9010A

Micro-System Troubleshooter

Programming Manual

P/N 609289
June 1981

FLUKE®

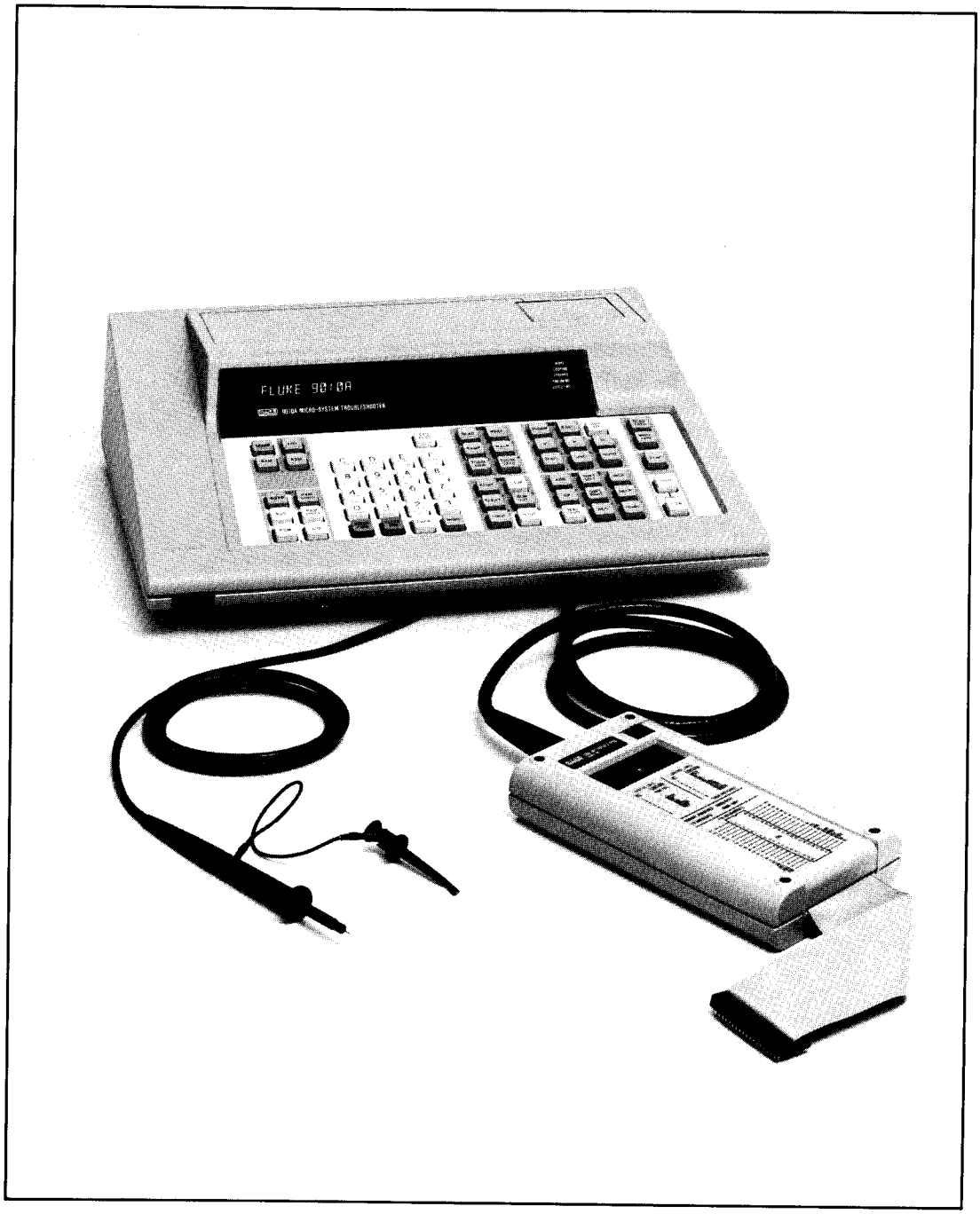
Table of Contents

(continued on page ii)

TABLE OF CONTENTS, *continued*

List of Tables

List of Illustrations



9010A Micro-System Troubleshooter

Section 1
Introduction

1-1. PROGRAMMING OVERVIEW

It is easy to learn 9010A programming. Experience or knowledge of complicated programming languages is not required. If the programmer understands how to use the 9010A in the Immediate Mode (as described in the 9010A Operating Manual), then the programmer already knows most of what is required to program the 9010A.

Programs are sequences of 9010A tests, functions, and operations. Any of the built-in tests or troubleshooting functions may be included in programs, as well as the Learn, Read Probe, and Arithmetic operations. In addition, test sequencing keys are available to help direct the flow of the programs, and allow the construction of conditional and unconditional branches. Specific 9010A features related to programming include the following:

- An internal memory of 10K bytes that can store up to 100 programs and up to approximately 1000 total program steps.
- Up to 16 labels for each program.
- Test sequencing keys for the construction of conditional and unconditional branch steps.
- A display step which allows the programmer to create operator messages.
- Programmable output to and input from the operator during program execution.
- Ability for one program to execute another program in a subroutine-like fashion.
- Programmed implementation of the Learn and Probe operations, the functional tests, and the troubleshooting functions.
- Full use of the sixteen 32-bit internal registers for storage and manipulation of data during program execution.
- Full use of the Arithmetic operations.
- Programmed implementation of the Run UUT Mode.
- Nonvolatile storage of programs on cassette.
- Transfer of programs from the 9010A to another 9010A or a remote device via the optional AUX I/F RS-232 Interface.
- Programmable output to and input from the optional AUX I/F RS-232 Interface.

1-2. ORGANIZATION OF THE 9010A PROGRAMMING MANUAL

The 9010A Programming Manual is written with the assumption that the reader is already familiar with 9010A Immediate Mode operation. If this is not the case, it may be necessary for the reader to refer to the 9010A Operator Manual.

The 9010A Programming Manual is divided into seven sections:

1 INTRODUCTION
2 GETTING STARTED
3 CREATING, EDITING, AND EXECUTING PROGRAMS
4 CREATING PROGRAM STEPS
5 PROGRAMMING EXAMPLES AND TECHNIQUES
6 ERROR HANDLING IN THE EXECUTING MODE
7 DATA FORMAT FOR AUX I/F IMMEDIATE MODE OPERATION

Section 1 provides an overview of programming and the 9010A Programming Manual. Section 2 consists of a sample program that may be entered and executed to simply get started. Section 3 explains how to initially create, enter, edit, exit, and execute a program. Section 4 is a definition and reference section which defines all possible program steps that may be created and the keys that are used to create the steps. Section 5, the heart of the manual, is organized according to topic, and explains how to effectively use the programming steps defined in Section 4. Section 5 includes numerous programming examples. Section 6 describes error handling. Section 7 provides a detailed explanation of the data format for the AUX I/F Immediate Mode operation.

Much of the programming may be learned without having an interface pod or a UUT connected to the 9010A main instrument. This is because a considerable number of the programming features (such as the creation of display messages) do not require the 9010A to communicate with an interface pod or UUT. Wherever possible, the example program steps given in this manual are written so that they may be used in a program and executed without having an interface pod or a UUT connected to the 9010A.

The following definitions are provided as a reminder of the differences in 9010A operation in the three operating modes. For more information, refer to the 9010A Operator Manual.

IMMEDIATE MODE	9010A actions are performed as soon as they are selected and the specification is complete.
PROGRAMMING MODE	When 9010A actions are selected and specified, they are not performed, but are stored as program steps.
EXECUTING MODE	9010A actions are performed as specified by the steps in the program that is being executed.

1-3. THE 9010A INSTRUCTION MANUAL SET

The 9010A Programming Manual is part of a set of manuals that document the 9010A Micro-System Troubleshooter. The manual set also includes the Operator Manual, the Service Manual, and a Reference Guide. For reference, all the manuals in the set are described as follows:

OPERATOR MANUAL	Instrument description and specifications, operating instructions, probe use, execution of programs, options and accessories, and routine maintenance.

PROGRAMMING MANUAL	Description of instrument programming capabilities, writing, editing, and execution of programs. Little or no previous programming experience required.
SERVICE MANUAL	Specifications, theory of operation, troubleshooting, repair and maintenance information, a list of replaceable parts, and schematics. Intended for use by a qualified technician.
REFERENCE GUIDE	Quick-reference operating and programming information.

In addition, an Interface Pod Manual is available for each interface pod, and provides the following information:

INTERFACE POD MANUAL	Specifications, theory of operation, maintenance, a list of replaceable parts, and schematics.

Section 2
Getting Started

2-1. INTRODUCTION

This section provides a sample program which may be entered and executed to get a feeling for the ease and simplicity of 9010A programming. Note that it is not necessary for the 9010A to be connected to a unit under test (UUT) or an interface pod to execute the program.

2-2. SAMPLE PROGRAM

At this point it is not necessary to understand how to create or execute programs. Simply follow the instructions listed below.

1. Turn off the 9010A power (if on) and turn on again. This clears the memory of any previous programs. The 9010A displays the following power-on message:

FLUKE 9000 POWER-UP OK VER-nn

2. Press the PROGM key, the 1 key, and then the ENTER key. The PROGMING annunciator begins flashing and the 9010A displays the following message:

PROGRAM 1 CREATED

3. Press the keys in the order listed below to enter the program steps listed. The keys are separated by commas for ease in reading. If a mistake is made when entering a step, press the CLEAR key until the STEP CLEARED message is displayed, and then begin reentering the step.

PRESS	DISPLAY
REG, 1, 2, 0, ENTER	*REG1 = 20*
LABEL, 1	*LABEL 1*
DISPL, RUN UUT, 1, ENTER	*DPY-@1*
REG, 2, 1, 0, ENTER	*REG2 = 10*
LABEL, 2	*LABEL 2*
DECR, 2	*DEC REG2*
IF, REG, 2, >, 0, GOTO, 2	*IF REG2 > 0 GOTO 2*
DECR, 1	*DEC REG1*
IF, REG, 1, >, 0, GOTO, 1	*IF REG1 > 0 GOTO 1*
DISPL, D, >, IF, E, ROM VIEW, ENTER	*DPY-DONE* ◭

4. Press the PROGM key, which closes the program (turning off the PROGMING annunciator) and causes the 9010A to enter the Immediate Mode and display the following message:

PROG 1 CLOSED-10141 BYTES LEFT

To execute the program, press the EXEC key, the 1 key, and then the ENTER key. As the program is being executed, the display shows a countdown beginning with 32 and ending with 1. The final message on the display is the word DONE, and it is accompanied by the audible beep. Note that the EXECUTING annunciator flashes while the program is being executed and stops flashing after the execution is complete. More sample programs are presented in Section 5.

Section 3
Creating, Editing, and Executing Programs

3-1. INTRODUCTION
This section describes how to enter, edit, exit, and execute programs. It also includes information about program numbering and program deletion.

3-2. PROGRAM NUMBERING
The 9010A can store up to 100 programs, which are stored in the tape-transferable memory. Each program is assigned a number by the programmer when it is first created. Program numbers are displayed in decimal and may range from 0 to 99.

The 9010A also stores a list of the program numbers for existing programs. To examine the list of programs that are stored in the 9010A, press the PROGM key and then the = (equals) key. If there are any programs stored in the 9010A, the 9010A displays the following message:

PROGRAMS dd dd dd dd dd dd etc.

The letters *dd* represent the program numbers for existing programs. The numbers are listed in numeric order, beginning with the smallest number. Up to eight program numbers may be listed on the display. If the list is too long to fit on one line, the MORE annunciator flashes to indicate that more information is available. Subsequent numbers in the list may be brought to the display with the MORE and PRIOR keys.

If no programs are stored in the 9010A, when the PROGM key and the = keys are pressed the following message is displayed:

NO PROGRAMS DEFINED

3-3. CREATING A NEW PROGRAM
To create a new program, do the following:

1. Press the PROGM key. The 9010A displays a prompt for the program number by displaying the following message:

PROGRAM _

2. Select a number for a program that does not already exist in 9010A memory. Key in the number in decimal and press the ENTER key. The 9010A displays the following message:

PROGRAM dd CREATED

Notice at this point that the PROGMING annunciator begins flashing, indicating that the 9010A is no longer in the Immediate Mode, but is in the Programming Mode. Now the programmer may create the desired program steps.Program steps are stored in the program in the order in which they are created, with one line for each step. For example, if the programmer presses the BUS TEST key, the 9010A stores the Bus Test as a single program step as follows:

BUS TEST

If the Read function is specified at an address *aaaa,* the Read function is stored as a single program step as follows:

READ @ aaaa

The creation of program steps is described in detail in Section 4.

3-4. OPENING AND EDITING AN EXISTING PROGRAM

An existing program is opened in the same way a new program is created, by pressing the PROGM key and then entering the program number. The message that appears on the display is slightly different, however, and is as follows:

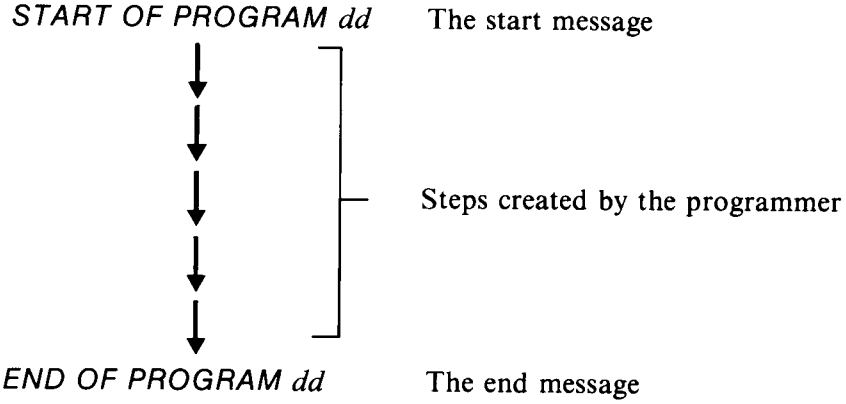
PROGRAM dd OPENED - nn BYTES

Notice that the display now includes additional information. The letters *nn* represent a decimal number which tells how many bytes are presently needed to store the program.

The 9010A provides a default value for the program number if the programmer presses the PROGM key and then the ENTER key without specifying the program number. The 9010A opens the last program that was opened. If the last program opened has since been deleted, the 9010A creates it. Note that if no programs have been created after power-on, the default program number supplied by the 9010A for the first program created is zero.

3-5. Moving Around in the Program

When a program is opened, the program steps may be brought to the display one at a time. The 9010A displays a start message and an end message to indicate the start and end of the program as follows:



The start message and end message are only for reference when examining the program as it is stored in the 9010A. The start message and end message are not part of the program listing that is transferred via AUX I/F, and do not appear when a program is printed out or viewed on a video display.

There are two ways to move around in the program and examine the program steps. One way is to move forward or backward step-by-step with the MORE and PRIOR keys.

The other way of moving around in the program requires an understanding of labels. Labels are simply program steps that are inserted into a program to provide points of entry for branching steps. Each program may contain up to 16 distinct labels (numbered 0 through 9, A through F). Labels are explained in detail in Section 4. At this point it is not important to understand everything about labels. It is only important to know that if a label exists in a program, pressing the hexadecimal digit key for that label causes the program step containing the label to appear on the display. This provides a convenient way of moving around in the program.

If the number zero is not used as a label, pressing the 0 key causes the start message to appear on the display. Similarily, if the number F is not used as a label, pressing the F key causes the end message to appear on the display. Table 3-1 shows how the hexadecimal keys are used in scrolling.

Table 3-1. Moving Around in a Program

START OF PROGRAM 1	← Pressing the 0 key brings this message to the display.
REG1 = 20	
LABEL 1	← Pressing the 1 key brings this message to the display.
DPY-@ 1	
REG2 = 10	
LABEL 2	← Pressing the 2 key brings this message to the display.
DEC REG2	
IF REG2 > 0 GOTO 2	
DEC REG1	
IF REG1 > 0 GOTO 1	
DPY-DONE	
END OF PROGRAM 1	← Pressing the F key brings this message to the display.

