9100 SERIES • SOLUTIONS

9100A SYSTEMS TRAINING — PART I

FLUKE

John Fluke Mfg. Co., Inc.

Customer Support Services
P.O. Box 9090 MS 239D
Everett, Washington 98206
(206) 347 6100 or
(800) 44-FLUKE Ext 73

Rev. November 20, 1991

Table of Contents

Section 1

Test and Troubleshooting Approaches

Section 2

System Configuration

Section 3

Understanding the UUT

Section 4

Bus Test

Section 5

RAM Testing

Section 6

ROM Testing

Section 7

Parallel Interface Adapter

Section 8

Fault Isolation

Section 9

Editor Operation

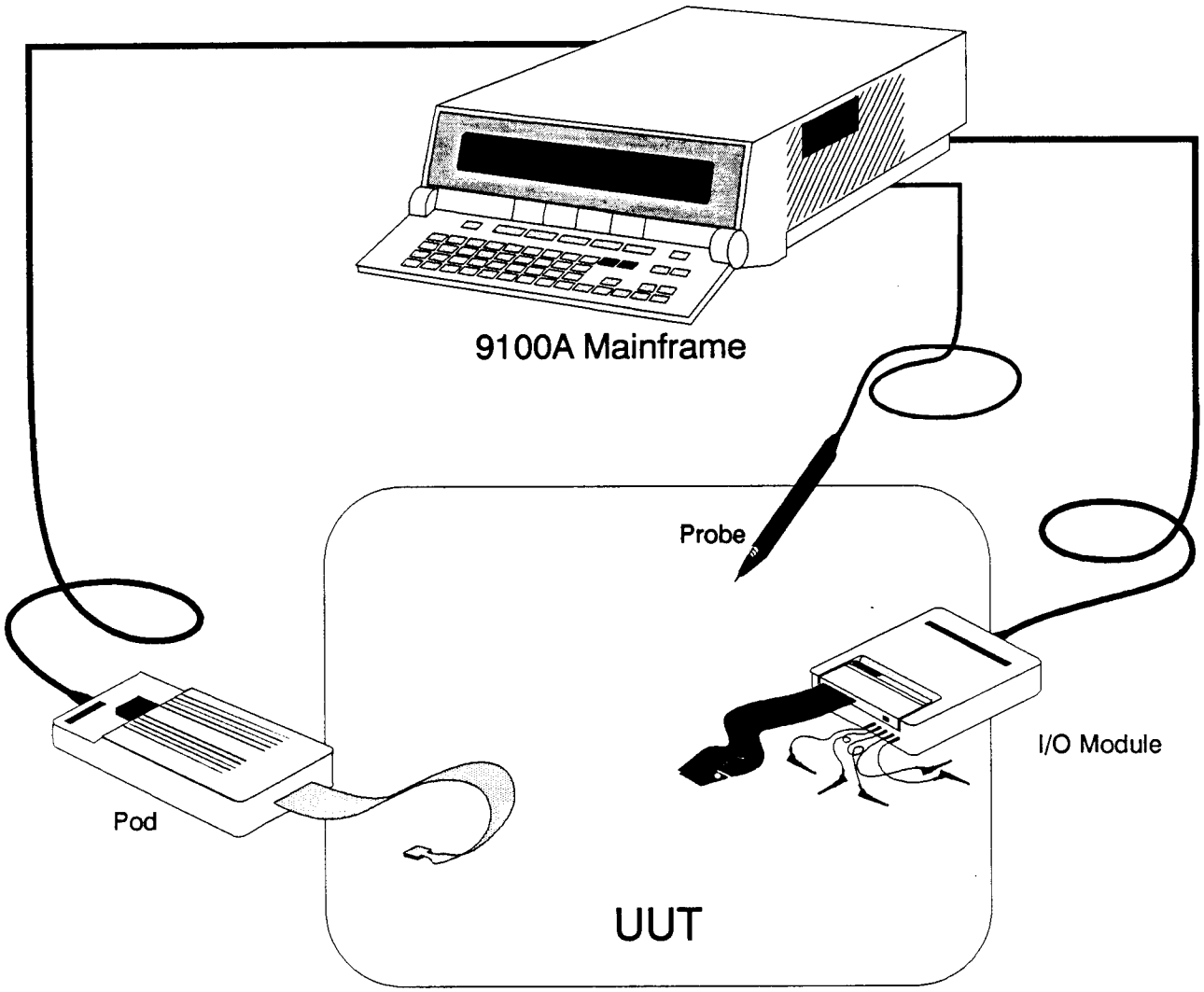
Section 10

Functional Test

Section 11

The TL/1 Test Language

Section 1
Test and Troubleshooting Approaches


9100A Mainframe
Probe
I/O Module
Pod
UUT

9100A System

The 9100A system is designed to test and troubleshoot digital electronic boards that are controlled by microprocessors. Testing precedes troubleshooting. If a board proves to be defective in a test, you can troubleshoot to locate the fault. The board under test is called a UUT (Unit Under Test).

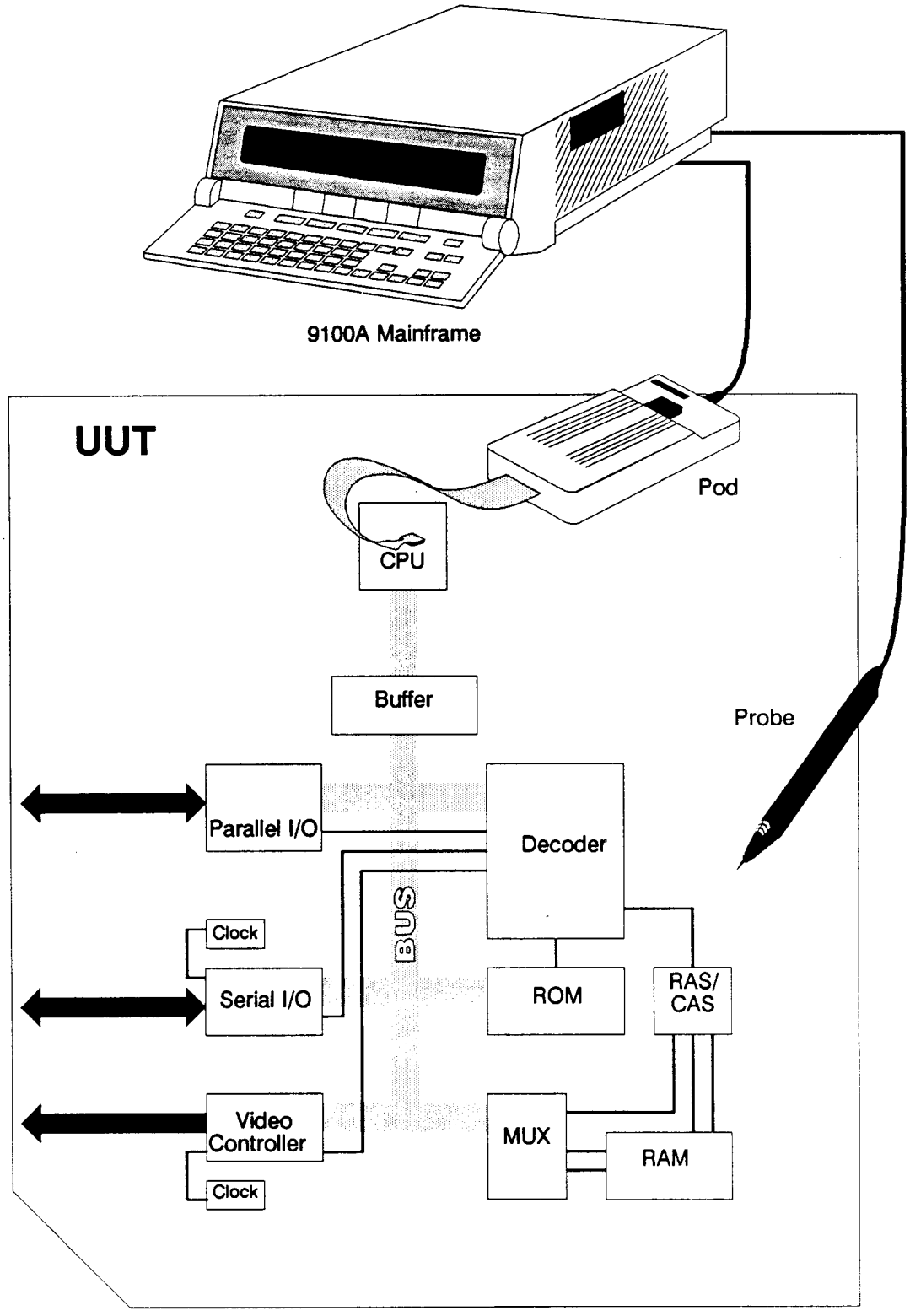
Testing and troubleshooting can be done in two ways:

Immediate Mode

The user typically employs a combination of automated procedures and manual keypad commands to test or troubleshoot a UUT. To do so, the user should be familiar with both the UUT and the test system.

