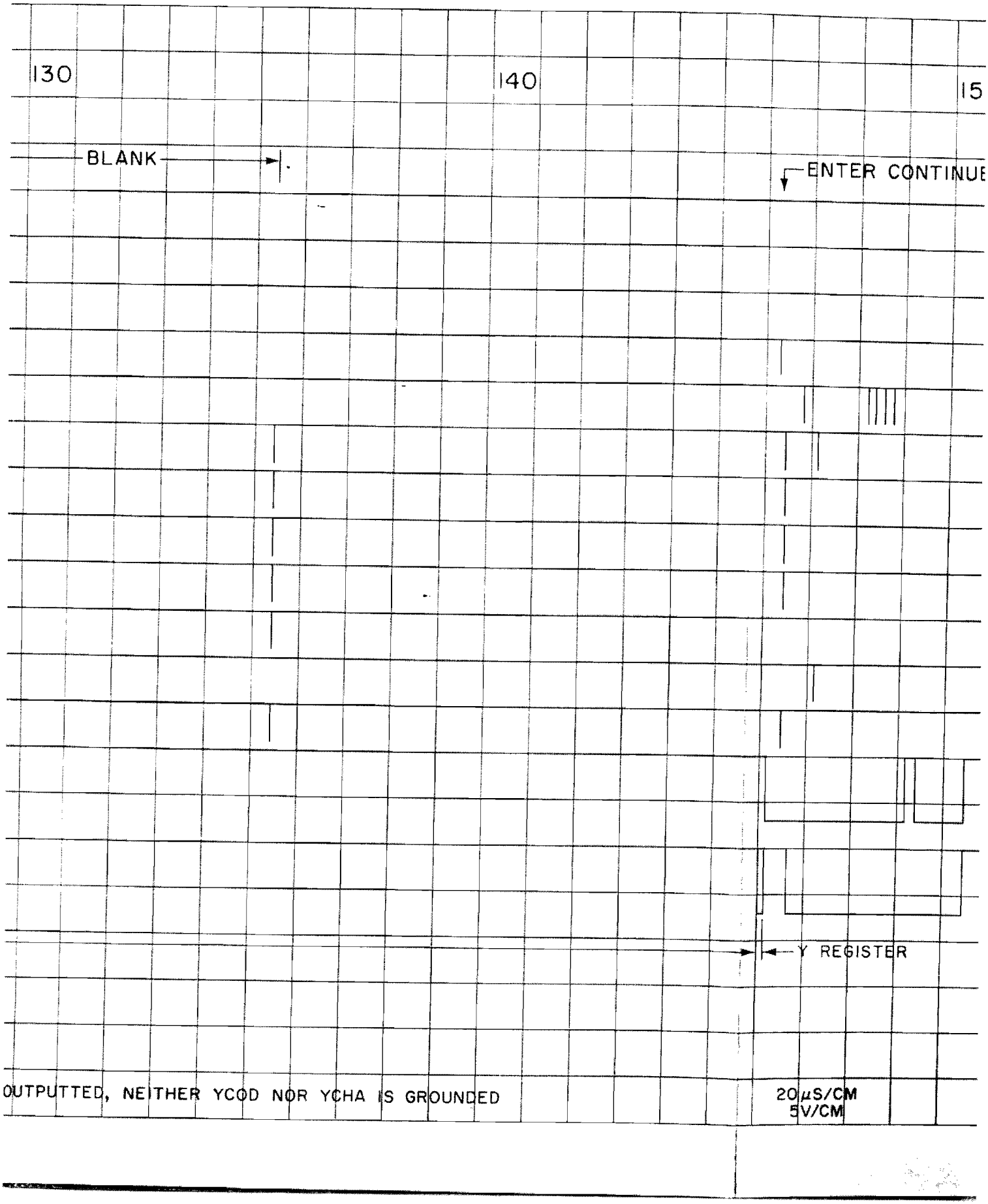

60
70
( )
9

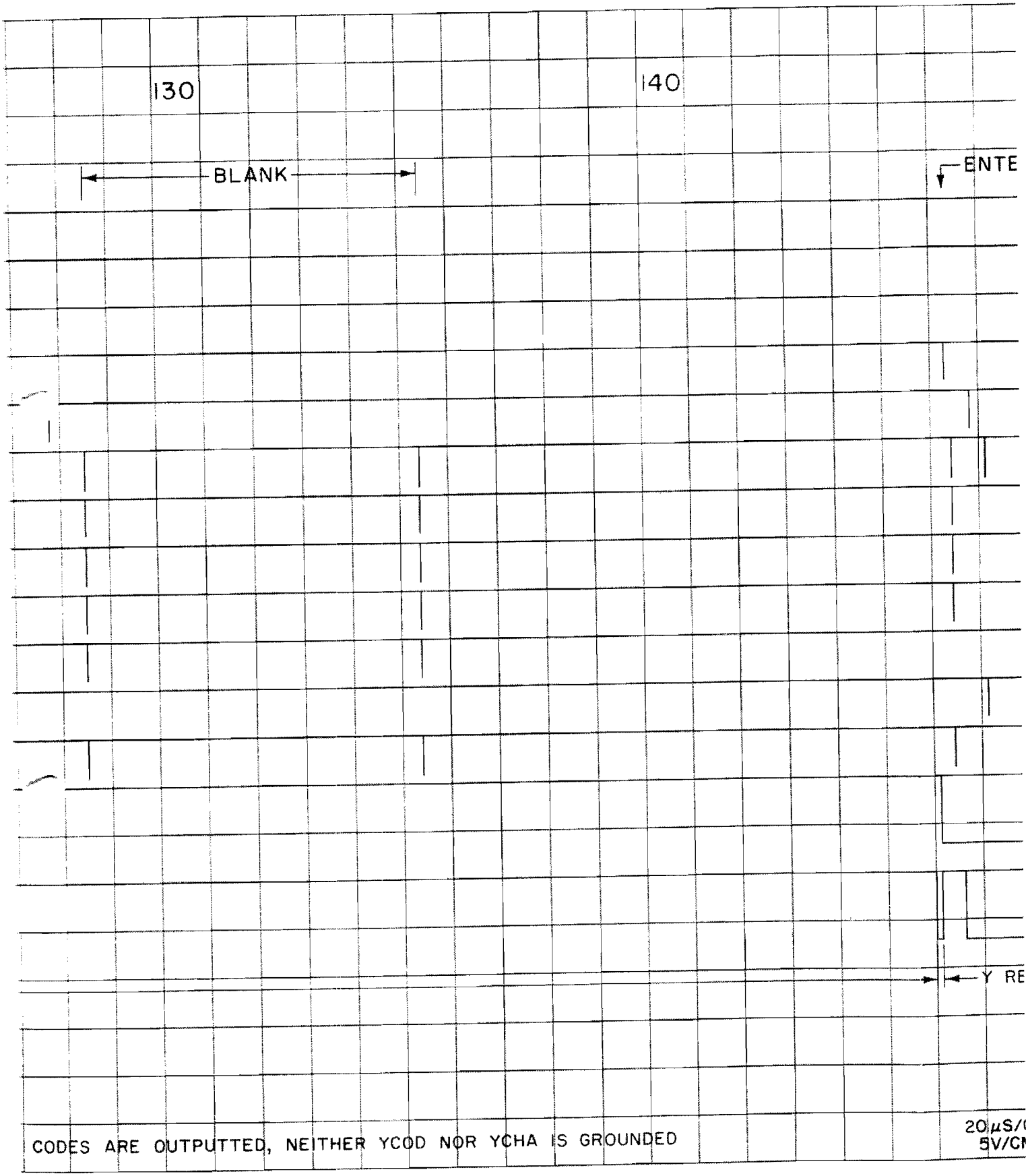

80
ICHF
HOUT
HFMT
IOPF
IADV
ITBB
F20B
F21B
F22B
F23B
F24B
F25B
F40B
F41B


80
ICHF
HOUT
HFMT
IOPF
IADV
ITBB
F20B
F21B
F22B
F23B
F24B
F25B
F40B
F41B
ALL TRACES START AT ZERO VOLTS


90
100
110
(NOTE ICHF)
FORMAT √X
DECIMAL POINT
NTER SYNC STATE
DECIMAL WHEEL SET TO 0
ALL SIGNALS VIEW


110
120
3
DISPLAY X REGISTER
IGNALS VIEWED ACROSS 2KΩ LOAD
ONLY BCD CHARACTER CODES AR


130
140
15
BLANK
ENTER CONTINUE
Y REGISTER
OUTPUTTED, NEITHER YCOD NOR YCHA IS GROUNDED
20 μS/CM
5V/CM


130
140
BLANK
ENTE
Y RE
CODES ARE OUTPUTTED, NEITHER YCOD NOR YCHA IS GROUNDED
20μS/
5V/C

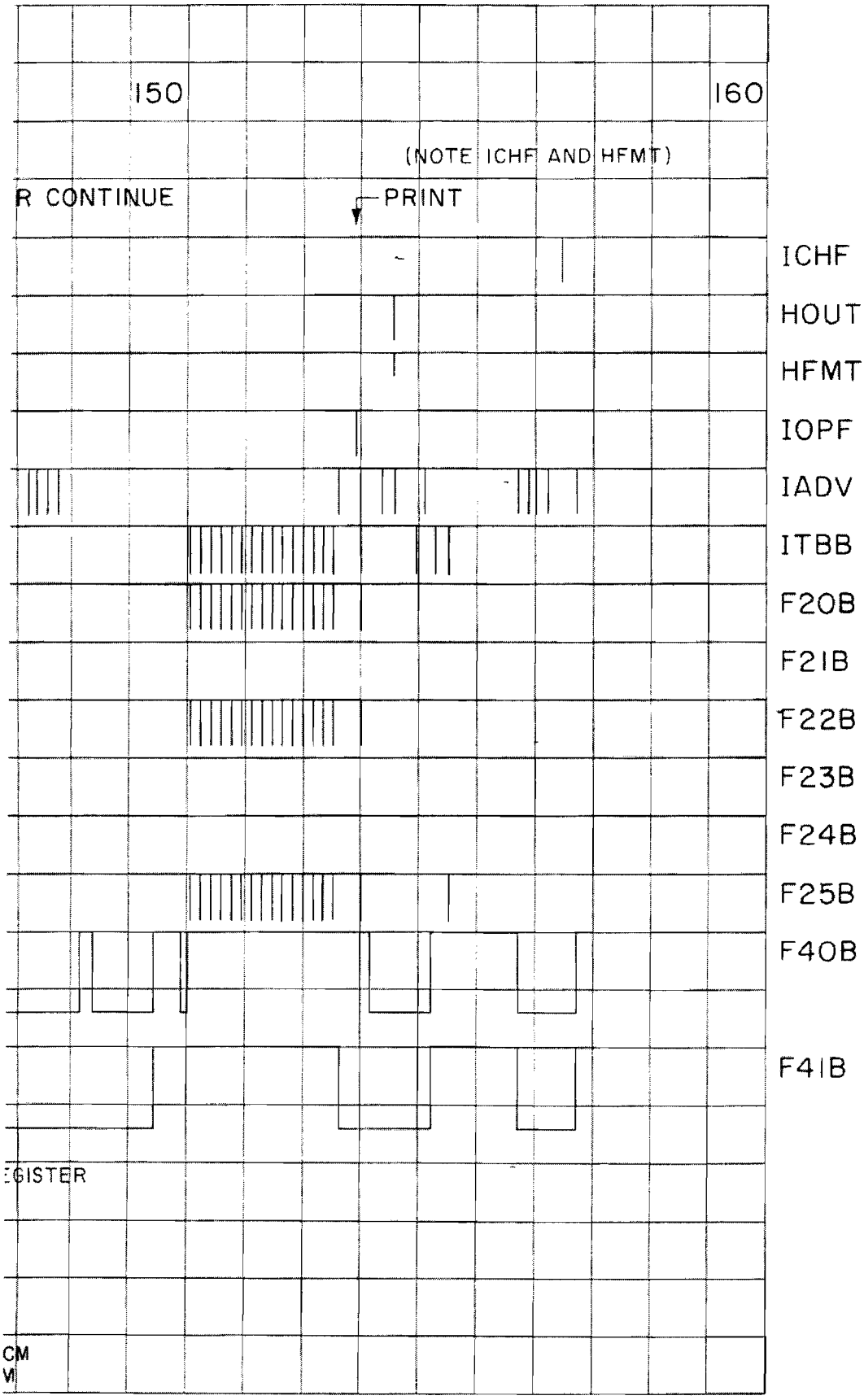


Figure 4. 9100A output signals during program segment of Figure 3.

SUMMARY OF 9100 I/O SIGNALS

YRAD	Grounding YRAD forces the 9100 to work in degrees. If YRAD sees a high impedance to ground the DEGREES/RADIANS switch on the Keyboard will determine which is used.
YFLT	Grounding YFLT forces the Display to be in fixed point. If YFLT sees a high impedance to ground the FLOATING/FIXED POINT switch on the Keyboard will determine the type of Display. See NFLT.
NFLT	Grounding NFLT will force floating point Display. NFLT will override YFLT if both are grounded. If NFLT sees a high impedance to ground the FLOATING/FIXED POINT switch on the Keyboard will determine the type of Display.
YRUN	Grounding YRUN will force the 9100 into Program Mode. If YRUN sees a high impedance to ground the PROGRAM/RUN switch on the Keyboard will determine whether the 9100 is in Run or Program Mode.
YINH	Grounding YINH will disable all keys on the 9100 Keyboard, except the STOP and PAUSE keys. See YKDN, and Page 76.
NHLD	Grounding NHLD will cause the 9100 to remain in the Sync State once it gets there from the Z, P or T Registers until NHLD is released. See NDGO, and Appendix V.
NDGO	Grounding NDGO causes the 9100 to never wait for sync. If both NDGO and NHLD are grounded NDGO will override NHLD.
YAFZ	YAFZ determines how the Format operation is treated. If YAFZ sees a high impedance to ground, the 9100A will not look for nor present the key-code following the execution of the Format key-code on the I/O terminals. There is no YAFZ on the 9100B. See page 7.
KEY LINES NK20 - NK25	NK20-NK25 are six inputs that are used to encode the logical complement of a desired key-code.
YKDN	Making YKDN go negative will cause the New Errand Qualifier to be met, provided it is being asked at least 12 msec after YKDN has gone negative. An escapement mechanism is involved so that a key-code will be processed only one time for each time that YKDN goes negative. If YINH is grounded while YKDN is negative, the 9100 will act as if YKDN were removed for the period of time that YINH is at ground.

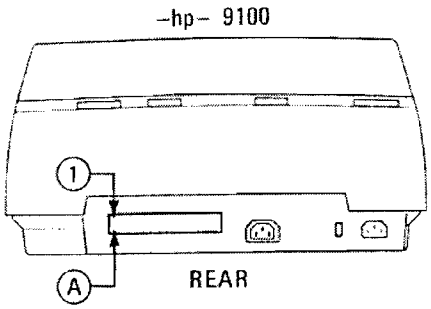
YCHA	Grounding YCHA while in the Display causes the 9100 to repeatedly output the BCD character code for the character currently being formed. See Appendix I.
YCOD	Grounding YCOD while in Display causes the 9100 to repeatedly output the unblanking code for the character half about to be formed. See Appendix I.
HOUT	HOUT results from the normal execution of the Print operation, and is used to trigger the -hp- 9120A Printer.
HFMT	HFMT is given by the normal execution of the Format operation. For 9100A's see Appendix II.
ITBB	ITBB indicates there is information currently being strobed on the Buffered Bits.
BUFFERED BITS F20B - F25B	F25B - F20B are the six output lines used to output codes from the 9100.
BUFFERED WORDS F40B - F41B	The Buffered Words are used to indicate the current register being displayed. See Table 4.
ISTP	ISTP is an output of the 9100 that goes negative as long as the STOP key on the Keyboard is pressed. ISTP is not affected by YINH.
IOPF	IOPF is given immediately before the processing of any operation. Taken together, F41B and IOPF can be used to tell if the operation is a statement or an errand. If 41B is a 1 when IOPF is given the key-code came from a program. If the 9100 is in RUN Mode, then shortly after IOPF ITBB will strobe the Buffered Bits to indicate the key-code to be executed. If the 9100 is in Program Mode the next ITBB after the IOPF will indicate the key-code being processed, as long as it is not 51_8 (Step Program).
IADV	IADV has a dual application. During the Display it occurs about 8 μsec before each ITBB. While the 9100 is in the Display IADV can be used to ready an external device to receive data. IADV will also occur at the end of the Z or P Register. Also, there is a method using IOPF, ITBB and IADV to determine the general classification of each key-code as it is processed. This would be needed to construct a printer that would print the key-code and the result of that operation, as it would be desirable to print after some keys, but not after others. Whether the 9100 is in Run or Program Mode also enters into the decision of whether or not to print. See Appendix V, The Print-After-Printer, for complete details.
ICHF	ICHF is given each time the 9100 finishes processing an operation and heads back towards the Display to be available for use on errands. In the 9100A ICHF will appear just a few μsec before the Sync State is entered. In the 9100B it occurs approximately 100 μsec before the Sync State.

Table 2. I/O terminals.

Terminal Connections

NK25	A	1	NK24
NHLD	B	2	NK20
YRAD	C	3	NK21
YFLT	D	4	NK22
YRUN	E	5	NK23
HOUT	F	6	HFMT
F20B	H	7	F21B
F22B	J	8	F23B
F24B	K	9	F25B
NFLT	L	10	— —
ITBB	M	11	ISTP
IADV	N	12	ICHF
IOPF	P	13	F41B
YINH	R	14	F40B
YKDN	S	15	YCOD
YGNPS	T	16	YDSP
YSKIC	U	17	YCHA
YAFZ	V	18	NDGO
— —	W	19	SPARE 3
GND	X	20	GND
SPARE 2	Y	21	SPARE 1
-15V*	Z	22	-15V*

Terminal Location



*A total of 50ma may be drawn from this supply.

REGISTER FORMATION

Characters are formed by unblanking the CRT Beam in the appropriate manner as it moves across the screen of the CRT in a fixed path. The path for floating point is shown in Figure 5.

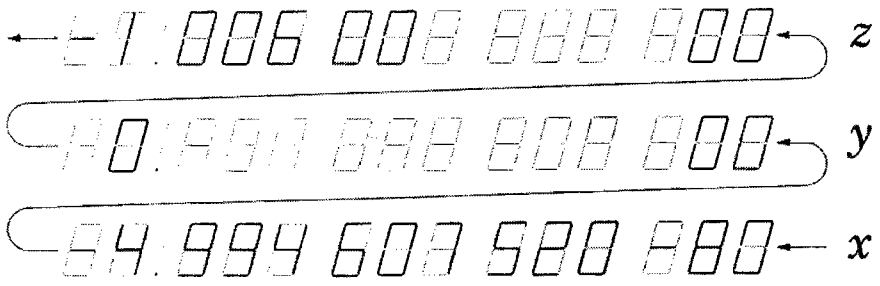


Figure 5. Floating point Display.

In fixed point the path depends upon the size of the number to be displayed and the setting of the Decimal Wheel.

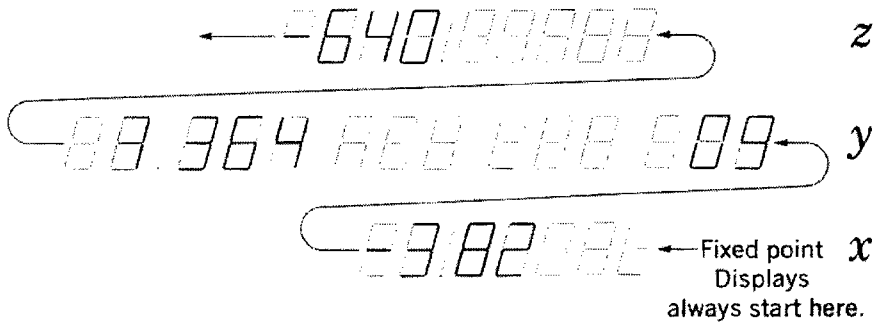


Figure 6. Mixed Display.

In Figure 6 the Decimal Wheel is set to 5. The number in the X Register (–3.82) has been entered to 2 places to the right of the decimal. Hence, the additional 3 spaces are displayed as blanks. If, however, the number in the X Register had been entered as –3.82000 then those characters would be displayed as zeros.

BLANK SPOTS

Note that the space to the left of the minus sign on the 3 in Figure 6 is not formed by blanks. The beam never goes there under those conditions. The same is true about the space to the right of the characters of the X Register and under the Y Register Exponent.

Thus, not all blank spots in the Display are blank characters. In particular, the spaces between characters in the floating point Display (see Figure 5) are **not** blank characters. They are formed by rapidly shifting the beam to the left.

APPENDIX I — THE DISPLAY PROCESS

CHARACTER FORMATION

With the exception of the decimal point each character in the Display is composed of two halves, labelled left and right. The right half is formed first and consists of segments 1, 2, 4, 6, 7, 9, 10 and 12 are done slowly; the rest are done quickly.

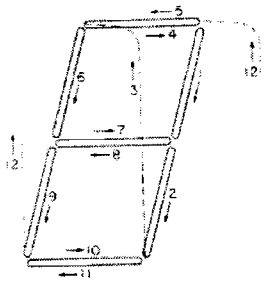


Figure 7. Character formation.

Segments 4, 6, 7, 9, and 10 comprise the maximum visible portion of the left half.

UNBLANKING CODES

Each character half has an **unblanking code** that is used to turn the beam on as it moves through the Display. These unblanking codes are available to the user (they are there primarily for the -hp- 9120A Printer) if he grounds YCOD. The unblanking code about to be implemented will be repeatedly outputted as long as YCOD is at ground, when the YCOD output option is reached.

TABLE OF UNBLANKING CODES

Table 3. Character codes and unblanking codes.

CHARACTER	CHARACTER CODE	UNBLANKING CODES	
		RIGHT HALF	LEFT HALF
0	0 2 0 0 0 0	0 1 0 0 0 0	0 0 0 1 0 0
1	0 2 0 0 0 1	1 1 0 0 0 0	0 1 1 1 1 1
2	0 2 0 0 1 0	0 1 1 0 0 0	0 0 0 0 1 0
3	0 2 0 0 1 1	0 1 0 0 0 0	0 0 1 0 1 0
4	0 2 0 1 0 0	0 1 0 0 0 0	0 1 1 0 0 1
5	0 2 0 1 0 1	0 1 0 0 1 0	0 0 1 0 0 0
6	0 2 0 1 1 0	0 1 0 0 1 0	0 0 0 0 0 0
7	0 2 0 1 1 1	0 1 0 0 0 0	0 1 1 1 1 0
8	0 2 1 0 0 0	0 1 0 0 0 0	0 0 0 0 0 0
9	0 2 1 0 0 1	0 1 0 0 0 0	0 0 1 0 0 0
—	0 2 1 0 1 0	0 1 1 0 1 0	0 1 1 0 1 1
a	0 2 1 0 1 1	0 1 0 0 0 0	0 0 0 0 1 0
b	0 2 1 1 0 0	0 1 0 0 1 0	0 0 0 0 0 1
.	0 2 1 1 0 1		0 0 1 1 1 1
c	0 2 1 1 1 0	0 1 1 0 1 0	0 0 0 0 1 1
d	0 2 1 1 1 1	0 1 0 0 0 0	0 0 0 0 1 1
BLANK = +	1 2 1 1 1 1	0 1 1 0 1 0	0 1 1 1 1 1
	F25 F24 F23 F22 F21 F20	F25 F24 F23 F22 F21 F20	F25 F24 F23 F22 F21 F20

2 = 1 or 0

On a right half, segments 1 and 2 are controlled by bits 21 and 23, respectively, of the unblanking code. On a left half, segments 5, 6, 7, 9, and 10 are controlled by bits 20 through 24, respectively. A segment is unblanked if its corresponding bit is 0.

It is possible to tell a left half unblanking code from a right half unblanking code. No left half unblanking code has the form 212020, where every right half unblanking code does, (2 indicates either 1 or 0).

BLANK CHARACTERS

There are 16 non-blank characters that the 9100 can display. Since 4 binary lines can encode 16 codes, only 4 lines are needed to specify a given character. However, any character can be a blank as well. To indicate this and preserve the character, a 1 or a 0 in the most significant bit (leftmost of six, say, F25B on the Buffered Bits) is used to mean blank or non-blank respectively. The least four bits encode the character. See Appendix V, Core Memory Conventions.

In the Display there is a mechanism to trade BCD character codes for unblanking codes. It would be wasteful if it had to recognize 16 different codes for a blank, although that can happen in terms of core memory encodings. Before the BCD character code is exchanged the most significant bit of the BCD character code is inspected. If it is a 1, then a special code for a blank is forced into the rest of the bits. The code is the same as for d except the 1 in the most significant bit (the 1 in F25B) is retained.