Note that this program is the sample program from Section 2.

3-6. Deleting or Adding Program Steps

To delete a step that appears on the display, press the CLEAR key. The 9010A deletes the step and places the following message on the display:

STEP DELETED

A program step may be added anywhere in the program. When a program step is added to the middle of a program, the step is stored in the program after the step that appears on the display. Note that if a step is added when the end message (END OF PROGRAM *dd)* appears on the display, the step is stored before the end message.

3-7. CLOSING A PROGRAM

To close a program, press the PROGM key. The 9010A displays the following message:

PROG dd CLOSED-nnnnn BYTES LEFT

The letters *nnnnn* represent a decimal number which tells how much space is left in the 9010A memory. There are 10,192 bytes available in memory for program storage. Notice that when a program is closed, the PROGMING annunciator stops flashing, indicating the 9010A has entered the Immediate Mode.

A program cannot be closed if it contains duplicate labels (using the same label more than once in the program) or is missing a label (a label is missing if a branch step points to a label that is not present in the program). If the programmer attempts to close a program that contains duplicate labels, the following message appears on the display:

DUPLICATE LABEL h

If the programmer presses the MORE key, the step that appears on the display is the second occurrence of the label in the program.

If the programmer attempts to close a program that is missing a label, the following message appears on the display:

MISSING LABEL h

If the programmer presses the MORE key, the step that appears on the display is the first branch step that points to the missing label.

3-8. EXECUTING A PROGRAM

To execute a program, do the following:

1. First make sure the 9010A is in the Immediate Mode. Then press the EXEC key. The 9010A prompts for the program number by displaying the following message:

EXECUTE PROGRAM _

2. Key in the program number and press the ENTER key.

If the program does not exist in 9010A memory, the following message appears on the display:

EXECUTE PROGRAM dd - NOT FOUND

If the program exists in 9010A memory, the EXECUTING annunciator begins flashing and the program begins executing. The following message appears on the display:

EXECUTE PROGRAM dd

The 9010A provides a default value for the program number if the operator presses the EXEC key and then the ENTER key without specifying the program number. The 9010A attempts to execute the last program that was executed (note that this may not be the same as the last program opened). The default program number for program execution after power-on is program zero.

When the 9010A executes a program, the 9010A performs the actions specified for each step, one step at a time. For example, if a Read function is specified as a step, the 9010A performs the Read function as specified when it encounters the step during program execution.

In addition to the EXECUTE PROGRAM *dd* message, there are two other types of messages that may appear on the display: programmer-created display messages or error messages.

Programmer-created display messages may be strictly informative, telling the operator the results of tests or operations. Or the messages may require the operator to enter data or perform some action on the UUT. The programmer-created display messages are created with the DISPL key, and are described in Sections 4 and 5.

The other messages that may appear during program execution are error messages. Error handling and error messages that may appear during execution are described in Section 6.

After the program has completed execution (and the EXECUTING annunciator stops flashing), the last message displayed remains on the display until the operator initiates some other action.

During program execution, the operator may interrupt program execution by pressing the STOP key, causing the STOPPED annunciator to flash along with the EXECUTING annunciator. The operator may continue program execution by pressing the CONT key, or abort program execution by selecting any Immediate Mode operation such as the Bus Test. When program execution is aborted, the EXECUTING annunciator stops flashing, the 9010A enters the Immediate Mode, and the 9010A presents the display message associated with the Immediate Mode operation selected.

3-9. DELETING A PROGRAM

To delete a program, do the following:

1. Open the program by pressing the PROGM key and specifying the program number. If the program is already open, move to the START OF PROGRAM *dd* message.

2. Press the CLEAR key. The 9010A responds with the following message:

DELETE PROG dd - ARE YOU SURE?

If the programmer presses the NO key, the start message for the program appears on the display. If the programmer presses the YES key, the program is deleted and the following message appears on the display:

PROG dd DELETED-nnnnn BYTES LEFT

3-10. REPORTING INSUFFICIENT MEMORY

As the programmer adds steps to a program, the 9010A keeps track of the amount of memory that has been used and reports to the programmer when no more steps may be stored. For example, if there is insufficient memory to store a program step in a program, the following message appears when the programmer completes the specifications for the step:

INSUFFICIENT MEM TO STORE STEP

If a program step has been successfully stored in a program but there is not enough memory left to store the RPEAT or LOOP modifiers, the following message is displayed:

INSUFFICIENT MEM TO STORE KEY

If there is not enough memory to allow the programmer to create a new program, the following message is displayed:

INSUFFICIENT MEM TO CREATE PROG

If the 9010A memory is full, the programmer may store the contents on a cassette tape, clear the 9010A memory, and then continue writing programs.

Section 4
Creating Program Steps

4-1. INTRODUCTION

There are 29 keys that may be used to initiate program steps, with each key initiating a particular type of step. This section lists the keys that initiate steps and describes the resulting steps. The keys that provide supplementary functions are also listed and their functions described. The sixteen 32-bit registers are described, with emphasis on how their function in the Executing Mode differs from the Immediate Mode.

Note that this section is not an exhaustive discussion of programming techniques. Instead, it provides extended definitions of the functions relevant to programming, with short examples where appropriate. This section is organized according to the keys on the keyboard and provides a foundation for understanding Section 5. Section 5 is a discussion of programming techniques and is organized according to topic.

The 9010A keyboard is shown in Figure 4-1. The keys are grouped in three categories: (1) the 29 keys that initiate program steps, (2) the keys that do not initiate program steps but perform supplementary functions in the creation of program steps, and (3) the disallowed keys that are not used in the Programming Mode.

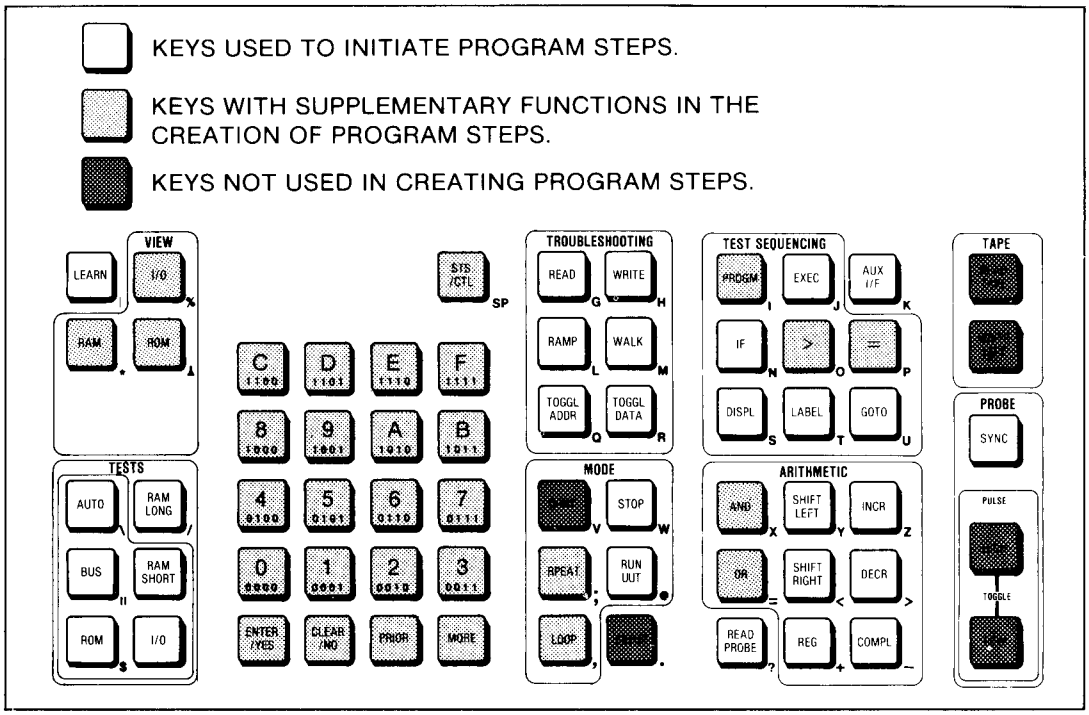


Figure 4-1. Function of 9010A Keys in The Creating of Program Steps

Most of the keys in the first two categories operate in the same way in the Programming Mode as in the Immediate Mode. These keys are listed in the following paragraphs, with important differences in Programming Mode behavior pointed out. The remainder of the keys in the first two categories have applications in the Programming Mode that are either considerably different from the Immediate Mode or do not exist in the Immediate Mode at all. The function of each of these other keys is described following the review of the Immediate Mode keys.

Note that there is a limit to the length of a program step. The maximum length ranges from 35 to 47 keystrokes, depending on the type of keystrokes used to create the step. If the limit is exceeded, all the keystrokes for the step are deleted and the following message is displayed:

STEP TOO LONG

The programmer may then begin entering another program step.

4-2. REVIEW OF IMMEDIATE MODE KEYS USED TO CREATE STEPS

The Immediate Mode keys that initiate program steps or perform supplementary functions are shown in Figure 4-2. These keys are used to create program steps in the Programming Mode in the same way they are used to specify operations in the Immediate Mode. The same prompts occur during specification, and the same default values are available. The main difference is that when the specification of the operation is complete in the Programming Mode, the 9010A stores the specified operation as a program step instead of performing the operation.

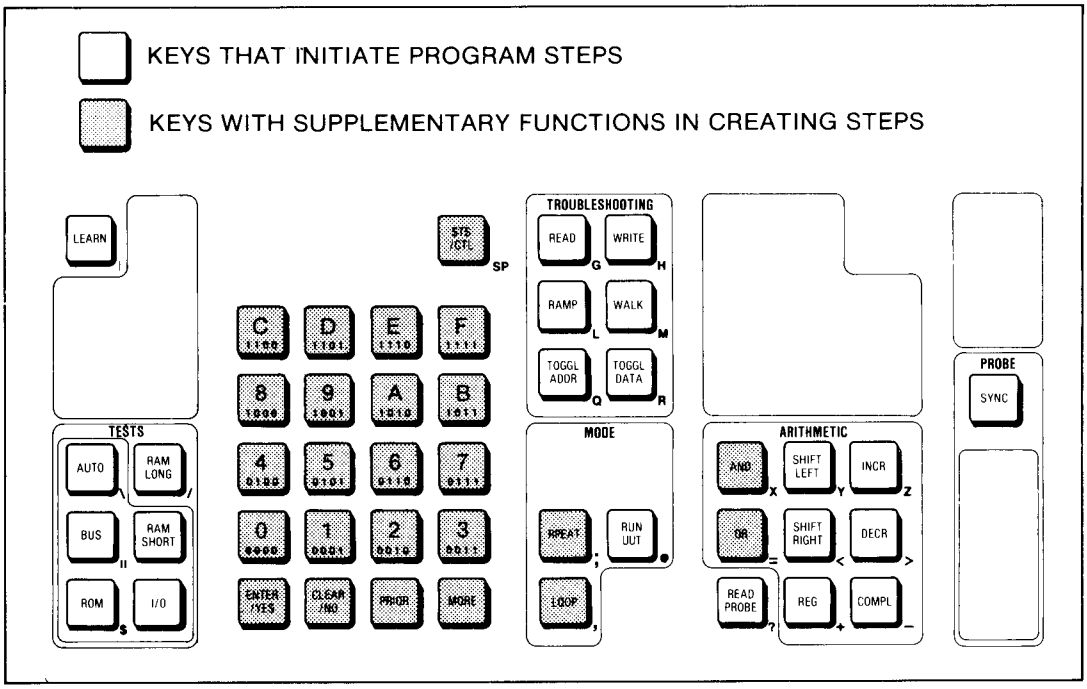


Figure 4-2. Immediate Mode Keys With Similar Applications in the Programming Mode

For example, to specify a Bus Test in the Programming Mode, the operator presses the BUS TEST key. The 9010A stores Bus Test as a program step. For another example, to specify the Ramp troubleshooting function, the operator presses the RAMP key. After the operator specifies the address, the 9010A stores the final specification as a program step. Other important notes about programming with the Immediate Mode keys follow.

- The Repeat function may not be specified as a step by itself, but may be specified one or more times as a 'modifier' after a test or a troubleshooting function has been specified. The word REPT is appended on the same line as the specification, with one REPT appearing for each time the Repeat function is specified. This is illustrated as follows:

  READ @ 50E3 REPT REPT

- Like the Repeat function, the Loop function may not be specified as a step by itself, but may be specified once as a "modifier" after a troubleshooting function or test has been specified. The word LOOP is appended on the same line as the specification. This is illustrated as follows:

  WRITE @ 4C25 = F7 LOOP

- The Loop modifier may be specified after the Repeat modifier has been specified one or more times. The Repeat modifier may not be specified after the Loop modifier has been specified.

- When executing a Run UUT step, the interface pod is placed in the Run UUT mode and the 9010A continues to execute program steps. The interface pod remains in the Run UUT mode until a step is executed which requires use of the interface pod, such as a read or write operation.

Another difference between the Programming Mode and the Immediate Mode is that expressions involving registers or Arithmetic operations are not evaluated until the step containing the expression is executed. This includes the use of the default registers during the specification of a test or function. For example, if the operator selects the Ramp function by pressing the RAMP key, the 9010A displays a prompt for the address. If the operator presses the ENTER key, the 9010A stores the following specification as the program step:

RAMP @ REGF

When the program step is executed, the 9010A supplies the value contained in the address default register, Register F, and performs the Ramp function at that address.

Another example involves the use of a troubleshooting function, a default register, and an Arithmetic operation. Consider the following sequence of keystrokes: READ, INCR, ENTER. When these keys are pressed in the order listed while in the Immediate Mode, the 9010A increments the value provided by the address default register, Register F, and

performs the Read function at the resulting address. In the Programming Mode, when the READ, INCR, and ENTER keys are pressed in order, the 9010A stores the following program step.

READ @ REGF INC

When the program step is executed, the 9010A increments the value provided by Register F and performs the Read function at the resulting address.

4-3. KEYS WITH APPLICATIONS UNIQUE TO PROGRAMMING

The keys shown in Figure 4-3 have applications that are unique to programming or are considerably different from Immediate Mode operation. The PROGM key allows the programmer to enter or exit programs and the Programming Mode, as described in Section 3. The other keys are used to create program steps as described in the following paragraphs.

4-4. The EXEC Key

The EXEC key allows the programmer to create Execute steps. Execute steps cause the 9010A to execute one program from within another program in a subroutine-like fashion. To create the Execute step, press the EXEC key and then enter the program number *dd* of the program that is to be executed. When the Execute step is stored in the program, it looks like the following:

EXECUTE PROGRAM dd

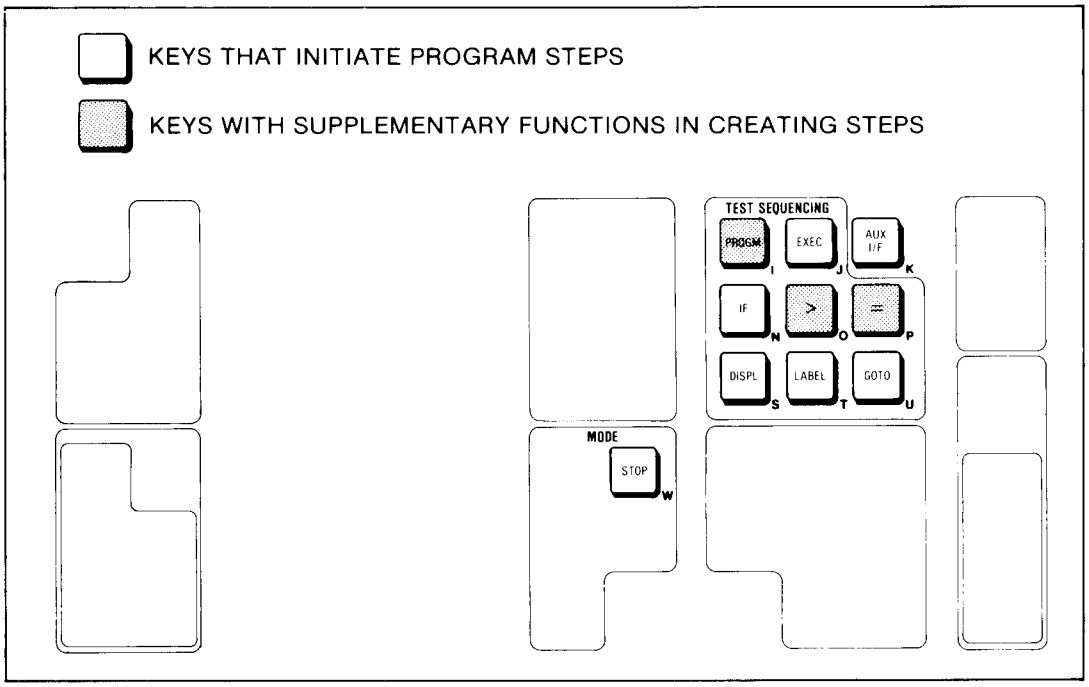


Figure 4-3. Keys With Applications Unique to Programming

The Execute step is illustrated in Figure 4-4. The program that is selected for execution is program 1. When the 9010A encounters the Execute step, the 9010A "calls" program 2 and begins executing it. When the 9010A completes the execution of program 2, it returns to the original program, program 1, and begins executing the step that follows the Execute step.