Automated Mode

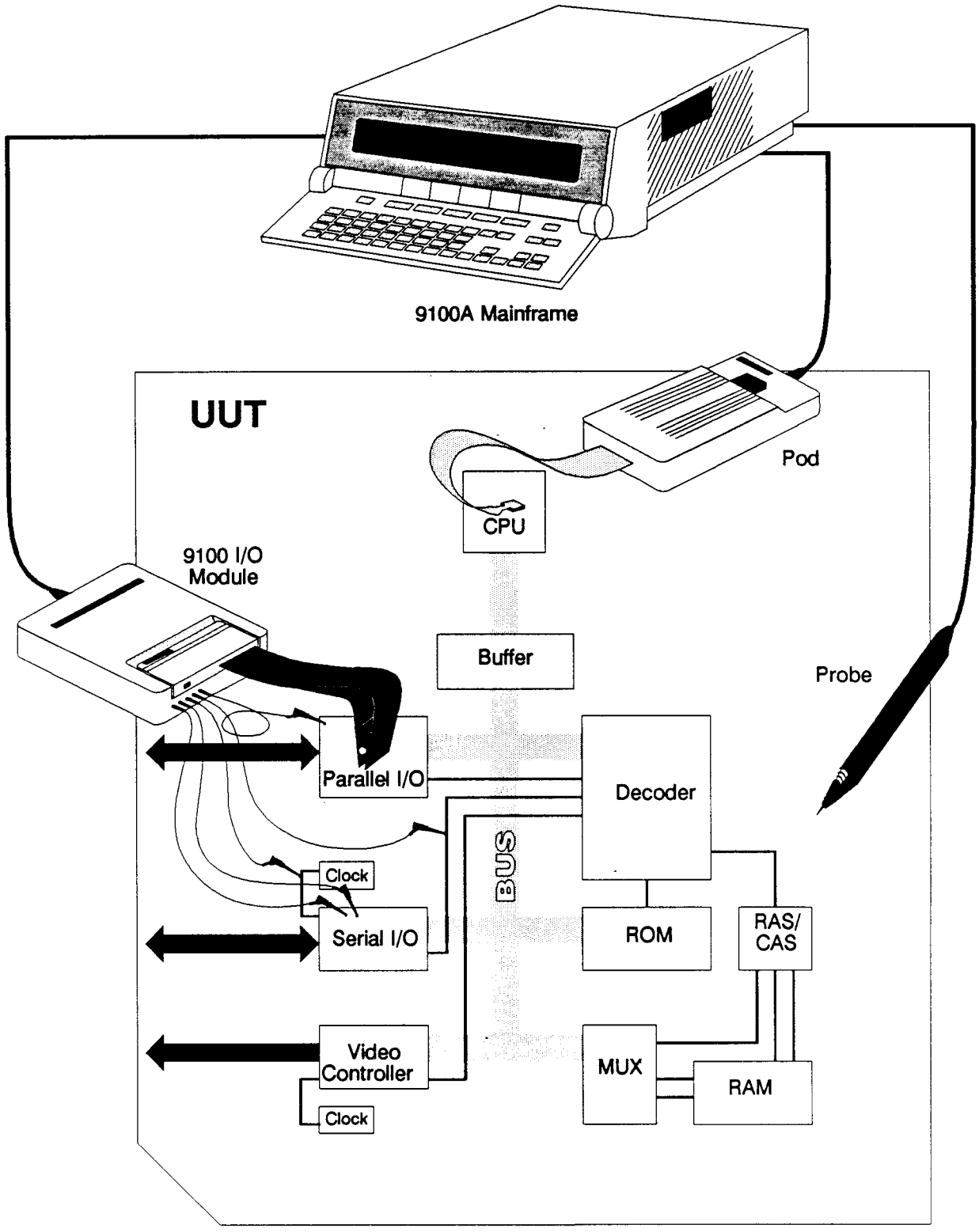
The user employs test or troubleshooting sequences that are stored on a user disk. Most often the user should be familiar with the basic methods of implementing test or fault isolation test sequences.


9100A Mainframe
UUT
Pod
CPU
Buffer
Probe
Parallel I/O
Decoder
BUS
Clock
Serial I/O
ROM
RAS/
CAS
Video
Controller
MUX
RAM
Clock

Emulative Testing

The operation between an Emulative Test system and a UUT is based on the capability of a microprocessor to read and write data at an address. The tester takes control of the UUT microprocessor bus and allows the operator to specify read and write operations anywhere in the UUT address space through a UUT connection at the microprocessor socket. All the tester's operations are derived from this fundamental ability to manipulate data at an address.

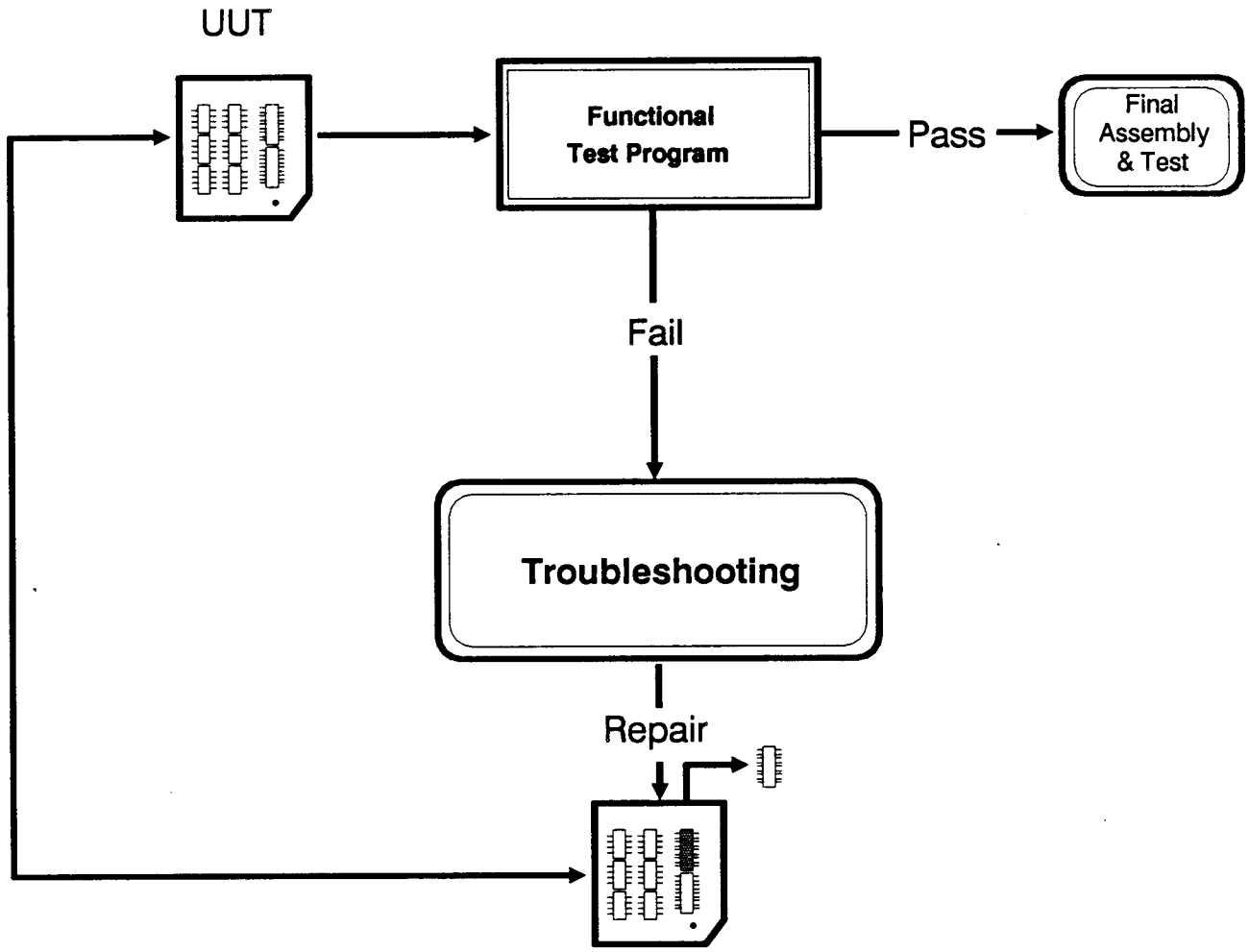
When the test system is connected to the UUT's microprocessor socket, the system takes over the digital activity normally done by the UUT's microprocessor. The test system exercises and tests ROM, RAM, I/O, or any other circuit that is related to the microprocessor bus. Some systems can also emulate the UUT's microprocessor and execute programs that reside in the UUT memory.