SIGNS

The fifth bit (F24B on the Buffered Bits) is used to encode the sign attached to that digit—important only if that digit is the most significant digit of the number or of its exponent. If F24B is a 1 the digit is negative. See Appendix V, Core Memory Conventions.

Another internal detection and exchange mechanism looks at the fifth bit (F24B) of the BCD character code of the most significant digit of the number and of its exponent to determine what sign to form after them. The code for the proper sign is then forced to be the next BCD character. A plus sign is formed as a blank.

DECIMAL POINT

There is a detection mechanism for the decimal point. Numbers stored in the core memory of the 9100 are always stored in floating point. The position of the decimal point is always assumed to be between the two leftmost significant digits of the number. The Display Process has a Decimal Point Counter that can afix the decimal point whether the Display is in fixed point or floating point.

When it is time to do the decimal point, the detection mechanism synthesizes the BCD code for the decimal point and it is displayed as any other character, except that it has only a left half. During the formation of that left half the normal ability to form horizontal motions, (as in segments 4, 7, and 10), is removed and the would be segment 10 is unblanked. The result is a spot in the proper position for the decimal point.

BCD CODES AVAILABLE WITH YCHA

It is these modified BCD character codes that are available with YCHA. The only time a 1 on F25B will occur (while looking at BCD character codes) is when a blank is being formed, and then the rest of the code will be the same as for d. All other BCD character encodings will have F25B as a zero. F24B of the BCD character encodings will depend upon previous events.

APPENDIX I—THE DISPLAY PROCESS

NEGATIVE CHARACTERS IN THE BODY OF THE DISPLAY

A close look at Figure 8 reveals that all characters formed before the 5 in the significant digit portion of the number in the X Register are negative in terms of their BCD character codes. This is because of the sequence of key strokes used to enter the number in the X Register.

It was:

CLEAR X
1
.
2
3
4
5
CHG SIGN
6
7
8
9
0
ENTER EXP
4
5

The normal execution of the Change Sign operation changes the sign of the most significant digit (for significant digit or exponent portion—depending on previous operations) and as a by product of that, a flag is set which causes any further digit entries to have the new sign attached to them. This in no way interferes with arithmetic, as the only signs that count are those of the most significant digit of the number and of its exponent.

The following are interesting points about the Display and can be verified by looking at Figure 8.

1) The first two spaces formed in a floating point register (for the sake of grouping, like the comma in 2,153) are marked by a double IADV. The third and last space is not. See the 12th, 23rd and 34th cms.

2) There are no spaces for grouping in a fixed point register.

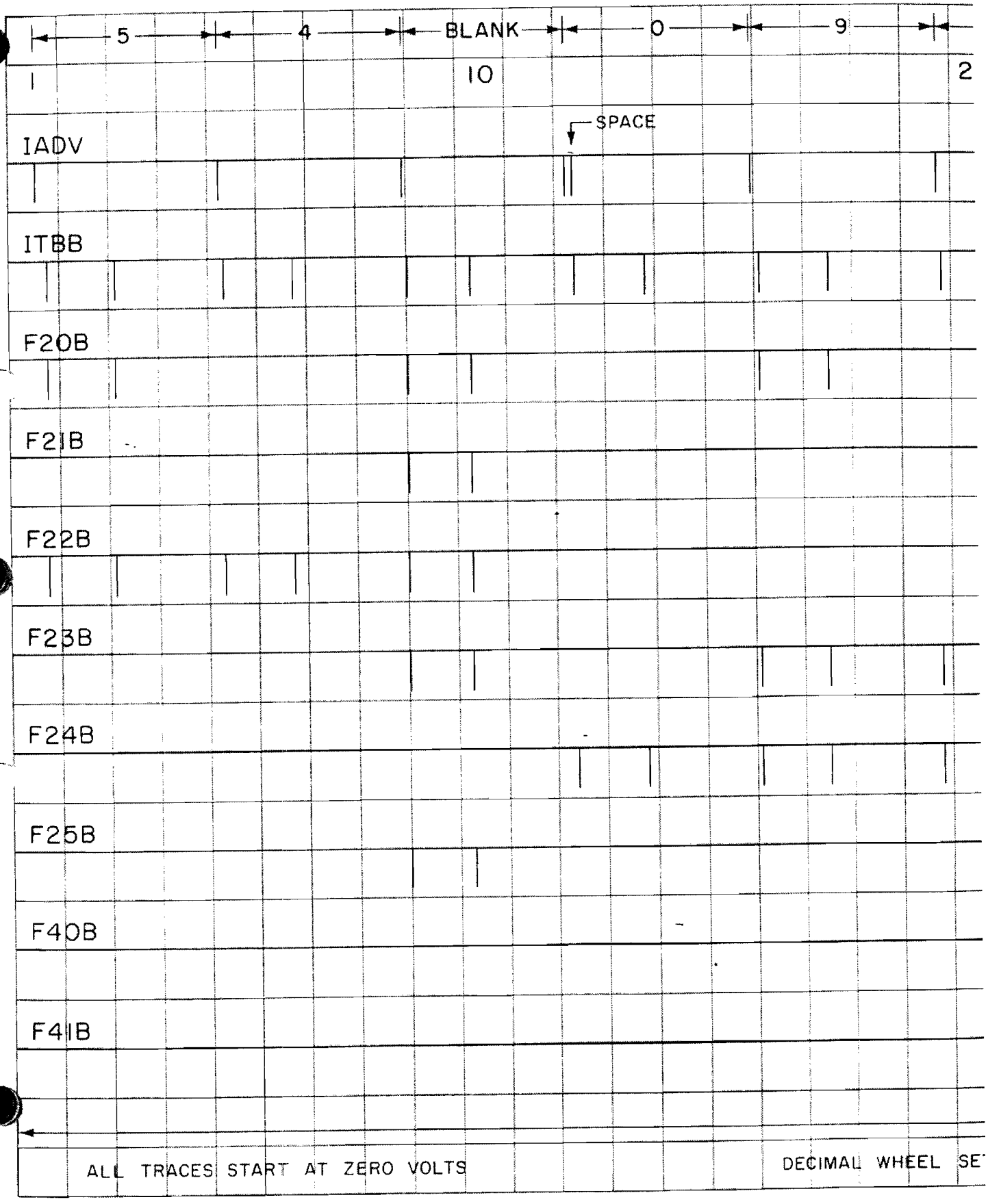
3) The Buffered Words indicate what register is currently being displayed. See Table 4.

4) After the display of the frame, the Buffered Words indicate the T Register. They will do this for the length of time the 9100 waits for sync in the Sync State. If NDGO is at ground the 9100 never waits for sync and the Buffered Words will not indicate the T Register.

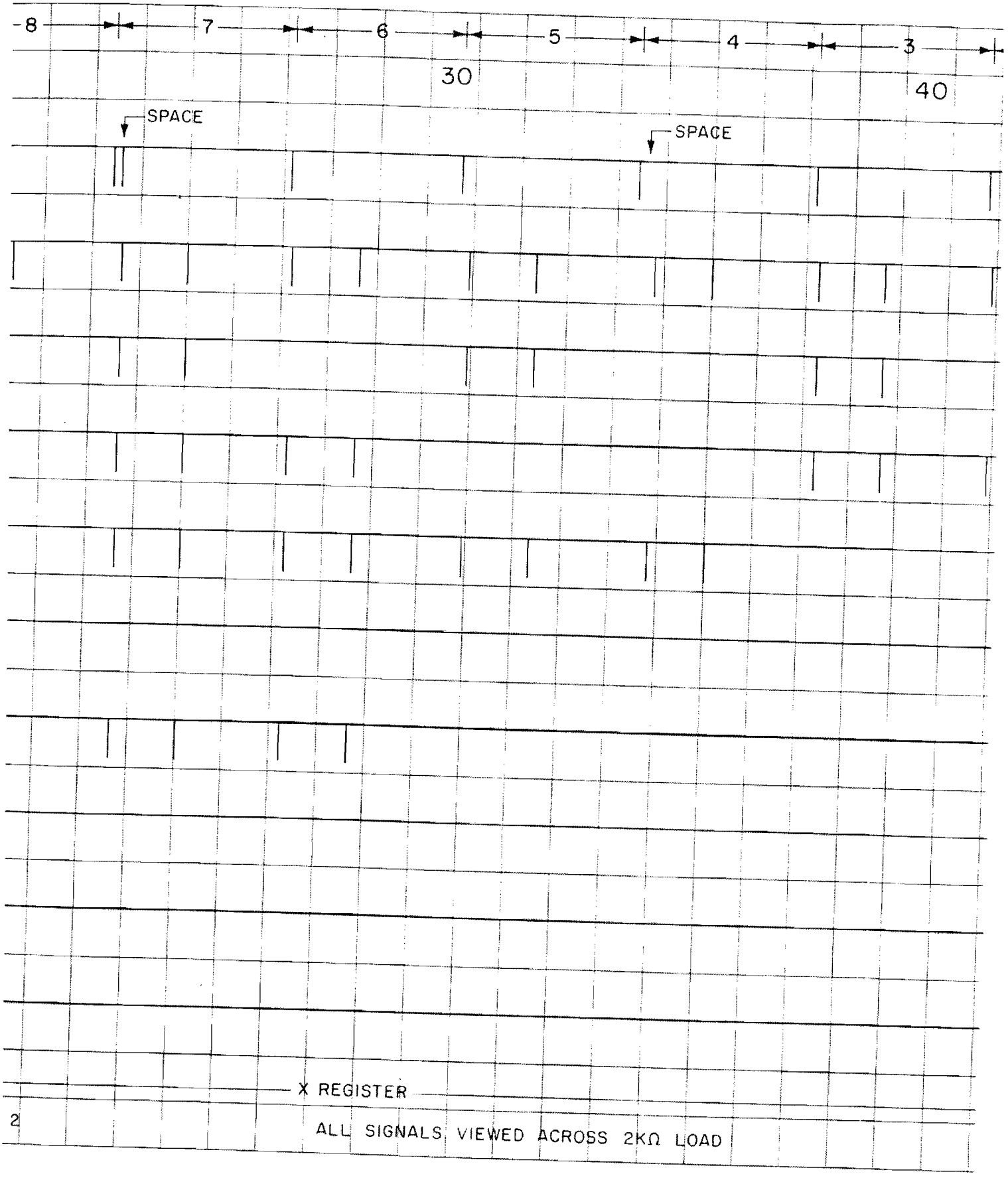
Table 4. Meaning of the Buffered Words.

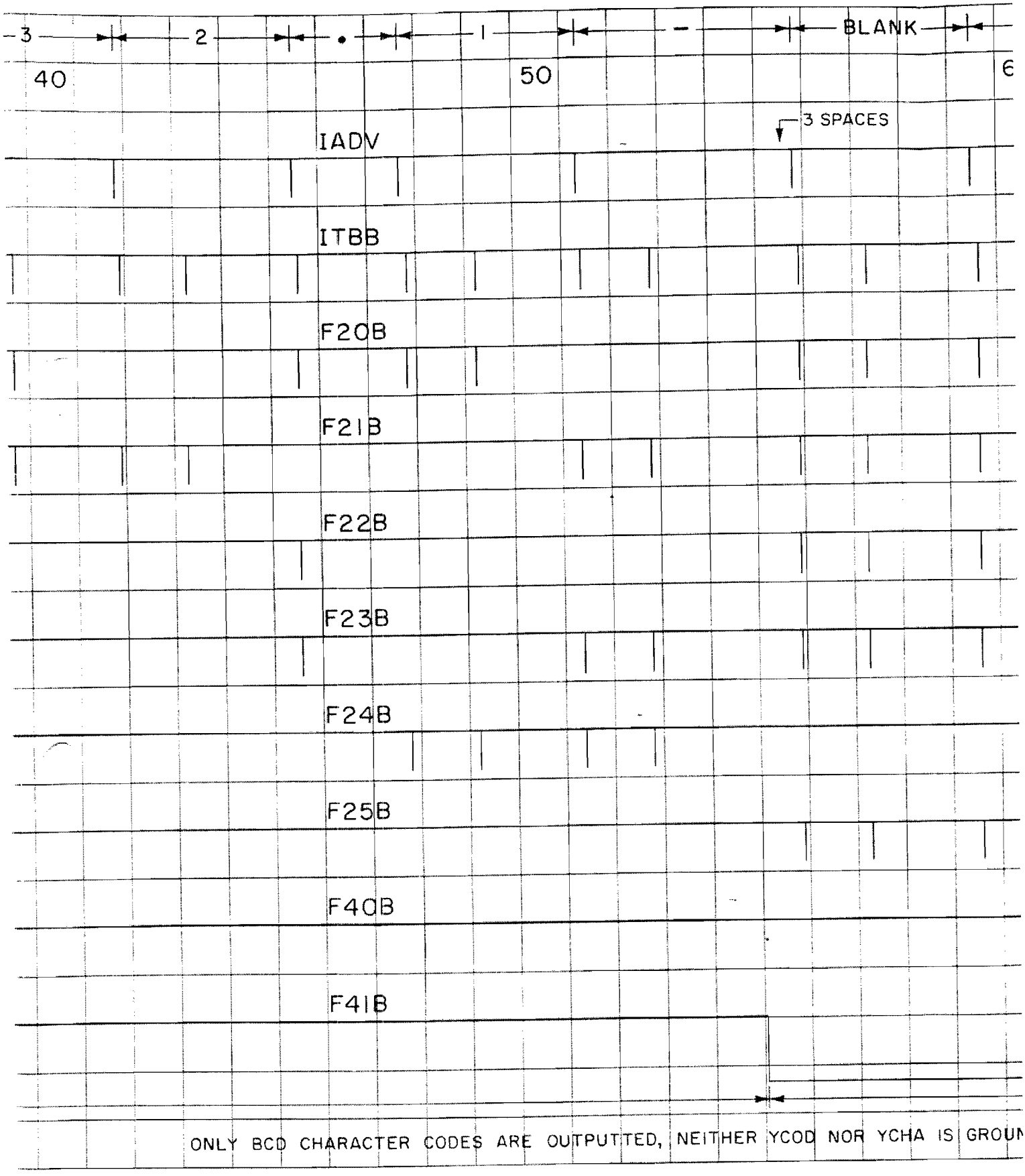
9100A				9100B			
F40B	F41B	YRUN	REGISTER	F40B	F41B	YRUN	REGISTER
0	0	2	X	0	0	2	X
0	1	2	Y	0	1	2	Y
1	0	2	Z	1	0	1	Z
1	1	2	T	1	0	0	P
				1	1	2	T

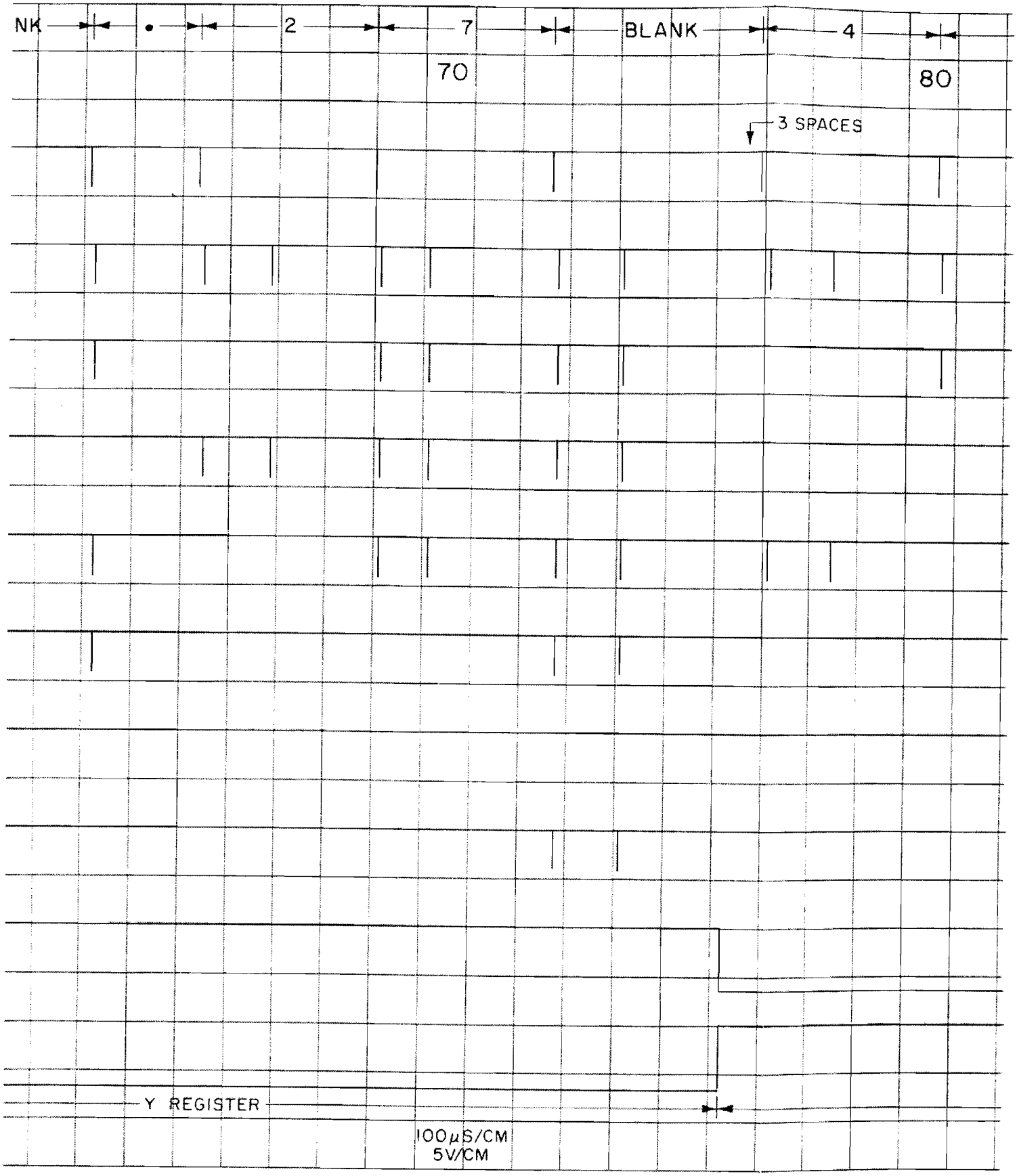
Note: 2 is either 1 or 0.

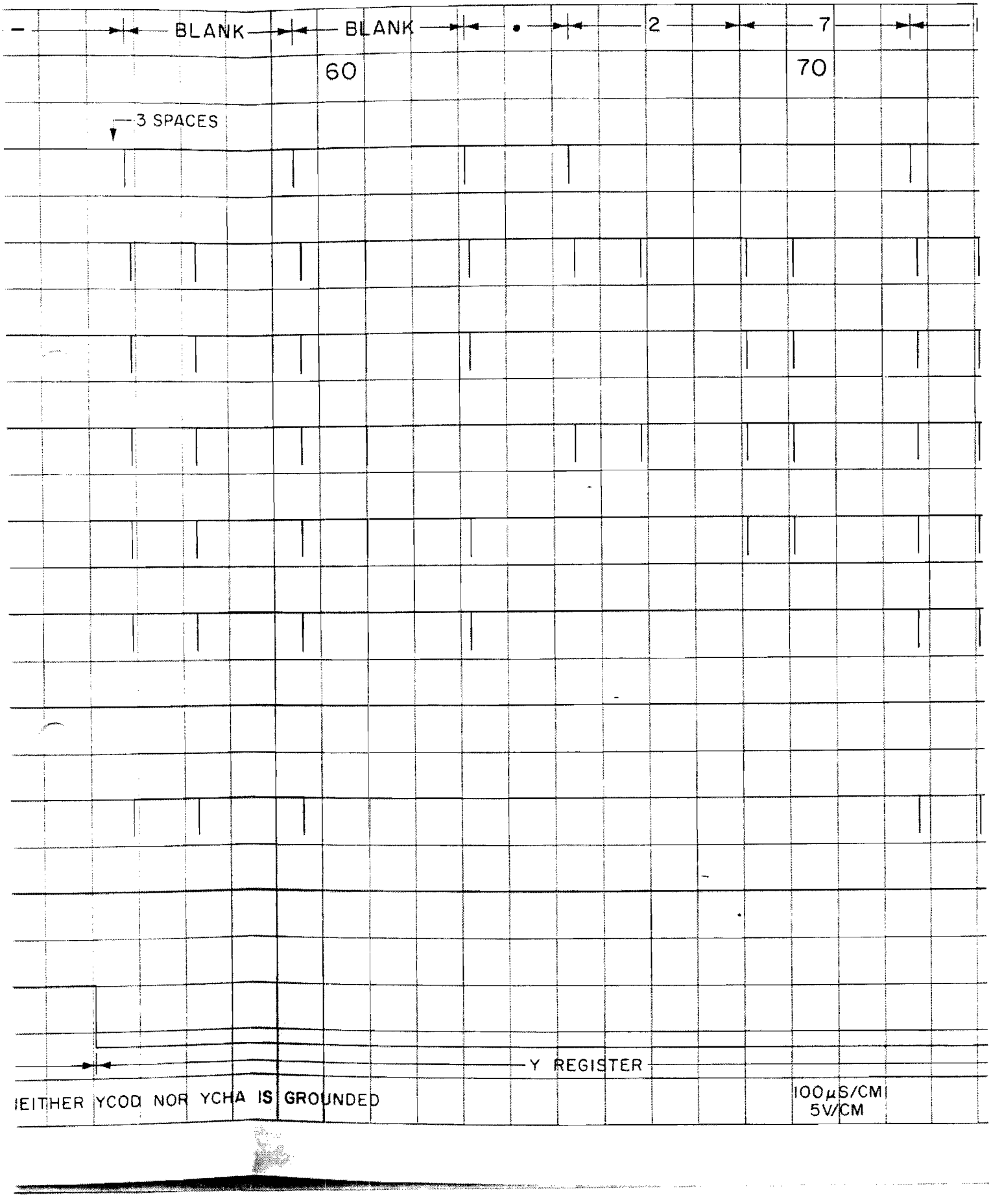


9100A-J-50078


-8
7
6
5
4
3
30
40
SPACE
SPACE
X REGISTER
ALL SIGNALS VIEWED ACROSS 2KΩ LOAD


3
2
.
1
-
BLANK
40
50
6
3 SPACES
IADV
ITBB
F20B
F21B
F22B
F23B
F24B
F25B
F40B
F41B
ONLY BCD CHARACTER CODES ARE OUTPUTTED, NEITHER YCOD NOR YCHA IS GROUN


NK
•
2
7
BLANK
4
70
80
3 SPACES
Y REGISTER
100μS/CM
5V/CM


BLANK
BLANK
2
7
60
70
3 SPACES
Y REGISTER
EITHER YCOD NOR YCHA IS GROUNDED
100μS/CM
5V/CM

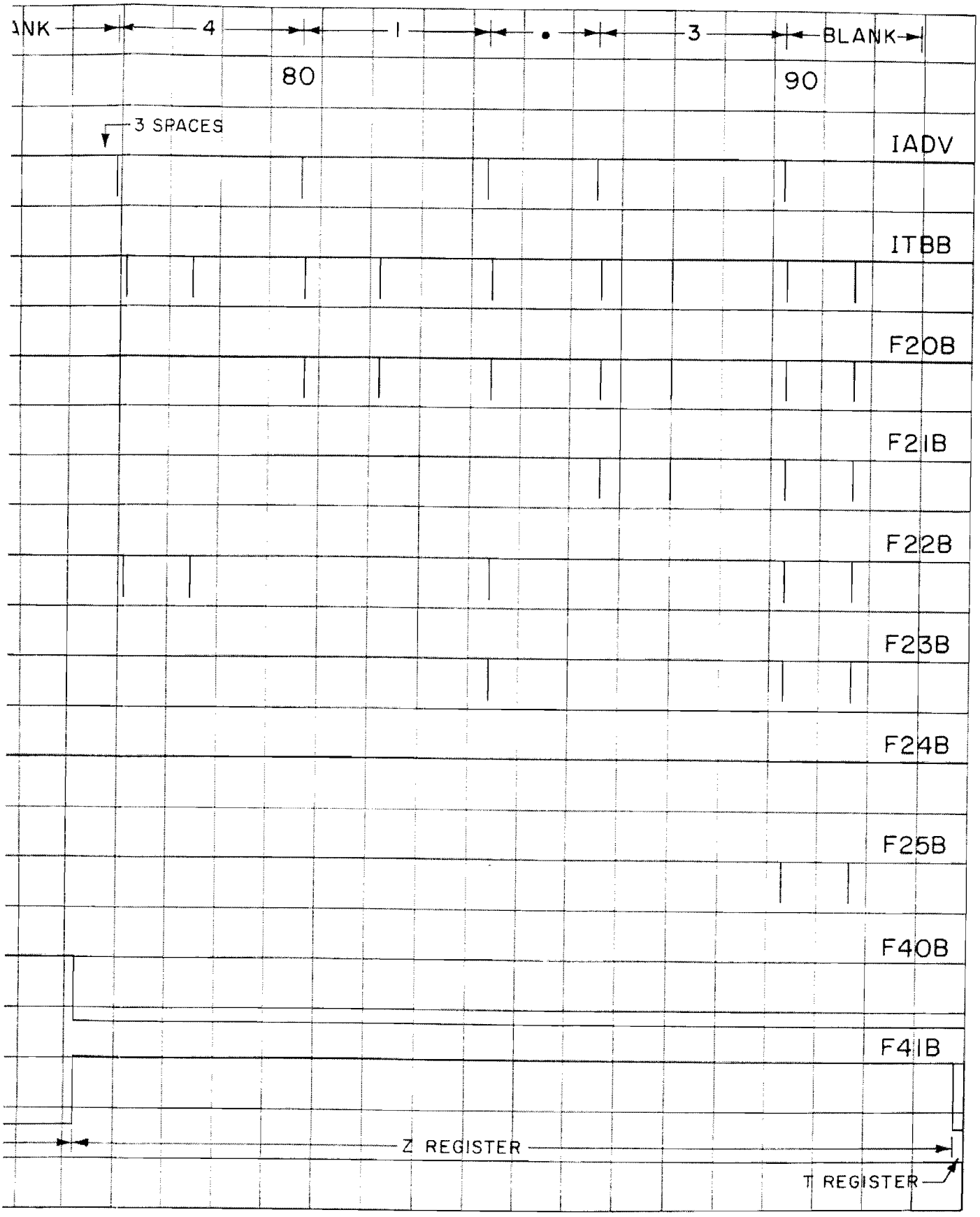


Figure 8. Calculator output signals during the Display.