Notice in Figure 4-4 that Registers 0 through 7 are "local" registers whose values are accessible only within the currently executing program. When program 2 is called, the values in Registers 0 through 7 are stored and then Registers 0 through 7 are set to zero. After program 2 is executed and program control returns to program 1, the original values are restored in Registers 0 through 7. Registers 8 through F are "global" registers whose values are accessible throughout the calling or called programs. The values in Registers 8 through F are not affected when passing between the calling or the called programs. For reference, the function of the 16 registers is presented in Table 4-1.

- A program may not call itself.
- A program may call a program which in turn calls another program. Programs may be called up to 10 levels away from the original program.
- If multiple levels of programs are called, a program may not call any program from a previous level. For example, if program 1 calls program 2 which calls program 3, program 2 may not call program 1, and program 3 may not call program 2 or program 1.
- When creating the Execute step, the program number may be specified as an expression involving registers, decimal numbers, or Arithmetic operations (such as EXECUTE PROGRAM REG7).

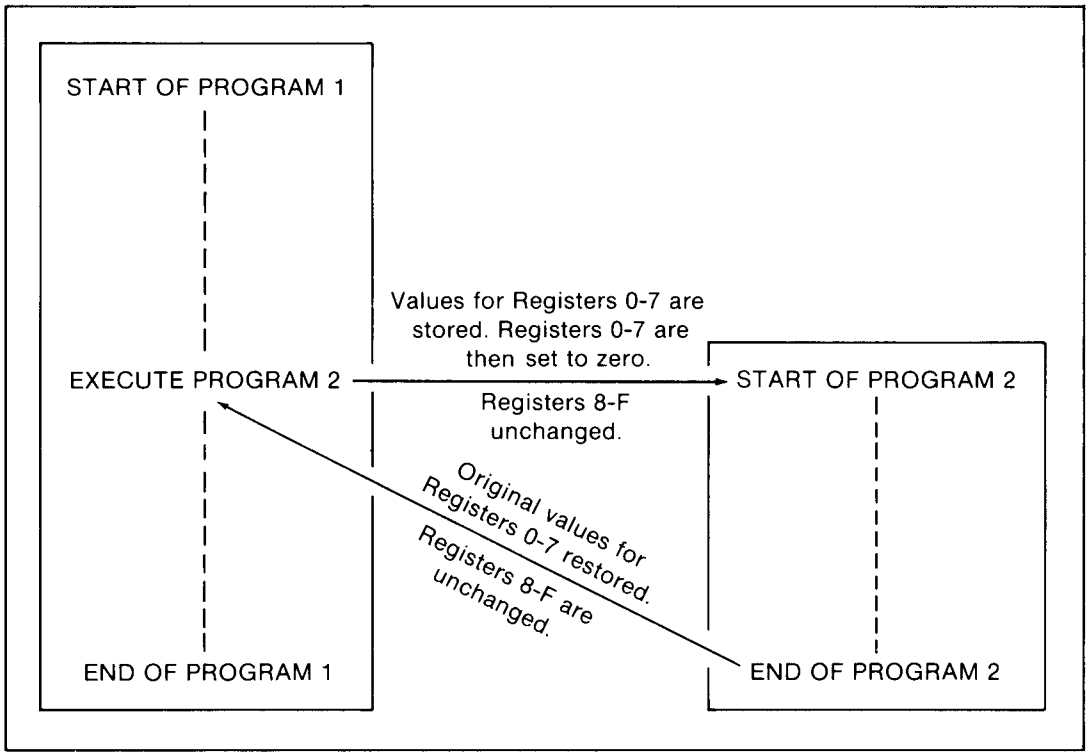


Figure 4-4. The Execute Step

Table 4-1. Function of the Sixteen 32-bit Registers

TYPE OF REGISTER	REGISTER	FUNCTION
DEDICATED	A	Stores the last bit mask specified by the programmer. Also, if the UUT I/O address descriptors are invoked as default values when the I/O Test is performed, the bit mask specified by the last I/O address descriptor is stored in Register A.
	B	Stores the last ROM signature specified by the programmer. Also, if the UUT ROM address descriptors are invoked as default values when the ROM Test is performed, the ROM signature specified by the last ROM address descriptor is stored in Register B.
	C	Stores the last status/control information specified by the programmer for the Write Control or Toggle Data Control functions, or generated by the 9010A during performance of the Read Status function.
	D	Stores the last bit number (in the range 0-31) specified by the programmer for the Toggle Address, Toggle Data, or Toggle Data Control functions.
	E	Stores the last data specified by the programmer or generated by the 9010A during operation.
	F	Stores the last address specified by the programmer or generated by the 9010A during an operation involving the interface pod.
	0	Stores data accumulated during the Read Probe operation.
NON-DEDICATED	1-9	Non-dedicated registers for sole use by programmer for storage and manipulation of data as specified by the programmer.

NOTES:

1. Dedicated Registers A through F and 0 are also available to the programmer for storage and mainipulation of data.

2. Registers 0 through 7 are local registers whose values are local to the currently executing program.

3. Registers 8 through F are global registers and unaffected by passing between called and calling programs.

4-5. The STOP Key

The STOP key is used to create Stop steps which suspend program execution at desired points in the program. To specify the Stop step, press the STOP key. The specification is complete (pressing the ENTER key is not required), and the 9010A displays the step as follows:

STOP

The Stop step may not be modified or combined with another function in the same step.

When the Stop step is executed, the STOPPED annunciator begins flashing and the 9010A suspends program execution. The Stop step itself does not cause a message to appear on the display. The message that appears on the display is the message that is present when the program execution is suspended.

To cause the 9010A to resume program execution, the operator must press the CONT key. While the 9010A is in the stopped state (as indicated by the flashing STOPPED annunciator), the selection of any Immediate Mode test, function, or operation causes the program execution to be aborted. The message that is displayed is the message associated with the selection of the Immediate Mode action.

4-6. The LABEL Key

The LABEL key allows the programmer to create labels. Labels are program steps that are inserted into programs to provide points of entry for branching steps.

- There are 16 possible labels for each program. Each label is identified by a single hexadecimal digit (0 through 9 and A through F).
- Each label may be used only once in a program.
- Labels may appear in any order.
- Labels allow the programmer to move around when examining a program in the Programming Mode, as described in Section 3. If a label exists in a program, pressing the key for the hexadecimal digit of the label causes the label to appear on the display.
- A label may exist without the need for a branch (Goto) step designating that label.

To specify a label as a program step, press the LABEL key and then press a single hexadecimal digit. The specification is complete (pressing the ENTER key is not required), and the 9010A displays the following message.

LABEL n

The letter *n* represents any of the hexadecimal digits.

4-7. The GOTO Key

The GOTO key allows the programmer to construct Goto (unconditional branch) steps. A Goto step is a program step which redirects program execution to a label in the program.

To create a Goto step, press the GOTO key and a single hexadecimal digit. The specification is complete (pressing the ENTER key is not required), and the 9010A displays the following message:

GOTO n

The letter *n* corresponds to the hexadecimal digit of the label where program execution is to be redirected. The use of a Goto step is shown in Figure 4-5. When the program step GOTO 7 is executed, the program execution is redirected to LABEL 7.

- More than one Goto step may redirect program execution to the same label.
- The label to which program execution is redirected by the Goto step may appear anywhere in the program.

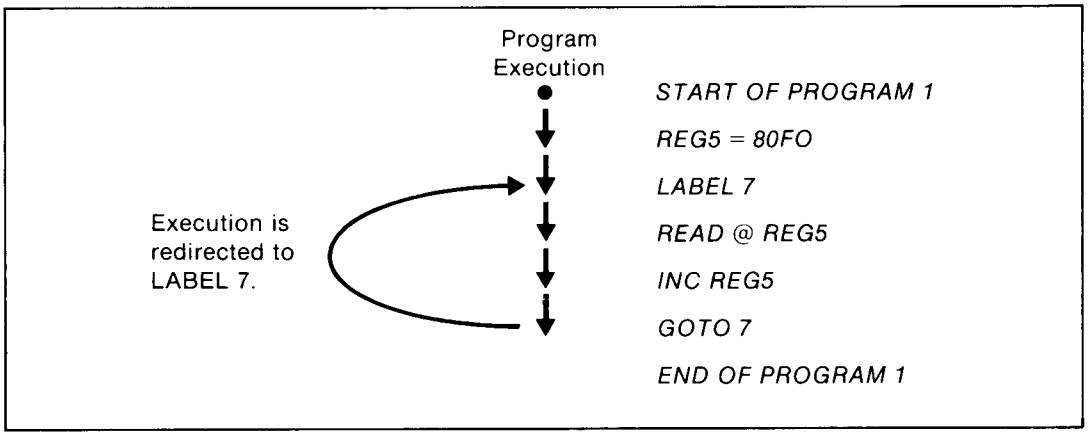


Figure 4-5. The Goto (Unconditional Branch) Step

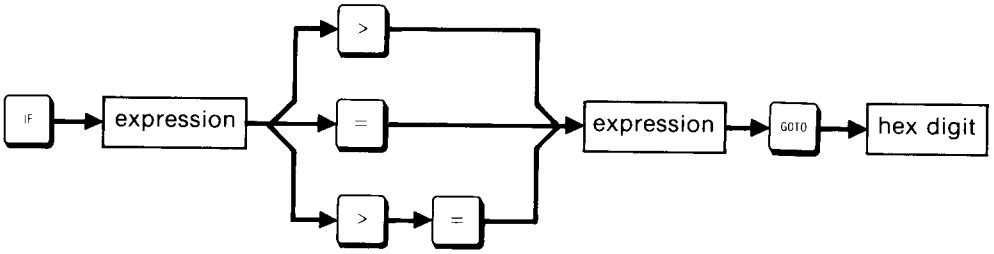
4-8. The IF, >, and = Keys

The IF, >, and = keys may be combined with the GOTO key to create If (conditional branch) steps. An If step differs from a Goto step in that, depending on certain conditions specified by the programmer, the program execution may or may not be redirected to a label in the program.

The conditions that determine whether the branch occurs are based on a comparison of two expressions for greater than, equal, or greater than or equal. The syntax for the specification of the If step is illustrated in Figure 4-6, along with examples. Note that an If step is initiated with the IF key and always includes two expressions separated by either or both of the > and = keys, followed by the GOTO key and a hexadecimal digit.

The use of an If step is shown in Figure 4-7. Note that program 2 in Figure 4-7 is identical to program 1 in Figure 4-5, except that the Goto step is replaced by an If step. When the If step in program 2 is executed, the program execution is redirected to Label 7 only if the contents of Register 5 are less than 80FF. When program 2 is executed, the 9010A reads the data at addresses 80F0 through 80FF and completes the execution of the program.

- More than one If step may redirect program execution to the same label.
- The label to which program execution is redirected by the If step may appear anywhere in the program.



EXAMPLES: *IF REG4 > 2000 GOTO 5*

IF REGE OR FFF7 = FFFF GOTO B

IF 8000 > REG6 GOTO 9

IF REGE CPL >= REG 3 GOTO A

NOTES:

expression = a sequence of keystrokes consisting of numeric values and/or registers. It may also include Arithmetic operations.

hex digit = single hexadecimal digit which designates the label where the branch redirects program execution.

Figure 4-6. The Syntax for the Specification of the If Step

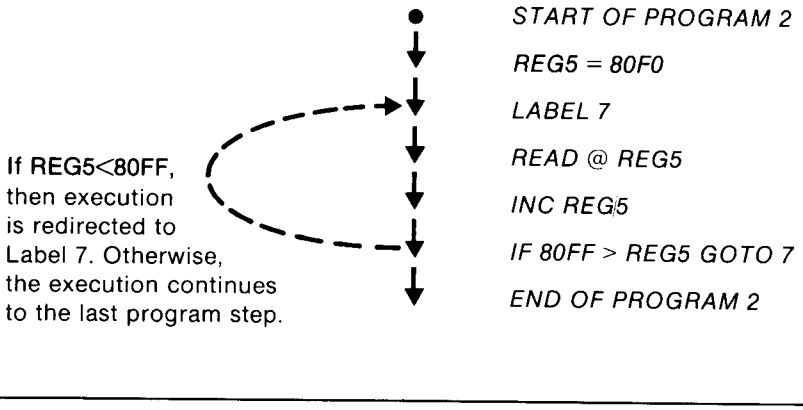


Figure 4-7. The If (Conditional Branch) Step

4-9. The DISPL Key

The DISPL key allows the programmer to compose messages that are presented to the operator when a program is executed. Some of the messages simply provide information, such as the data at an address or the contents of a register. Other messages suspend program execution and display a prompt for operator input. The program steps that are created with the DISPL key are called Display steps.

To create a Display step, press the DISPL key. The 9010A responds by presenting the following message:

DPY-_

After the DISPL key has been pressed, a message may be created one character at a time by pressing the appropriate key. The numerals 0 through 9 and the letters A through F are entered with the corresponding hexadecimal keys. Additional characters and symbols available for use are located on the keyboard panel near the lower right corner of the associated key. For example, the letter Q is located next to the TOGGL ADDR key. When the TOGGL ADDR key is pressed, the letter Q is specified and appears in the display message when the Display step is executed. Similarly, when the OR key is pressed, the equal sign (=) is specified and appears in the display message when the Display step is executed.

Characters that have been entered in the message may be deleted one character at a time by pressing the PRIOR key. The entry of characters or symbols in the message is terminated by pressing the ENTER key.

Table 4-2 presents a list of all the keys that have valid display characters or symbols. The available characters and symbols include all numbers 0 through 9, all 26 uppercase letters, common punctuation marks, and several special symbols.

Table 4-2. Function of the 9010A Keys When Used In the Display Step

CHARACTER OR SYMBOL	KEY NAME	CHARACTER OR SYMBOL	KEY NAME	CHARACTER OR SYMBOL	KEY NAME
0	0	J	EXEC	=	OR
1	1	K	AUX I/F	<	SHIFT RIGHT
2	2	L	RAMP	>	DECR
3	3	M	WALK	,	LOOP
4	4	N	IF	.	SETUP
5	5	O	>	?	READ PROBE
6	6	P	=	+	REG
7	7	Q	TOGGL ADDR	-	COMPL
8	8	R	TOGGL DATA	'	LEARN
9	9	S	DISPL	%	I/O VIEW
A	A	T	LABEL	*	RAM VIEW
B	B	U	GOTO	\	AUTO TEST
C	C	V	CONT	/	RAM LONG
D	D	W	STOP	"	BUS TEST
E	E	X	AND	$	ROM TEST
F	F	Y	SHIFT LEFT	space	STS/CTL
G	READ	Z	INCR	◢ (bell)	ROM VIEW
H	WRITE	;	RPEAT		
I	PROGM	@	RUN UUT		

The following items pertain to the creation of Display steps:

- During execution of the Display step, only the characters following the initial display (DPY-) appear on the display. The characters DPY- do not appear on the display.
- Up to 27 characters may be entered for a Display step. The 9010A emits a beep if the programmer attempts to enter another character.
- Whenever a Display step is executed, previous information on the display is always cleared unless the first character of the Display step message is the + symbol. If the first character is the + symbol, the Display step message is appended to any previous message on the display.
- Six of the keys are not allowed during the creation of a Display step. The 9010A emits a beep when they are pressed. The keys are: RAM SHORT, I/O TEST, MORE, READ TAPE, WRITE TAPE, and SYNC.