9100A Mainframe
UUT
Pod
CPU
9100 I/O
Module
Buffer
Probe
Parallel I/O
Decoder
BUS
Clock
Serial I/O
ROM
RAS/
CAS
Video
Controller
MUX
RAM
Clock

9100A Theory of Testing

The 9100-Series uses a microprocessor pod as the primary mode of stimulating the UUT. The pod either replaces the UUT microprocessor or clips over the processor if the processor can be tri-stated. Once the pod is in place, the entire kernel can be tested, including the bus, RAM, ROM, and any device that is accessible from the bus.

The 9100-Series is the first emulative-type tester that finds faults beyond the bus in a cost effective way. The 9100-Series uses a high speed probe and up to 160 stimulus/measurement lines. By adding the I/O Module and Probe to the Mainframe, testing can be performed on any portion of the UUT.


UUT
Functional
Test Program
Pass
Final
Assembly
& Test
Fail
Troubleshooting
Repair

9100A General Test and Troubleshooting Flow

The 9100A testing strategy identifies two areas in the test process:

- Functional Testing
- Troubleshooting

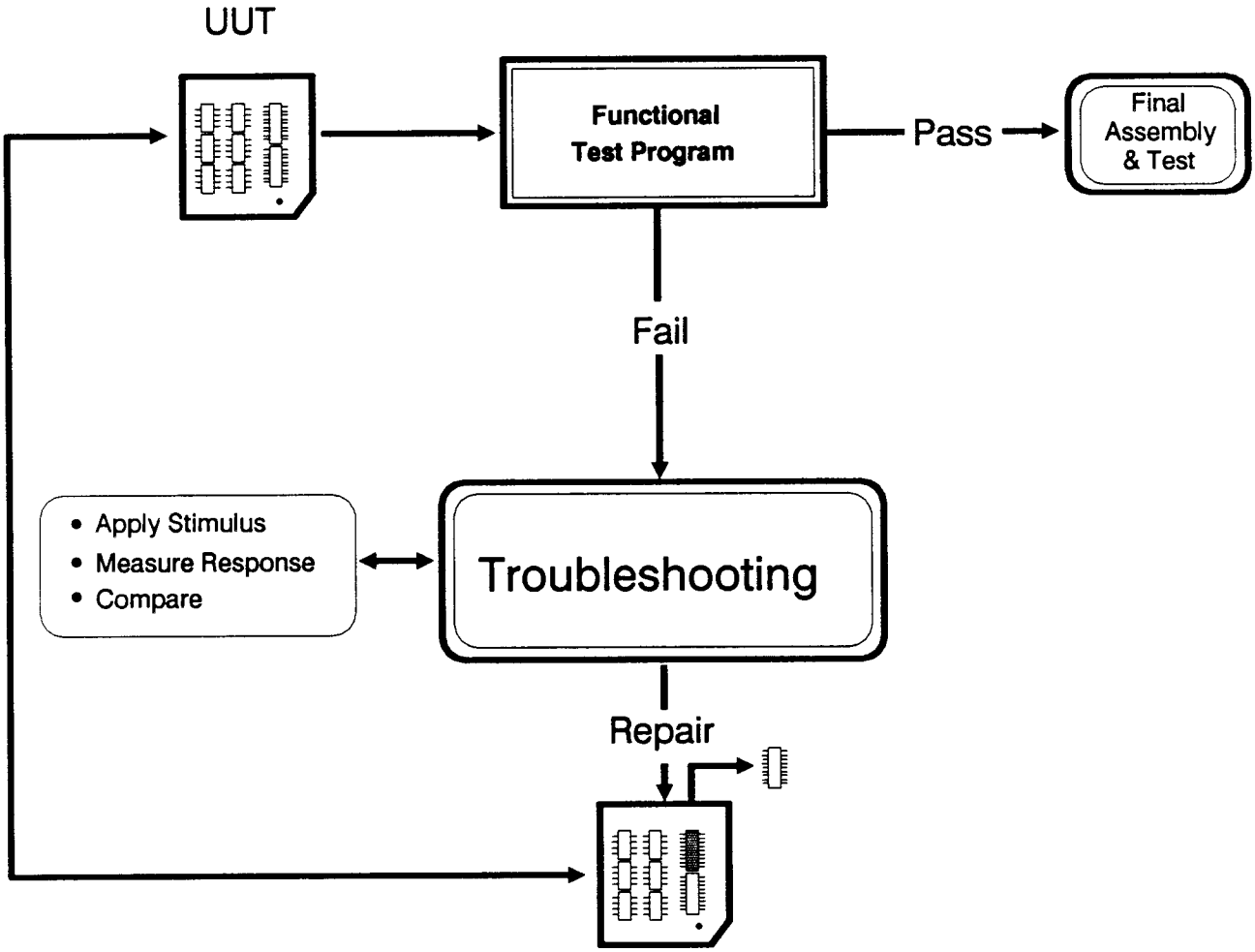
The functional test area could be subassembly, final assembly, or final test. At whatever stage the functional test is performed, the ultimate goal is to verify the proper operation of the UUT as quickly and as accurately as possible.

If a fault is encountered in any stage of functional testing, troubleshooting is the next step.

The 9100-Series test strategy identifies three modes of fault isolation:

- Application Keypad (Immediate Mode)
- Unguided Fault Isolation (UFI)
- Guided Fault Isolation (GFI)

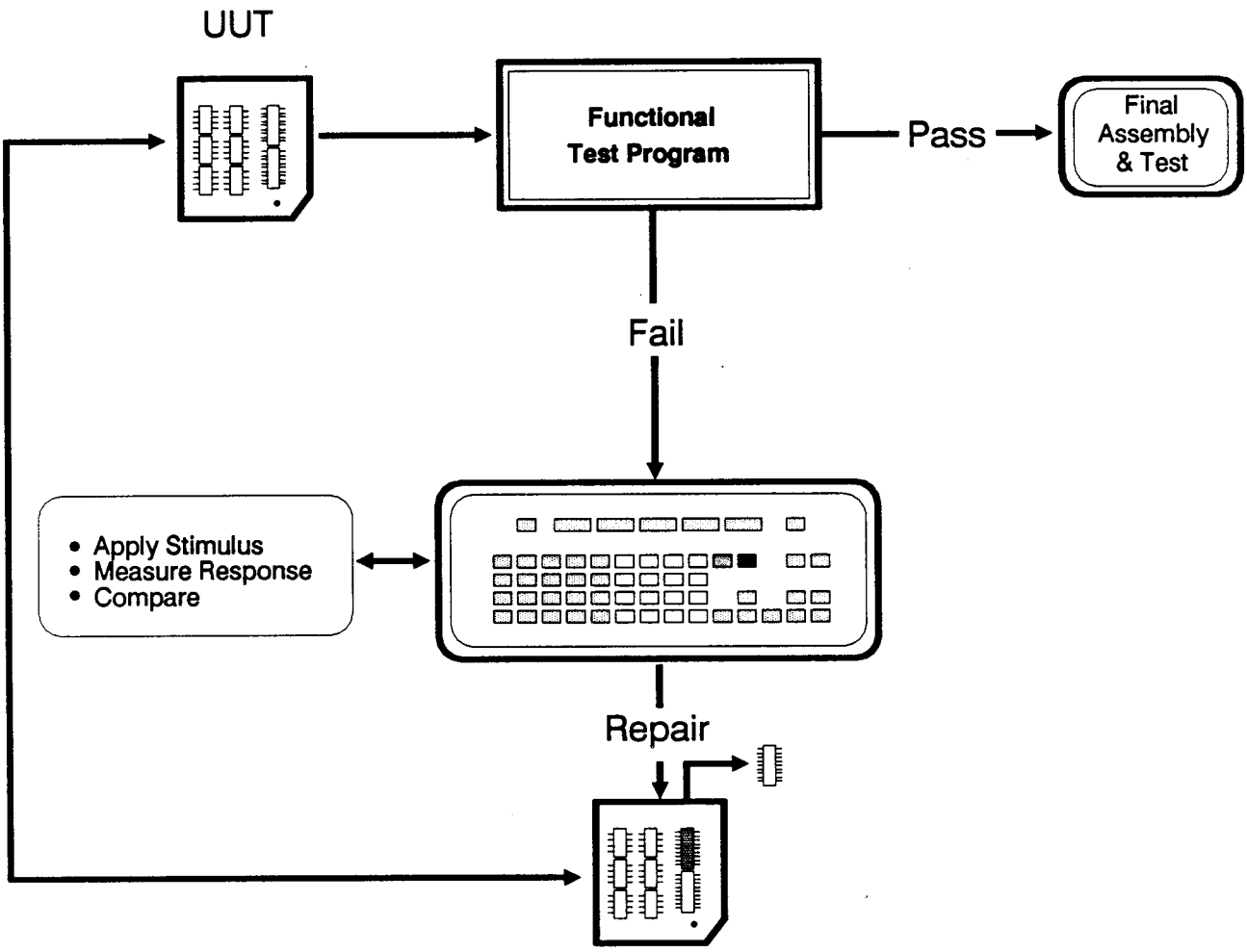
The Application Keypad method is a manual method of troubleshooting. The Unguided Fault Isolation method is semiautomated troubleshooting while Guided Fault Isolation is fully automated troubleshooting.