DIRECTOR AND SYNC STATE

FLOW CHARTS DESCRIBE THE 9100

Figures 10 and 11 show the four major processes that characterize the 9100A and 9100B respectively. As flowcharts Figures 10 and 11 are functional equivalents of the internal workings of the 9100. There are pronounced similarities between various sections of Figure 10 (and of Figure 11). Very often these similar sections are actually the same section of internal Calculator logic with the various functions separated by internal flagging. This Appendix explains only three of the four processes; the Display Process is explained in Appendix 1.

After the display of each register the 9100 checks for a new errand. Displaying the three visible registers constitutes a frame. At the end of the frame the 9100 checks to see if the Display is in the context of a Pause, and then checks for a new errand.

PAUSE CHECK

The Pause check is made by asking if the 9100 is running a program, since the only time the Calculator is running a program while it is in the Display is during a Pause.

If a Pause is in progress the frame count is decremented once (a Pause consists of nine frames) and the Keyboard lock-out mechanism is renewed. This is an internal mechanism that prevents YKDN (but not YDSP) from causing the New Errand Qualifier to be met.

A frame count of zero determines the end of a Pause. When it is reached the Display Process is left altogether and the 9100 goes back to the Program Execution Process to get the next step in the program.

While a Pause is in progress, putting the key-code for Pause on the Key Lines will not cause the 9100 to stop, as pushing the PAUSE key on the Keyboard will. This is true no matter whether YKDN or YDSP is used to enter the key-code. First, YKDN is disabled by the Pause in progress. If YDSP were used it still wouldn't do any good since the way the 9100 tells that the PAUSE key is down is by a signal from the Keyboard that is not found on the I/O terminals.

PROLONGED PAUSE

However, there is a way to prolong the Pause, depending upon what is to be done during that time. The method is to ground NDGO before the Pause gets started. The execution of the Pause will still set the frame count and return to the Display. Once back in the Display, however, the answer to the question "Was that the Z or P Register?" will always be no, even when it should have been yes. Hence, the frame count is never decremented and the Display goes on indefinitely. During the time that NDGO is grounded the Sync State will change the Buffered Words from Z to X*. (Should the 9100B be put into Program Mode while NDGO is grounded during a Prolonged Pause, the P Register will replace the Z Register and the situation remains as described. However, the P Register won't indicate the proper data unless a Stop errand is executed prior to switching to Program).

*The normal sequence is $\left(\frac{Z}{P}\right) \rightarrow T \rightarrow X$.

DIRECTOR AND SYNC STATE

Should the user desire to process errands while the 9100 is in a Prolonged Pause, the first such errand should be a Stop. After that NDGO can be released as the 9100 is truly stopped. Failure to do this can lead to strange results. The reason is that each time a key-code is executed the 9100 will return to a place that NDGO was otherwise preventing it from getting to, so that the frame count gets decremented one time for each errand executed. Eight such errands can be executed and still retain the Prolonged Pause. The ninth errand will be duly executed but the program will also resume its execution. In addition to this certain errands (Clear, Tan x, and half a dozen others) will cause the program to resume execution immediately as these errands destroy the frame count during their execution.

PAUSE ERRANDS

The execution of a Pause errand while the 9100 is displaying does not cause a Pause. This is because even though the 9100 comes back to the Display, it never gets to the frame count mechanism because it is not running a program. The frame counter is still waiting to count down 9 frames, but it will never get the chance since the only way to get the 9100 to begin executing a program is to execute a Continue errand, and that removes the Pause frame count information.

IADV

If the end of the frame is at hand (and NDGO is not grounded) the 9100 forces to the T Register and an IADV is given. Also, when the Display is being entered from some other activity these things are accomplished before the start of the Display.

SYNC STATE

The Sync State has a dual function. It is responsible for changing registers in the fashion indicated by the flowchart, and it has a second function enabled only when the Buffered Words indicate the Z, P or T Register. (This will normally be after the display of the Z or P Registers.) The second function is to hold the 9100 at the Sync State until a pulse derived from the powerline releases it. This prevents a waviness in the Display.

NEW ERRAND QUALIFIER

After the 9100 leaves the Sync State it checks to see if there is a new errand. Either YKDN or YDSP can be used to indicate this. If there is not a new errand the 9100 goes back to the Display.

If there is a new errand the key-code is entered and an IOPF (with F41B = 0) is given. Then the 9100 determines whether it is in Run Mode or Program Mode and goes to the Key-Code Execution Process or the Program Storage Process, respectively. If the 9100 goes to the Key-Code Execution Process an ITBB reveals the key-code about to be executed. If the 9100 goes to the Program Storage Process the next ITBB reveals the key-code about to be stored as part of a program, except when the original errand was Step Program (51$_8$). Then the next step in an existing program is the code on the Buffered Bits when the ITBB is given.

KEY-CODE EXECUTION PROCESS

FORMAT IN THE 9100A CONTINUED

> **NOTE**
> Only YKDN can be used to enter the key-code. Grounding YDSP with a key-code on the Key Lines will have absolutely no affect.

After the key-code is entered the program execution flags are removed. This allows whatever device being activated to have as much time as necessary to do whatever it is that it has to do. Then, if the Format operation was in the context of a program, the external device will enter a Continue errand after the event is over.

FORMAT IN THE 9100B

The way in which the 9100B executes the Format operation is nearly the same way as the 9100A executes it. The difference is that there is no YAFZ in the 9100B. Accordingly, after giving HFMT the 9100B checks to see if the Format is being done in the context of a program. With that check everything else is the same as for the 9100A.

GO TO ( ) ( ) IN THE 9100A

The first thing that the 9100A determines when executing a Go To operation is if it originated as an errand (which happens if the 9100A is not running a program). If it did the 9100A waits for the next two errands.

> **NOTE**
> Those errands must be either keystrokes or remote entries made with YKDN. YDSP will have absolutely no affect. Also, YKDN must be allowed time to recycle between the entries.

During the time that the 9100 is waiting there is no Display or any other effective activity.

If the Go To came from a program, the address following the Go To is still imbedded in the program. What has to be done is to increment the program address and get the next step in the program. If it is alpha-numeric then increment the address again and actually perform the address modification. If the first half of the upcoming "address" is not alpha-numeric the Go To sequence is aborted and those key-codes following the Go To are simply executed for what they are.

Notice that in the Key-Code Execution Process of Figure 10 there is a segment that contributes nothing to the execution of the Go To operation, yet accompanies each increment of the program address. The segment strongly resembles the first part of the Program Execution Process, and in fact, is just that. It was shown to convey that there is a considerable amount of this sort of thing in the Calculator. It was felt that to present a flowchart that closely describes these processes would be as complicated as the internal logic of the Calculator. The point is that fragments of certain I/O options are imbedded in others, although they are not intended to be used. This is the result of a good deal of overlapping in the internal software of the Calculator, which saves hardware.

QUALIFIER OPERATIONS IN THE 9100A

First the answer to the qualifier is found. If it is yes, the 9100A simulates a Go To statement by entering the Go To sequence at the proper place. From there it is exactly like a Go To from a program. If the answer to the qualifier is no, the program address is incremented twice before the Program Execution Process is entered again. This results in the 9100A skipping the two program steps after the qualifier.

KEY-CODE EXECUTION PROCESS

GO TO ( ) ( ) IN THE 9100B

In the 9100B the Go To operation is just an intializer; it sets a flag that is detected by a subsequent operation. The actual address modification sequence is contained in the execution of the alpha-numeric key-codes. Note that it makes no difference whether the Go To originated as a statement or an errand as far as its execution is concerned.

QUALIFIER OPERATIONS IN THE 9100B

If the qualifier is met, the Go To flag is set. If the qualifier is not met the program address is incremented twice before returning to the Program Execution Process. The result is that the 9100B skips the two program steps after the qualifier.

ALPHA - NUMERIC OPERATIONS IN THE 9100B

The first thing that is done for an alpha-numeric operation is to see if the Go To flag is set. If it is the key-code is stored as part of an upcoming address modification. Should that key-code be the first half of the address the 9100B exits from the Key-Code Execution Process to get the next key-code. The second part of the address is stored and then the actual address modification is performed.

Should the Go To flag not be set the 9100B checks for other initializer flags such as x→( ) and y→( ), etc., and performs the proper activities to accomplish the meaning of the operation.

Note that the inspection of an address (to see if it is alpha-numeric) following a Go To or met qualifier is implicity accomplished by the sequence of operations after the Go To flag is set. For instance, if Go To (Print) (6) is in a program for a 9100B, the Go To flag will be set to a 1, but the execution of the Print will cause a printout on the Printer. When the 9100B gets to the 6 it will simply be entered as a digit into the X Register. This is because most of the operations that cannot be part of an address following a Go To or a qualifier, clear the Go To flag when they are executed. The Plus and Minus operations are notable exceptions: during their execution the Go To flag is checked, and if it is set the Plus and Minus operations become further initializers on the upcoming address.

THE OTHER OPERATIONS

To go beyond the operations that have been described to this point would be extremely difficult, and for the most part, would be of minimal interest anyway.

The strongest statement that can be made is: all of the 9100's operations fit into one of the three categories outlined for the Print-After-Printer. However, the actual sequence of ITBB's and IADV's is not predictable without the most extensive training in the machine level activity accompanying each operation. Even then, iterative techniques complicate the picture by making the size of the numbers involved play a part in the internal logic.

In addition, there are several operations whose execution involve the outright execution of many other operations. This is used extensively and the Calculator has a way to keep track of whether an operation is being executed for its own sake at the request of the user, or whether it is being used to help in execution of another operation. The operations that make the most use of this are the Transcendentals, such as tan x.

PROGRAM EXECUTION PROCESS

The first thing that is done in the Program Execution Process is to increment the program address. Then the Format flag is checked to see if the key-code at the new program address is part of a Format operation. If it is the key-code is read and outputted with ITBB. Also, the program execution flags are removed so that the 9100 will stop execution of the program.

If the Format flag is not set the 9100 enters whatever key-code happens to be on the Key Lines, checks YSKIC (which should be normally true), enters the key-code again and then asks if it is in Run Mode (i.e., what is the setting of the PROGRAM/RUN switch on the Keyboard, as modified by YRUN on the I/O terminals). The only time that the Calculator gets into the Program Execution Process is when it is in Run Mode. All this latter activity amounts to little. The key-code that is entered is not used. It happens this way because of overlap in the internal logic of the Calculator; the Program Execution Process and the Program Storage Process are actually the same section of internal Calculator logic.

PROGRAM INTERRUPTION WITH YSKIC

However, it does present a fine opportunity to interrupt the execution of a program without using a Stop errand. All one need do is ground YSKIC, and the next time the 9100 gets to this section of internal logic it will lock up — harmlessly. The only thing to be aware of is that several operations (Go To, qualifiers) make use of that same segment during their execution. So without going to elaborate methods, the user won't necessarily know where he has stopped the Calculator.

PROGRAM INTERRUPTION WITH A STOP ERRAND

The next thing that the 9100 does while in the Program Execution Process is to check and see if key-code 41_8 (Stop) is on the Key Lines. It is not necessary for either YDSP or YKDN to be active. The "Enter key-code" just completed supplies the needed information. If the key-code for Stop is on the Key Lines the program execution flags are removed and the statement about to be executed is executed anyway. After that the 9100 will go back to the Display — provided that the last statement is not a Print, Format or Continue, which undo the effect of the Stop.

The last bit of activity in the Program Execution Process concerns the giving of output signals. In the 9100A some (the number depends on the program address) ITBB's are given followed by an IADV and finally an IOPF. During the IOPF F41B will always be a 1. The ITBB's given will reveal the key-code about to be executed, and it will be revealed again after the IOPF by the ITBB just before the entrance to the Key-Code Execution Process.

In the 9100B the process is similar, the difference being that there is only one ITBB before the IADV, and that a Step Program - caused operation will cause F41B to be set to a 0 for the IOPF.

PROGRAM STORAGE PROCESS

The Program Storage Process of the 9100B is functionally the same as that of the 9100A. They do differ in their internal make-up, however. But the differences are of a timing nature, and are less than a few tens of microseconds. Since any interfacing will take place with millisecond timing, no problems should arise.

IMPORTANCE OF STEP PROGRAM

The first thing the Program Storage Process does is check if a Step Program errand was entered. This is important, and allows flagging to be set up to result in either the storage of a new statement into a program, or the inspection of an existing program.

If the errand was Step Program the internal Step Program flag is set to a 1 and the 9100 reads the upcoming step from the program and puts it on the Buffered Bits.

If the errand was not Step Program the internal Step Program flag is set to a 0 and the 9100 skips around the reading of the next program step.

YGNPS OUTPUT OPTION

Either way the 9100 arrives at the YGNPS output option. If the errand entered was Step Program, and if YGNPS is grounded the 9100 repeatedly outputs the key-code that is the next step in the program being inspected. If this is not desired YGNPS is left alone. If the errand was not Step Program the code on the Buffered Bits will be the key-code for the errand just entered.

YSKIC INPUT OPTION

Next the 9100 checks to see if YSKIC is at ground. If it is the YSKIC input option is desired and the 9100 skips over the setting up of the Program Mode Display in the X Register. If YSKIC is not at ground the X Register is set up to display the last program address and step.

Then the program address is incremented. At this point the 9100 is ready to accept a new key-code to be stored as the next step in a program. If YSKIC is at ground the new key-code will be entered repeatedly until YSKIC is released. Then the key-code is entered one more time. (Because of the last entry, the key-code must stay on the Key Lines for a short time, say 5 μsec, after YSKIC goes true).

> **NOTE**
>
> If the 9100 is in Program Mode and a program is being entered with one of the remote methods the key-code **must** remain on the Key Lines until after the last "Enter key-code" in the Program Storage Process.

PROGRAM STORAGE PROCESS

At this point the 9100 checks to see if it is in Program Mode. The answer is always yes, since there is no way to get into the Program Storage Process unless the machine is in Program Mode. However, if the user is forcing Program Mode by grounding YRUN he must make sure that YRUN stays at ground for as long as necessary.

Next the 9100 checks to see if the Step Program flag is a 1. If it is not, the key-code just entered is stored into program memory at the new program address. If the flag is a 1 the 9100 skips that activity.

Either way, the exit from the Program Storage Process is accomplished by going to the beginning of the execution of the Stop operation.

MAGNETIC CARD READER

Even though the Magnetic Card Reader is contained in the case of the 9100 it is in principle an interface of a separate device to the 9100. The interface is based on YSKIC, YGNPS and the properties of Step Program. A detailed discussion of this interface is in Appendix IV.

APPENDIX II – 9100 FUNCTIONAL DESCRIPTION

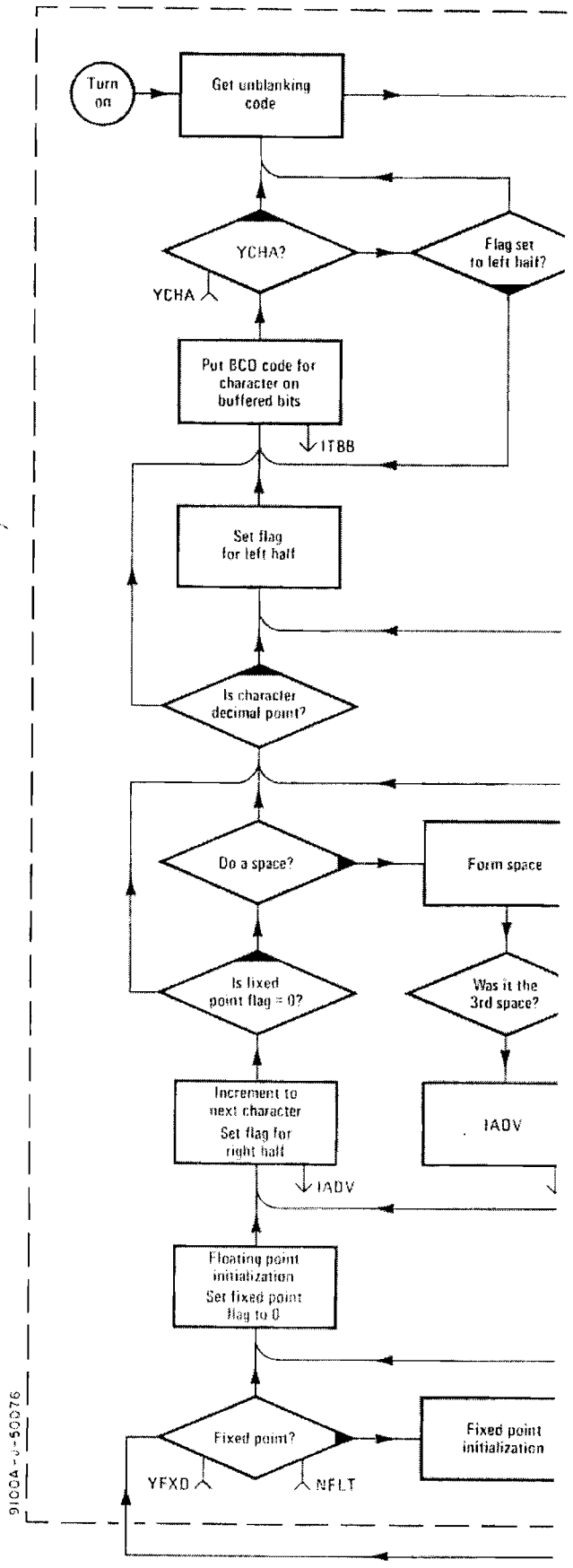
THE DIRECTOR

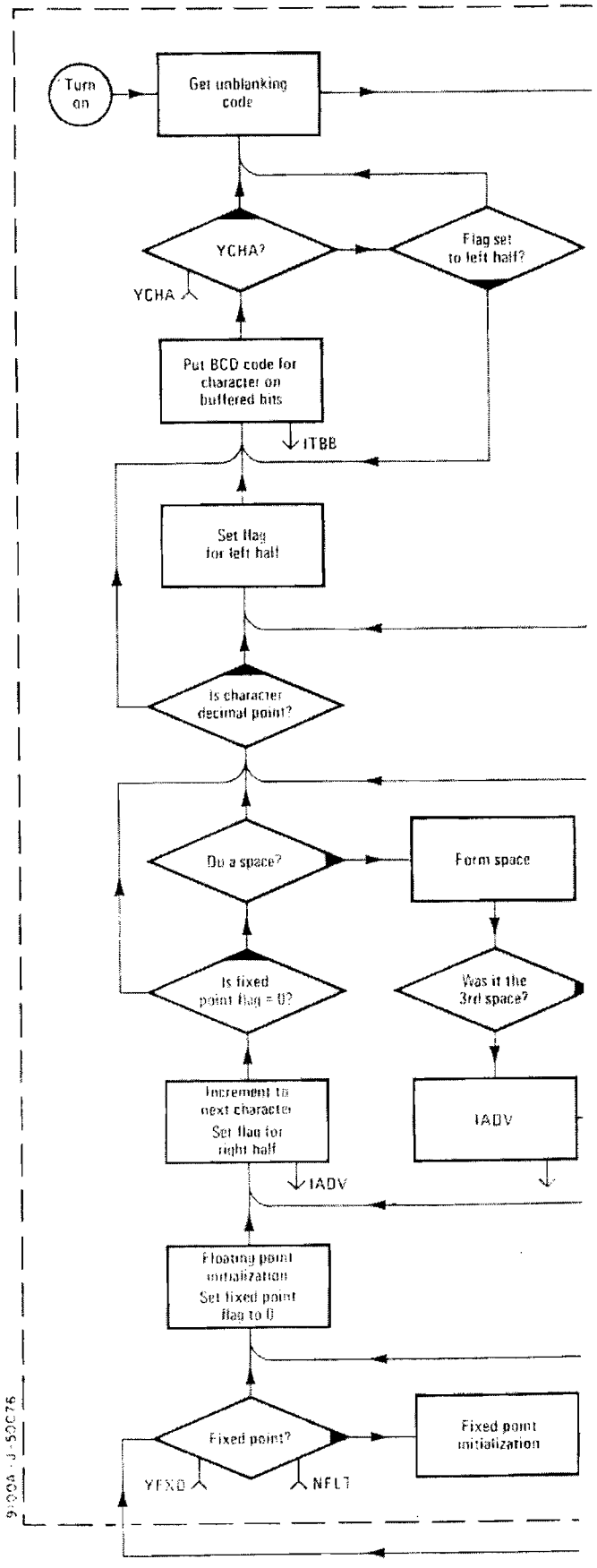
Each of the three major processes discussed in this Appendix eventually terminates at the same place, called the Director. Its function is to keep track of what the Calculator is up to and steer it to the proper activity. If the Calculator is running a program the Director will steer the Calculator to the Program Execution Process unless the last statement executed was a Pause. In that and all cases where the Calculator is not running a program the Director steers the Calculator to the Display Process.