The following example shows the keystrokes necessary to create the message REPLACE U27. The keys are separated by commas for ease in reading.

PRESS	DISPLAY (PROGRAMMING MODE)
DISPL	*DPY-_*
TOGGL DATA, E, =, RAMP, A, C, E	*DPY-REPLACE_*
STS/CTL, GOTO, 2, 7, ENTER	*DPY-REPLACE U27*

When the Display step is executed (in the Executing Mode), the message is displayed as follows:

REPLACE U27 (The portion *DPY-* does not appear)

In addition to the text-creating capability of the display function, there are eight symbols which have special meaning when they are specified in the proper format in a Display step. The function of each of these symbols is described in Table 4-3. Examples of the use of each symbol are provided in Section 5.

Table 4-3. Function of Special Symbols in the Display Step

CHARACTER OR SYMBOL	KEY NAME	ACTION CAUSED
◭ (bell)	ROM VIEW	Causes the 9010A to beep when the Display step is executed. The bell symbol does not appear on the display when the step is executed.
$	ROM TEST	When followed by a hexadecimal digit, it causes contents of the register designated by the digit to be displayed in hexadecimal on the display.
@	RUN UUT	The same as for the $ symbol, except that the contents are displayed in decimal.

Table 4-3. Function of Special Symbols in the Display Step (cont)

CHARACTER OR SYMBOL	KEY NAME	ACTION CAUSED
/	RAM LONG	When followed by a hexadecimal digit, it suspends program execution and prompts for input. When the operator enters a hexadecimal value terminated by ENTER, the 9010A places the value in the register designated by the digit and resumes program execution. Pressing ENTER without specifying a hexadecimal value causes the value to default to the previous contents of the register.
\	AUTO TEST	The same as for the / symbol, except that the 9010A accepts only a decimal entry.
?	READ PROBE	When followed by a hexadecimal digit, it suspends program execution and displays the question mark (?). If the operator presses the ENTER/YES key, the 9010A places a 1 in the designated register. If the operator presses the CLEAR/NO key, the 9010A places a 0 in the designated register. After the 1 or 0 is placed in the register, the 9010A removes the question mark and then resumes program execution.
%	I/O VIEW	When followed by a hexadecimal digit, it enables or disables asynchronous input from the operator during execution. Asynchronous input is stored in the register designated by the hexadecimal digit. Asynchronous input is described in Section 5.
+	REG	When it is the first character in the specification, it causes the following characters in the specification to be appended to the text that is on the display at the time the Display step is executed.

NOTE: In order to causes the $, @, /, \ , or % symbols to appear in the displayed text when the Display step is executed, the symbols must appear twice in the specification.

4-10. The AUX I/F Key

The Auxiliary Interface (AUX I/F) functions much differently in the Programming and Executing Modes than it does in the Immediate Mode. In the Immediate Mode, information sent to the AUX I/F may consist of programs, program numbers, Setup parameters, or address descriptors. In the Executing Mode, information sent to the AUX I/F consists of programmer-created messages that are composed character-by-character in a fashion that is identical to the Display step messages. The main difference is that

instead of pressing the DISPL key, the programmer presses the AUX I/F key. Instead of presenting DPY-_ on the display, the 9010A presents the following message:

AUX-_

At this point the programmer may compose messages that are sent to the AUX I/F when the AUX I/F step is executed in the program. Note that some of the symbols that have special meaning in the Display steps (as described in Table 4-3) have a slightly different meaning when used in an AUX I/F step. The function of these symbols is described in Table 4-4. The use of these symbols in programs is discussed in Section 5.

When an AUX I/F step is executed the terminator sequence is sent (as determined by the "NEWLINE" Setup parameter in the Immediate Mode) unless the + symbol is the final character in the AUX I/F message.

Table 4-4. Function of Special Symbols in the AUX I/F Step

CHARACTER OR SYMBOL	KEY NAME	ACTION CAUSED
◭ (bell)	ROM VIEW	Sends a control G (bell) to the RS-232 interface.
$	ROM TEST	When followed by a hexadecimal digit, it causes the contents of the register designated by the digit to be transmitted in hexadecimal to the RS-232 interface.
@	RUN UUT	The same as for the $ symbol, except that the contents are transmitted in decimal.
/	RAM LONG	When followed by a hexadecimal digit, it suspends program execution, waits for the next byte of data from the RS-232 interface, and places the value of the byte in the register designated by the hexadecimal digit (the upper three bytes of the register equal zero). If the RS-232 interface is configured to transfer eight data bits, then eight data bits appear. Otherwise the eighth data bit (bit 7) is zero.
\	AUTO TEST	When followed by a hexadecimal digit, it places the status of the RS-232 interface in the lower five bits of the register designated by the hexadecimal digit (the upper 27 bits are zero). The five status bits are as follows: Bit 0: 1 = Parity Error/0 = No Parity Error Bit 1: 1 = Framing Error/0 = No Framing Error Bit 2: 1 = Overrun Error/0 = No Overrun Error Bit 3: Status of Receive Buffer 1 = Character Received 0 = No Character Received Bit 4: Status of Transmit Buffer 1 = Transmit buffer is empty, ready for next character 0 = Character still being sent
?	READ PROBE	Not used with AUX I/F.

Table 4-4. Function of Special Symbols in the AUX I/F Step (cont)

CHARACTER OR SYMBOL	KEY NAME	ACTION CAUSED
%	I/O VIEW	When followed by a hexadecimal digit, it transmits the low-order byte contained in the register designated by the hexadecimal digit. This provides a way for the programmer to send the full range of ASCII characters to the AUX I/F. Eight data bits are sent if the RS-232 interface is configured to transfer eight bits.
+	REG	When it is the last character in an AUX I/F step, it prevents a line termination sequence from being sent at the end of the line.

NOTE: In order to causes the $, @, /\ , or % symbols to appear in the text when the AUX I/F step is executed, the symbols must appear twice in the specification.

Section 5
Programming Examples and Techniques

5-1. INTRODUCTION

This section expands on the step definitions provided in Section 4 and explains how the steps may be used in a program. Programming techniques for each type of step are discussed and illustrated by examples.

Note that program steps are presented in this section in the AUX I/F format in which they appear when printed out on a printer or viewed on an auxiliary video display. To distinguish these program steps, they are shown in this manual in **PRINTER TYPEFACE**. Messages that appear on the 9010A display are shown in *UPPERCASE ITALIC TYPEFACE*. Comments that appear beside the program steps are shown in the regular typeface.

The sample program from Section 2 is shown below to illustrate the difference between the AUX I/F format and 9010A display format. The program is shown at the left as it appears through AUX I/F; the program steps are shown at the right as they appear when viewed one-by-one on the 9010A display in the Programming Mode. Remember that the steps shown on the right are not the messages that appear on the 9010A display when the program is executed.

AUX I/F FORMAT	9010A DISPLAY FORMAT
PROGRAM 1 51 BYTES	
	START OF PROGRAM 1
REG1 = 20	*REG1 = 20*
1: LABEL 1	*LABEL 1*
DPY-@1	*DPY-@1*
REG2 = 10	*REG2 = 10*
2: LABEL 2	*LABEL 2*
DEC REG2	*DEC REG2*
IF REG2 > 0 GOTO 2	*IF REG2 > O GOTO 2*
DEC REG1	*DEC REG1*
IF REG1 > 0 GOTO 1	*IF REG1 > O GOTO 1*
DPY-DONE#	*DPY-DONE* ◭
	END OF PROGRAM 1

The AUX I/F format differs from the 9010A display format in the following ways: the program number is listed along with the size of the program (51 bytes); label numbers are printed in the left margin so they are easily recognized; the start message and end message do not appear. Notice also in the last line of the AUX I/F listing that the AUX I/F transmits the # symbol for the ◭ (bell) symbol.

5-2. CREATING DISPLAY MESSAGES

When the program is executed, the only messages that may appear on the display are error messages, the initial specification message (EXECUTE PROGRAM nn), or display messages created by the programmer. Display steps allow the programmer to display text, register contents, or cause the 9010A to beep. The following paragraphs contain programs illustrating the use of Display steps.

5-3. Displaying Text

The programmer may use the alphabetic and numeric characters to create messages that provide information or instructions during program execution. The following steps provide examples. Note that the program steps are presented at the left as they appear in the AUX I/F program listing. The corresponding display messages at the right appear on the 9010A display when the steps are executed.

```
DPY-PROGRAM 15 COMPLETED          PROGRAM 15 COMPLETED
DPY-REPLACE U27                   REPLACE U27
DPY-PRESS CONT KEY WHEN READY     PRESS CONT KEY WHEN READY
DPY-PLACE PROBE @ U4 PIN6         PLACE PROBE @ U4 PIN6
```

Whenever the Display step is executed, the 9010A display is cleared unless the Display step begins with the + symbol. Sometimes it is useful to append a message onto a previous message using the + symbol as follows:

```
DPY-PLACE PROBE @ U4 PIN6     PLACE PROBE @ U4 PIN6
DPY-+-PRESS CONT              PLACE PROBE @ U4 PIN6-PRESS CONT
```

5-4. Displaying Register Contents

The $ and @ symbols allow the programmer to display register contents in either hexadecimal or decimal respectively:

```
REG1 = 12E4
DPY-$1          12E4
DPY-@1          4836
```

In order to display the $ and @ symbols, they must be followed by either a non-hexadecimal digit, or inserted twice:

```
REGE = 27AA3
DPY-ABCDEFG$H      ABCDEFG$H
DPY-ABCD$EFGH      ABCD27AA3FGH
DPY-ABCD$$EFGH     ABCD$EFGH
```

5-5. Displaying Text and Register Contents Together

Display steps may also display text and register contents in the same step:

```
REG7 = 3B7
DPY-REG7 CONTAINS $7 HEX     REG7 CONTAINS 3B7 HEX
```

Text and register contents may be appended to a previous step with the + symbol:

```
REG7 = 3B7
DPY-REG7 CONTAINS $7 HEX     REG7 CONTAINS 3B7 HEX
DPY-+ OR @7 DEC              REG7 CONTAINS 3B7 HEX OR 951 DEC
```

The $ and @ symbols may be used more than once in the same step:

```
REG1 = 1111
REG2 = 2222
DPY-REG1 = $1, REG2 = $2      REG1 = 1111,  REG2 = 2222
```

To perform hexadecimal-to-decimal conversion, place the desired value in Register 6 in the Immediate Mode and then execute a program consisting of the following step:

```
DPY-$6 HEX EQUALS @6 DECIMAL
```

5-6. Causing the 9010A to Beep

The 9010A beep may be used to call attention to a particular step or operator instruction when the program is executed:

```
DPY-PLACE PROBE AT U15 PIN3 #     PLACE PROBE AT U15 PIN3 (beep)
```

When this step is executed, the 9010A emits a beep to call the operator's attention to a displayed instruction. Note that the ◭ (bell) symbol appears in the program listing as the # symbol. Consecutive occurrences of the # symbol do not cause multiple beeps. The execution of a Display step causes the previous message to be cleared from the display unless the Display step begins with the + symbol:

```
DPY-REPLACE U8        REPLACE U8
DPY-#                 (display cleared, 9010A beeps)
DPY-REPLACE U14       REPLACE U14
DPY-+#                REPLACE U14 (beep)
```

5-7. CONTROLLING PROGRAM FLOW

There are three ways of controlling the program flow during execution: (1) with the If, Goto, or Stop steps, (2) with the Execute step, (3) with the Loop or Repeat modifiers. All three types of control are discussed in the following paragraphs.

5-8. Using Goto, Stop, and If Steps to Control Program Flow

The sample programs in this section demonstrate the use of the If, Goto, and Stop steps. Simple loops are illustrated, as well as the use of registers and Arithmetic operations.

Most of the sample programs in this section involve a Read operation which requires that an interface pod and a UUT be connected to the 9010A. The address specified for the Read operations in these programs is arbitrarily selected as 8000 (and up). In order to obtain stable data when executing these sample programs, it is recommended that the programmer replace the address 8000 with a ROM address in the UUT.

The following program performs a very simple but useful task; it reads the data at an address and displays the data:

```
READ @ 8000
DPY-ADDRESS $F DATA $E  Reg F contains address, Reg E contains data
```

The $ and @ symbols may be used more than once in the same step:

```
REG1 = 1111
REG2 = 2222
DPY-REG1 = $1, REG2 = $2      REG1 = 1111,  REG2 = 2222
```

To perform hexadecimal-to-decimal conversion, place the desired value in Register 6 in the Immediate Mode and then execute a program consisting of the following step:

```
DPY-$6 HEX EQUALS @6 DECIMAL
```

5-6. Causing the 9010A to Beep

The 9010A beep may be used to call attention to a particular step or operator instruction when the program is executed:

```
DPY-PLACE PROBE AT U15 PIN3 #     PLACE PROBE AT U15 PIN3 (beep)
```

When this step is executed, the 9010A emits a beep to call the operator's attention to a displayed instruction. Note that the ◭ (bell) symbol appears in the program listing as the # symbol. Consecutive occurrences of the # symbol do not cause multiple beeps. The execution of a Display step causes the previous message to be cleared from the display unless the Display step begins with the + symbol:

```
DPY-REPLACE U8        REPLACE U8
DPY-#                 (display cleared, 9010A beeps)
DPY-REPLACE U14       REPLACE U14
DPY-+#                REPLACE U14 (beep)
```

5-7. CONTROLLING PROGRAM FLOW

There are three ways of controlling the program flow during execution: (1) with the If, Goto, or Stop steps, (2) with the Execute step, (3) with the Loop or Repeat modifiers. All three types of control are discussed in the following paragraphs.

5-8. Using Goto, Stop, and If Steps to Control Program Flow

The sample programs in this section demonstrate the use of the If, Goto, and Stop steps. Simple loops are illustrated, as well as the use of registers and Arithmetic operations.

Most of the sample programs in this section involve a Read operation which requires that an interface pod and a UUT be connected to the 9010A. The address specified for the Read operations in these programs is arbitrarily selected as 8000 (and up). In order to obtain stable data when executing these sample programs, it is recommended that the programmer replace the address 8000 with a ROM address in the UUT.

The following program performs a very simple but useful task; it reads the data at an address and displays the data:

```
READ @ 8000
DPY-ADDRESS $F DATA $E  Reg F contains address, Reg E contains data
```

The preceding program can be expanded with the use of a Goto step and a loop. This program reads and displays the data at a sequence of addresses:

```
   REG1 = 8000                          Initial address loaded into Reg 1
1: LABEL 1                              Entry point for branch
   READ @ REG1
   DPY-ADDRESS $1 DATA $E
   INC REG1                             Reg 1 incremented for next address
   GOTO 1                               Branch back and read next address
```

When the preceding program is executed, a problem arises; the 9010A reads, displays, and increments the addresses so fast that the addresses and data cannot be read by the operator. One way of overcoming this difficulty is with the Stop step, as shown in the following program:

```
   REG1 = 8000
1: LABEL 1
   READ @ REG1
   DPY-ADDRESS $1 DATA $E
   DPY-+-PRESS CONT                     Append instruction
   STOP                                 Suspend execution, wait for CONT
   INC REG1
   GOTO 1
```

When the preceding program is executed, the 9010A reads the first address and displays the data. When the Stop step is executed, the 9010A suspends execution with the address and data present on the display and turns on the flashing STOPPED annunciator. When the operator presses the CONT key, program execution is resumed and the STOPPED annunciator turns off. The program increments Register 1, loops back to Label 1, and reads and displays the data at the next address. This use of the Stop step gives the operator control over the Read operations.