UUT
Functional
Test Program
Pass
Final
Assembly
& Test
Fail
• Apply Stimulus
• Measure Response
• Compare
Troubleshooting
Repair

Characteristics of Troubleshooting

The following characteristics of troubleshooting are the same for each mode of operation.

1. Stimulate the circuitry being tested.
2. Learn the known good responses that are created by the stimulus. This learned response information is either documented or remembered.
3. Measure the response on the board under test. The method of measurement could be as simple as a visual response or as complex as a multiple point measurement.
4. Determine if the measured response is good. This is done by comparing the measured response to a known good response.
5. Determine where to stimulate and measure next. This determination could be based on experience, electronics knowledge, knowledge of UUT operation, relationship of one circuit to the next, or even intuition.

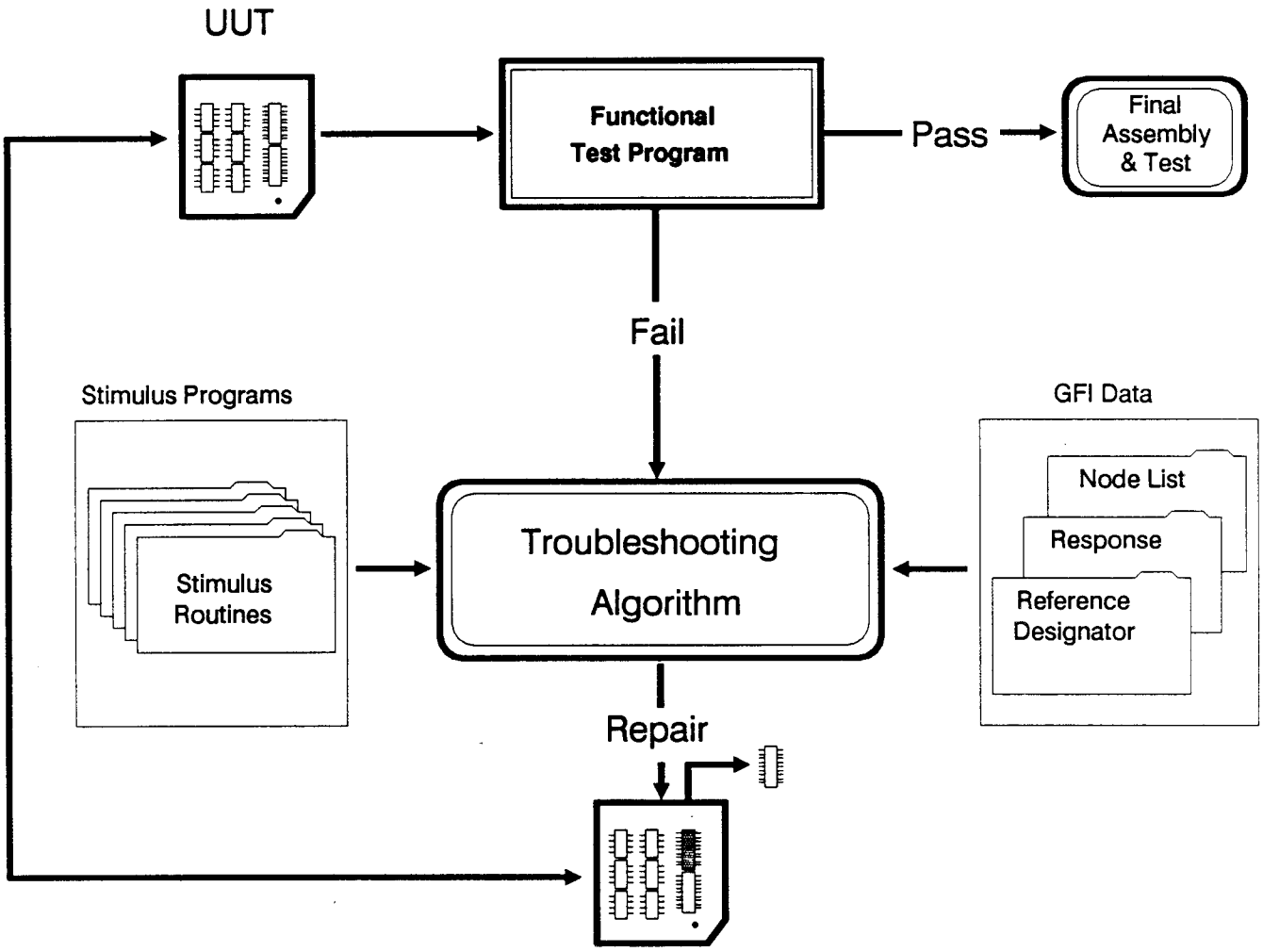

UUT
Functional
Test Program
Pass
Final
Assembly
& Test
Fail
• Apply Stimulus
• Measure Response
• Compare
Repair

Application Keypad Method

The Application Keypad method of troubleshooting is the most common way of identifying a fault.

In Application Keypad troubleshooting, each step of the troubleshooting process is initiated by the operator. The operator must remember the proper stimulus and response, know how to measure the response, and decide where to test next.

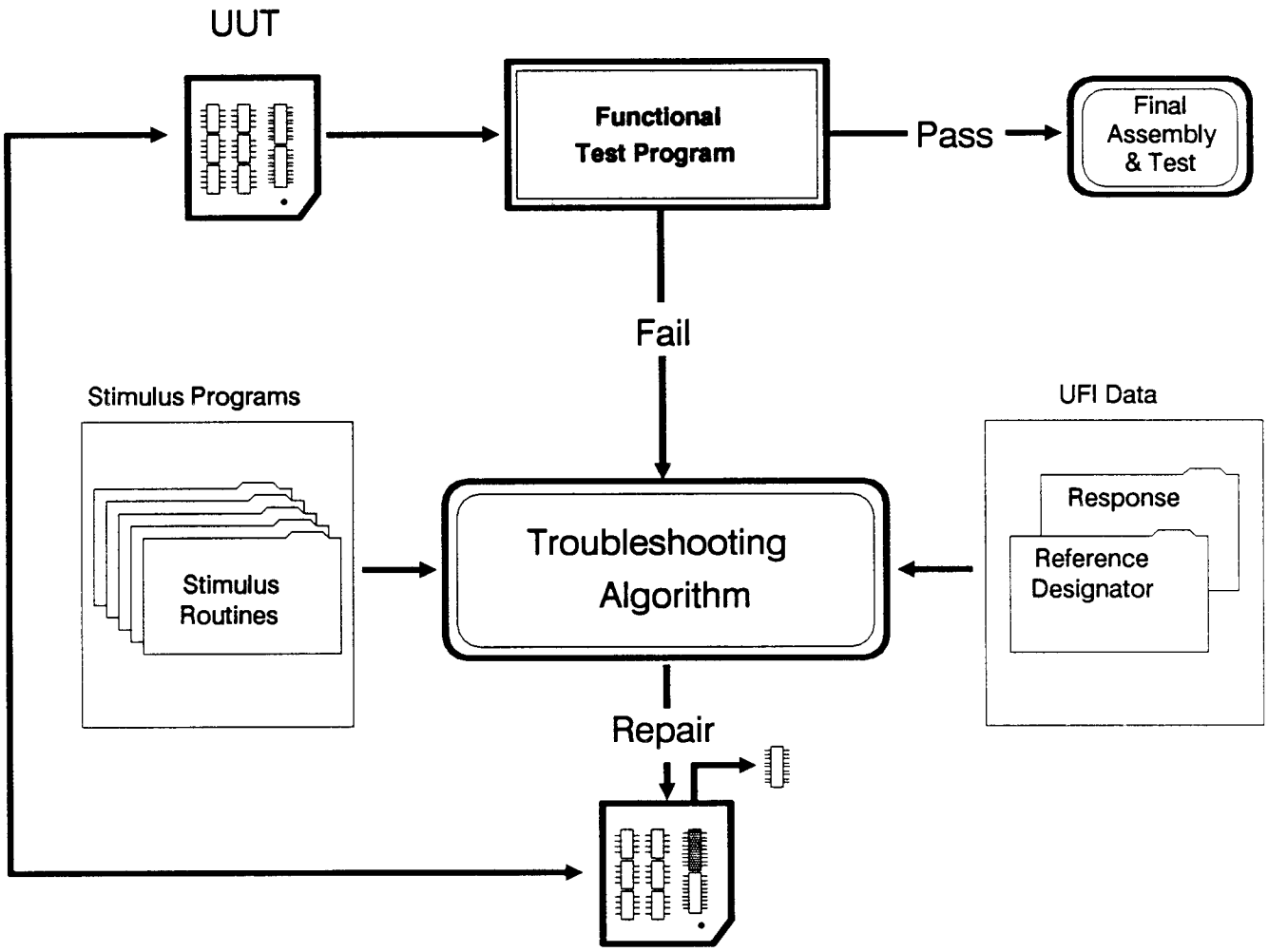
Once the fault has been identified and corrected, the UUT goes back to functional testing. This loop continues until the UUT passes all the functional tests.