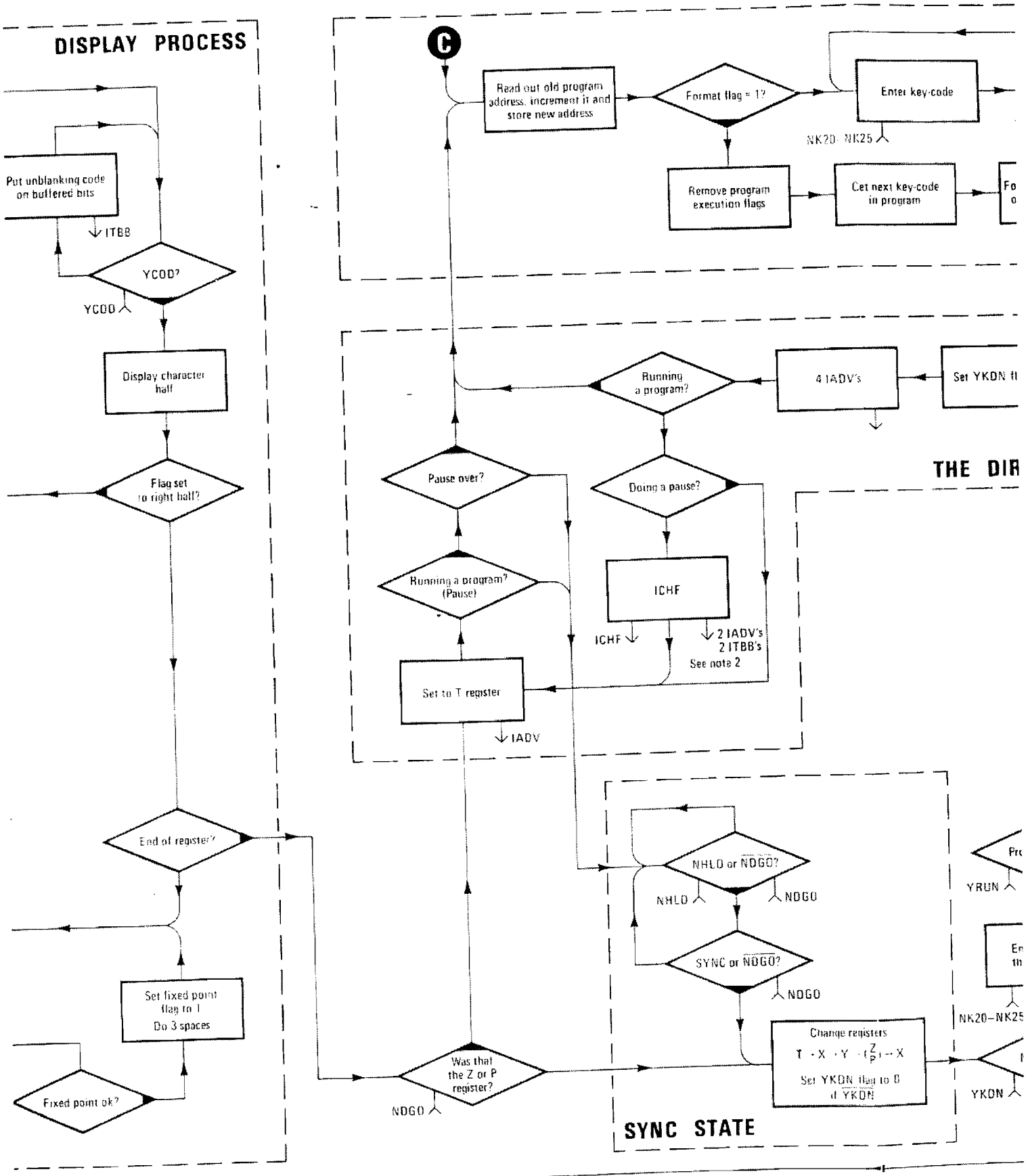
ICHF

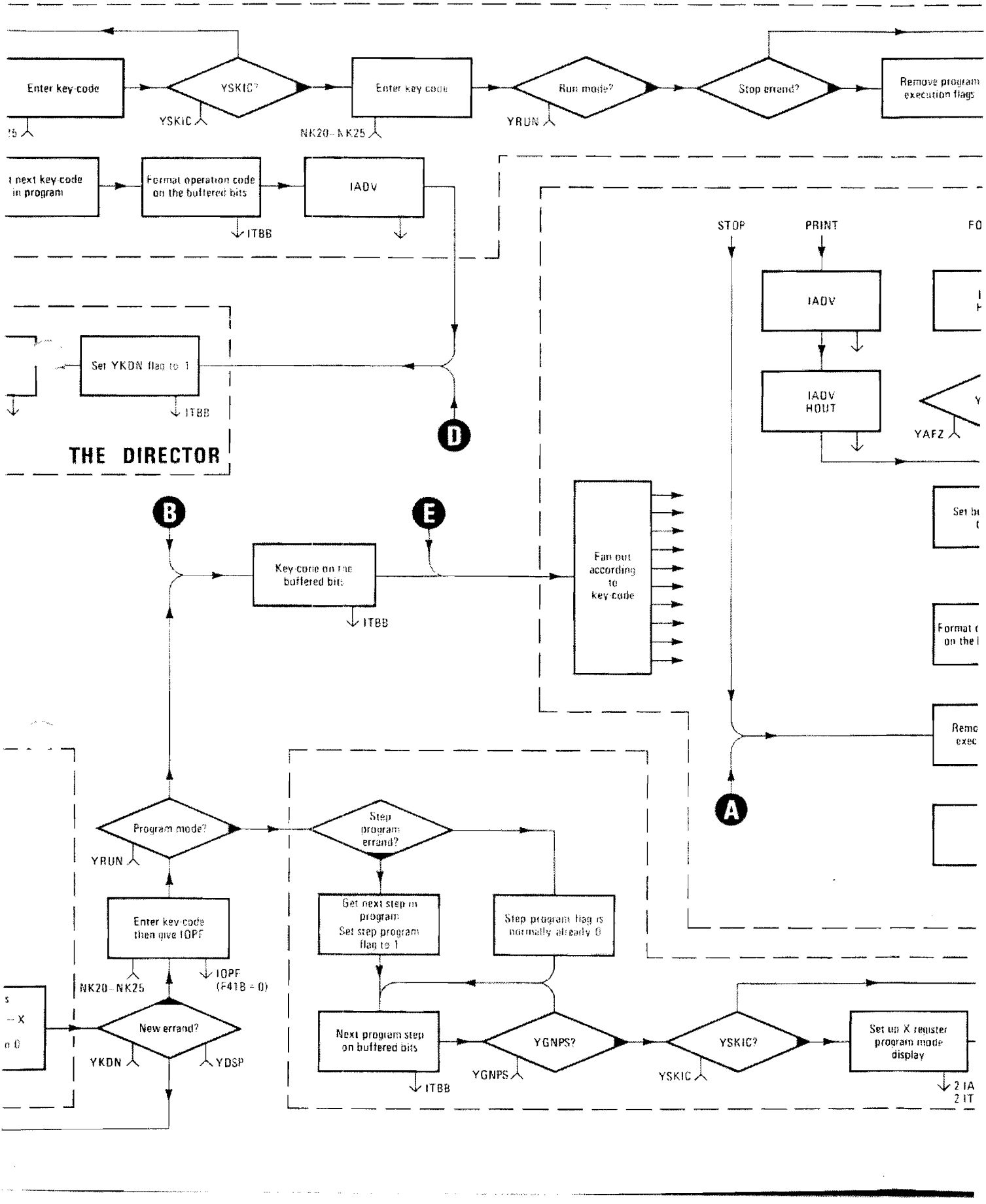
If the Director sends the 9100 back to the Display an ICHF is given. The exception to this is that the block that gives ICHF is bypassed if the Display is for a Pause.

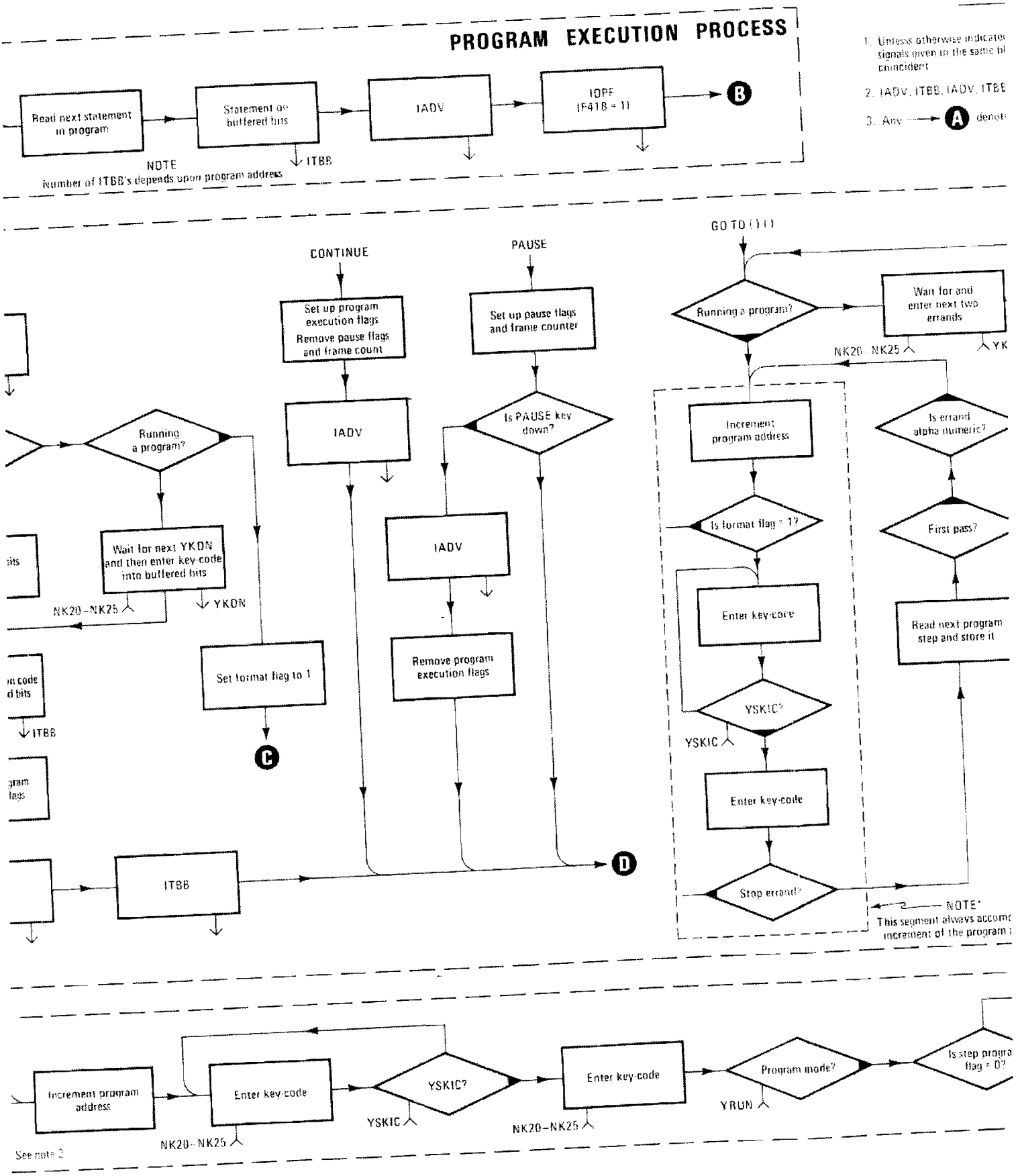
In the 9100A the ICHF is immediately before the entrance to the Display oriented logic previously discussed at the beginning of this Appendix. In the 9100B the P Register is set up (approximately 100 μsec) before that logic is reached. Note that in the 9100B the P Register is not set up for a Pause.

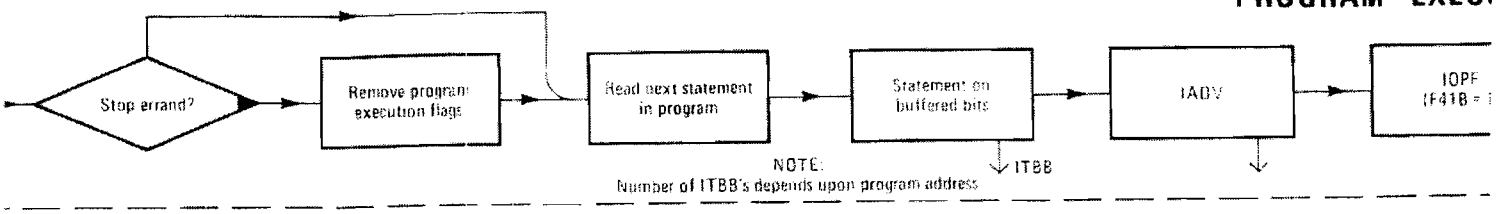

Turn on
Get unblanking code
YCHA?
YCHA
Flag set to left half?
Put BCD code for character on buffered bits
ITBB
Set flag for left half
Is character decimal point?
Do a space?
Form space
Is fixed point flag = 0?
Was it the 3rd space?
Increment to next character
Set flag for right half
IADV
IADV
Floating point initialization
Set fixed point flag to 0
Fixed point?
YFXD
NFLT
Fixed point initialization
91004A-3-50076

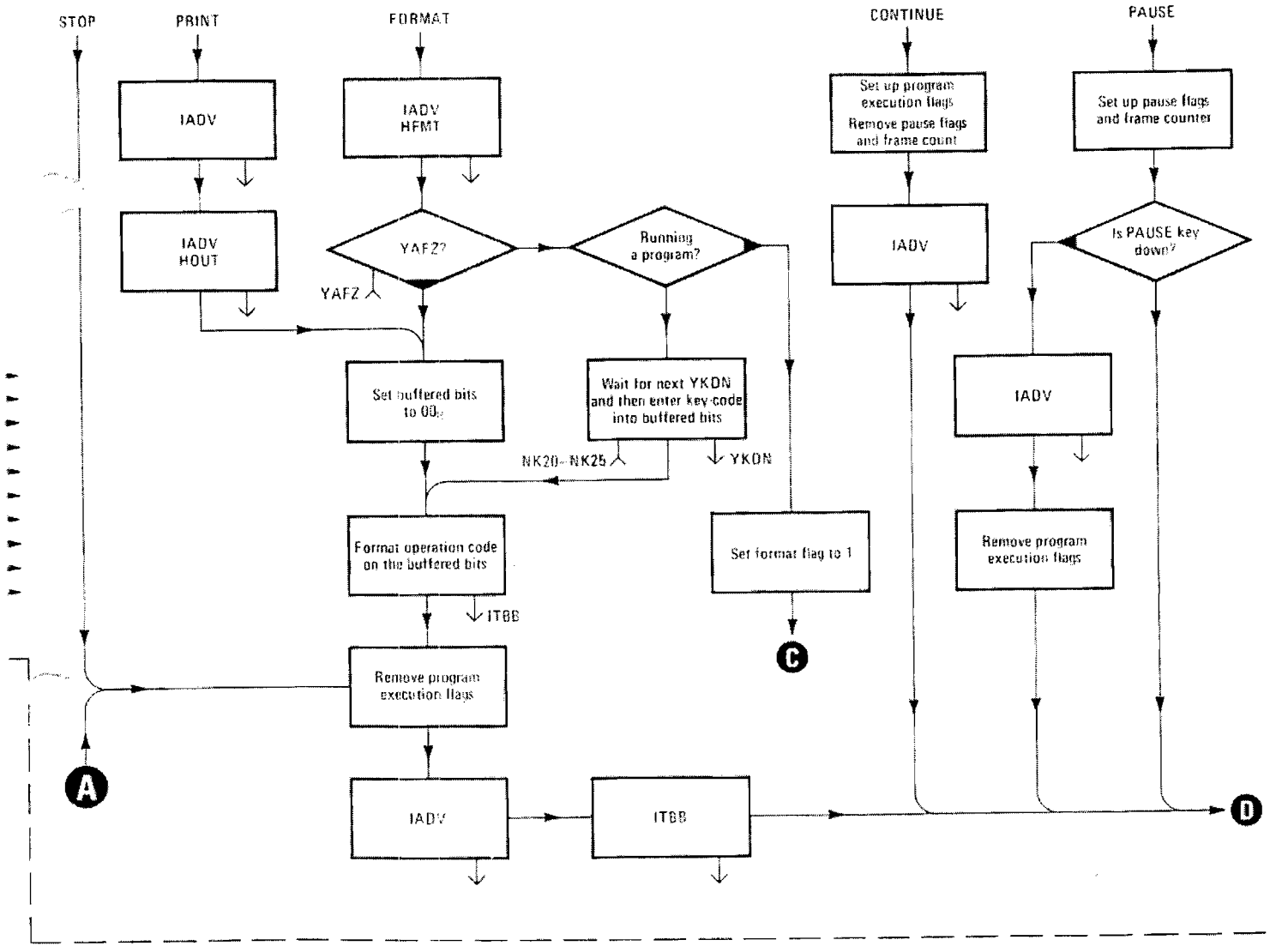

Turn on
Get unblanking code
YCHA?
Flag set to left half?
YCHA
Put BCD code for character on buffered bits
ITBB
Set flag for left half
Is character decimal point?
Do a space?
Form space
Is fixed point flag = 0?
Was it the 3rd space?
Increment to next character
Set flag for right half
IADV
IADV
Floating point initialization
Set fixed point flag to 0
Fixed point?
Fixed point initialization
YFXD
NFLT
9-004-J-50076

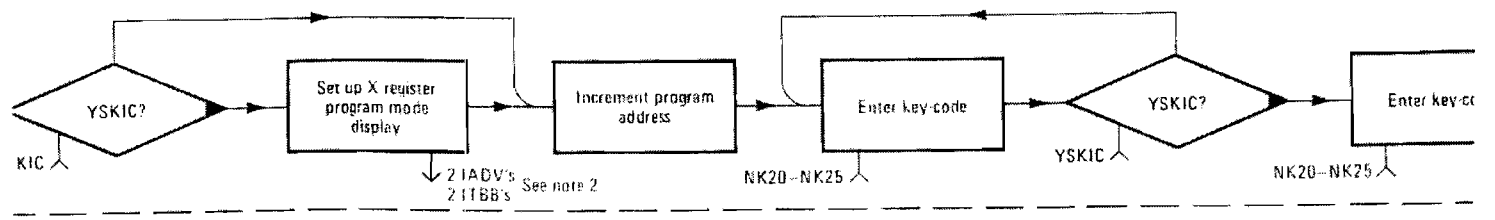

DISPLAY PROCESS
C
Read out old program address, increment it and store new address
Format flag = 1?
Enter key-code
NK20- NK25
Remove program execution flags
Get next key-code in program
Put unblanking code on buffered bits
ITBB
YCOD?
YCOD
Display character half
Flag set to right half?
Running a program?
4 IADV's
Set YKON fl
THE DIR
Pause over?
Doing a pause?
Running a program? (Pause)
ICHF
ICHF
2 IADV's
2 ITBB's
See note 2
Set to T register
IADV
End of register?
NHLD or NDGO?
NHLD
NDGO
SYNC or NDGO?
NDGO
Set fixed point flag to 1
Do 3 spaces
Fixed point ok?
Was that the Z or P register?
NDGO
Change registers
T - X - Y - (Z/P) - X
Set YKON flag to 0
d YKON
SYNC STATE
YRUN
NK20–NK25
YKON


Enter key-code
YSKIC?
YSKIC
Enter key code
NK20–NK25
Run mode?
YRUN
Stop errand?
Remove program execution flags
t next key-code in program
Format operation code on the buffered bits
ITBB
IADV
Set YKDN flag to 1
ITBB
D
THE DIRECTOR
STOP
PRINT
IADV
IADV HOUT
YAFZ
B
E
Key-code on the buffered bits
ITBB
Fan out according to key-code
A
Program mode?
YRUN
Step program errand?
Enter key-code then give IOPF
NK20–NK25
IOPF (F41B = 0)
Get next step in program Set step program flag to 1
Step program flag is normally already 0
New errand?
YKDN
YDSP
Next program step on buffered bits
ITBB
YGNPS?
YGNPS
YSKIC?
YSKIC
Set up X register program mode display


PROGRAM EXECUTION PROCESS
Read next statement in program
Statement on buffered bits
IADV
IOPF (F41B = 1)
B
ITBB
NOTE
Number of ITBB's depends upon program address
1. Unless otherwise indicated signals given in the same bl coincident
2. IADV, ITBB, IADV, ITBB
3. Any A deno
GOTO( ) ( )
CONTINUE
PAUSE
Set up program execution flags
Remove pause flags and frame count
Set up pause flags and frame counter
Running a program?
Wait for and enter next two errands
NK20–NK25
YK
IADV
Is PAUSE key down?
Increment program address
Is errand alpha numeric?
Running a program?
Is format flag = 1?
First pass?
Wait for next YKDN and then enter key-code into buffered bits
IADV
Enter key-code
Read next program step and store it
NK20–NK25
YKDN
Set format flag to 1
Remove program execution flags
YSKIC?
YSKIC
C
Enter key-code
ITBB
Stop errand?
D
NOTE*
This segment always accomc increment of the program
Increment program address
Enter key-code
YSKIC?
Enter key-code
Program mode?
Is step progra flag = 0?
YSKIC
NK20–NK25
YRUN
NK20–NK25
See note 2


Stop errand?
Remove program execution flags
Read next statement in program
Statement on buffered bits
IADV
IOPF (F41B = 1
ITBB
NOTE:
Number of ITBB's depends upon program address


STOP
PRINT
FORMAT
CONTINUE
PAUSE
IADV
IADV HFMT
IADV HOUT
YAF2?
YAF2
Running a program?
Set up program execution flags
Remove pause flags and frame count
Set up pause flags and frame counter
IADV
Is PAUSE key down?
Set buffered bits to 00₈
Wait for next YKDN and then enter key-code into buffered bits
NK20–NK25
YKDN
IADV
Format operation code on the buffered bits
Set format flag to 1
Remove program execution flags
ITBB
C
Remove program execution flags
A
IADV
ITBB
D


YSKIC?
KIC
Set up X register program mode display
2 IADV's
2 ITBB's
See note 2
Increment program address
Enter key-code
NK20–NK25
YSKIC?
YSKIC
Enter key-co
NK20–NK25