Another way to control the preceding program is with an If step as follows:

```
   REG1 = 8000
1: LABEL 1
   READ @ REG1
   DPY-ADDRESS $1 DATA $E
   REG5 = 30                            Sets number of delay loops
2: LABEL 2
   DEC REG5                             Decrement delay loop register
   IF REG5 > 0 GOTO 2
   INC REG1
   GOTO 1
```

When the preceding program is executed, the 9010A steps through the addresses in sequence, reading and displaying the data. However, the four new steps in this program (beginning with REG5 = 30) make up a delay loop. The sole purpose of the delay loop is to increase the amount of time that each address and data appear on the display so that they may be read by the operator. The length of the delay can be adjusted by changing the initial value that is placed in Register 5.

The delay loop could also be constructed as follows:

```
    REG5 = 0
2:  LABEL 2
    INC REG5
    IF 30 > REG5 GOTO 2
```

One disadvantage of the preceding construction is that the adjustment of the length of the delay involves changing the longer If step rather than the shorter first step.

The following If step is a useful addition at the end of the program, for it sets an upper limit for the address range:

```
    REG1 = 8000
1:  LABEL 1
    READ @ REG1
    DPY-ADDRESS $1 DATA $E
    REG5 = 30
2:  LABEL 2
    DEC REG5
    IF REG5 > 0 GOTO 2
    IF REG1 = 801F GOTO 3        Branch when address = 801F
    INC REG1
    GOTO 1
3:  LABEL 3
    DPY-+-COMPLETE               Word appended when program complete
```

After the data at address 801F is read and displayed, the program control branches to Label 3, where the final message is displayed and execution is completed.

The following program demonstrates more uses of loops, the If step, and Arithmetic operations. The program counts the number of one bits in a hexadecimal number and displays the result. To use the program, enter the desired hexadecimal number into Register 1 and then execute the program.

```
                                 Original number placed in Reg 1
    REG2 = 20                    Loop counter set
    REG3 = 0                     Initialize bit counter
    REG4 = REG1                  Original number placed in Reg 4
1:  LABEL 1
    IF REG4 AND 1 = 0 GOTO 2     Check farthest right bit
    INC REG3                     Increment Reg 3 if bit is 1
2:  LABEL 2
    SHR REG4                     Bits are shifted right
    DEC REG2                     Decrement loop counter
    IF REG2 > 0 GOTO 1           Branch if more bits to test
    DPY-$1 HEX HAS @3 ONE BITS   Display number and list count
```

In the preceding program, Register 2 is the loop counter which counts the 32 loops that are required to examine all 32 bits in Register 1 (32 decimal equals 20 hexadecimal). Register 3 counts the number of one bits that are determined by the logical AND operation in the first If step.

5-9. Using the Execute Step to Control Program Flow

The Execute step allows the programmer to use modular design when creating programs. Sequences of frequently used steps such as delay loops can be stored as separate programs and called by any other program stored in memory. This can save program steps in a large program, make it easier to read, and also save registers. The following two programs show how the Execute step may be used. For reference, the program numbers are included at the top of the program.

```
PROGRAM 40   21 BYTES

    REG1 = REG8
1:  LABEL 1
    DEC REG1
    IF REG1 > 0 GOTO 1

PROGRAM 10   78 BYTES

    REG1 = 8000
1:  LABEL 1
    READ @ REG1
    DPY-ADDRESS $1 DATA $E
    REG8 = 20                      Sets number of delay loops
    EXECUTE PROGRAM 40             Calls delay loop program
    IF REG1 = 801F GOTO 2
    INC REG1
    GOTO 1
2:  LABEL 2
    DPY-+-COMPLETE
```

Program 40 is a simple delay loop program which is called by program 10. Notice in the first step in Program 40 that the delay loop counter (Register 1) is set equal to Register 8, a global register. This provides more flexibility in using the delay loop, because the program that calls Program 40 can set the length of the delay by placing a value in Register 8 and passing the value to Program 40.

Notice also that Register 1, a local register, is used in both Program 40 and Program 10, and the value in Register 1 in one program is not carried over to another program. In Program 10, Register 1 contains the address for the Read operation. When Program 40 is executed from Program 10, the contents of Register 1 are saved and then Register 1 is set to 0. The first step in Program 40 places a new value in Register 1. When Program 40 has completed execution, the value saved from Program 10 is restored in Register 1 and execution of Program 10 resumes.

Another example of the use of the Execute step is provided in the following program:

```
DPY-PROBE U17 PIN6-PRESS CONT#     Operator instructions
STOP                               Wait for CONT
EXECUTE PROGRAM 20                 Probe program called
DPY-PROBE U9 PIN11-PRESS CONT#     Operator instructions
STOP                               Wait for CONT
EXECUTE PROGRAM 20                 Probe program called
DPY-PROBE U10 PIN3-PRESS CONT#     Operator instructions
STOP                               Wait for CONT
EXECUTE PROGRAM 20                 Probe program called
DPY-PROGRAM COMPLETE
```

In the preceding program, the operator is given instructions about where to place the probe and then Program 20 is called. Program 20 (not listed) could be a sequence of steps involving the probe which applies to all three of the nodes listed.

5-10. Using the Loop or Repeat Modifiers to Control Program Flow

The Loop or Repeat modifiers may be appended to any of the built-in tests or troubleshooting functions when a step is created. When a step with the Repeat modifier is executed, the test or function is performed twice. When a step with the Loop modifier is executed, the 9010A turns on the flashing LOOPING annunciator and repeatedly performs the test or function.

When a test or function is looping, the operator has the same control as in the Immediate Mode: pressing the STOP key while looping causes the 9010A to turn off the LOOPING annunciator, turn on the STOPPED annunciator, and enter the stopped state; pressing the LOOP key while stopped causes the 9010A to resume looping (indicated by the LOOPING annunciator). Unlike the Immediate Mode, however, if the CONT key is pressed while looping, the 9010A continues program execution and executes the next program step.

The following program demonstrates one use of the Loop modifier. The program allows the operator to examine the waveforms that occur at data bit 3 on an oscilloscope. The operator is instructed where to place the scope probe and then a looping Write operation is performed which forces data bit 3 high. The operator may stop and resume the looping Write operation (by pressing the CONT key while in the stopped state). The next looping Write operation forces data bit 3 low, and the operator has the same choices as described for the first looping Write operation.

```
DPY-PUT SCOPE @ U7 PIN9          Operator instruction
DPY-+-PRESS CONT
STOP                             Wait for CONT
DPY-DATA BIT 3 IS HIGH           Tells value of bit 3
WRITE @ 8000 = 8 LOOP            Set data bit 3 high
DPY-DATA BIT 3 IS LOW            Tells value of bit 3
WRITE @ 8000 = 0 LOOP            Writes data bits low
DPY-TEST COMPLETE                Final message
```

Note that a Display step is specified before each of the looping Write operations that describes what takes place during the operation. The reason for the message is that when the looping Write operation takes place, the operator has no way of knowing what is taking place except for the flashing LOOPING annunciator. The Display step message provides more information about what the 9010A is doing.

5-11. SYNCHRONOUS OPERATOR INPUT

Synchronous operator input is input that may only be provided while program execution is suspended. When a Display step is executed that requires synchronous input, program execution is suspended at that step until the operator enters a valid input. The valid input is placed in the designated register and program execution resumes. While program execution is suspended, the STOPPED annunicator flashes; when the program execution resumes the annunciator stops flashing.

When the /,\, or ? symbols are followed by a hexadecimal digit in the Display step, the 9010A accepts hexadecimal, decimal, or Boolean (YES or NO) input. Each type of input is discussed and illustrated in the following paragraphs.

5-12. Hexadecimal Input

This hexadecimal-to-decimal conversion program illustrates hexadecimal input and output, and decimal output. When the first step in the program is executed, the 9010A suspends program execution and displays the cursor which prompts for a hexadecimal value. When a value is entered, the value is placed in Register 1 and program execution is continued. The second step displays the hexadecimal value and the decimal equivalent.

```
   DPY-ENTER A HEX VALUE /1          Prompt for the value
   DPY-$1 HEX EQUALS @1 DECIMAL      Display in hex and decimal
```

5-13. Decimal Input

This program illustrates decimal input and output. When execution begins, the program displays a prompt for a hexadecimal number. After a number is entered, the program displays a message which prompts the operator to guess the decimal equivalent of the hexadecimal number. If the guess is incorrect, the program displays a prompt for another guess. If the guess is correct, the 9010A emits a beep and displays the correct value. Notice that the 9010A only accepts decimal input when the decimal guess is entered (otherwise the 9010A emits a beep).

```
   DPY-ENTER THE HEX NUMBER /1          Hex input to Reg 1
   DPY-ENTER YOUR DECIMAL GUESS         Prompt for guess
1: LABEL 1
   DPY-+ \2                             Decimal input to Reg 2
   IF REG2 = REG1 GOTO 3                Guess is correct
   IF REG2 > REG1 GOTO 2                Guess too high
   DPY-@2 IS TOO LOW-TRY AGAIN          Guess too low
   GOTO 1
2: LABEL 2
   DPY-@2 IS TOO HIGH-TRY AGAIN         High guess displayed
   GOTO 1
3: LABEL 3
   DPY-*YES* HEX $2 = DECIMAL @2#       Correct guess displayed
```

5-14. Boolean (YES or NO) Input

This program illustrates Boolean (YES or NO) input. When this program is executed, the question ARE YOU READY? appears. The characters ?A in the first step cause the program execution to be suspended. If the YES key is pressed, a 1 is stored in Register A and program execution is continued. If the NO key is pressed, a 0 is stored in Register A and program execution is continued. Notice that unlike the other special symbols for the Display step, the ? symbol appears on the display as the prompt for a response. The ? symbol is removed from the display when a response is entered.

```
   DPY-ARE YOU READY?A                  Prompt for input to Reg A
   IF REGA = 0 GOTO 1                   Branch if NO key pressed
   DPY-+ YES                            YES response displayed
   GOTO 2
1: LABEL 1
   DPY-+ NO                             NO response displayed
2: LABEL 2
```

5-15. Multiple Inputs and Outputs

More than one input request may be included in a single Display step as illustrated in the following example. Note that when the step is executed the 9010A only displays the text

portion to the left of the first prompt (ENTER ADDR). When a value is entered in response to the first prompt, the 9010A appends the remaining portion of text to the message on the display along with the second prompt. The first value entered is stored in Register 1, and the second value entered is stored in Register 2.

```
DPY-ENTER ADDR /1 ENTER DATA /2
```

The following program illustrates multiple inputs and outputs. During execution, the 9010A displays prompts for the first and last address of an address block and for an operator-specified value. When the addresses and data are specified, the 9010A searches through the address block for a memory location with the specified value. If the specified value is found, the 9010A displays the data, the memory location where it was found, and asks if the search is to continue.

```
   DPY-FIRST /1 LAST /2 DATA /3          Prompts for addresses, data
1: LABEL 1
   READ @ REG1                           Read performed
   INC REG1
   IF REGE = REG3 GOTO 3                 Data found
   IF REG1 > REG2 GOTO 2                 Data not found in block
   GOTO 1
2: LABEL 2
   DPY-DATA NOT FOUND-
   GOTO 4
3: LABEL 3
   DPY-$3 FOUND AT $F - CONTINUE?5       Data and address displayed
   IF REG5 = 1 GOTO 1                    Branch if YES key pressed
   DPY-                                  Clears the display
4: LABEL 4
   DPY-+SEARCH COMPLETE
```

5-16. ASYNCHRONOUS OPERATOR INPUT

During program execution, it is often desirable to have the 9010A be responsive to input from the operator while it continues to execute program steps. This is not possible with synchronous input since program execution is always suspended when synchronous input is requested, and program execution is not continued until a value is entered. With asynchronous input, however, the 9010A may be executing program steps but still be responsive to input from the operator.

Asynchronous input is characterized as follows:

- Asynchronous input is enabled when a Display step is executed which contains the % symbol followed by a hexadecimal digit.
- While asynchronous input is enabled, pressing any key (except STOP, HIGH, or LOW) causes the associated value from Table 5-1 to be entered into the register designated by the hexadecimal digit. This allows the programmer to redefine the keyboard during program execution.
- Once asynchronous input is enabled, it remains enabled until it is disabled. Asynchronous input is disabled if the operator presses any key (except STOP, HIGH, or LOW), or if a Display step is executed which contains the % symbol followed by the hexadecimal digit for the enabled register.
- Only one register can be open for asynchronous input at a time. Enabling a register for asynchronous input disables any register previously enabled.

asynchronous input is disabled, the associated value is displayed, and the program control returns to the first step to begin the sequence again.

One of the advantages of asynchronous input is that the programmer can redefine the use of the keys during program execution. The following program demonstrates this redefinition of the keys. When the program is executed, the 9010A prompts for an address. When the address is entered, the 9010A reads and displays the data at the address and waits for another key to be pressed. The keys that are defined by this program and may be used during execution are as follows:

MORE	Read and display the data at the next address
PRIOR	Read and display the data at the previous address
ENTER	Prompt for new starting address
Other Keys	Beep when pressed (except STOP, HIGH, and LOW)

```
1: LABEL 1                          Arrive here if ENTER pressed
   DPY-ADDRESS /1                   Prompt for new address
2: LABEL 2
   READ @ REG1                      Read UUT data
   DPY-ADDRESS $1 DATA $E           Display address and data
3: LABEL 3
   REG2 = 40                        No key has this value
   DPY-+%2                          Enable async input for Reg 2
4: LABEL 4
   IF REG2 = 40 GOTO 4              Branch if no key pressed
   IF REG2 = 1C GOTO 1              Branch if ENTER pressed
   IF REG2 = 1B GOTO 5              Branch if MORE pressed
   IF REG2 = 1A GOTO 6              Branch if PRIOR pressed
   DPY-+#                           Beep if other key pressed
   GOTO 3                           Branch and wait for key
5: LABEL 5                          Arrive here if MORE pressed
   INC REG1                         Set to next address
   GOTO 2
6: LABEL 6                          Arrive here if PRIOR pressed
   DEC REG1                         Set to previous address
   GOTO 2
```

In the preceding program, notice that pressing any of the asynchronous input keys besides ENTER, MORE, and PRIOR causes a beep. This is accomplished by the DPY-+# step below Label 4. When this program is executed, it is also possible to cause the 9010A to beep if the MORE or PRIOR key is pressed very rapidly. This is because initially pressing the MORE or PRIOR key causes the asynchronous input to be disabled until the program control loops back to Label 2 and performs the step (DPY-+%2) which enables asynchronous input again. If any asynchronous key is pressed during the time the asynchronous input is disabled, the 9010A beeps as usual.