UUT
Functional
Test Program
Pass
Final
Assembly
& Test
Fail
Stimulus Programs
Stimulus
Routines
Troubleshooting
Algorithm
GFI Data
Node List
Response
Reference
Designator
Repair

Guided Fault Isolation Method

The Guided Fault Isolation (GFI) technique is the ultimate method in the troubleshooting process. All the elements of troubleshooting are contained in files. The test equipment knows the types of devices, how the various devices are connected together, what nodes are inputs, what nodes are outputs, what nodes are bidirectional, how to stimulate each of the nodes, what measurement tool determines if the response is correct, what the correct response is, and where to test next.

If the UUT in this environment fails the functional test, the test equipment knows what portion failed. Based on this information, troubleshooting begins. The GFI algorithm takes over and guides the operator through the troubleshooting process until the fault is identified. The UUT then goes back to the functional test. This process continues until all functional tests pass.

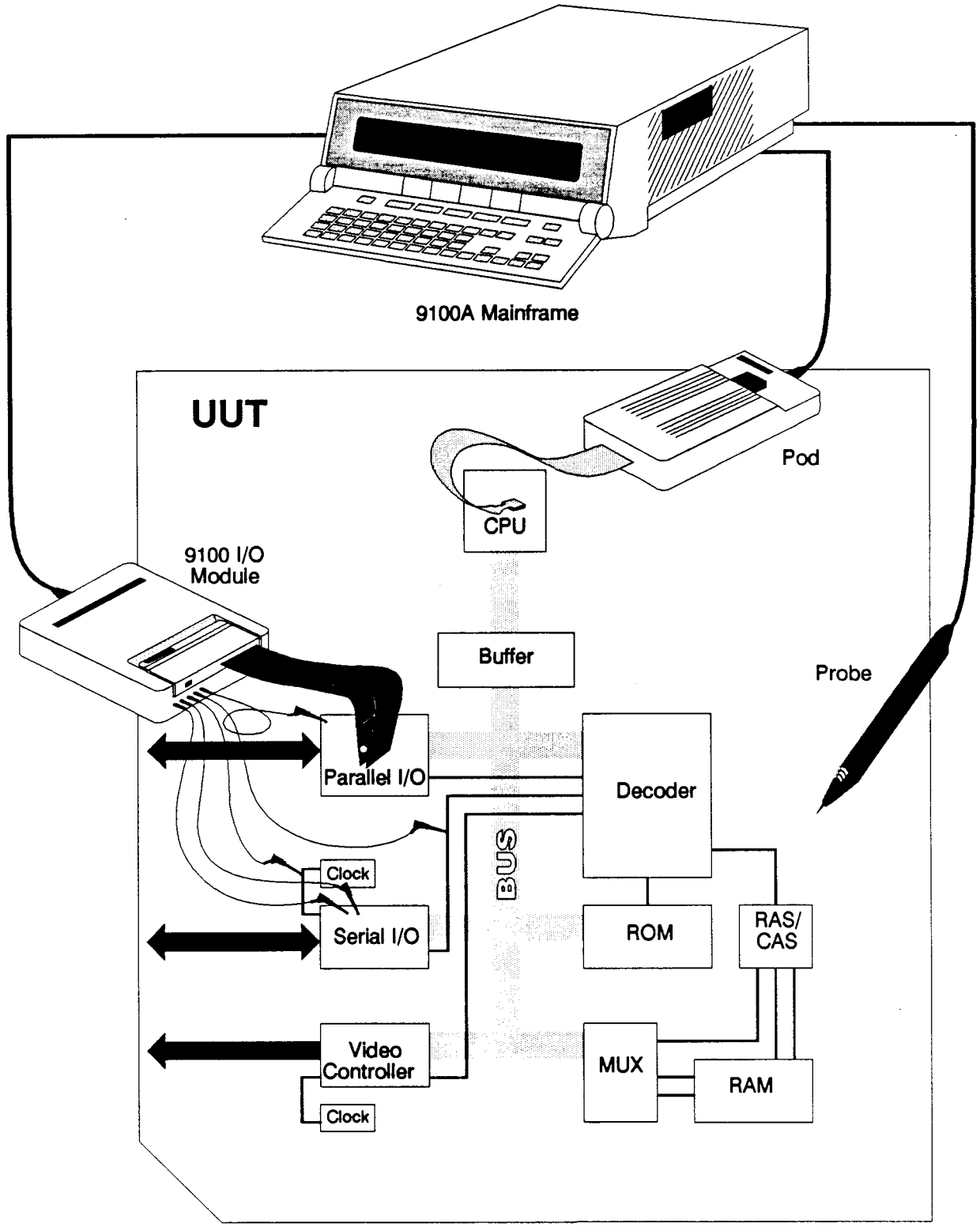

UUT
Functional
Test Program
Pass
Final
Assembly
& Test
Fail
Stimulus Programs
Stimulus
Routines
Troubleshooting
Algorithm
UFI Data
Response
Reference
Designator
Repair

Unguided Fault Isolation Method

The Unguided Fault Isolation (UFI) method of troubleshooting incorporates semiautomation. Portions of the troubleshooting process are contained in a data base.

One of the most difficult things the test operator must do is remember the proper stimulus for a particular circuit. After the operator determines the stimulus, the next step is to measure the response, and then decide if the response is correct.

UFI makes all these decisions for the operator. The operator tells the test equipment which node to test. For example, the operator tells the test equipment to test U1 pin 1. The equipment goes to the stimulus routine file and pulls out the proper routine for that node. The file tells the operator which measurement tool to use (such as a probe or an I/O module). When the stimulus is complete, the tester accesses the response file and pulls out the correct response for the node and determines if the measured response is the same. The operator then decides the next measurement point based on the results of the test. This process continues until the operator identifies the cause of the fault.


9100A Mainframe
UUT
Pod
CPU
9100 I/O
Module
Buffer
Probe
Parallel I/O
Decoder
BUS
Clock
Serial I/O
ROM
RAS/
CAS
Video
Controller
MUX
RAM
Clock

Summary

In this section, we have found that you need to understand the types of faults that are expected so that you can choose the proper test equipment.

You must also understand the level of testing or troubleshooting that is being performed to choose the proper test equipment.

Finally, you must apply the chosen test equipment, based on the characteristics of troubleshooting and the troubleshooting process.

During this week, you will begin to understand the flexibility of the 9100A Mainframe and be able to choose the characteristics that best fit your application.

Section 2
System Configuration

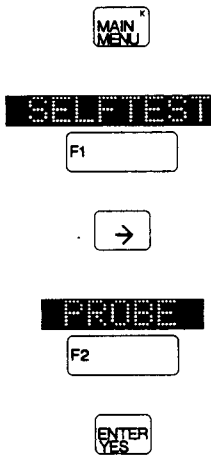
Introduction

This section introduces you to the 9100-Series Programming station. To use the equipment efficiently, you have to understand how each of the pieces work together. Therefore, this section demonstrates the communication protocol of each piece of the test station and the correct connections. You will also perform the calibration procedure for the probe and the I/O Module. The calibration procedures ensure accurate and repeatable measurements. These procedures must be done to ensure correct testing and test development.