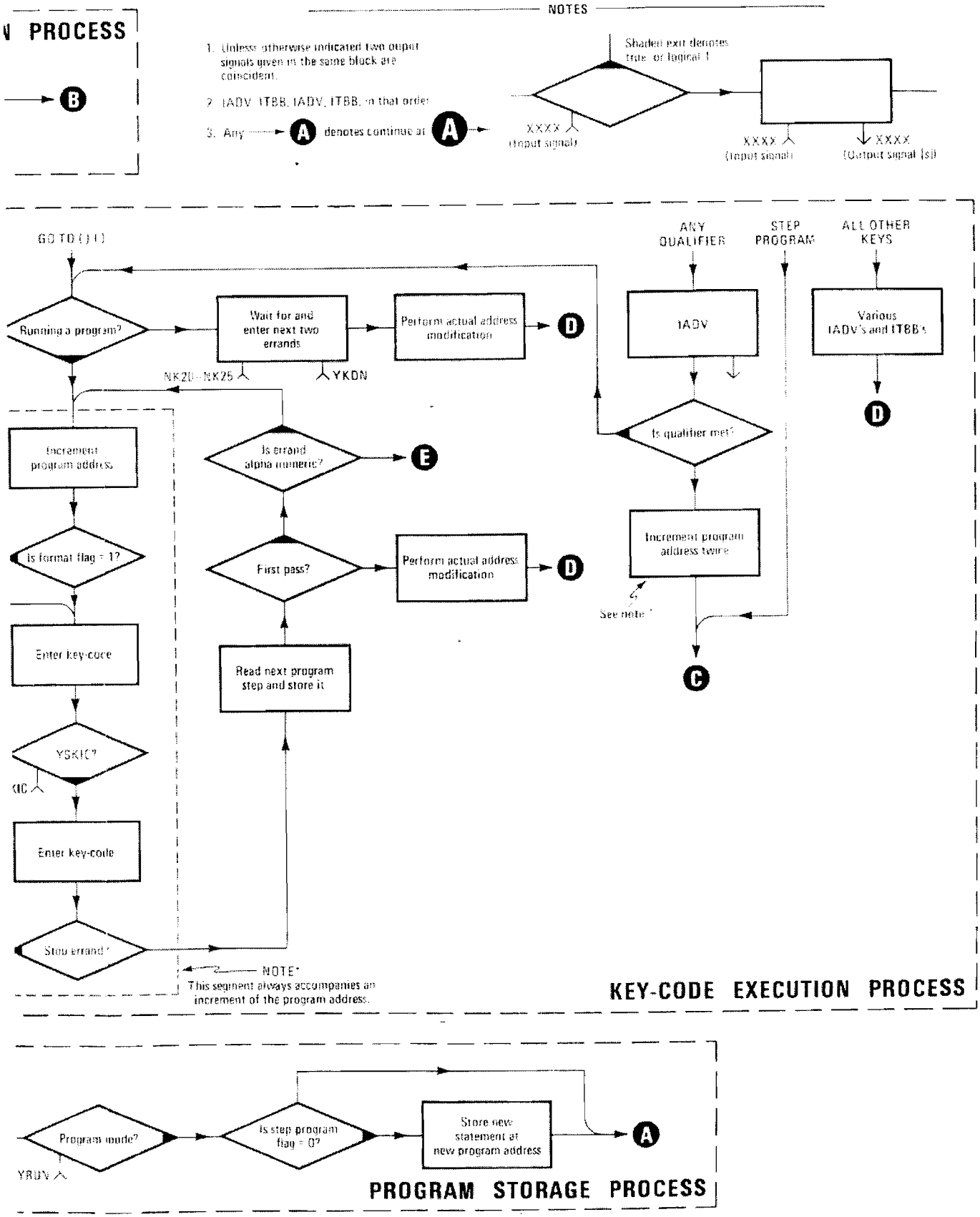
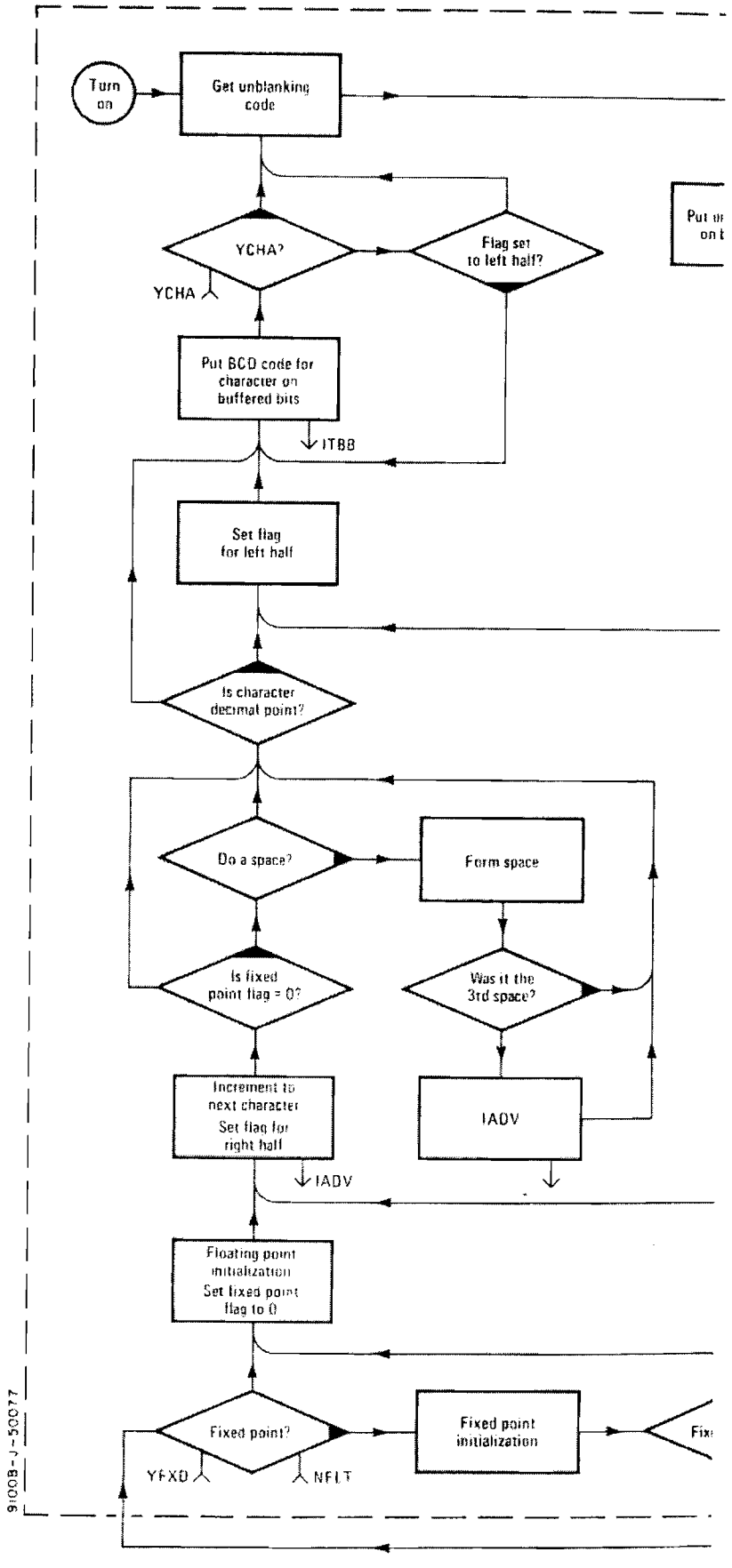
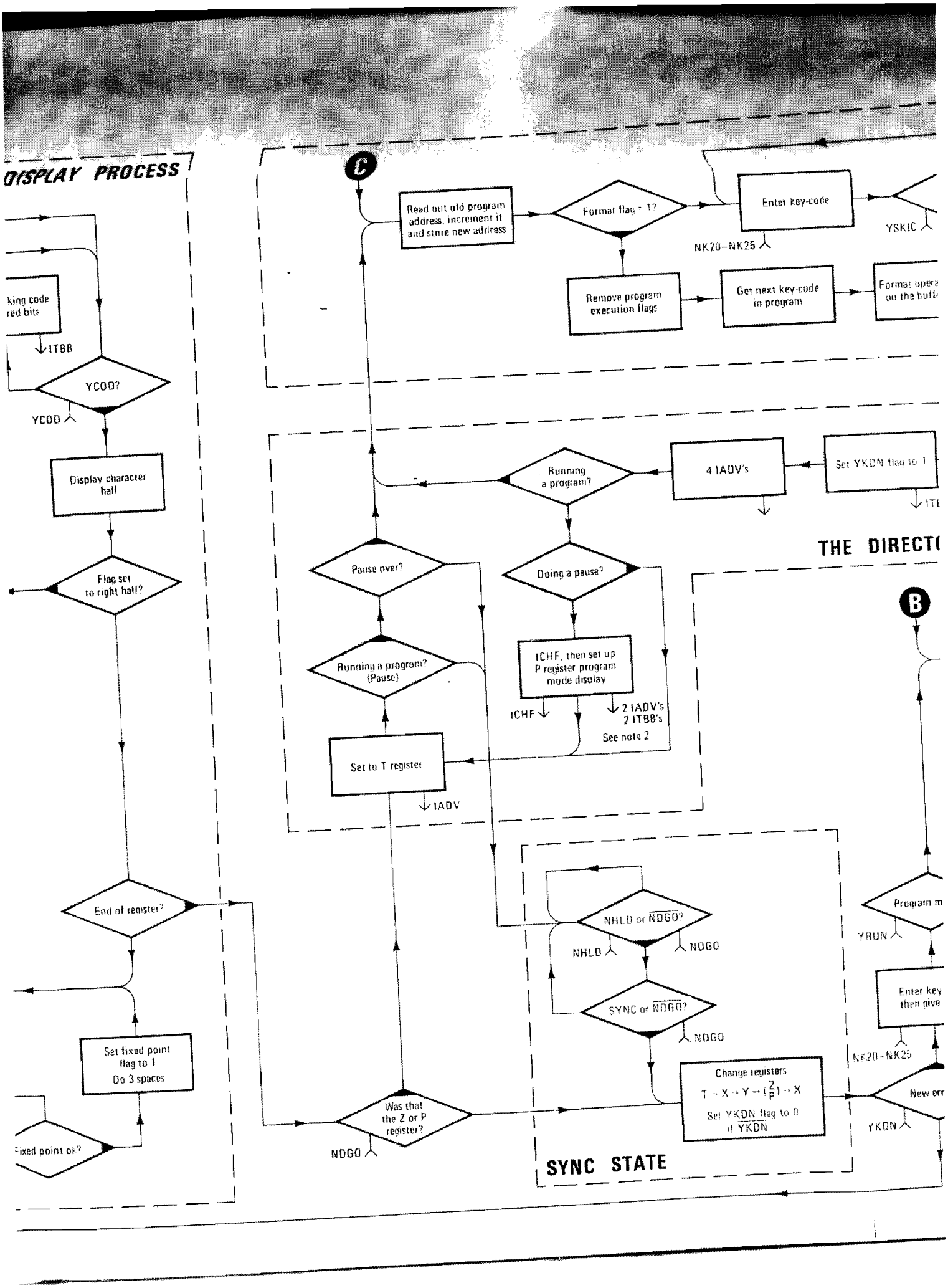
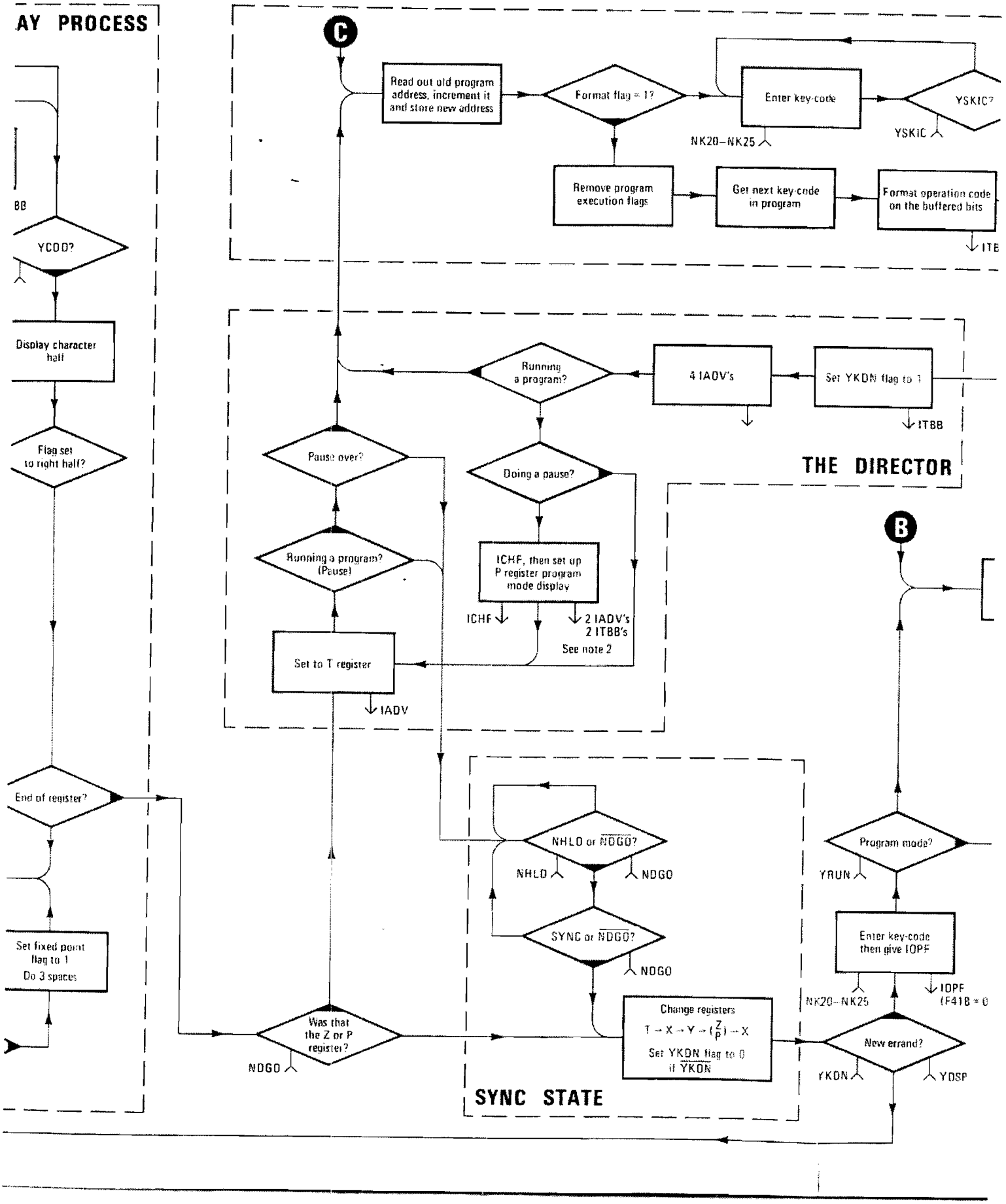
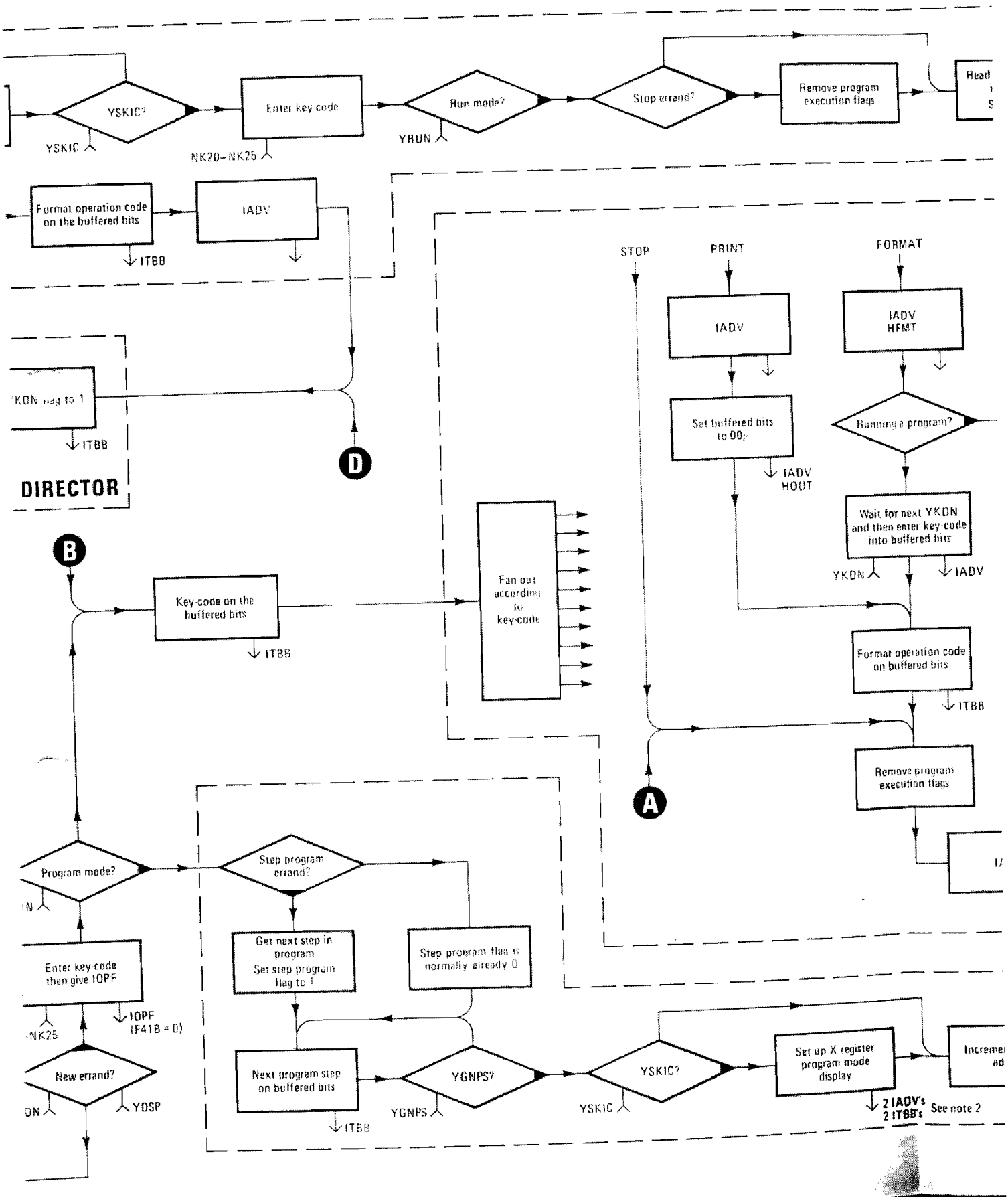


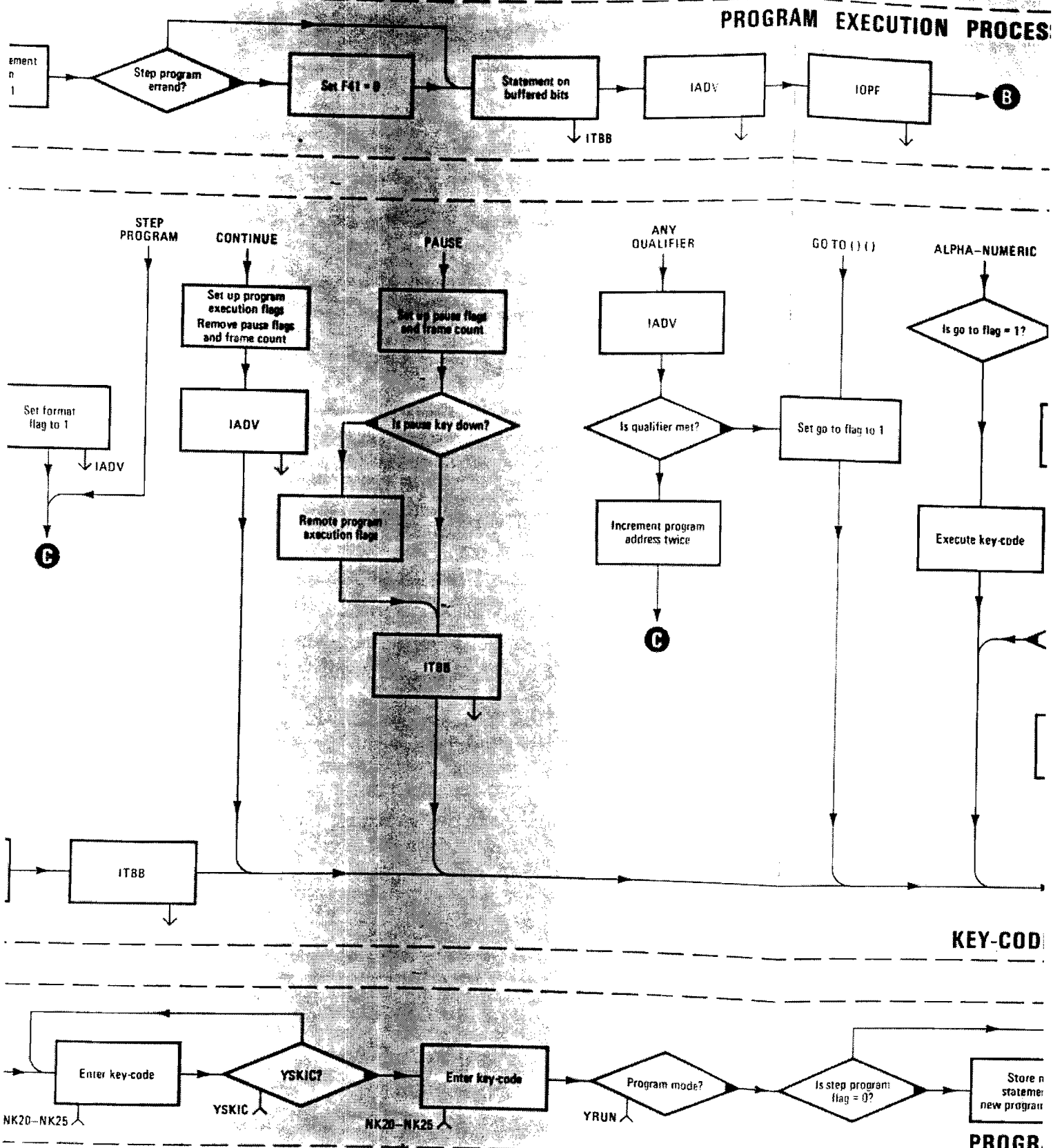
Figure 10. 9100A functional flowchart.

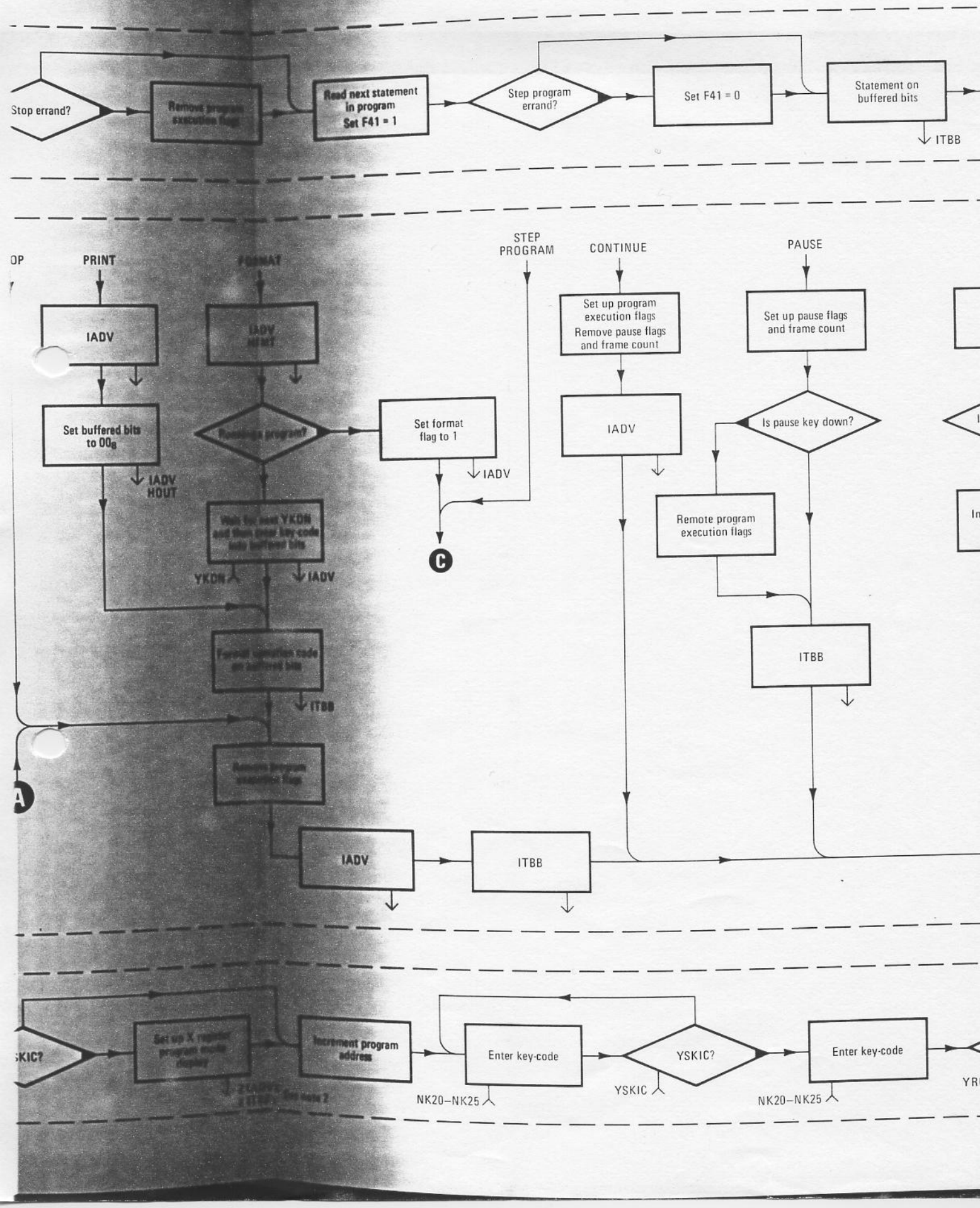

Turn on
Get unblanking code
YCHA?
Flag set to left half?
YCHA
Put BCD code for character on buffered bits
ITBB
Set flag for left half
Is character decimal point?
Do a space?
Form space
Is fixed point flag = 0?
Was it the 3rd space?
Increment to next character
Set flag for right half
IADV
IADV
Floating point initialization
Set fixed point flag to 0
Fixed point?
Fixed point initialization
YFXD
NFLT
9008-J-50077


DISPLAY PROCESS
king code
red bits
ITBB
YCOD?
YCOD
Display character half
Flag set to right half?
End of register?
Set fixed point flag to 1
Do 3 spaces
Fixed point ok?
C
Read out old program address, increment it and store new address
Format flag = 1?
Enter key-code
YSKIC
NK20–NK25
Remove program execution flags
Get next key-code in program
Format opera on the buff
Running a program?
4 IADV's
Set YKDN flag to 1
ITE
THE DIRECT
Pause over?
Doing a pause?
Running a program? (Pause)
ICHF, then set up P register program mode display
ICHF
2 IADV's 2 ITBB's
See note 2
Set to T register
IADV
Was that the Z or P register?
NDGO
NHLD or NDGO?
NHLD
NDGO
SYNC or NDGO?
NDGO
Change registers
T – X → Y → (Z/P) → X
Set YKDN flag to 0 if YKDN
SYNC STATE
B
Program m
YRUN
Enter key then give
NK20–NK25
New err
YKDN


AY PROCESS
BB
YCOD?
Display character half
Flag set to right half?
End of register?
Set fixed point flag to 1
Do 3 spaces
C
Read out old program address, increment it and store new address
Format flag = 1?
Enter key-code
YSKIC?
YSKIC
NK20–NK25
Remove program execution flags
Get next key-code in program
Format operation code on the buffered bits
ITE
Running a program?
4 IADV's
Set YKDN flag to 1
ITBB
THE DIRECTOR
Pause over?
Doing a pause?
Running a program? (Pause)
ICHF, then set up P register program mode display
ICHF
2 IADV's
2 ITBB's
See note 2
Set to T register
IADV
B
NHLD or NDGO?
NHLD
NDGO
SYNC or NDGO?
NDGO
Was that the Z or P register?
NDGO
Change registers
T → X → Y → (Z/P) → X
Set YKDN flag to 0 if YKDN
SYNC STATE
Program mode?
YRUN
Enter key-code then give IOPF
NK20–NK25
IOPF (F41B = 0
New errand?
YKDN
YDSP


YSKIC?
YSKIC
Enter key-code
NK20–NK25
Run mode?
YRUN
Stop errand?
Remove program execution flags
Format operation code on the buffered bits
ITBB
IADV
YKDN flag to 1
ITBB
DIRECTOR
D
STOP
PRINT
FORMAT
IADV
Set buffered bits to 00₂
IADV HOUT
IADV HFMT
Running a program?
Wait for next YKDN and then enter key-code into buffered bits
YKDN
IADV
Format operation code on buffered bits
ITBB
Remove program execution flags
B
Key-code on the buffered bits
ITBB
Fan out according to key-code
A
Program mode?
Step program errand?
Get next step in program
Set step program flag to 1
Step program flag is normally already 0
Enter key-code then give IOPF
IOPF (F41B = 0)
NK25
New errand?
YDSP
Next program step on buffered bits
ITBB
YGNPS?
YGNPS
YSKIC?
YSKIC
Set up X register program mode display
2 IADV's
2 ITBB's
See note 2