The following program demonstrates the ability of the 9010A to perform operations and yet still be responsive to asynchronous input from the keyboard at the same time. The keys that are defined by this program and may be used during execution are as follows:

MORE	Step through addresses in ascending order, reading and displaying data
PRIOR	Step through addresses in descending order, reading and displaying data
ENTER	Prompt for new starting address

Other Keys Beep when pressed (except for STOP, HIGH, and LOW)

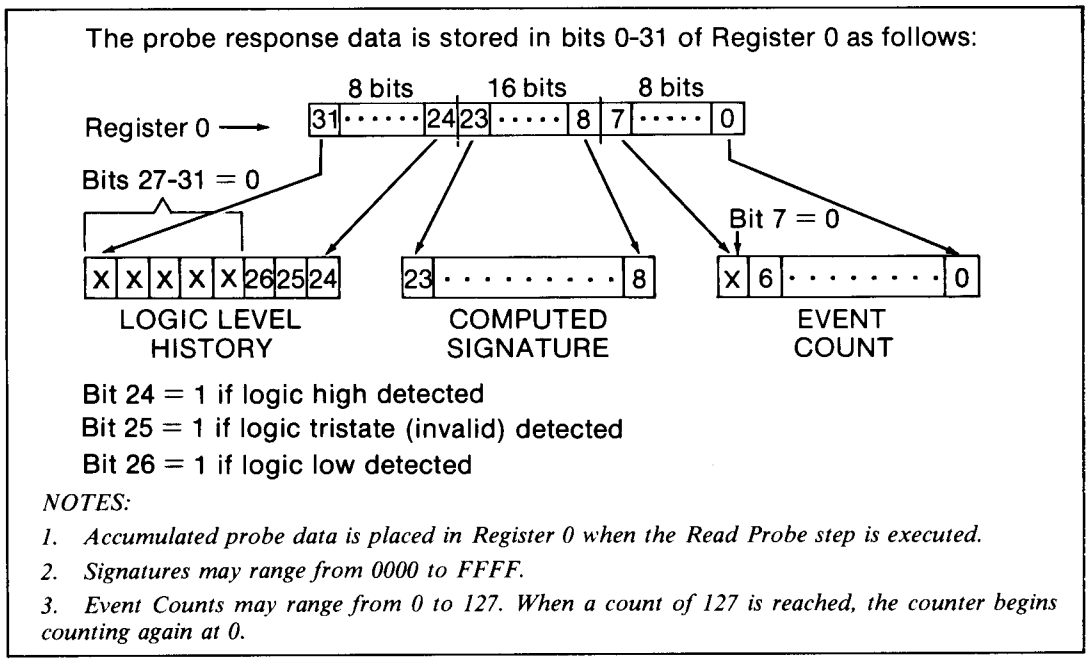
```
1: LABEL 1
   REG0 = 0                          Up/down counter set to up
   DPY-ADDRESS /1                    Prompt for starting address
2: LABEL 2
   REG2 = 40                         No key has this value
   DPY-+%2                           Enable async input for Reg 2
3: LABEL 3
   IF REG2 = 40 GOTO 6               Branch if no key pressed
   IF REG2 = 1C GOTO 1               Branch if ENTER pressed
   IF REG2 = 1B GOTO 4               Branch if MORE pressed
   IF REG2 = 1A GOTO 5               Branch if PRIOR pressed
   DPY-+#                            Beep if other key pressed
   GOTO 2                            Branch to restart
4: LABEL 4
   REG0 = 0                          Up/down counter set to up
   GOTO 2                            Branch to restart
5: LABEL 5
   REG0 = 1                          Up/down counter set to down
   GOTO 2                            Branch to restart
6: LABEL 6
   IF REG0 = 1 GOTO 7                Branch if counter set to down
   INC REG1                          Increment reg with address
   GOTO 8
7: LABEL 7
   DEC REG1                          Decrement reg with address
8: LABEL 8
   READ @ REG1                       Read address in Reg 1
   DPY-ADDRESS $1 DATA $E            Display address and data
   REG6 = 10                         Delay loop count
9: LABEL 9
   DEC REG6
   IF REG6 > 0 GOTO 9                Delay loop branch
   GOTO 3
```

In the preceding program, the 9010A steps through addresses and displays the data in either ascending order (selected by pressing the MORE key) or descending order (selected by pressing the PRIOR key). When the program execution begins, the 9010A prompts for a starting address. A new starting address may be specified by pressing the ENTER key and then entering the address. Register 1 is used to store the address that is being read. Register 0 is used as an up/down counter. When Register 0 is zero, the 9010A steps through addresses in ascending order, and when Register 0 is one, the 9010A steps through addresses in descending order.

5-17. USING THE PROBE IN PROGRAMS

The READ PROBE key and the SYNC key may each be used to create program steps which control probe operation in programs. The probe stimulus capabilities controlled by the HIGH and LOW keys are not subject to program control during program execution, although they are still available for use and may be manually controlled by the operator.

The use of the Read Probe step and the Sync step are discussed in the following paragraphs. The discussion is divided into three parts, with one part for each of the three types of information available with the Read Probe operation: the logic level history, the computed signature, and the event count. The sample programs included show how to set up the Sync and Read Probe steps, and how to obtain the data from Register 0 and display it. For reference, the format for the probe response data obtained with Read Probe is presented in Figure 5-1. Note that executing the Read Probe step also extinguishes the indicator lights in the probe.



NOTES:

1. *Accumulated probe data is placed in Register 0 when the Read Probe step is executed.*
2. *Signatures may range from 0000 to FFFF.*
3. *Event Counts may range from 0 to 127. When a count of 127 is reached, the counter begins counting again at 0.*

Figure 5-1. Probe Data Format for Register 0

5-18. Using the Logic Level History

A basic application of the Read Probe step involves the determination of the logic level of a static node, as shown in the following program:

```
    SYNC FREE-RUN                          Set the Sync mode
    DPY-PUT PROBE @ U6 PIN12               Operator instruction
    DPY-+-PRESS CONT#                      Bell (#) calls attention
    STOP                                   Wait for CONT
    READ PROBE                             Clears previous probe data
    READ PROBE                             Places data in Reg 0
    DPY-NODE LOGIC LEVEL WAS               First part of message
    IF REG0 AND 1000000 ) 0 GOTO 1         Check for high logic level
    IF REG0 AND 2000000 ) 0 GOTO 2         Check for invalid logic level
    DPY-+ LOW                              Remaining choice is low
    GOTO 3
1:  LABEL 1
    DPY-+ HIGH
    GOTO 3
2:  LABEL 2
    DPY-+ INVALID
3:  LABEL 3
```

The second and third steps of the preceding program instruct the operator how to set up the Read Probe operation. The beep (# symbol) in the third step calls the operator's attention to the instruction. Two Read Probe steps are specified, the first to clear previous probe information from Register 0, and the second to place the accumulated data in Register 0. Since the node is static, no stimulus is needed between the two Read Probe steps. The two If steps determine what the logic state was at the node.

The first step in the preceding program sets the synchronization mode to free-run. Note that the synchronization mode setting is maintained when passing between the Immediate Mode and the Executing Mode.

The following program can be used to identify address lines on the bus:

```
    SYNC ADDRESS
1:  LABEL 1
    DPY-PLACE PROBE AND PRESS CONT      Operator instruction
    STOP                                Wait for CONT
    REG1 = 1                            Set address reg to 1
    REG2 = 0                            Set bit counter to 0
2:  LABEL 2
    READ PROBE                          Clear previous probe data
    READ @ REG1                         Read stimulates address bit
    READ PROBE                          Places data in Reg 0
    IF REG0 AND 1000000 > 0 GOTO 3      Check if probe detected bit
    IF REG2 = F GOTO 4                  Check bit count
    SHL REG1                            Move to next address bit
    INC REG2                            Increment bit counter
    GOTO 2
3:  LABEL 3
    DPY-PROBE ON ADDRESS BIT @2         Display probe location
    GOTO 5
4:  LABEL 4
    DPY-NOT AN ADDRESS LINE             No bits found high by probe
5:  LABEL 5
    DPY-+-NEW PT?3                      Prompt for repeat
    IF REG3 = 1 GOTO 1                  Branch and repeat if YES
    DPY-PROGRAM COMPLETE
```

To use the preceding program, place the probe on some random address bit when prompted. The program stimulates the address bits beginning with bit 0 and continuing through all 16 bits, checking each time to see if the probe detected the high bit. Register 1 contains the address where the Read operation is performed, and Register 2 keeps track of the bit number (the microprocessor is assumed to have 16 address lines). The address synchronization is specified to limit the probe data accumulation to the address activity. The technique shown in this program can be useful for checking the address decoding circuitry.

5-19. Gathering Signatures

The following program shows how to isolate and display the signature. Assume that the probe is already placed when the program is executed.

```
   SYNC DATA
   READ PROBE                          Clears previous probe data
   RAMP @ 5000                         Provides stimulus for sig
   READ PROBE                          Places data in Reg 0
   REG0 = REG0 SHR SHR SHR SHR         Removes first hex digit
   REG0 = REG0 SHR SHR SHR SHR         Removes second hex digit
   REG0 = REG0 AND FFFF                Masks logic level information
   DPY-SIGNATURE IS $0
```

In the preceding program, the eight shift-right operations remove the two event count digits, and the logical AND operation masks off the logic level digit, leaving the four-digit signature in Register 0 as desired.

The following program shows how to gather a signature for an 8-bit microprocessor and display the results:

```
   SYNC DATA
   DPY-PROBE ON DATA BIT 0             Data bit 0 selected
   DPY-+-PRESS CONT
   STOP                                Wait for CONT
   READ PROBE
   RAMP @ 4000
   READ PROBE
   REG0 = REG0 SHR SHR SHR SHR         Removes first hex digit
   REG0 = REG0 SHR SHR SHR SHR         Removes second hex digit
   REG0 = REG0 AND FFFF                Masks off logic level
   IF REG0 = 96EC GOTO 1               Compares signatures
   DPY-SIG INCORRECT-
   DPY-+WAS $0 NOT 96EC
   GOTO 2
1: LABEL 1
   DPY-SIG CORRECT-$0
2: LABEL 2
```

In the preceding program the operator is prompted to place the probe on data bit 0. The Read Probe operation is performed with the Ramp function providing the stimulus. The signature is isolated and compared with the expected signature. For an 8-bit microprocessor, a Ramp function produces a signature of 96EC at data bit 0.

5-20. Using the Event Count

The following program shows how the event count may be used to test an I/O device:

```
    DPY-PUT PROBE @ U9 PIN7          Probe placed on I/O output
    DPY-+-PRESS CONT
    STOP                             Wait for CONT
    READ PROBE                       Clear previous probe data
    REG1 = 40                        Loop counter set to 64 (dec)
1:  LABEL 1
    DTOG @ C000 = 0 BIT 0            Toggle Data performed
    DEC REG1
    IF REG1 > 0 GOTO 1
    READ PROBE                       Places data in Reg 0
    REG0 = REG0 AND 7F               Isolates event count in Reg 0
    IF REG0 = 10 GOTO 2              Compares count
    DPY-COUNT INCORRECT-
    DPY-+WAS @0 NOT 16
    GOTO 3
2:  LABEL 2
    DPY-COUNT CORRECT-@0
3:  LABEL 3
```

In the preceding program the operator is instructed to place the probe at the output pin of the I/O device that is being tested. There is a divide-by-four counter on the output port so that the count at the input to the I/O device is divided by four at the output. The Toggle Data function is performed 64 times at the input, so a count of 16 is expected at the output. The event count is isolated and compared with the expected count and the results are displayed.

Note that the maximum number of counts that may be counted by the event counter is 127. After 127 counts, the counter starts over at 0.

5-21. USING RUN UUT IN PROGRAMS

Sometimes it is useful to execute program code out of UUT memory during programs, particularly if execution speed is important. The following programs show how to load a program into UUT RAM, and then execute the program with the Run UUT step. All of the following examples use the 8080 microprocessor instruction set and assembly language mnemonics. The machine instructions also execute properly on the 8085 and Z-80 microprocessors.

Since instructions are fetched from UUT RAM, it is important to verify that the UUT timing supports instruction fetches from RAM. Some processors, such as the Z-80, have tighter timing margins for instruction fetches than for data fetches.

5-22. Running a Program from UUT RAM

In this example, there is an output port at I/O address 20. Assume that bit 0 of this port drives additional circuitry, and that the normal operating software for the UUT generates short pulses on the line by setting it high and then low. When troubleshooting, it would be useful to have the 9010A generate similar pulses. The pulses can be generated using either the Write troubleshooting function:

```
WRITE @ 10020 = 1
WRITE @ 10020 = 0
```

Or the Toggle Data troubleshooting function:

```
DTOG @ 10020 = 0 BIT 0
```

The width of the resulting pulses is several milliseconds (the exact value depends on the type of interface pod and the UUT clock frequency). If the circuit being driven must have shorter pulses, then the troubleshooting functions cannot be used. Instead, a short program may be loaded into UUT RAM, then executed using the Run UUT function. The program can contain the same instructions as the normal operating program, thus providing an operator-controlled stimulus that duplicates normal operating conditions. The following assembly language program generates a short pulse on bit 0 of port 20.

```
8000 3E 01 MVI A, 1      ; set accumulator to 1
8002 D3 20 OUT 20        ; raise the line
8004 AF    XRA A         ; set accumulator to zero
8005 D3 20 OUT 20        ; lower the line
8007 76    HLT
```

The following 9010A program loads and executes the program at address 8000. Since the assembly language program does not contain any absolute addresses, it may be loaded and executed at any address.

```
WRITE @ 8000 = 3E
WRITE @ 8001 = 01
WRITE @ 8002 = D3
WRITE @ 8003 = 20
WRITE @ 8004 = AF
WRITE @ 8005 = D3
WRITE @ 8006 = 20
WRITE @ 8007 = 76
RUN UUT @ 8000
```

Each time the preceding 9010A program is executed, the machine instructions are loaded into UUT RAM and executed, generating one pulse. Note that once the assembly language program is loaded into UUT RAM, it may be executed again without reloading.

In the following program, the same assembly language program is used, but the 9010A program is more flexible, allowing the machine instructions to be loaded and executed at any address. Register 8 is assumed to contain the RAM address at which to load and execute.

```
WRITE @ REG8 = 3E
WRITE @ REGF INC = 1
WRITE @ REGF INC = D3
WRITE @ REGF INC = 20
WRITE @ REGF INC = AF
WRITE @ REGF INC = D3
WRITE @ REGF INC = 20
WRITE @ REGF INC = 76
RUN UUT @ REG8
```

5-23. Communicating With a Machine Language Program in UUT RAM

The following program shows one method for allowing a 9010A program to communicate with a downloaded machine language program. The machine language program reads the data at an operator-specified address, then stores it at a known address. The 9010A may then gain access to the stored data using the Read function.

The machine language program is as follows:

```
8000 3A bb aa LDA aabb
8003 32 07 80 STA 8007
8006 76       HLT
8007 00       DB  0
```

The 9010A program is as follows:

```
    WRITE @ 8000 = 3A                    LDA instruction
    WRITE @ 8003 = 32                    STA instruction
    WRITE @ 8004 = 07                    low byte of address 8007
    WRITE @ 8005 = 80                    high byte of address 8007
    WRITE @ 8006 = 76                    HLT instruction
1:  LABEL 1
    DPY-ADDRESS /1                       Prompt for and read address
    WRITE @ 8001 = REG1 AND FF           Fill in low byte of LDA address
    REG1 = REG1 SHR SHR SHR SHR          Shift Reg 1 four places right
    REG1 = REG1 SHR SHR SHR SHR          Shift Reg 1 four places right
    WRITE @ 8002 = REG1 AND FF           Fill high byte of LDA address
    RUN UUT @ 8000                       Execute UUT program
    READ @ 8007                          Get result of UUT program
    DPY-+ DATA $E                        Display result
    STOP                                 Wait for CONT
    GOTO 1
```

The first five steps load all of the program except the address in the LDA instruction. The operator is then prompted for an address. The operator-supplied address is loaded into the second and third bytes of the LDA instruction; the program is then executed using the Run UUT step. After the Run UUT step is executed and the program in UUT RAM is being run by the interface pod, the 9010A executes the program steps following the Run UUT step. Once the interface pod is placed in the Run UUT mode, the 9010A does not communicate with the interface pod. If a subsequent step requires use of the interface pod, the interface pod is automatically removed from the Run UUT mode to begin communicating with the 9010A, even if the program loaded in the UUT is not completed.

In the preceding 9010A program, the step READ @ 8007 removes the interface pod from the Run UUT mode. The data from 8007 (which was stored by the machine language program) is then appended to the display. Note that if the program loaded in the UUT must run for more than a few milliseconds, a delay should be placed between the Run UUT step and the first step that uses the interface pod. Otherwise, the interface pod will be removed from the Run UUT mode before the machine language program has completed execution.

5-24. USING THE AUX I/F IN PROGRAMS

Like the Display step, the AUX I/F step provides a way for external input and output during program execution. With the AUX I/F step, however, the input and output take

place through the RS-232 interface. The input may be single bytes which are stored in a programmer-designated register. The output may include text and register contents. Input and output are described in the following paragraphs.