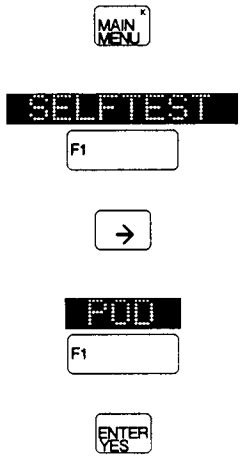
Exercise 2-1
Preparing the 9100A for Testing

1. Cable connections.
2. Description of each part.
3. Power-up the 9100A before UUT to activate Pod protection circuits.
4. Selftests:

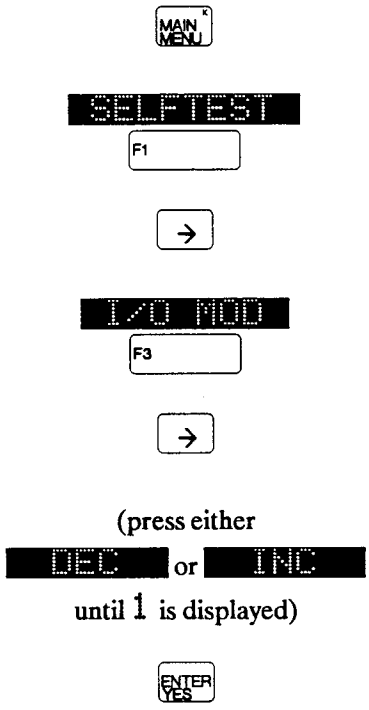
   a. Probe



   b. Pod



b. I/O Module



Repeat as necessary for each additional I/O Module (2 through 4) connected to the system.

5. UUT connection.

The 9100A System

The 9100A system services printed circuit boards, instruments, and systems that are controlled by microprocessors. The system has two basic functions: testing and troubleshooting.

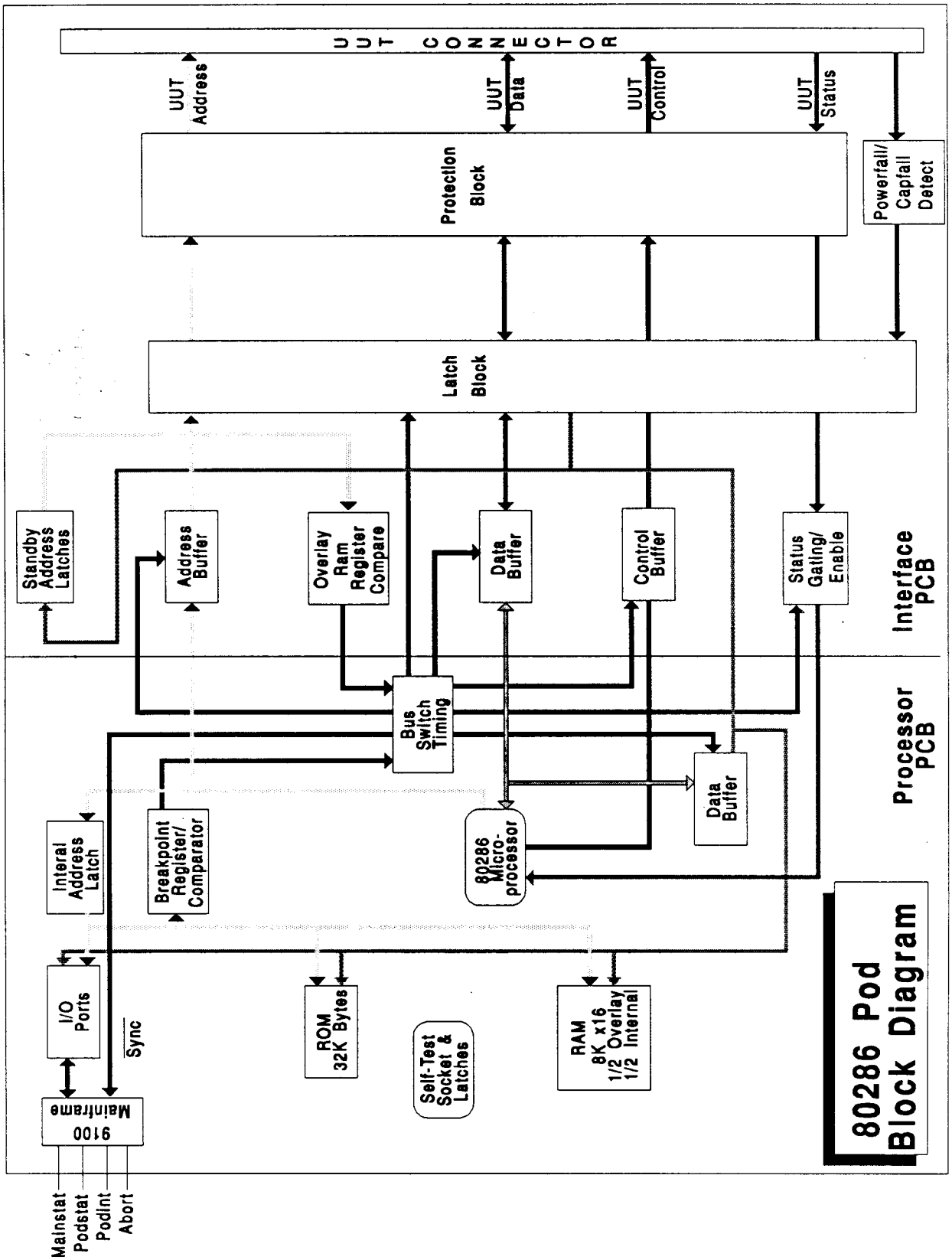
The pod directly controls or senses anything on the UUT that can be controlled or sensed by the UUT microprocessor. The probe and I/O Module can stimulate and measure all the UUT digital circuitry, even circuitry that is not accessible to the UUT microprocessor.

9100A Overview

The 9100A system consists of a 9100A Digital System without the programming option, version 5.0 System Software Disks, version 5.0 Master User Disks (Note that the System Disk and Master Userdisk software is preloaded onto the 9100A hard disk), and a version 1.0 9100A UUT Test Disk which contains the Learn and Auto Test program.

In addition, the system includes 31 standard Interface Pods and a 9132A Memory Interface Pod with three microprocessor support packages.

Testing and troubleshooting digital boards with the 9100A is done in keystroke mode, using built-in tests and stimuli manually initiated at the operator's keypad. A test may be constructed from a sequence of keystrokes that can be stored and later executed.


UUT CONNECTOR
UUT Address
UUT Data
UUT Control
UUT Status
Protection Block
Powerfail/ Capfail Detect
Latch Block
Standby Address Latches
Address Buffer
Overlay Ram Register Compare
Data Buffer
Control Buffer
Status Gating/ Enable
Interface PCB
Bus Switch Timing
Data Buffer
Processor PCB
Interal Address Latch
Breakpoint Register/ Comparator
80286 Micro-processor
I/O Ports
Sync
ROM 32K Bytes
Self-Test Socket & Latches
RAM 8K x16 1/2 Overlay 1/2 Internal
Mainframe 9100
Mainstat
Podstat
PodInt
Abort
80286 Pod Block Diagram

Interface Pod General Description

The illustration to the left shows the 80286 pod block diagram. The basic block diagram for all pods is essentially the same. Each pod uses the same method to communicate with the Mainframe, contains interface boards and processor boards, and replaces the UUT microprocessor.
The pod has two modes of operation:

- Normal
  The pod adapts the architecture of the Mainframe to the specific pin layout of the UUT microprocessor. This allows the Mainframe to exercise all UUT buses while monitoring UUT activity. Typically, the pod executes READs or WRITEs, and the Mainframe displays the results.
- RUN UUT
  The pod's microprocessor is connected through buffers to the UUT, and serves as a substitute microprocessor.