PROGRAM EXECUTION PROCES
Step program errand?
Set F41 = 0
Statement on buffered bits
IADV
IOPF
B
ITBB
STEP PROGRAM
CONTINUE
PAUSE
ANY QUALIFIER
GO TO ( ) ( )
ALPHA–NUMERIC
Set up program execution flags
Remove pause flags and frame count
Set up pause flags and frame count
IADV
Is go to flag = 1?
Set format flag to 1
IADV
Is pause key down?
Is qualifier met?
Set go to flag to 1
Remote program execution flags
Increment program address twice
Execute key-code
C
ITBB
ITBB
KEY-COD
Enter key-code
YSKIC?
Enter key-code
Program mode?
Is step program flag = 0?
Store n statemer new program
NK20–NK25
YSKIC
NK20–NK25
YRUN
PROGR


Stop errand?
Remove program execution flags
Read next statement in program
Set F41 = 1
Step program errand?
Set F41 = 0
Statement on buffered bits
ITBB
OP
PRINT
FORMAT
STEP PROGRAM
CONTINUE
PAUSE
IADV
Set buffered bits to 00₈
IADV HOUT
Set format flag to 1
IADV
Set up program execution flags
Remove pause flags and frame count
IADV
Set up pause flags and frame count
Is pause key down?
Remote program execution flags
ITBB
YKDN
IADV
C
A
IADV
ITBB
Increment program address
Enter key-code
NK20–NK25
YSKIC?
YSKIC
Enter key-code
NK20–NK25

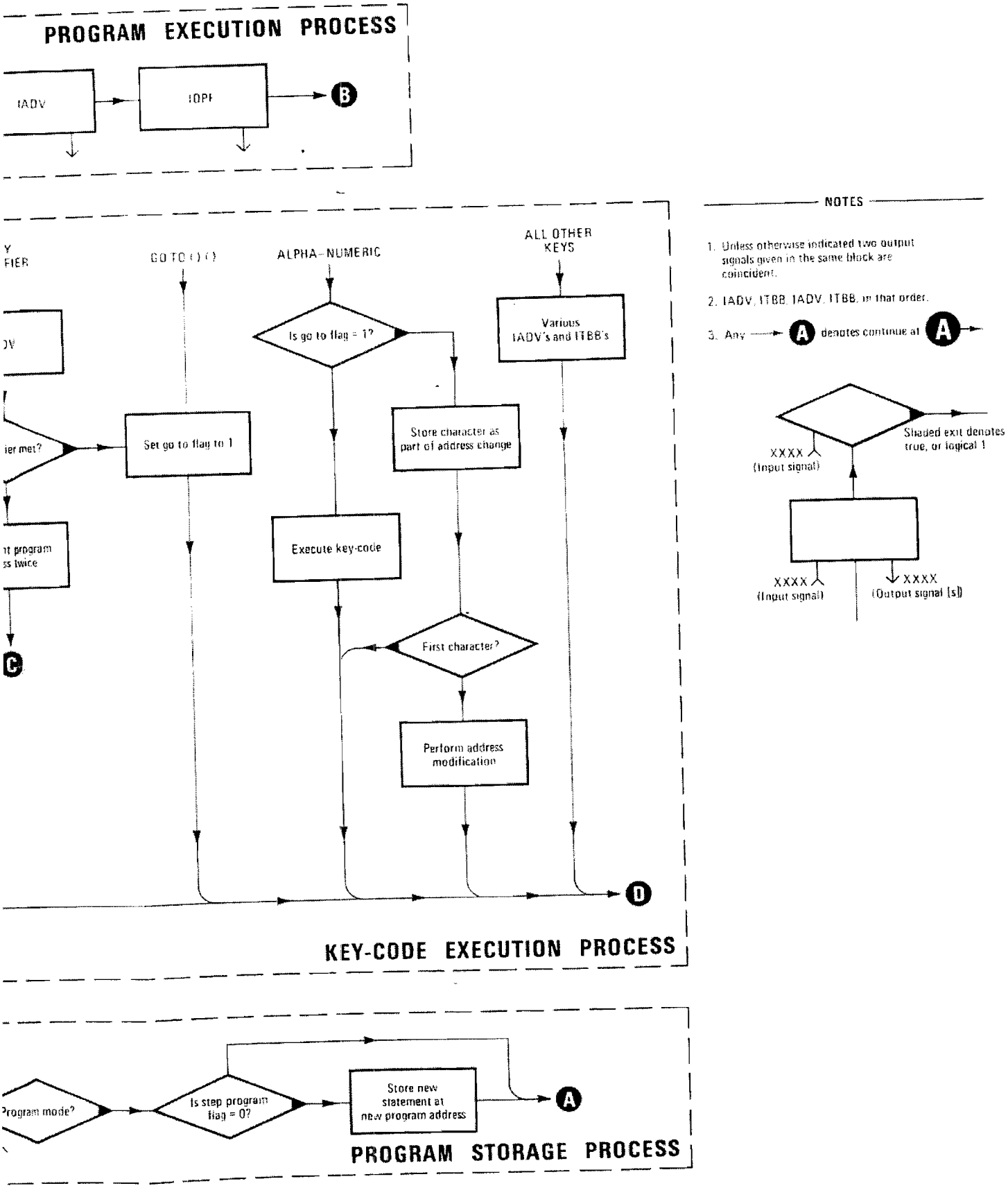


Figure 11. 9100B functional flowchart.

APPENDIX III – RECOMMENDED CIRCUITS

ELECTRICAL CONSIDERATIONS

THE SIMPLE CASE

If a private interface is the only peripheral attached to the 9100 the device can be attached directly to the 9100 by short cables, or by buffers and longer cables.

SOME COMPLICATIONS

The situation is radically altered by the addition of any other peripherals. The basic problem is that to insure good signal to noise ratio, most of the level sensing circuitry of ·hp· peripherals are current sinks. Once such a peripheral is attached it is impractical to use voltage sensing techniques to determine signal levels, and since it's also impractical to run two current sinks in parallel from the same output, one peripheral is essentially all that can be attached to the 9100. (The ·hp· 9160A and ·hp· 9150A are exceptions to this — they don't have any level sensing circuitry.) An additional problem is that the amount of stray capacitance that can be safely applied without buffering to an I/O terminal is limited to about 200 pF.

SIGNAL BUFFERING

In order to run two or more peripherals having current sink inputs, some sort of fan-out is needed. For a Calculator-Printer-Plotter combination, the Plotter plugs into the Calculator and the Printer plugs into the Plotter. The Plotter performs the level detection and buffering of 9100 output signals and supplies them to the Printer via the rear plug on the Plotter.

BUFFER BOX

The Extended Memory and the Coupler cannot connect to the Calculator through the Plotter. They must either be connected directly to the Calculator or to the Calculator through the 9102A Calculator Buffer. The Calculator Buffer will supply three connectors that are essentially the same as the I/O terminals of the 9100, yet isolated from each other, so that multiple peripherals can be used.

PROBLEMS OF BUFFERING

The construction of suitable buffers for long cable is far from simple. Aside from the usual problems of power dissipation for low impedance driver circuitry, and rise time problems, there is an additional consideration of propagation delay.

The first consideration is the effect of lumped uniform delay between the 9100 and the interface. Buffer networks inducing a 200 nsec delay (uniformly — 200 nsec for each signal) would not interfere with the logic of the system.

A second consideration is differences in propagation time for the different signals being transmitted. Those differences should be held to less than 50 nsec.

These delays are not so critical for 9100 signals being transmitted between the 9100 and a private interface, as the interface can be designed around the delay. But for remote installation of standard peripherals, these delays are very important.

VIEWING SIGNALS

Most output signals originate through a diode some distance from the I/O terminals. The stray capacitance in the line from the diode to the terminal is charged by the signal, and due to the diode, won't be adequately discharged unless a load of from 2K to 5K is attached. In order to view the signals the load must be attached.

–15 VOLT SUPPLY

The –15V supply on the rear plug of the 9100 is intended for use with the ·hp· 9120A and ·hp· 9160A, and not as a general purpose supply for an interface. A maximum of 50 ma can be taken from this supply.

ELECTRICAL CONSIDERATIONS

Figure 12 shows the recommended input circuits for the 9100.

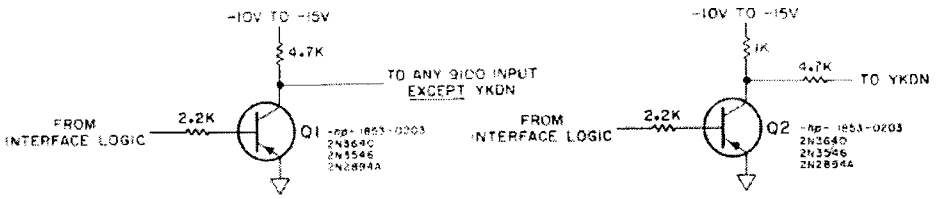
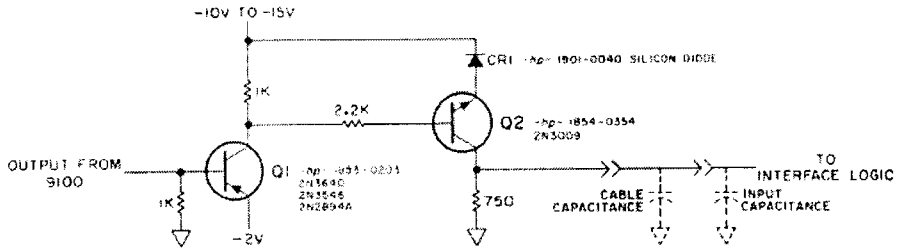


Figure 12. Recommend input circuits.

Figure 13 shows the recommended level sensing circuit for the 9100.



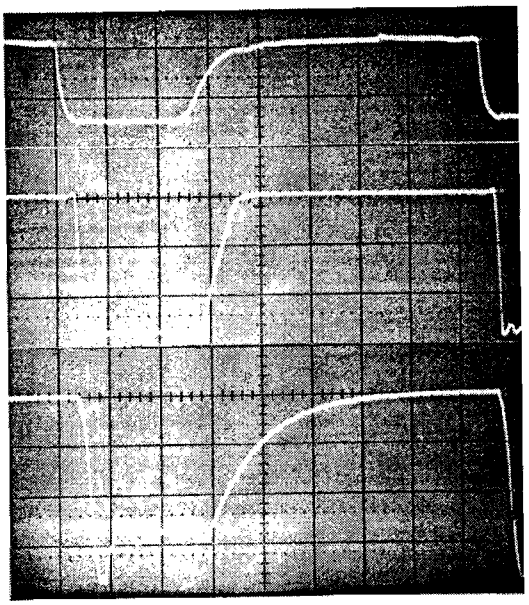


Figure 13. Recommended level sensing circuit.

These circuits assume that the distance from the interface to the 9100 is relatively short. For longer distances these circuits should be kept at a close distance to the 9100 and suitable buffering used to match the circuits to the required cable length.

9100 INPUT CIRCUITS

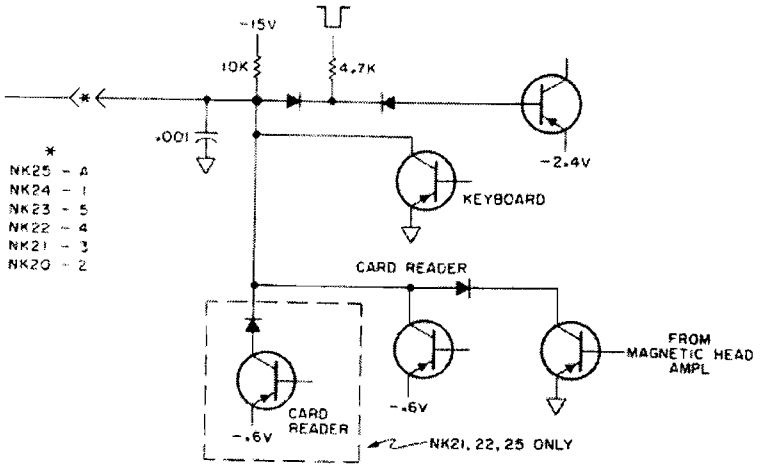


Figure 14. Typical internal circuitry for 9100 Key Lines.

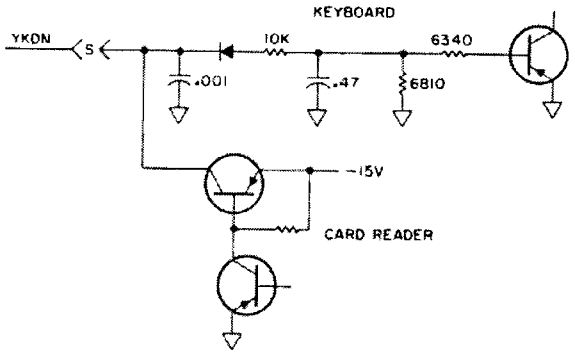


Figure 15. 9100 internal circuit for YKDN.

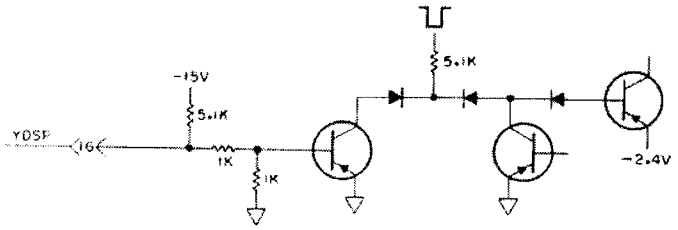


Figure 16. 9100 internal circuit for YDSP.

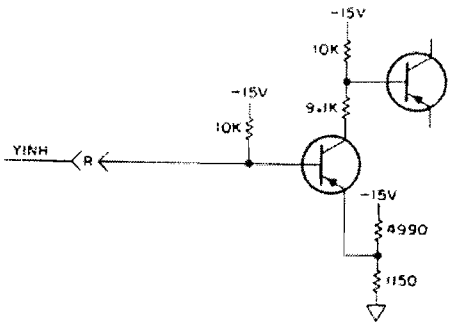


Figure 17. 9100 internal circuit for YINH.

9100 INPUT CIRCUITS

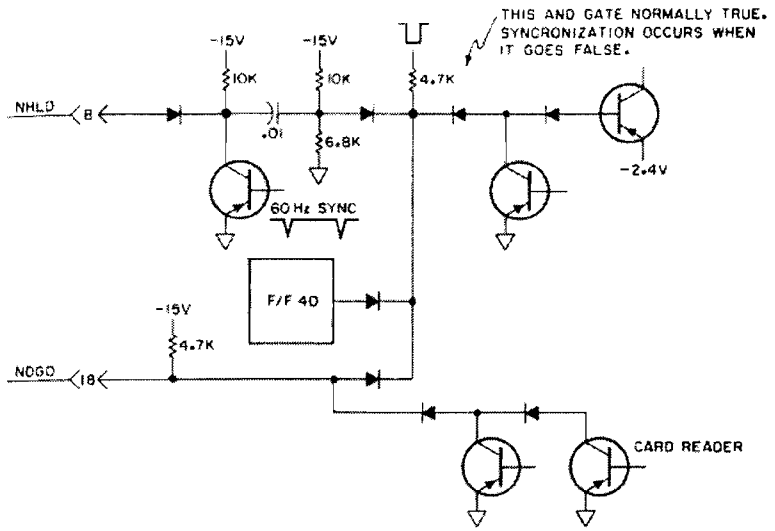


Figure 18. 9100 internal circuit for NHLD and NDGO.

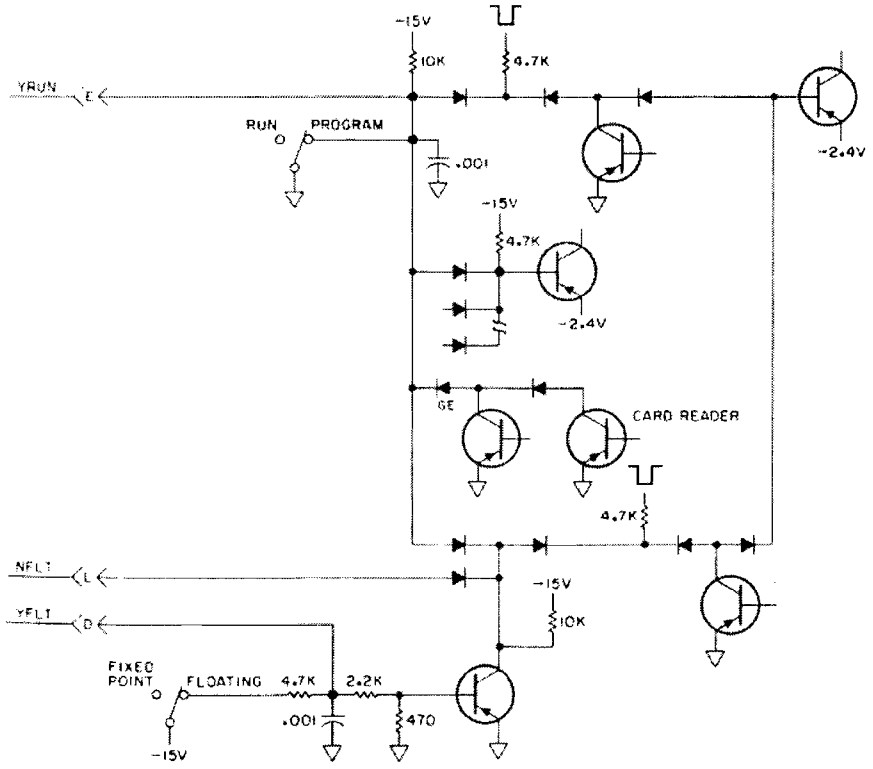


Figure 19. 9100 internal circuitry for YRUN, NFLT and YFLT.

9100 INPUT CIRCUITS

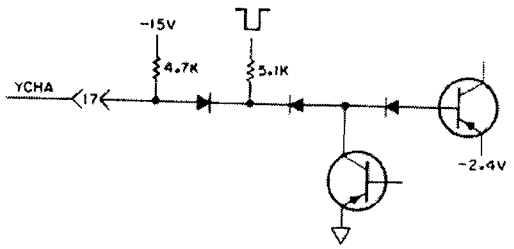


Figure 20. 9100 internal circuit for YCHA.

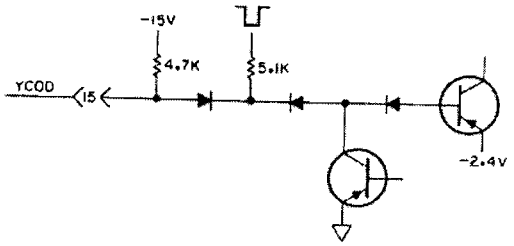


Figure 21. 9100 internal circuit for YCOD.

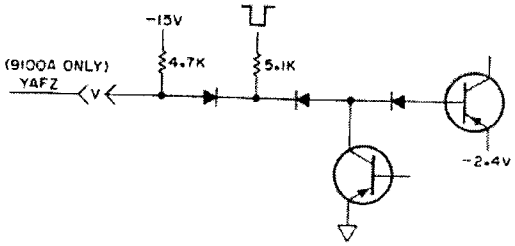


Figure 22. 9100 internal circuit for YAFZ.

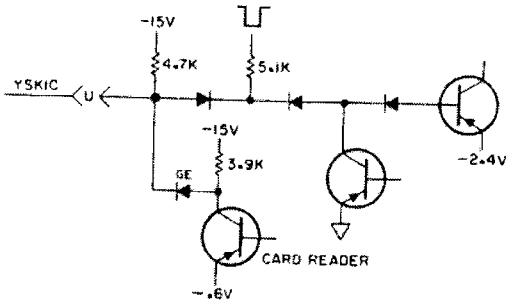


Figure 23. 9100 internal circuit for YSKIC.

9100 INPUT CIRCUITS

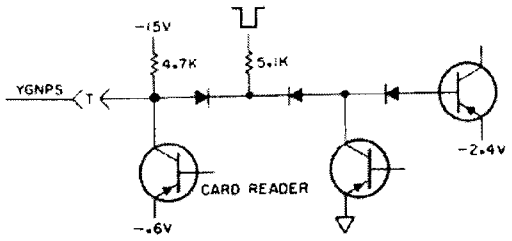


Figure 24. 9100 internal circuit for YGNPS.

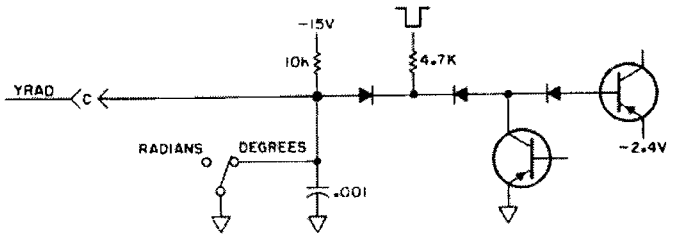


Figure 25. 9100 internal circuit for YRAD.

9100 OUTPUT CIRCUITS

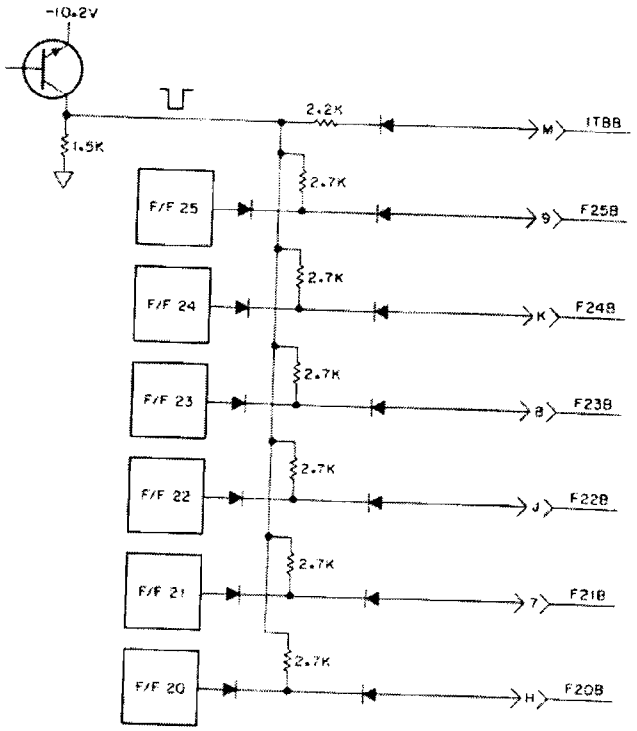


Figure 26. 9100 internal circuit for the Buffered Bits and ITBB.

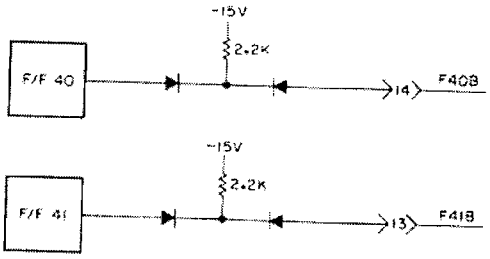


Figure 27. 9100 internal circuit for the Buffered Words.

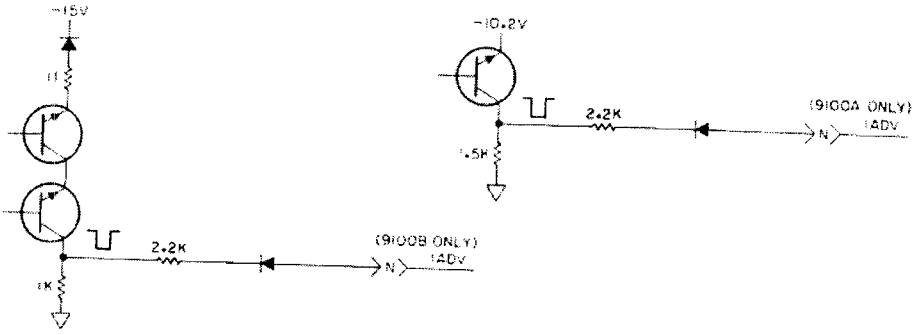


Figure 28. 9100 internal circuit for IADV.

9100 OUTPUT CIRCUITS

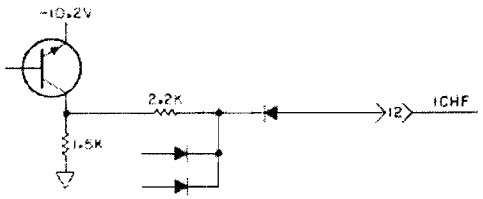


Figure 29. 9100 internal circuit for ICHF.