AUX I/F data is transmitted in the same way in the Executing Mode as in the Immediate Mode with one exception: the line size is not controlled by the Setup LINESIZE parameter. The line size is determined by the length of the text in the AUX I/F step. A line terminator (as determined by the NEWLINE Setup parameter) is sent at the end of each AUX I/F step unless the + symbol is the last character in the AUX I/F step.

5-25. Output to the AUX I/F

Operator output via the AUX I/F can have a variety of uses. The following program demonstrates a typical application: logging test results on a printer. The program is used in testing a printed circuit board which contains an 8-bit analog-to-digital converter which provides digital output at address 8000.

```
1: LABEL 1
   REGA = 1                              Reg for voltage settings
   DPY-ENTER SERIAL NUMBER \1            Prompt for information
   DPY-CONNECT POWER-PRESS CONT          Operator instruction
   STOP
   AUX-SERIAL NUMBER - @1                Serial number printed out
   AUX-                                  Insert blank line in table
   AUX-VOLTAGE (V)  DATA                 Heading for table
   AUX-----------------                  Dashed line under heading
2: LABEL 2
   DPY-SET TO @A VOLTS-PRESS CONT        Operator instruction
   STOP
   READ @ 8000                           Read digital side of a/d
   AUX-   @A          $E                 Print setting and data
   INC REGA                              Increment to next setting
   IF 9 >= REGA GOTO 2                   Branch back for next setting
   DPY-DISCONNECT POWER                  Operator instruction
   DPY-+-PRESS CONT#
   STOP
   GOTO 1                                Repeat test for next pcb
```

During the execution of the preceding program, the operator receives instructions through the 9010A display. The operator is instructed to: (1) enter the serial number for the printed circuit board, (2) connect a power supply to the analog input of the analog-to-digital converter, and (3) set the power supply voltages.

The voltage settings are stored in Register A, which is initially set to 1. After the operator sets the voltage, the 9010A reads the digital output of the analog-to-digital converter (at address 8000) and sends the voltage setting and the data to the printer via the AUX I/F. This process is repeated for eight more voltage settings. After the test is completed the operator is instructed to disconnect the power supply, and the test branches back to the beginning for the next printed circuit board.

The following program demonstrates how to read and list the contents of UUT memory:

```
   DPY-FIRST /1  LAST /2          Prompt for address range
   REG1 = REG1 AND FFF0           Last byte 0 in first address
   AUX-                           Ensures printer at left side
1: LABEL 1
   IF REG1 AND F ) 0 GOTO 3       Selects every sixteenth address
   AUX-
   IF REG1 ) FFF GOTO 2
   AUX-0+
   IF REG1 ) FF GOTO 2            Formats and prints address
   AUX-0+
   IF REG1 ) F GOTO 2
   AUX-0+
2: LABEL 2
   AUX-$1+
3: LABEL 3
   AUX- +                         Inserts single space
   IF REG1 AND 7 ) 0 GOTO 4
   AUX-  +                        Inserts two spaces
4: LABEL 4
   READ @ REG1                    Reads data
   IF REGE ) F GOTO 5
   AUX-0+                         Inserts 0 if data F or less
5: LABEL 5
   AUX-$E+                        Sends data to AUX I/F
   INC REG1                       Increment to next address
   IF REG2 )= REG1 GOTO 1         Branch back if not finished
   AUX-                           Returns printer to left side
```

When the preceding program is executed, the 9010A prompts for the first and last address. After the operator enters the addresses, the 9010A reads the data at the addresses and sends the data to the AUX I/F. Figure 5-2 is a sample printout using this program with addresses 0100 through 01FF. The first column at the left of Figure 5-2 lists every sixteenth address. The rows beside the column of adresses list the data for the addresses.

```
0100   17 CA E9 01 35 6E 29 91   43 75 C1 B4 62 94 5C 21
0110   13 46 D4 63 87 33 A9 40   81 4E 9F 60 03 49 0D 34
0120   DF 79 E6 FE 4F C9 2A E4   8F CD 3A 0D 79 17 D2 44
0130   3A DF 8F 5F CD 4E 02 CD   6E 02 F1 2A B4 05 39 38
0140   37 01 22 E4 8F 7B 32 DF   8F C3 1D 01 E1 3E 10 32
0150   D8 8F 2A EC 8F 16 11 CD   48 02 CD 6E 02 FA 07 06
0160   1C 1B 15 14 14 79 E6 02   CA 80 01 35 6D 56 81 90
0170   13 B7 F4 53 28 9A EA 59   22 EC 7A 36 71 23 04 D9
0180   12 42 90 F8 C5 3A 72 66   81 30 02 55 A7 83 FE 70
0190   2A EC 8F 41 68 03 A5 CA   97 63 5B 88 14 52 74 0A
01A0   02 5E E1 CD 4E 02 CD 6E   02 F0 07 55 15 1B 24 24
01B0   2B FE 01 CA 5C 01 E5 21   48 05 06 08 CD 61 02 D8
01C0   50 7A 31 68 A3 8E 42 74   9B 83 41 29 7C 89 42 63
01D0   24 14 BA E5 04 22 46 F9   8B 67 88 92 D9 07 35 66
01E0   54 02 36 78 F2 14 E6 81   26 71 A4 D5 62 83 0B DE
01F0   FE 4A 3B 87 40 91 06 B8   71 23 4A 74 89 03 A2 7B
```

Figure 5-2. Sample Output to AUX I/F From Program that Reads UUT Memory

Sometimes it is useful to send single byte information via the AUX I/F. For example, assume the programmer wants to send information to a printer which requires the ASCII character with the hexadecimal value 0C to do a form feed. The programmer can accomplish this with the following two steps:

```
REG6 = 0C
AUX-%6
```

The first step loads the hexadecimal value (0C) for the character into Register 6. When the second step is executed, the 9010A sends the lower order byte contained in Register 6 to the AUX I/F. This technique provides a way for the programmer to send the full range of ASCII characters to the AUX I/F.

5-26. Input from the AUX I/F

Input from the AUX I/F is similar to the synchronous input from the 9010A keyboard with a Display step. When an AUX I/F step is executed which contains the / symbol followed by a hexadecimal digit, the 9010A suspends program execution, waits for the next byte of data from the RS-232 interface, and places the byte in the designated register (the upper three bytes of the register equal zero). The following program demonstrates how the operator may interact with the 9010A via the AUX I/F during program execution. The program also demonstrates how the RS-232 status register may be used.

In this program, a keyboard and video display are connected to the AUX I/F. The operator may interact with the 9010A during the execution of the program by pressing any one of four keys which have functions defined as follows:

S key: Suspends program execution until the C key is pressed
C key: Continues program execution if suspended
R key: Restarts the program execution at the beginning
X key: Terminates the program execution

```
1: LABEL 1
   REG1 = 0                          Initial address set to 0
2: LABEL 2
   READ @ REG1                       Read at address
   AUX-ADDRESS $1 DATA $E            Send address and data to AUX
   INC REG1                          Increment address register
   AUX-\5+                           Read RS-232 status into Reg 5
   IF REG5 AND 8 = 0 GOTO 2          Check to see if key pressed
   AUX-/6+                           Place hex value of key in Reg 6
   IF REG6 = 53 GOTO 3               If key was S, branch to 3
   IF REG6 = 52 GOTO 1               If key was R, branch to 1
   IF REG6 = 58 GOTO 4               If key was X, branch to 4
   AUX-#+                            Key pressed was none of above
   GOTO 2
3: LABEL 3
   AUX/6+                            Wait for key, place in Reg 6
   IF REG6 = 43 GOTO 2               Program continues if C pressed
   IF REG6 = 52 GOTO 1
   IF REG6 = 58 GOTO 4
   AUX-#+
   GOTO 3
4: LABEL 4
   AUX-EXIT PROGRAM                  Final message if X pressed
```

When the preceding program is executed, the 9010A begins reading data at a sequence of addresses, beginning with address 0. After the data is read at an address, the 9010A sends the address and data to the AUX I/F. The register containing the address is incremented, and then the next two steps check to see if a key was pressed. The AUX-\5 step reads the status of the RS-232 into register 5. The next step (IF REG5 AND 8 = 0 GOTO 2) checks the fourth bit (bit 3) to see if it is 1. If the fourth bit is 1, then a character has been received and is waiting in the RS-232 receive buffer (meaning a key was pressed). The AUX-/6 step places the hexadecimal value of the key into Register 6, and the If steps that follow identify the key and branch to the appropriate place.

Section 6
Error Handling in the Executing Mode

6-1. INTRODUCTION

Any of the errors that may be detected and reported in the Immediate Mode may be detected and reported in the Executing Mode. These include the Timeout errors, UUT System errors, and Test errors. In addition, there are five fatal error messages that may be encountered only when executing or attempting to execute programs. All the possible errors are discussed in the following paragraphs.

6-2. TIMEOUT ERRORS, UUT SYSTEM ERRORS, AND TEST ERRORS

The detection and reporting of Timeout errors, UUT System errors, or Test errors is handled in the Executing Mode in a manner very similar to the Immediate Mode. The error messages are the same, and the operations performed by the 9010A when looping on the errors are the same. When an error is detected, program execution is suspended and the error is reported. During the reporting of or looping on errors, the behavior of the MORE, LOOPING, and STOPPED annunciators is the same as in the Immediate Mode.

For more details about the Timeout errors, UUT System errors, and Test errors, refer to the 9010A Operator Manual.

6-3. FATAL ERRORS

The five fatal error messages are called fatal because they abort the program that is being executed and return the 9010A to the Immediate Mode. There is no way to loop on a fatal error or disregard it and continue with program execution. The five fatal error messages are listed and described in Table 6-1.

Each fatal error message consists of two lines. The first line identifies the type of error. The second line consists of from one to eleven program numbers. If a single number is listed, the number indicates the program that was executing. If more than one number is listed, the program numbers trace the "calling path" for the programs involved. For example, assume that Program 1 calls Program 2 which then calls Program 1. When Program 1 is executed, the first line of the following two-line error message is presented on the display:

FATAL-ATTEMPTED RECURSION MORE annunciator begins flashing
01 02 01

The first line indicates that a program called a preceding program in the calling path. The second line indicates the order of programs that did the calling, i.e., Program 1 called Program 2 which called Program 1. The MORE and PRIOR keys may be used to bring the desired line to the display.

The last three error messages can occur only if the executing program contains an illegal Execute step. In this case, the next-to-last number in the calling path is the program that contains the illegal Execute step. The last number in the calling path is the program designated by the Execute step.

Table 6-1. Fatal Error Messages

FIRST LINE OF ERROR MESSAGE*	DESCRIPTION
FATAL-MEMORY EXCEEDED FOR LEARN	The address descriptors obtained during a Learn operation have exceeded the 9010A internal memory. This situation is described in the 9010A Operator Manual.
FATAL-NUMERIC VALUE OUT OF RANGE	A value (such as data or an address) was specified which is out of the valid range permitted for that quantity (such as a bit number greater than 31).
FATAL-ATTEMPTED RECURSION	A program attempted to call itself, or to call a program which preceded it in the calling path.
FATAL-DEPTH EXCEEDED	A program attempted to call another program more than ten levels deep. The maximum number of programs that can be involved in the calling path is ten.
FATAL-PROG NOT FOUND	A program was called which does not exist in memory.

* *The second line of the error message consists of the calling path for the programs involved.*

Section 7
Data Format for AUX I/F Immediate Mode Operation

7-1. INTRODUCTION

This section contains a detailed description of the data format used in the AUX I/F Write and AUX I/F Read operations. These operations allow information to be sent to and received from remote devices. The reader is assumed to be familiar with the Setup parameters and their meanings.

Note that the operator does not need this information to use the AUX I/F Write and AUX I/F Read operations to transfer 9010A-generated information from a 9010A to a remote device or to another 9010A. However, if the information is destined for a computer and subsequent manipulation, the information described in this section is required.

7-2. THE AUX I/F WRITE AND I/F READ OPERATIONS

When the AUX I/F Write operation is selected, the following information is sent to the RS-232 interface:

1. Setup parameters
2. Address space descriptors
3. All programs

Note that this is exactly the same data that is recorded on tape when the Write Tape operation is performed. In fact, the AUX I/F Write and AUX I/F Read operations can be thought of as RS-232 counterparts of the Write Tape and Read Tape functions. The main difference between the RS-232 port and the tape is that information sent to the RS-232 may be accessed and possibly modified by the user.

The AUX I/F Read operation attempts to read the same information back into the 9010A. The format of the incoming data is expected to be the same as that produced by the AUX I/F Write operation. As a result, it is possible to send information from one 9010A to another by connecting the RS-232 ports together, performing an AUX I/F Read on one unit and an AUX I/F Write on the other.

7-3. DATA FORMAT FOR AUX I/F WRITE

The AUX I/F Write operation sends information as a series of records. Each record begins with a colon and ends with the NEWLINE terminator (referred to as ⟨terminator⟩), as specified in Setup. There are two types of record formats, one for programs, and one for all other information. The second type consists of fixed-length records and is described in the following paragraphs. The record format for programs is described later.

Each fixed-length record contains specific data, e.g. a Setup parameter or an address descriptor. The general form of a fixed-length record is

:⟨record type⟩⟨data⟩⟨checksum⟩⟨terminator⟩

For example, the following record specifies a default Run UUT address of 12345678.

:06341278561A⟨terminator⟩

All characters between the colon and the terminator are interpreted as hexadecimal digits, with each pair of digits representing one eight-bit byte. The first byte (first two characters) is the record type. The next four bytes (next eight characters) are the data - in this case the Run UUT address. The last byte is the checksum. The checksum is computed by adding the record type and all data bytes and taking the two lower-order bytes of the sum. The number of data bytes is fixed for each record type, but varies with record type. Table 7-1 lists all fixed-length record types, their lengths (number of data bytes), and contents.

A record of the form :00⟨terminator⟩ is sent at the end of an AUX I/F Write operation.

7-4. Detailed Description of Fixed-Length Record Types

A detailed description of the data portion for each record type is given in the following paragraphs. References to bit numbers follow the convention of numbering bits from least to most significant, starting at zero.

Table 7-1. Fixed-Length Record Types

Record Type	Length (bytes)	Contents
01	1	Error mask for traps
02	1	Mask of enabled forcing lines
03	1	Beep on error flag
04	1	Exercise errors flag
05	4	Bus test address
06	4	Default Run UUT address
07	4	Stall character
08	4	Unstall character
09	4	Line size
0A	4	Timeout length
0B	4	NEWLINE information
0C	7	Pod name
0D	1	Mask of forcing lines that may be enabled
0E	28	Names of first 4 forcing lines that may be enabled
0F	28	Names of last 4 forcing lines that may be enabled
10-17	32	Reserved - data bytes will be zero for AUX I/F WRITE
18	—	Not used
19	18	One address descriptor
1A	1	Program number

7-5. TYPE 01 - ERROR MASK FOR TRAPS

The single data byte contains the Setup values for the seven "trap" parameters (e.g. "TRAP ADDR ERR"), one bit per parameter. A 1 indicates YES, and a 0 indicates NO. The bits are mapped as shown below.

BIT	SETUP PARAMETER
0	TRAP DATA ERR
1	TRAP ADDR ERR
2	TRAP CTL ERR
3	TRAP ACTIVE FORCE LINE
4	TRAP ACTIVE INTERRUPT
5	TRAP ILLEGAL ADDRESS
6	not used - always 1
7	TRAP BAD PWR SUPPLY

7-6. TYPE 02 - MASK OF ENABLED FORCING LINES

The data byte is a mask of the forcing lines that may be enabled which are enabled. Each bit corresponds to one Setup parameter of the form ENABLE xxxxxx. A 1 indicates that the line associated with that bit is enabled. See the discussion of record types 0D, 0E, and 0F for more information on forcing lines that may be enabled.

7-7. TYPE 03 - BEEP ON ERROR FLAG

A value of 01 for the data byte means that the BEEP ON ERR TRANSITION parameter is set to YES. A value of 00 means NO.