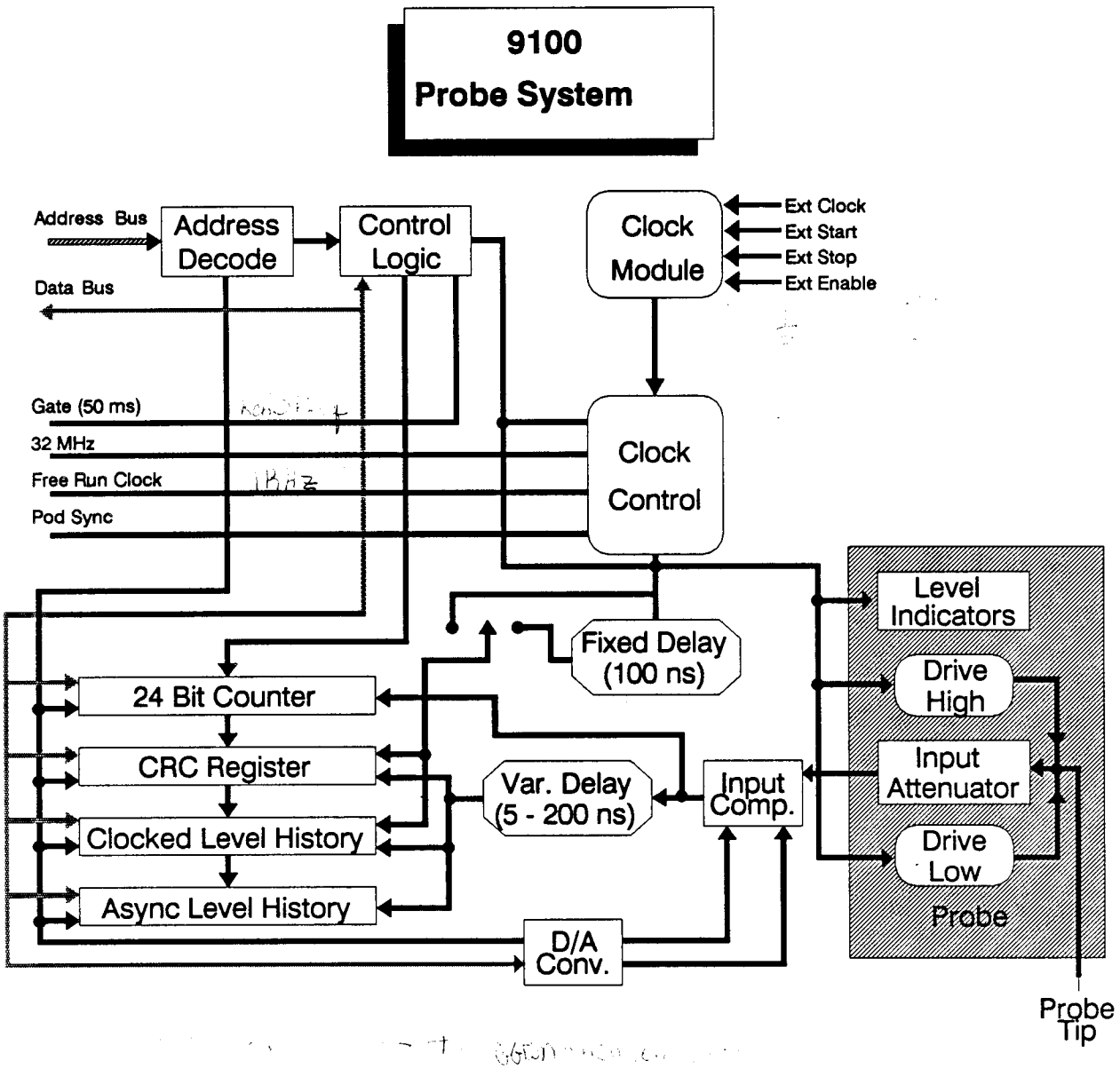
The pod provides stimulus to the UUT using READ, WRITE, combinations of READ and WRITE, and built-in tests.

Status lines are control inputs to the microprocessor. Sometimes these lines can stop test execution entirely. The Mainframe, therefore, is capable of measuring these lines or, to continue execution and ignore the lines.

The pod communicates with the Mainframe using the following lines from the processor PCA:

- MAINSTAT
- PODSTAT
- PODINT
- ABORT

The pod communicates with the UUT using the address, data, control, and status lines that are directly associated with those lines on the UUT.


9100
Probe System
Address Bus
Address Decode
Control Logic
Clock Module
Ext Clock
Ext Start
Ext Stop
Ext Enable
Data Bus
Gate (50 ms)
32 MHz
Free Run Clock
Pod Sync
Clock Control
Level Indicators
Fixed Delay (100 ns)
Drive High
24 Bit Counter
CRC Register
Var. Delay (5 - 200 ns)
Input Comp.
Input Attenuator
Clocked Level History
Drive Low
Async Level History
Probe
D/A Conv.
Probe Tip

The Purpose of the Probe and Clock Module

The function of the probe is dual purpose. The probe acts as an input measurement device that senses activity on the circuitry and as an output stimulus device that provides activity to the circuitry.

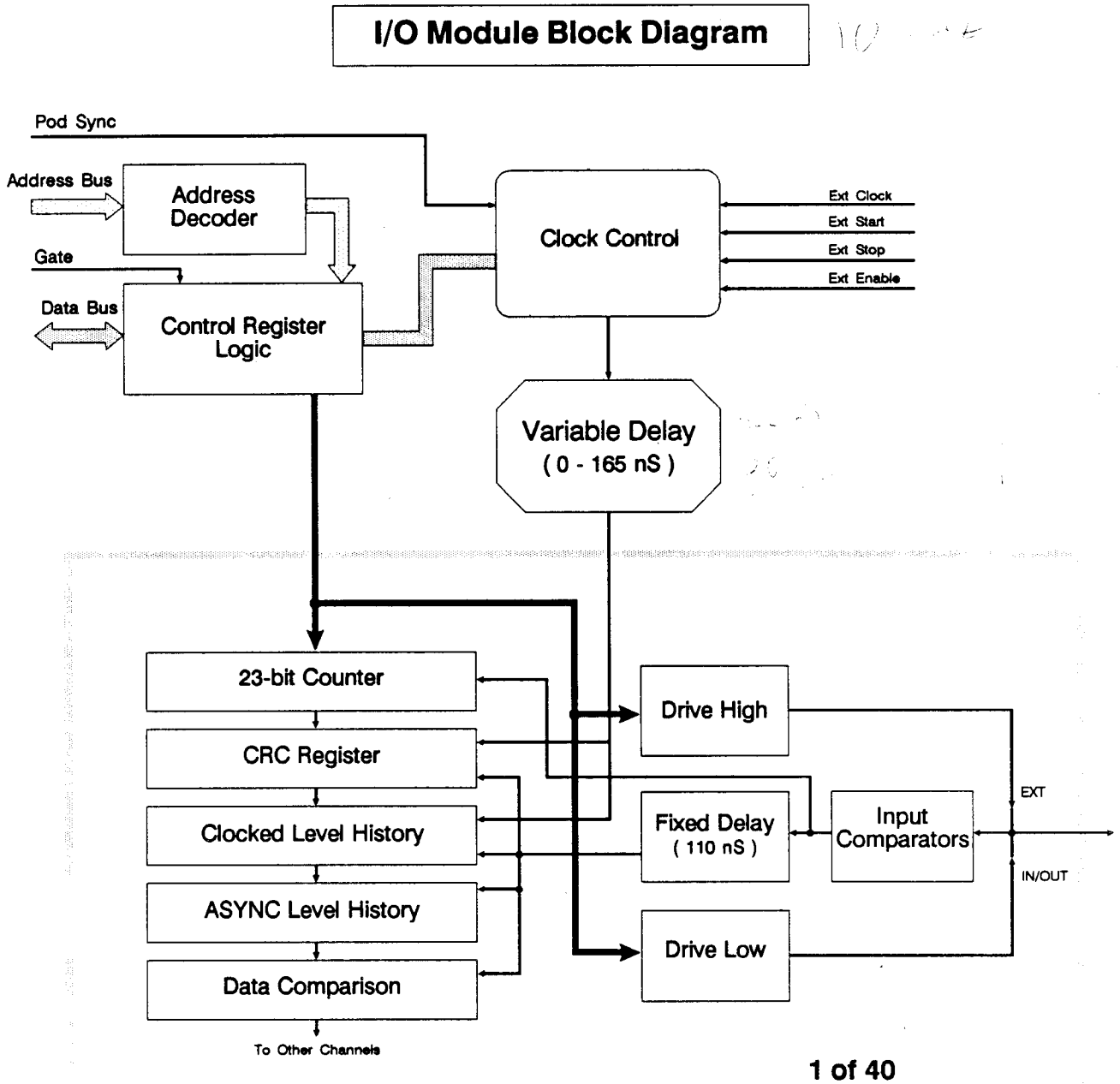
As a measurement tool the probe detects level activity (whether that activity is synchronous or asynchronous to microprocessor activity), transition counts, frequency, or CRC signatures. The activity measured by the probe can either be synchronized to the pod, the clock module, or a free-running clock.