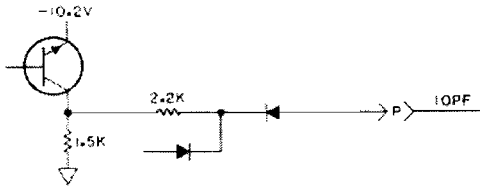


Figure 30. 9100 internal circuit for IOPF.

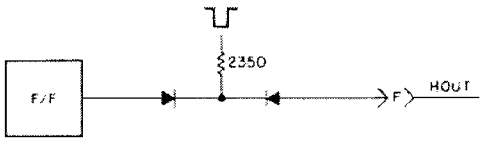


Figure 31. 9100 internal circuit for HOUT.

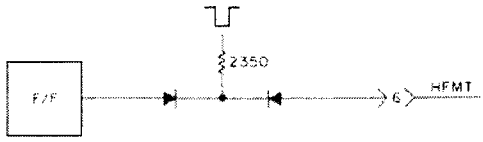


Figure 32. 9100 internal circuit for HFMT.

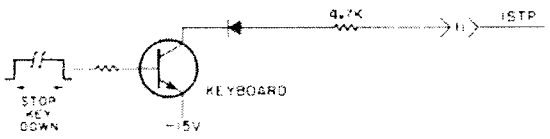


Figure 33. 9100 internal circuit for ISTP.

APPENDIX IV – A SURVEY OF -hp- PERIPHERALS

This Appendix contains a brief general description of the interface considerations of the standard -hp- peripherals. The information is given here to show what can be done, and also to aid designers of private interfaces in avoiding compatibility problems between their interface and a standard peripheral.

MAGNETIC CARD READER

The Magnetic Card Reader has a four track head. One track is used as a timing strobe, and the other three are used for data. A two part multiplexing process is used to accommodate the six bits of data that make up each key-code recorded on the card. The recording is done in a non-return-to-zero fashion.

Any Card Reader operation grounds YRUN, NDGO, and YDSP. This puts the 9100 into Program Mode, frees it from waiting for sync (which is slower than the Card Reader) and gives the Card Reader immediate access to the Program Storage Process.

ENTER

On "Enter" the Card Reader grounds YSKIC but lets YGNPS remain true. The Card Reader keeps track of the strobe channel and demultiplexes the bits recorded on the card. The reconstructed key-code is applied to the Key Lines. By the time this happens the 9100 is waiting at the "Enter key-code" in the YSKIC input option. Then YSKIC is made true for 30 μsec, releasing the 9100 to enter the key-code again and store it. By the time the 9100 reaches the Program Storage Process again YSKIC is back at ground. This allows the 9100 to skip setting up the Program Mode Display in the X Register. It also allows the 9100 to wait for the Card Reader to reconstruct the next key-code.

The Card Reader has a circuit that detects an End statement read from the card. After the End is stored the Card Reader ceases to read any more bits from the card. YRUN and YDSP go true immediately after the End is stored and the key-code for End (46_8) is again put on the Key Lines. Then YKDN goes true to cause the End errand to be **EXECUTED**.

RECORD

On "Record" the Card Reader grounds YGNPS and lets YSKIC remain true. In addition the Card Reader puts the key-code for Step Program (51_8) on the Key Lines. Since YGNPS is grounded the 9100 locks up while repeatedly outputting the next key-code to be recorded. (See NOTE below!) During this time the Card Reader multiplexes and records the key-code.

Then YGNPS is made true for 27 μsec and the rest of the activity in the 9100 is the same as for a Step Program errand. YGNPS wil be at ground again before the 9100 returns to the lock-up state.

NOTE

The Card Reader uses the internal "unbuffered bits". They are available to the Card Reader on a DC basis. The strobed nature of the Buffered Bits will not suffice in attempts to hook up a remote Card Reader of the type used in the 9100, unless additional circuitry is used to compensate for this. Also, for a 9100B there can be no automatic page changing feature.

—hp— 9120A PRINTER

The -hp- 9120A Printer is an electro-sensitive printer. It forms characters on special conductive paper by sweeping (right to left) closely spaced and slanted print fingers across the paper. Characters are formed by selectively applying a pattern of voltage to each print finger as the sweep proceeds, a situation roughly analogous to the unblanking of the CRT Beam. In fact, the Printer uses the unblanking codes from the Display in forming its characters on paper. This method prints one register per sweep, and can reproduce any valid display on the CRT.

The Printer is triggered by HOUT, or by the LIST pushbutton. Immediately YINH goes to ground. The Printer uses the pushbuttons on the front of the Printer to determine which registers to print (their order is determined by the direction of the paper advance). The two general cases are any Print operation (with any selected registers) and a List operation.

ANY PRINT

The Printer will be triggered by HOUT and will wait until the Buffered Words begin to indicate the first register to be printed. Then YCOD is grounded and the first unblanking code is obtained. After that unblanking code is implemented YCOD is made true for 5 μsec to allow the 9100 to get to the next character half, and its unblanking code.

DECIMAL POINT

There are two interesting cases that deserve attention. The first is the decimal point. It has only a left half, and it has a unique unblanking code. Since the Printer does not care about left and right halves per se, the fact that the decimal point has only a left half does not cause any problem; the Printer prints each half just as it falls in the Display. But in order to make a more attractive decimal point the Printer synthesizes its own code for the print fingers. This is done upon recognizing the unique unblanking code for the decimal point.

FIGURE 1 SHIFT

The second point of interest is the character 1. In the Display it is formed as a completely unblanked right half, but shifted over to the left by a half character width for easy viewing. The Printer does the same thing. It can tell when to shift because the unblanking code for the right half of a 1 is also unique; it is the only unblanking code that has a 1 in bit 25.

END OF THE REGISTER

The Printer detects the end of the register by the change in the Buffered Words. If there are more registers to print the Printer will latch onto the start of the next register to be printed by grounding YCOD as soon as that register is indicated by the Buffered Words.

END OF THE PRINT CYCLE

When all the selected registers have been printed the Printer will give a Continue to the 9100 if the Print was in the context of a program. The Printer continually monitors IOPF and F41B. When HOUT is given the preceding combination of IOPF and F41B is used to decide whether or not to give the Continue at the end of the print cycle. If the command to print was an errand no Continue is given.

When the Printer gives a Continue it does it by putting the key-code for Continue (47_8) on the Key Lines, making YKDN go negative, and grounding NDGO. This is done for 30 msec.

IMPORTANT REMARK

Assume that there is a Print in a program containing Format operations for an interface. At the conclusion of the print cycle YKDN goes true with 47_8 on the Key Lines. It will take anywhere from 8 to 16 msec, typically 10 msec, for YKDN to result in the internal signal required to meet the New Errand Qualifier.

–hp– 9120A PRINTER

IMPORTANT REMARK CONTINUED

By now the 9100 can be anywhere in the Display Process. It will take a maximum of 6 msec, possibly only .5msec, to get to the New Errand Qualifier. From there it takes less than 150 μsec to execute the Continue. From then on the 9100 is back running the program. But for a while, say 10 to 20 msec, the Continue is still on the Key Lines. That in itself does not interfere with the execution of the program, for the same reason that pushing keys on the Keyboard during the execution of a program has no affect. However, if a Format operation should occur during that time, the external device might try to use the Key Lines, expecting them to be free. Also, if the external device enters its key-codes by YKDN, the two YKDN's could overlap, preventing YKDN from recycling and thus resulting in no entries at all from the external device. If a Printer is to be used with a program that has Format operations for a non-standard peripheral, some sort of delay is necessary. It may be through the use of "dummy" program steps, or by putting the Format operations ahead of the Print, or by use of built in delay in the interface.

PRINT SPACE OPERATION

Two or more consecutive Print statements in a program result in the first Print causing a printout and the remaining Prints causing paper advances. The Printer can tell that HOUT's are consecutive by looking for two consecutive IOPF's with no HOUT between them. If that condition does not occur the second and succeeding HOUT's cause paper advance.

Much the same sort of operation results from Keyboard activation of the Printer. If the PRINT key is held down for a length of time, a printout results, followed by continuous spacing for the time the key is down. This is an apparent violation of the "single execution per single keystroke" rule: a convention that all the other errands are subject to.

It is not actually such a violation. During the printout or the space, YINH is grounded by the Printer. The effect of YINH is such that the Keyboard cannot tell the difference between grounding YINH and releasing the key. At the end of the print cycle YINH is released and if the PRINT key is still down, it is as if it was just pressed again.

The PRINT key can be released and then pressed an indefinite number of times, each time resulting in a paper advance. This will continue until some other key is pushed, resulting in an IOPF not followed by an HOUT. This Print-Space behavior is for any Print operation, including remote entries.

LIST OPERATION

The List operation is initiated by the pushbutton on the front of the Printer, and continues until an End statement is encountered in the program being listed, or until the STOP key on the Keyboard is pressed.

The Printer grounds YRUN for the duration of the List operation. The key-code for Step Program (51_8) is applied to the Key Lines and YKDN made negative, while at the same time YGNPS and NDGO are grounded, each for 30 msec.

The processing of Step Program while the 9100 is in Program Mode sets up the X Register Program Mode Display which is printed by the Printer.

For a portion of the time that YGNPS is grounded the 9100 hangs up, repeatedly outputting the next step in the program. If it is End (46_8) the Printer detects this and alters the chain of logic that would otherwise reapply Step Program and print again after the next step, and so on. Instead, a final print is made and the List operation is over.

The other way to terminate the List operation is to press the STOP key on the Keyboard, which makes ISTP go negative. The printer senses this and terminates the List operation.

—hp— 9125A PLOTTER

GENERAL

The -hp- 9125A Plotter plots by drawing the straight line segment between the current (old) point and the next (new) point. At the time a new point is to be entered the Plotter is triggered by a Format operation, and the number in the X Register is taken as the abscissa and the number in the Y Register taken as the ordinate. The Plotter then moves the pen to the new position along the connecting straight line. The pen may or may not be in contact with the paper when it moves, depending upon the Format operation.

To simplify the internal logic of the Plotter it was decided to take the digits from a fixed point Display. That eliminates having to deal with the exponent. It was also decided to deal with only the integral portion of a number by ignoring any digits to the right of the decimal point. Furthermore, the number 500 represents a displacement (either on the abscissa or the ordinate) of one inch from the origin. Since the origin can be manually positioned anywhere on the plotting field, and since the paper is 11 inches by 17 inches, a maximum of ±8500 for the X Register and ±5500 for the Y Register must be displayable in fixed point. Now both of these numbers have four digits to the left of the decimal point. If a maximum of ten digits can be displayed, then only six of those can be to the right of the decimal point. Hence, the Decimal Wheel must be set to six or less.

TRIGGERING

The Plotter is triggered by an IOPF followed by a proper Format operation. The Format operation will automatically be treated by the 9100 as a two step Format operation because the Plotter grounds YAFZ. After the Format operation the 9100 returns to the Display, so that the new point can be outputted.

USE OF YINH

During the Display that accompanies the plot cycle the Plotter grounds YINH, provided that the Format operation came from a program. The reason that the Plotter does not ground YINH if the Format operation originated as an errand is that grounding and releasing YINH while the last errand was on the Key Lines (assuming the key is still down) causes another execution of that key-code. For instance, the keystrokes FORMAT ↓ would cause a normal plot followed by an execution of the ↓ key, if the ↓ key were held down until after the plot was over.

TRANSFER OF THE NUMERICAL DATA

While the Plotter is actually plotting it forces fixed point Display by grounding YFLT. YFLT remains grounded until the plot is over. Once triggered the Plotter looks at the Buffered Words to recognize the T Register (indicating that the 9100 is in the Sync State). That signals the Plotter that the X Register is next. When the Buffered Words change the Plotter begins counting ITBB pairs until the code for the decimal point is encountered on the Buffered Bits. (Each digit except the decimal point is accompanied by two ITBB's, see Appendix I). It is in this way that the Plotter determines if the Decimal Wheel is set to six or less. If the Decimal Wheel is set to more than six, or if the number in the Register is being displayed in floating point (as it would be if the number were too big) the count will exceed six and the Plotter lights the Improper Data Format Light and the plot cycle is aborted.

–hp– 9125A PLOTTER

TRANSFER OF THE NUMERICAL DATA CONTINUED

If the data format is proper, the characters after the decimal point (including the sign) are entered. The BCD codes for the digits are entered "on the fly" and neither YCHA nor YCOD are used in the entering process. When the Buffered Words indicate the Y Register the exact same process is used to check and enter the Y Register.

> **NOTE**
>
> If NDGO is grounded during Plotter operation, the Plotter will not function correctly. This is because grounding NDGO prevents the 9100 from setting the Word flip-flops to the T Register from inside the Display Process.

THE ACTUAL PLOTTING

If the plotting cycle is not aborted the numbers entered are used to drive Digital to Analog Converters, which in turn cause a servo-mechanism to move the pen to its new position. At the time that process begins a Continue is given if the original Format operation came from a program. The Continue is entered with YKDN and lasts until IOPF is given. A similar caution about interfacing applies to the Plotter as for the Printer. See the section titled IMPORTANT REMARK on page 55 of this Appendix.

HOLD-OFF

The Plotter has a hold-off mechanism that prevents the Plotter from acting on succeeding Format operations (though it will recognize them) until the previous plot is finished. This is necessary because the 9100 can be faster than the Plotter and can be ready with the next point to be plotted before the last plot is finished.

STOP KEY

If the STOP key is pressed ISTP goes negative. The Plotter uses ISTP as a command to lift the pen and return it to the origin. Now ISTP is given any time the STOP key is activated, even when the 9100 is in Program Mode. Certainly no plotting is being done while the 9100 is in Program Mode and it would be undesirable if the pen were to return to the origin just because the STOP key is pressed while the Calculator is being programmed. Accordingly, ISTP will have no effect if YRUN is at ground. The Plotter does not manipulate YRUN, but YRUN is connected to the circuit controlled by the PROGRAM/RUN switch. If the switch is in the PROGRAM position, YRUN is at DC ground, and the Plotter will ignore ISTP.

REAR TERMINALS OF THE PLOTTER

The rear plug of the Plotter is pin by pin a duplicate of the I/O terminals of the 9100. However, it is recommended that any private interface be attached to the 9100 directly or through the -hp- 9102A Calculator Buffer, rather than through the Plotter. The reason for this is that many of the signals from the 9100 that are used by the Plotter are buffered by the Plotter, and the resulting signals are not the same as the originals. Not only are the signal levels and output impedances slightly different, but their timing is stretched. A 550 ns signal going into the Plotter can come out as a 4 μsec signal.

–hp– 9160A MARKED CARD READER

GENERAL

The -hp- 9160A Marked Card Reader allows the operator to enter a sequence of key-codes into the 9100 from an IBM card marked in pencil. The entries may be made in either Run or Program Mode. The PROGRAM/RUN switch on the 9100 determines the mode, as the 9160A does not deal with YRUN.

HOW IT WORKS

The 9160A is activated by inserting a card. Immediately the motor starts to run and NHLD is grounded. This locks the 9100 into the Sync State before the first key-code is read from the card.

The key-code read from the card is used to set flip-flops to hold the key-code and apply it to the Key Lines. After the Key Line flip-flops are set up YDSP and NDGO are grounded for 30 to 35 μsec. That enters the key-code into the Calculator.

If the skip channel is marked the associated key-code is still put into the Key Line flip-flops, but YDSP and NDGO are prevented from going to ground.

The strobe channel consists of a set of marks that are physically centered between consecutive data marks. As the card moves through the 9160A the leading edge of the strobe mark causes YDSP and NDGO to go to ground for 30 to 35 μsec. The trailing edge of the strobe mark resets the Key Line flip-flops. The next code on the card will be set up when the marks are read.

NHLD is released as the card leaves the 9160A. The connector on the 9160A is of a feed-thru type and all signals on the I/O terminals of the 9100 are still available, but the three lines used for the -hp- 9150A Display Monitor are on special jacks.

–hp– 9150A DISPLAY MONITOR

If a 9100 is modified to be used with an -hp- 9150A (or an -hp- 1300A) I/O terminals 19, 21, and Y carry the analog voltages to drive the external CRT.

An adaptor (supplied with the modification of the 9100, or on the 9160A) allows the 9150A to be attached to the modified 9100.

If the 9100 is unmodified, pins 19, 21 and Y are not connected to anything.

THE OTHER –hp– PERIPHERALS

The other standard -hp- peripherals at this writing include the -hp- 9101A Extended Memory, and the components for the -hp- 2570A Coupler System.

The 9101A and the 2570A each use almost every signal available on the rear of the 9100. Both of these instruments are extremely complicated, and no brief general description is possible.

It is no overstatement to say that a designer who intends his interface to be compatible with either of these peripherals must be exceedingly careful. The guiding principles would be to treat the signals as carefully as possible, and construct the logic in such a way that signals are used for only as long as necessary.

THE OCTAL CODE CONVENTION

Key-codes are two digit octal (base 8) numbers. The Calculator logic encodes each octal digit as a pattern of voltages across three terminals, or lines. Hence, it takes six lines to encode a key-code.

An octal digit is encoded on its three lines by putting the ordinary binary representation of the digit on the lines, with negative voltage representing the 1's and ground repesenting the 0's. See Figure 34.

0	000	5	101	
1	001	6	110	
2	010	7	111	
3	011	8	1000	not permissible
4	100	9	1001	as octal digits

Figure 34. Binary encodings.

The Binary Coded Octal (BCO) for the two digit octal number is formed by replacing each of the octal digits with their binary encodings.

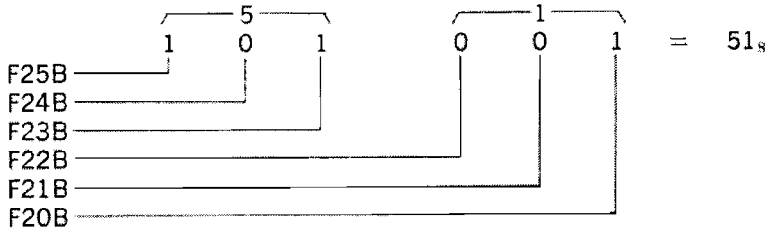


Figure 35. Binary coded octal on the Buffered Bits.

In every instance where a BCO code is used, whether for input or for output purposes, the left-most bit of the BCO code is called the most significant bit, while the right-most bit is called the least significant bit. The BCO code will always be across either the Buffered Bits, or the Key Lines. Each of these groups have numbers assigned to each of their lines, (i.e., the Buffered Bits are F25B through F20B). In each case the highest number is associated with the most significant bit, and the lowest number with the least significant bit. See Figure 35.

THE KEY-CODE CONVENTION

When inputting key-codes, the Key Lines are involved, and they work somewhat differently than the Buffered Bits. For a given Key Line to result in a 1 it is grounded, while to put in a 0 the Key Line is left alone. To put it another way, so that the negative - true logic still holds, the logical complement of the desired key-code is applied to the Key Lines, rather than the straight key-code (the complement of a 1 is a 0, and vice versa). See Figure 36.

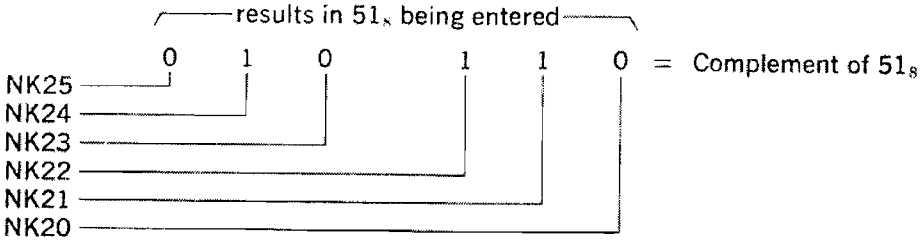


Figure 36. Binary coded octal on the Key Lines.

KEY-CODE EXECUTION TIMES

The tables of key-code execution times on the following pages were made by measuring the time between the IOPF and ICHF generated by the operation to be measured. The execution times of those operations that do not allow the 9100 to return to the Display immediately (such as Continue, or a failed qualifier) were measured by taking the time between the first and second IOPF's for two consecutive executions of the operation.

Except for Pause statements, whose execution times are power line frequency sensitive, the execution times of the 9100's operations are dependent upon the internal crystal oscillator that serves as a clock. The frequency tolerance of the crystal over the range of operating temperatures is ±.01%.

Table 5. 9100A Key-code execution times.

OPERATION	REMARKS	†	TIME
Any Digit	Previous operation makes a difference.		
	1. as digit entry into the X Register	3+	>145 μs*
	2. following x → ( ): for 0-7	1	173 μs
	for 8, 9	1	177 μs
	3. following y → ( ): for 0-7	1	330 μs
	for 8, 9	1	333 μs
	4. following y ⇄ ( ): for 0-7	1	434 μs
	for 8, 9	1	437 μs
Any Alpha	Previous operation makes a difference.		
	1. as recall into X Register	1	308 μs
	2. following x→( )	1	178 μs
	3. following y→( )	1	335 μs
	4. following y ⇄ ( )	1	439 μs
Enter Exponent	1. if X Register equals zero	3+	114 μs
	2. if X Register does not equal zero	3+	75 μs
Change Sign	1. for MSDΔ of number	3+	61 μs
	2. for exponent	3+	70 μs
Decimal Point	1. if MSDΔ of number is zero	3+	112 μs
	2. if MSDΔ of number is non-zero or for exponent entry	3+	55 μs
Format	YAFZ or running a program makes a difference.		
	1. YAFZ is true: for errands	1	68 μs
	for statements	1	68 μs
	2. YAFZ is false: for errands	2	N/A**
	for statements (Times are for complete Format operation)	2	>75 μs*
Continue	Starting program address makes a difference. While running a program, each operation requires a maximum of 110 μs extra due to time spent incrementing program address and fetching of key-code.	1	>150 μs*

(continued)

KEY-CODE EXECUTION TIMES

Table 5. 9100A Key-code execution times (continued).

OPERATION	REMARKS	†	TIME
Go To ( ) ( )	1. for statements (time is for entire sequence) 2. for errands	3+	>250 μs* N/A**
Any Qualifier	1. any met or failed qualifier except If Flag (time includes addressing sequence if qualifier is met) 2. If Flag met failed	1 1 1	>1.6 ms* >275 μs* >175 μs*
Pause	Nine frames of Display - power line frequency sensitive.	1	N/A
Step Program	1. while in Program Mode 2. while in Run Mode add a maximum of 110 μs (T_1) to the execution time (T_2) of the key-code about to be executed by the Step Program operation.	1 1	280 μs $T_1 + T_2$
Storing Program Steps	While Program Mode, for every operation except Step Program.	1	>265 μs*
Addition		1	>1.5 ms*
Subtraction		1	>1.5 ms*
Multiplication		1	>29 ms*
Division		1	>27 ms*
Square Root		1	>30 ms*
Accumulate +		1	>7 ms*
Accumulate −		1	>7 ms*
Recall		1	1.2 ms
x → ( )		1	55 μs
y → ( )		1	54 μs
y ⇄ ( )		1	56 μs
x ⇄ y		1	404 μs
↑		1	339 μs
↓		1	310 μs
Roll ↑		1	892 μs
Roll ↓		1	466 μs
|Y|		1	65 μs
Integer of X		1	>114 μs*

(continued)

KEY-CODE EXECUTION TIMES

Table 5. 9100A Key code execution times (continued).

OPERATION	REMARKS	†	TIME
Clear X		3+	112 μs
Clear		3+	2.6 ms
End		1	69 μs
Stop		1	57 μs
Print		2	73 μs
Pi		1	156 μs
Set Flag		1	54 μs
Arc		1	61 μs
Hyper		1	61 μs
Tan x		1	>350 ms*
Tan^{-1} x		1	>175 ms*
Tanh x		1	>200 ms*
$Tanh^{-1}$ x		1	>150 ms*
Sin x		1	>350 ms*
Sin^{-1} x		1	>300 ms*
Sinh x		1	>175 ms*
$Sinh^{-1}$ x		1	>170 ms*
Cos x	Same as for the Sin x in each case.		
e^x		1	>150 ms*
ln x		1	>100 ms*
log x		1	>100 ms*
To Polar		1	>250 ms*
To Rectangular		1	>375 ms*

† Indicates Print-After-Printer classification of operation.

* Time varies depending upon exact circumstances. For functions, execution time is not necessarily linearly related to size of the arguement, maximum time is shown.

** Time depends on rate of operator's entries.

Δ MSD - most significant digit.

For program execution times see Continue.

KEY-CODE EXECUTION TIMES

Table 6. 9100B Key-code execution times.