7-8. TYPE 04 - EXERCISE ERRORS FLAG

A value of 01 for the data byte means that the EXERCISE ERRORS parameter is set to YES. A value of 00 means NO.

7-9. TYPE 05 - BUS TEST ADDRESS

The four-byte data field contains the 32-bit Bus Test address. The data bytes map into the 32-bit address as follows:

first byte	address bits 16-23
second byte	address bits 24-31
third byte	address bits 0-7
fourth byte	address bits 8-15

This mapping of four data bytes into a 32-bit number is used for all four- byte records.

7-10. TYPES 06 THROUGH 0B

These records all have four data bytes which map into a 32-bit number. The mapping is identical to that for record type 05. Note that some parameters, such as STALL and UNSTALL, have upper limits which cause some of the data bytes to always be zero. Internally however, these parameters are treated as 32-bit numbers and will always be sent and received using four data bytes.

7-11. TYPE 0C - POD NAME

This record contains the name of the currently connected pod. The name may be up to six ASCII characters, with the first null byte terminating the name. If there is no pod connected at the time of an AUX I/F Write, the first byte of the data field will be 00. There are always seven data bytes, even if the pod name is less than six characters. In this case,

the bytes after the first null are meaningless. Note that each data byte (two characters) contains the ASCII value of one character in the pod name. The actual characters do not appear in the data stream.

7-12. TYPE 0D - MASK OF FORCING LINES THAT MAY BE ENABLED

This byte is a mask that is used with records 0E and 0F to determine which forcing lines may be individually enabled using Setup. There are up to eight such lines per pod, and they are different for each pod. Typical pods use only two or three of the bits. A 1 bit means that the corresponding line can be enabled, i.e., there will be a Setup entry of the form ENABLE xxxxxx, where xxxxxx is the name of the line. This byte should not be confused with the data byte in record type 02. A 1 bit in a 0D type record says 'this line can be enabled or disabled', whereas the corresponding bit in a type 02 record says 'it is enabled or disabled'. If the line cannot be enabled (record type 0D), the corresponding bit in record 02 is meaningless.

7-13. TYPES 0E, 0F - NAMES OF FORCING LINES THAT MAY BE ENABLED

These two records contain the names of the forcing lines that may be enabled. Each name may be up to six ASCII characters, terminated by a null byte. Therefore, each name occupies seven bytes in the record. (for names with less than six characters, the unused data bytes are meaningless) These are the names that appear in Setup entries of the form ENABLE xxxxxx. Record types 0E and 0F each contain four names, for a total of eight. There is a one-to-one correspondence between the names and the bits in the type 0D record.

BIT	LOCATION OF NAME
0	record 0E, bytes 0-6 (first 7 bytes)
1	record 0E, bytes 7-13
2	record 0E, bytes 14-20
3	record 0E, bytes 21-27 (last 7 bytes)
4	record 0F, bytes 0-6 (first 7 bytes)
5	record 0F, bytes 7-13
6	record 0F, bytes 14-20
7	record 0F, bytes 21-27 (last 7 bytes)

If a bit in the 0D record is zero, the corresponding name is meaningless.

7-14. TYPES 10-17 - RESERVED

These records are reserved for use by the John Fluke Mfg. Co., Inc. At present, all data bytes are output as 00.

7-15. TYPE 19 - ADDRESS DESCRIPTOR

Each type 19 record contains the information for one address descriptor. There may be up to 100 such records.

BYTE(S)	CONTENTS
0-3	32-bit low address of block, 4-byte format
4-7	32-bit high address of block, 4-byte format
8	block type (01 = I/O, 02 = RAM, 03 = ROM)
9-13	not used
14-17	dependent on block type I/O - 32-bit Read/Write mask, 4-byte format RAM - not used ROM - bytes 14-15 are ROM Sig, least significant byte first

7-16. TYPE 1A - PROGRAM NUMBER

This record contains only one data byte in the following format:

:1Ahh⟨checksum⟩ ⟨terminator⟩

The data byte hh is the number of a program. The actual program is contained in one or more program records which immediately follow the type 1A record.

7-17. Detailed Description of Format for Program Records

Program records (those records that are listed following a type 1A record) are actually a keystroke representation of a program. Each byte (two hex digits) corresponds to one key. For example, the step WRITE @ 123 = 45 is produced by the keystroke sequence WRITE 1 2 3 ENTER 4 5 ENTER, and would result in the following string of data bytes in a program record:

20 01 02 03 1C 04 05 1C

The numeric values associated with each key are shown in Table 7-2. Note that these are the same values that are used for asynchronous input. In addition, the following four special bytes may appear:

53 indicates the start-of-program, and is always the first byte in a program.
50 indicates the end-of-program, and is the last byte of a program (labels may follow, see below).
44 is used instead of 38 for REG when REG is the first key in a program step. For example, REG1 = REG2 would produce 44 01 38 02 1C.
7C indicates the end of the text string in a Display or AUX I/F program step. This byte appears instead of 1C, even though the step is ended with ENTER.

Table 7-2. Numeric Values For Keys in Program Records

VALUE	KEY	VALUE	KEY	VALUE	KEY	VALUE	KEY
0	0	10	LEARN	20	WRITE	30	AND
1	1	11	RAM VIEW	21	RAMP	31	OR
2	2	12	I/O VIEW	22	WALK	32	SHIFT LEFT
3	3	13	ROM VIEW	23	TOGGL ADDR	33	SHIFT RIGHT
4	4	14	AUTO TEST	24	TOGGL DATA	34	INCR
5	5	15	BUS TEST	25	CONT	35	DECR
6	6	16	ROM TEST	26	RPEAT	36	COMPL
7	7	17	RAM LONG	27	LOOP	37	EXEC
8	8	18	RAM SHORT	28	STOP	38	REG
9	9	19	I/O TEST	29	RUN UUT	39	READ PROBE
A	A	1A	PRIOR	2A	PROGM	3A	READ TAPE
B	B	1B	MORE	2B	LABEL	3B	WRITE TAPE
C	C	1C	ENTER/YES	2C	GOTO	3C	SYNC
D	D	1D	CLEAR/NO	2D	IF	3D	SETUP
E	E	1E	STS/CTL	2E	>	3E	DISPL
F	F	1F	READ	2F	=	3F	AUX I/F

NOTES:
1. *The keys have the same values that are used for asynchronous input.*
2. *The STOP key is included.*
3. *The HIGH and LOW keys are not included since they cannot be controlled by program steps.*

The ASCII characters in Display and AUX I/F steps have the usual ASCII codes, except that the eighth bit is always a one. For example, the ASCII code for "A" is 41, so in a Display or AUX I/F step it would appear as C1.

If a program contains one or more labels, three bytes per label are present immediately following the end of program byte (50). Each three-byte sequence has the format shown below:

first byte	label number (00-0F)
second byte	low byte of offset
third byte	high byte of offset

The offset is a 16-bit unsigned value equal to the byte position of the step following the label, relative to the start of the program. This is most easily shown with an example:

⟨start⟩	53
READ @ 12	1F 01 02 1C
LABEL 1	2B 01
READ @ 34	1F 03 04 1C
⟨end⟩	50
⟨label 1 description⟩	01 07 00

The step READ @ 34 follows LABEL 1, and is seven bytes beyond the beginning of the program.

7-18. PROCESSING OF RECORDS FOR AUX I/F READ

When AUX I/F Read is selected, the 9010A sends the terminator (selected by the NEWLINE Setup parameter), then reads records until the end-of-data record (:00) is encountered, or an error condition is detected. Although all of the previously defined record types are accepted, AUX I/F Read is most commonly used to download programs. The input records are processed according to the following rules:

1. Fixed-length records (types 01-19) may appear in any order.

2. It is not necessary to load all record types.

3. Programs must be loaded in numeric order.

4. If record types 10-17 are loaded, the data bytes should all be 00. Non-zero data will produce unpredictable results.

5. All characters between the end of a record and the next colon are ignored.

6. If there are multiple occurrences of the same record type, the last one is used. (This does not apply to types 19 and 1A.)

7. Program records may be any length, although extremely long records will result in weaker checksum protection.

8. Hex digits A through F must be in upper case only.

NOTE

Since the record contents are NOT checked when loaded, it is therefore possible to load meaningless information into the 9010A. This may produce unpredictable results, but cannot harm the instrument or UUT.

7-19. EXAMPLES OF DATA FORMAT FOR AUX I/F OPERATIONS

Figure 7-1 presents a sample printout of 9010A data for the AUX I/F Setup, AUX I/F Learn, and AUX I/F Program operations. For comparison, Figure 7-2 presents a sample printout of the same 9010A data for the AUX I/F Write operation. In Figure 7-2, notice that the first two characters in the records indicate the record type.

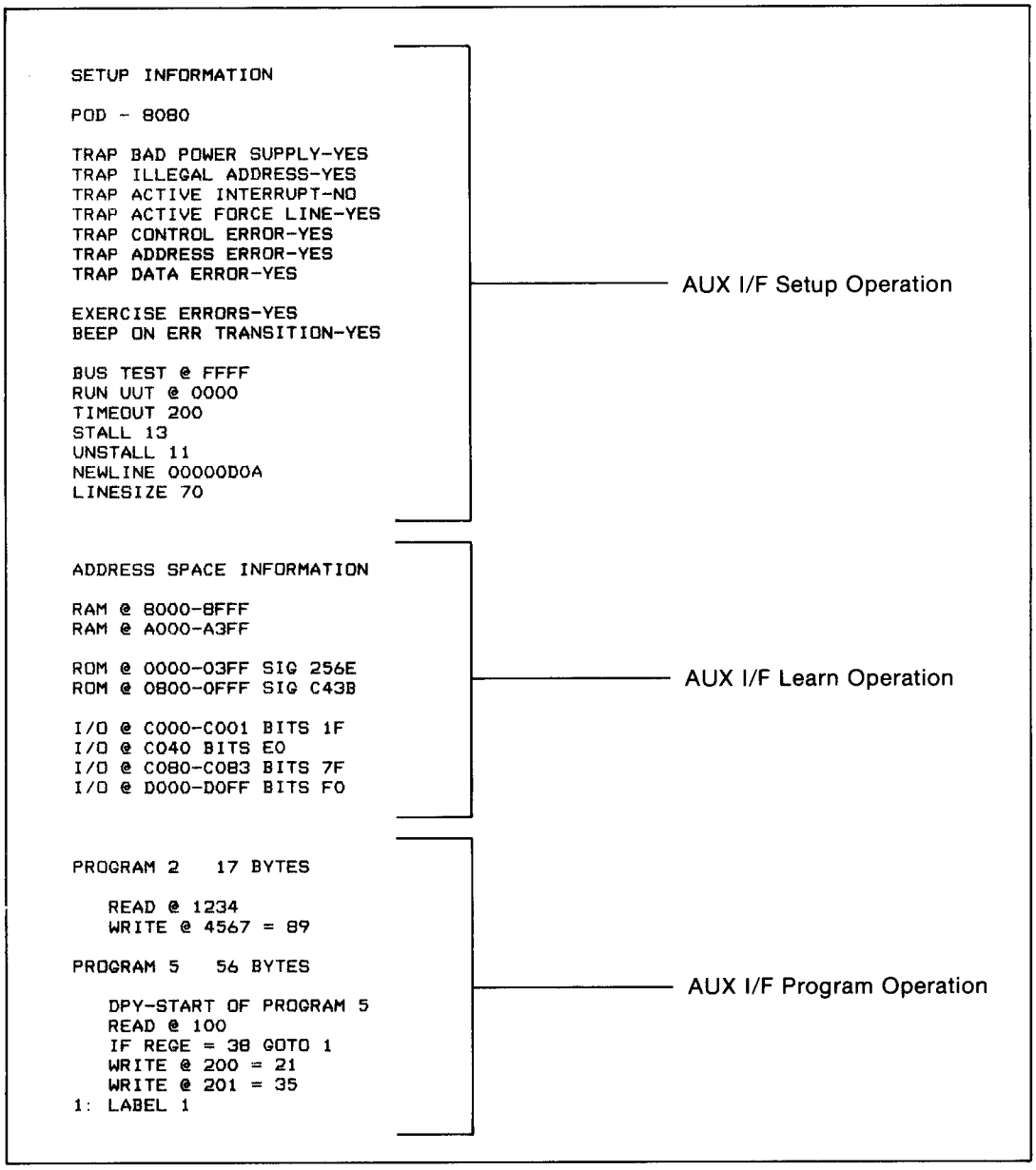


Figure 7-1. Data Format For AUX I/F Setup, AUX I/F Learn, and AUX I/F Program Operations

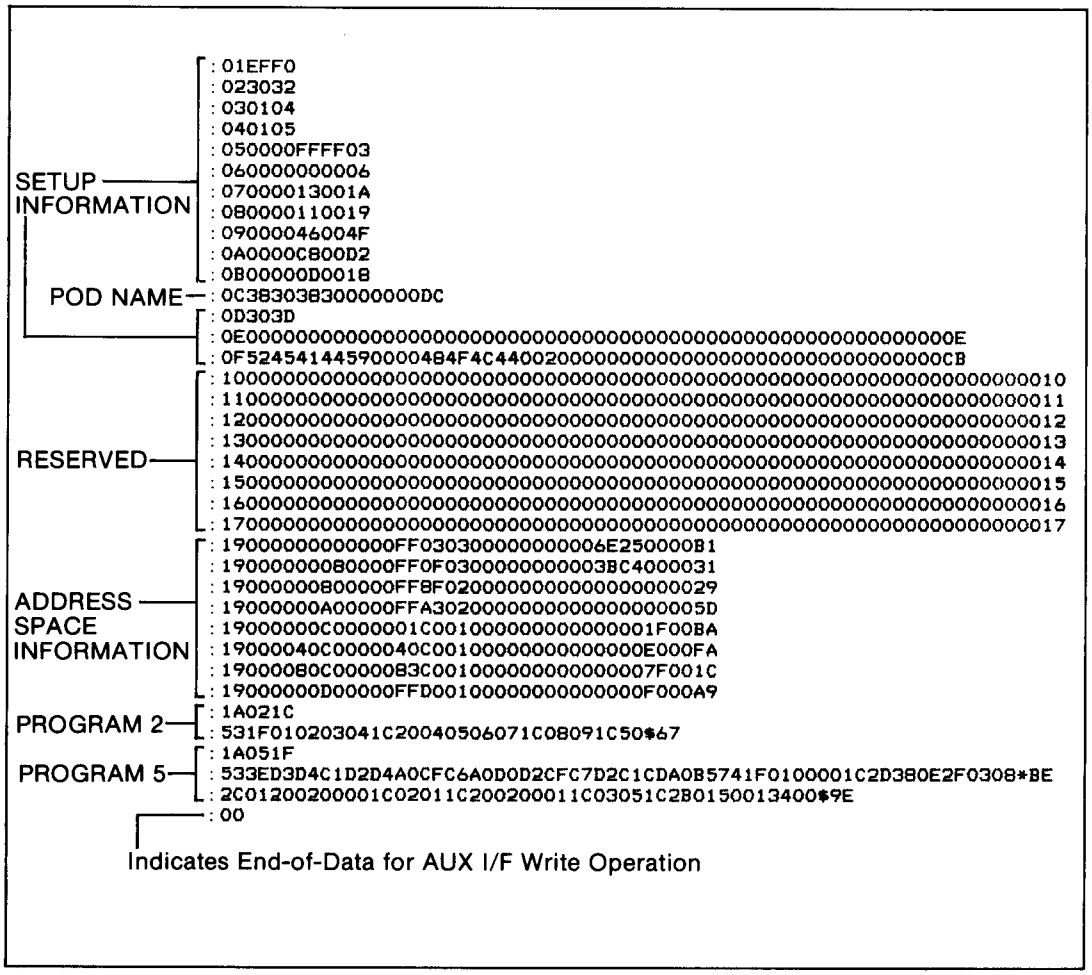


Figure 7-2. Data Format For AUX I/F Write Operation