OPERATION	REMARKS	†	TIME
Any Digit	Previous operation makes a difference.		
	1. as digit entry into the X Register	3+	>130 μs*
	2. following x →( )	1	173 μs
	3. following y →( )	1	330 μs
	4. following y ⇄ ( )	1	434 μs
	5. following x ←( )	1	311 μs
	6. first character following Go To ( ) ( )	3+	50 μs
	7. second character following Go To ( ) ( )	3+	65 μs
	8. first character following Go To Sub	3+	50 μs
	9. second character following Go To Sub	3+	198 μs
Any Alpha	Previous operation makes a difference.		
	1. as recall into X Register	1	312 μs
	2. following x →( )	1	175 μs
	3. following y →( )	1	331 μs
	4. following y ⇄ ( )	1	437 μs
	5. following x ←( )	1	313 μs
	6. first character following Go To ( ) ( )	3+	52 μs
	7. second character following Go To ( ) ( )	3+	67 μs
	8. first character following Go To Sub	3+	52 μs
	9. second character following Go To Sub	3+	200 μs
Enter Exponent	1. if X Register equals zero	3+	102 μs
	2. if X Register does not equal zero	3+	63 μs
Change Sign	1. for MSDΔ of number	3+	52 μs
	2. for exponent	3+	54 μs
Decimal Point	1. if MSDΔ of number is zero	3+	100 μs
	2. if MSDΔ of number is non-zero or for exponent entry	3+	43 μs
Format	Running a program makes a difference.		
	1. for errands	2	N/A**
	2. for statements (Times are for complete Format operation)	2	>75 μs*
Continue	Starting program address makes a difference. While running a program each operation requires a maximum of 110 μs extra due to time spent incrementing program address and fetching of key-code.	1	>110 μs*

KEY-CODE EXECUTION TIMES

Table 6. 9100B Key-code execution times (continued).

OPERATION	REMARKS	†	TIME
Go To ( ) ( )		3+	46 μs
Any Qualifier	1. any met or failed qualifier except If Flag (time includes addressing sequence if qualifier is met) 2. If Flag met failed	1 1 1	1.6 ms* >275 μs* >175 μs*
Pause	Nine frames of Display - power line frequency sensitive.	1	N/A
Step Program	1. while in Program Mode 2. while in Run Mode add a maximum of 110 μs (T_1) to the execution time (T_2) of the key-code about to be executed by the Step Program operation.	1 1	>195 μs* $T_1 + T_2$
Storing Program Steps	While in Program Mode, for every operation except Step Program.	1	>195 μs*
Plus	1. as addition 2. as part of an address	1 3+	>1.5 ms* >44 μs*
Minus	1. as subtraction 2. as part of an address	1 3+	>1.5 ms* >46 μs*
Sub/ Return	1. as subroutine call 2. as return statement	1 1	49 μs >300 μs*
Multiplication		1	>29 ms*
Division		1	>27 ms*
Square Root		1	>30 ms*
Accumulate +		1	>7 ms*
Accumulate −		1	>7 ms*
Recall		1	1.2 ms
x→( )		1	44 μs
y→( )		1	43 μs
y⇄( )		1	46 μs
x←( )		1	48 μs
x⇄y		1	399 μs
↑		1	328 μs
↓		1	300 μs

(continued)

KEY-CODE EXECUTION TIMES

Table 6. 9100B Key-code execution times (continued).

OPERATION	REMARKS	†	TIME
Roll ↑		1	881 μs
Roll ↓		1	456 μs
|Y|		1	58 μs
Integer of x		1	>134 μs*
Clear x		3+	102 μs
Clear		3+	2.6 ms
End		1	53 μs
Stop		1	46 μs
Print		1	58 μs
Pi		1	141 μs
Set Flag		1	46 μs
Arc		1	47 μs
Hyper		1	48 μs
Tan x		1	>350 ms*
$\tan^{-1}$ x		1	>175 ms*
Tanh x		1	>200 ms*
$\tanh^{-1}$ x		1	>150 ms*
Sin x		1	>350 ms*
$\sin^{-1}$ x		1	>300 ms*
Sinh x		1	>175 ms*
$\sinh^{-1}$ x		1	>170 ms*
Cos x	Same as for Sin x in each case.		
e^x		1	>150 ms*
ln x		1	>100 ms*
log x		1	>100 ms*
To Polar		1	>250 ms*
To Rectangular		1	>375 ms*

† Indicates Print-After-Printer classification of operation.

* Time varies depending upon exact circumstances. For functions, execution time is not necessarily linearly related to size of the argument, maximum time is shown.

** Time depends on rate of operator's entries.

Δ MSD - most significant digit.

For program execution times see Continue.

CORE MEMORY CONVENTIONS

Core memory can be used to hold two kinds of information: numerical data and program steps. Each kind of information is stored and handled differently.

ADDRESSING

An address in the core memory is determined by specifying a character location and a register (word). There are two conventions for the naming of character locations. Each address is six bits "deep". See Figure 37.

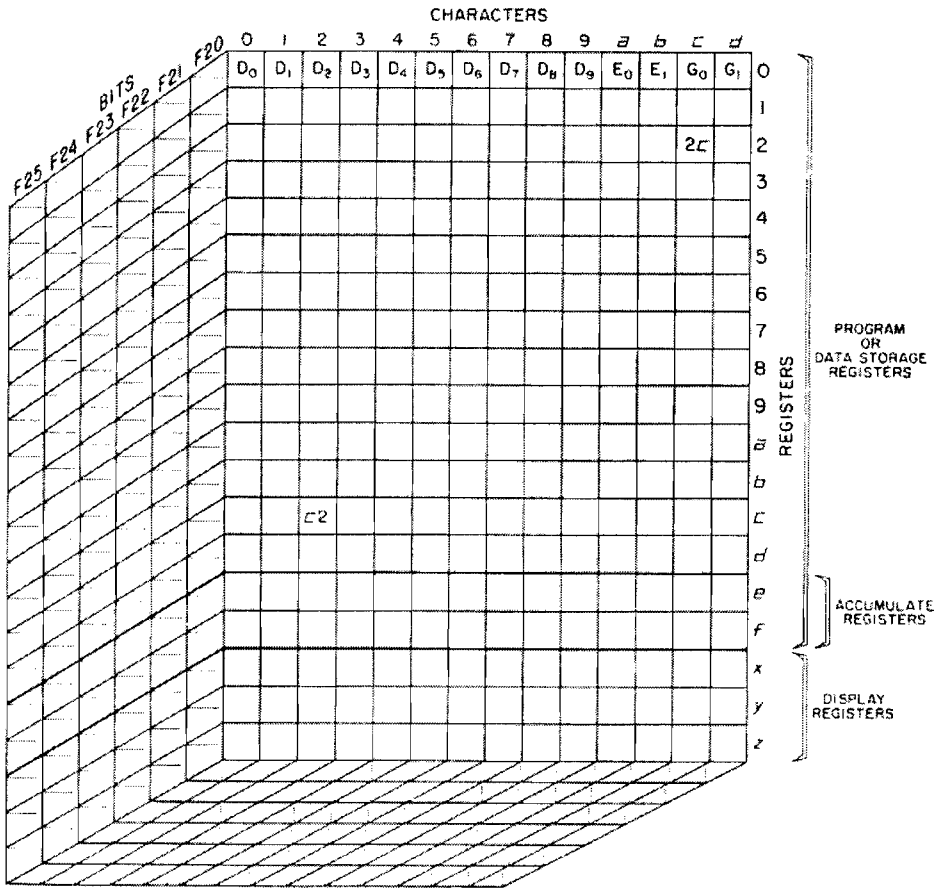


Figure 37. Core memory.

STORAGE OF PROGRAM STEPS

If a section of memory is being used to store program steps, a very definite order links the character and word locations together to form a sequence of addresses. It is that same order that is observed in the Program Mode Display when Step Program is pressed: 00 to dd. In such an address the first symbol defines the word and the second symbol defines the character location. Each address used in this fashion uses all six bits to encode the key-code being stored. Ordinarily a register containing program information is not purposefully displayed.

CORE MEMORY CONVENTIONS

STORAGE OF NUMERICAL DATA

If a register is to store numerical data or a Program Mode Display (generally for the X and P Registers only, and it's not to be confused with storing a segment of program steps) an altogether different scheme is used, and it is closely related to the properties of the Display Process.

With the exception of the Program Mode Display, all data that can be meaningfully displayed is numerical in nature. Numerical data is stored as a floating point number, no matter how it was entered. Fixed point Display merely changes the manner of presentation in the Display, not the way it is stored in core memory.

The digit locations in the floating point number have names as shown in Figure 38. Note the two Guard Digits, G_1 and G_0. They hold the number up to 12 significant digits, even though the Display Process only presents it to 10.

$$\pm D_9 \cdot D_8\ D_7\ D_6\ D_5\ D_4\ D_3\ D_2\ D_1\ D_0\ G_1\ G_0 \pm E_1\ E_0$$

Figure 38. Core locations of a data register.

THE SET OF DISPLAYABLE SYMBOLS

Each address in a register used to store numerical data will contain one of only 15 basic symbols. They are 0-9, a-d, -. They may be positive or negative while being blank or non-blank. Note that the decimal point is not in the collection. Why this is will be explained shortly.

Encoding does not take all of the six bits at any address containing one of these symbols unless it is negative, blank, or both. In fact, the least four significant bits (F23 - F20) allow 16 possible encodings, of which only 15 (for 0-9, a-d, -) will occur. The other encoding is used by the Display Process when it synthesizes the decimal point. It is never stored in core memory, however.

THE DECIMAL POINT

Because of the floating point format of numerical data, no decimal point is stored in core memory. It is always assumed to be in between D_8 and D_9, and is inserted into the Display in the proper place by the Display Process.

THE MINUS SIGN

A minus sign in numerical data is not stored all by itself in an address.* Instead, the sign of the number is attached to D_9, and the sign of the exponent is attached to E_1. The fifth bit (F24) of an address will contain a 1 if the digit stored in the least four significant bits is negative. Any digit in the register can be negative with this scheme and this happens. See Page 30. The only time when bit 24 matters is when forming the sign of D_9 or E_1. Granted, the signs on numbers are formed as separate characters in the Display, but they are not stored that way.

BLANKS

In a similar manner any address in a data register can represent a blank by having a 1 in the sixth bit (F25). The remaining five bits can contain any legitimate combination. Thus a blank in core memory is always a blank "something".

*The minus sign can be stored as a separate symbol in a Program Mode Display. The interesting case of a plus or minus minus sign does not occur because the BCD code for the minus sign is never stored in either D_9 or E_1.

CORE MEMORY CONVENTIONS

THE ORDERING OF ADDRESSES

The difference in the way the two kinds of information, program steps and numerical data, are handled has to do with the order in which the Calculator takes the addresses in a register. The natural right to left ordering of the addresses of a floating point number (going from E_0 to D_0) does not agree with the ordering for program step addresses. See Figure 37.

This explains the strange appearance of a register containing numerical data when it is inspected using Step Program and the Program Mode Display. Not only is the ordering of the digits somewhat scrambled, but the significance of bits 24 and 25 is no longer that of the numerical data convention, but that of the program step convention, i.e., they are part of a key-code — one that was never there.

If the user were to enter or remove numerical data by the YSKIC or YGNPS mechanisms he would have to observe not only the re-ordering of the digits, but also the way the signs are stored on D_0 and E_1.

ASSIGNMENT OF FORMAT CODES

Table 7 shows which codes have already been allocated for use with an -hp- peripheral. There is no reason why any of those codes can't be used for a private interface, except that such an interface will automatically be incompatible with an -hp- peripheral.

Five codes in Table 7 are not recommended. Format Step Program is not usable because it can't be stored as part of a program without going to an extreme amount of trouble. Format Sub/Return can be used if the interface is only expected to work with a 9100B. The Sub/Return key-code is 77_8, which is not available on the 9100A Keyboard. However, it is possible to overcome that by entering the 77_8 from the I/O terminals.

Table 7. Assignment of key-codes for Format operations.

KEY	KEY-CODE	USAGE
↓	25	9125A PLOTTER
↑	27	
−	34	9101A EXTENDED MEMORY
÷	35	
CLEAR x	37	
×	36	
+	33	
$y\rightarrow(\ )$	40	
GO TO ( )( )	44	
SET FLAG	54	
FMT	42	
END	46	
π	56	
1	01	2570A COUPLER
2	02	
3	03	
4	04	
5	05	
6	06	
7	07	
8	10	
9	11	
a	13	
b	14	
c	16	
d	17	
e	12	
f	15	
ENTER EXP	26	MCA**
ACC +	60	RESERVED
RCL	61	
ACC −	63	

KEY	KEY-CODE	USAGE
TO POLAR	62	RESERVED
int x	64	
ln x	65	
hyper ▾	67 *	
TO RECT	66	
e^x	74	
log x	75	
$\sqrt{x}$	76	
sin x	70	
tan x	71	
cos x	73	
arc ▾	72	
CLEAR	20	NOT ASSIGNED
$x\rightarrow(\ )$	23	
ROLL ↑	22	
$y\rightleftarrows(\ )$	24	
$x\rightleftarrows y$	30	
ROLL ↓	31	
CHG SIGN	32	
IF FLAG	43	
PRINT/SPACE	45	
CONTINUE	47	
$\lvert y\rvert$	55	
IF $x=y$	50	
IF $x>y$	53	
IF $x<y$	52	
•	21	
STOP	41	NOT RECOMMENDED
PAUSE	57	
0	00	
▲SUB▼/RETURN	77	
STEP PRGM	51	

** 5400A and 5401A Multichannel Analyzers * $x\leftarrow(\ )$ has the same key-code as hyper

ASSIGNMENT OF FORMAT CODES

Format Stop and Format Pause are definitely not usable because YINH does not inhibit the STOP and PAUSE keys. This means that if either of these Format operations were performed from the keyboard, the Stop or Pause key-code would most likely still be on the Key Lines while the interface was trying to use them. In addition, it is often desirable to use ISTP as a command to reset the interface, which would preclude the use of Format Stop.

The last combination that is not recommended is Format 0. The reason is not due to any difficulty in forming Format 0, and if the interface is only to work with a 9100B it can be used without problems. However, recall that a Print is followed by an ITBB with key-code 00_8 on the Buffered Bits. Furthermore, in the 9100A the execution of a Print results in a glitch on HFMT. The glitch and the ITBB combine to simulate a Format 0, in addition to the HOUT. Thus a Print could trigger an interface that was sensitive to Format 0.

Format x ← ( ) does not exist as a separate Format operation since x ← ( ) has the same key-code as Hyper (67_8).

THE PRINT-AFTER-PRINTER

NATURE OF A PRINT - AFTER - PRINTER

The idea behind the Print-After-Printer is that a printout of the Display would be desired after processing some operations, but not after others. For instance, it would not be desirable to print after any digit that was being used for digit entry into the X Register, because that digit might not be the last entry. After the last entry a print can be obtained by doing an operation like Stop, or Print. After most operations a printout is desired, and Tables 5 and 6 show the Print-After-Printer category for 9100 operations performed in Run Mode. In Program Mode all operations would be followed by a printout.

Each 9100 operation fits into one of three categories, depending upon the operation and the position of the PROGRAM/RUN switch. The category is determined by following certain counting rules involving IOPF, IADV and ITBB, as the operation is processed.

PULSE COUNTING

The rule for determining the classification of an operation as it is processed is: after the IOPF for that operation there will be one or more ITBB's followed by a group of consecutive IADV's, to be counted, the count to be terminated either when it gets to three or more, or when another ITBB appears. The number of IADV's in the count determines the classification. One means print after the operation is over, and three or more mean do not print. Print and Format have two IADV's, so that these operations can be treated separately. See Figure 39.

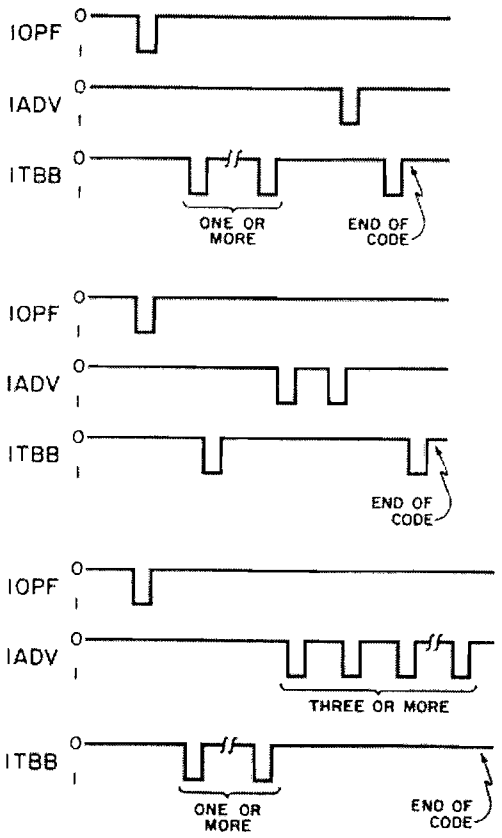


Figure 39. Pulse counting with IADV and ITBB.

A COMMENT ON NHLD

If NHLD is grounded the 9100 will wait in the Sync State until either NHLD is released and a sync pulse occurs, or until NDGO is grounded.

A PROBLEM

That rule holds with the following exception. If the 9100 is in the Sync State rapid grounding of NHLD can act like a sync pulse, releasing the 9100. See Figure 18. This could cause problems if, for instance, YDSP and NHLD were grounded together, with the idea being to process some key-code as soon as it is set up, by pulsing NDGO to ground. It could allow an unintentional key-code (whatever is on the Key Lines when the New Errand Qualifier is asked) to be processed, because the 9100 was prematurely released from the Sync State while YDSP was at ground.

A FIX

One cure for this situation is to always ground NHLD at least 15 msec before grounding YDSP. The reason for such a long delay is that once the 9100 gets back into the Display NHLD can't possibly take effect until the Word Flip-Flops indicate the T Register while the 9100 is in the Sync State, which won't happen until the end of the frame. But YDSP can take effect at the end of any register. The longest delay is determined by the time it takes to display three floating point registers.

THE WORST CASE

This problem is at its worst in 9100A interfacing situations where a Format operation from a program is recognized and immediate grounding of NHLD is desired. There is a time constant of about 70 μsec associated with NHLD. Within the 70 μsec after NHLD is grounded one cannot guarantee that NHLD will take effect. It just so happens that the 9100A takes about 70 μsec after HFMT is given to reach the Sync State. In the 9100B the setting up of the P Register provides an extra 80 to 100 μsec before the Sync State is reached, making it possible to successfully ground NHLD after recognizing a Format operation.

ANOTHER FIX

A method ·hp· has used to alleviate this situation is to ground NHLD, NDGO, and YDSP with key-code 41, (Stop) on the Key Lines as soon as a Format operation is recognized. Then, as soon as the IOPF for the Stop errand is given, unground NDGO. The Stop errand supplies extra time to make NHLD effective, but without wasting 15 msec.

For Keyboard or remote entries there is even less time between the ITBB that reveals the key-code after the HFMT, and the Sync State. However, the problem can be easily solved by grounding NHLD **every time** an HFMT is given, and returning it from ground if the key-code following the HFMT is the wrong key-code. This works for every situation except for 9100A's when YAFZ is not grounded (because there is no waiting period for the next errand after the HFMT). However, interfacing to a 9100A with YAFZ ungrounded is definitely not recommended.

A COMMENT ON YINH AND YKDN

RECYCLING YKDN IS IMPORTANT

In the 9100 the Step Program flag in the Program Storage Process is also used as a flag in the escapement mechanism for YKDN. Trouble can arise if YKDN is still negative from some peripheral (say, the -hp- 9120A Printer), or a key on the Keyboard is still down, while a Format operation is executed to trigger another peripheral into entering a program into the Calculator. In such a case the program source must wait either for YKDN to go away and also to recycle, or, it must ground YINH and wait for recycling. Recycling requires 6-8 msec followed by one pass through the Sync State.

Failure to do this can mean that the Step Program flag in the Program Storage Process will be a 1 on entry, preventing the program steps from being stored.

WHEN TO GROUND YINH

The only time YINH should be grounded by an external device triggered by a Format (or Print) operation is when the trigger operation originates from a program, **AND** the external device enters at least one errand (say, Continue). A Stop errand could be used if no other errand would normally be used. This rule has to be observed to prevent an unwanted execution of a key still down on the Keyboard. The remote errand sets to Keyboard lock-out flag (YKDN flag in Figures 10 and 11) so that when YINH is released the flag is set, preventing the execution of an activated key. This is needed since grounding YINH could result in clearing the Keyboard lock-out flag if the Calculator were allowed to display during the external operation.

If the trigger operation is an errand YINH should not be grounded for much the same reason. See the fourth paragraph on Page 57.

–hp– CONNECTORS

Figure 40 shows an exploded view of two connectors and a cable that may be of use in the design of a private interface. One connector is a feed-thru type, while the other is not. However, the feed-thru type does not feed-thru on pins 19, 21 and Y. Those pins carry the ·hp· 9150A analog Display waveforms, which are found on the three jacks instead.

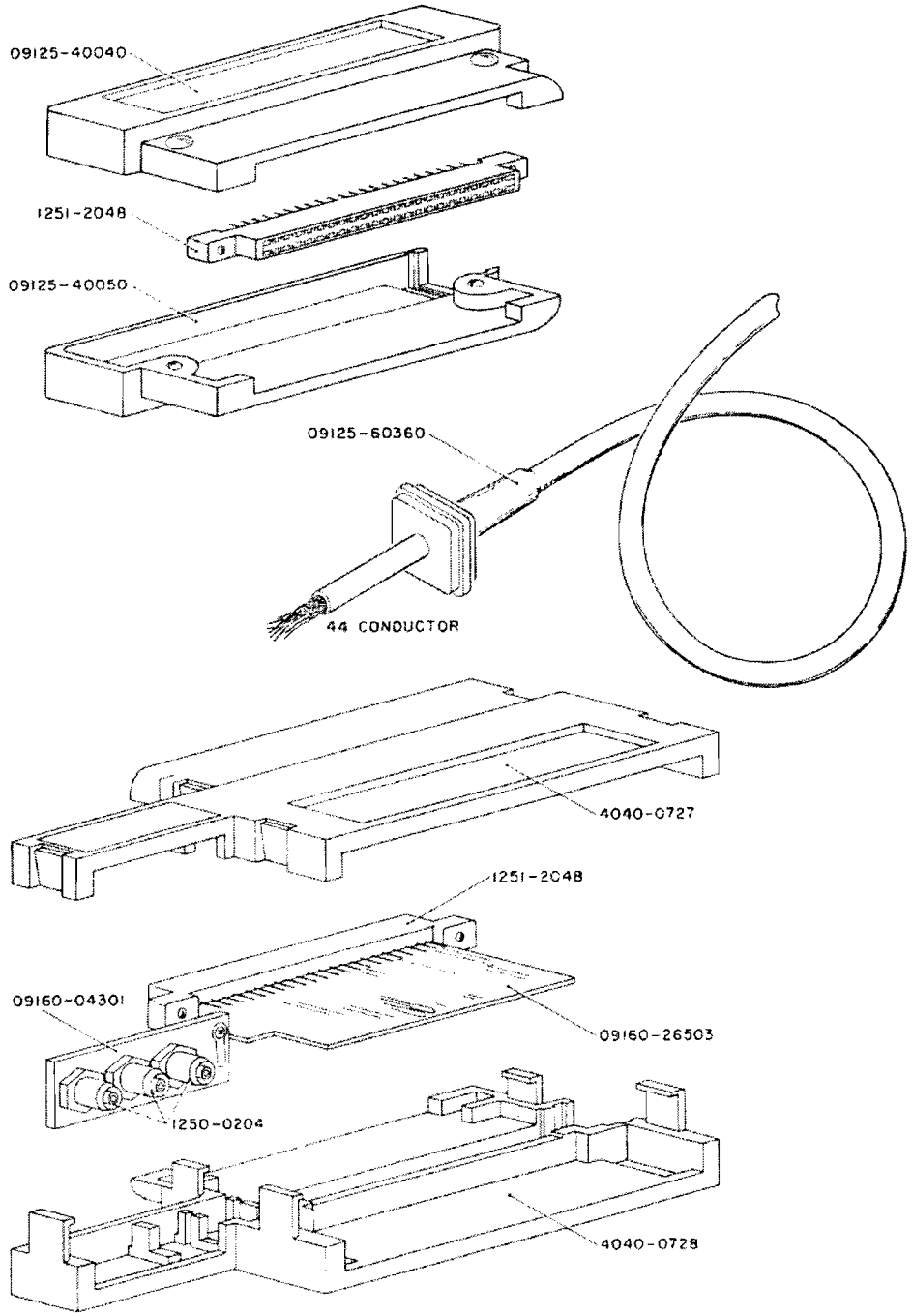


Figure 40. ·hp· connectors.