As a stimulus device, the probe provides pulses that are high, low, or toggling and are preset to 0-5 volts independent of thresholds TTL, CMOS, RS-232 or ECL.

The probe contains input circuitry that consists of an attenuator and level indicators. The indicators have circuitry that stretches pulses. Stretching the pulses allows an operator to see, by using lights, the levels of even very fast pulses.

The remainder of the probe circuitry is contained within the Mainframe. As you can see from the drawing on the previous page, there is a 24-bit counter, CRC register, clocked level history, and level history. All this information is available either through the Mainframe display in local mode or from within a Mainframe program.

The probe circuitry communicates with the Mainframe through the address and data bus of the Mainframe.


I/O Module Block Diagram
Pod Sync
Address Bus
Address Decoder
Gate
Data Bus
Control Register Logic
Clock Control
Ext Clock
Ext Start
Ext Stop
Ext Enable
Variable Delay
(0 - 165 nS)
23-bit Counter
CRC Register
Clocked Level History
ASYNC Level History
Data Comparison
To Other Channels
Drive High
Drive Low
Fixed Delay
(110 nS)
Input Comparators
EXT
IN/OUT
1 of 40
Channels

The Function of the I/O Module

The function of the I/O module is basically the same as the probe. The I/O module acts as an input measurement device that senses activity on the circuitry and as an output stimulus device that provides activity to the circuitry. The difference between the two is the I/O module has 40 channels (whereas the probe is a single channel). With the I/O module you have the equivalent of 40 probes for each module for total of 160 (four I/O modules can be used with the Mainframe).

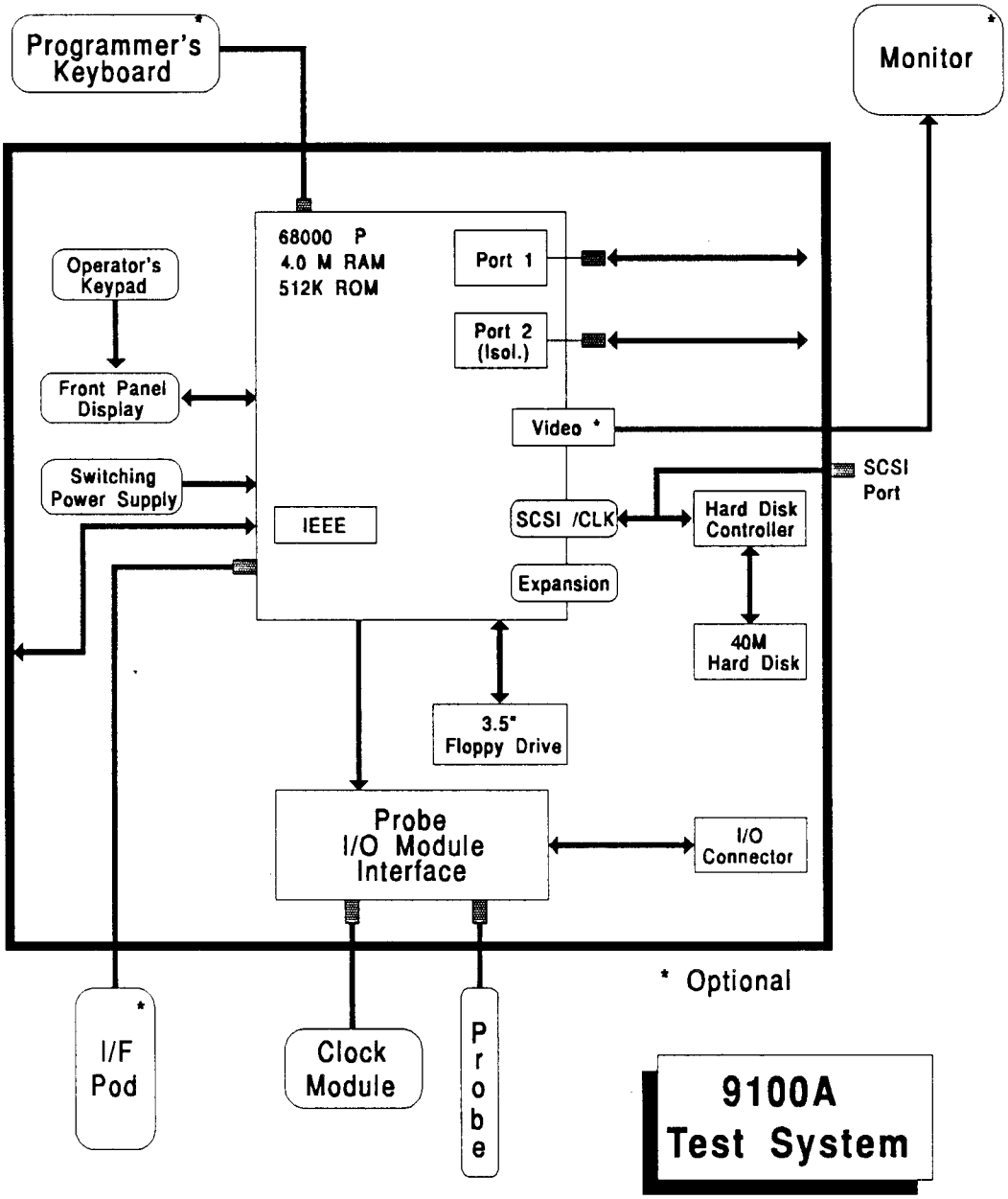
As a measurement tool the module can detect level activity, transition counts, frequency, and CRC signatures. The I/O module can also be used as a pattern reconizer.

As a stimulus device the module can pulse as the probe does, but can also drive/overdrive output patterns.

The I/O module consists of:

- Input comparator
- Drive circuitry for highs and lows
- 23-bit counter
- CRC register
- Clocked level history
- Data comparison

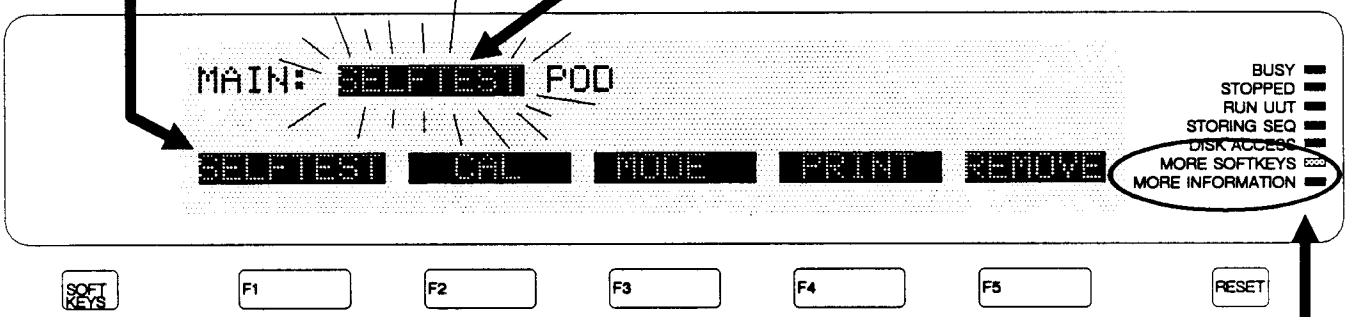
Communication with the Mainframe occurs through the address and data bus. Notice also that, unlike the probe, the I/O module does not have a free-running sync clock.


Programmer's Keyboard *
Monitor *
68000 P
4.0 M RAM
512K ROM
Operator's Keypad
Front Panel Display
Switching Power Supply
IEEE
Port 1
Port 2 (Isol.)
Video *
SCSI /CLK
Expansion
SCSI Port
Hard Disk Controller
40M Hard Disk
3.5" Floppy Drive
Probe I/O Module Interface
I/O Connector
* Optional
I/F Pod *
Clock Module
Probe
9100A Test System

Introduction to the Training Station

The illustration to the left is a simple block diagram of the Mainframe showing the connections to the pod, probe and so on.

Commands and information are entered through the keypad and are monitored on the display, which also provides the user with instructions, responses, error messages, and UUT fault messages.



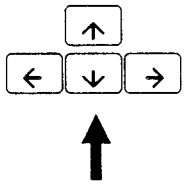
Softkey Labels
valid for highlighted Command Field

Highlighted Command Field

Function Keys
correspond to Softkey Labels directly above

"MORE" Annunciators

"Soft Keys" Key
displays additional softkeys when "MORE SOFTKEYS" LED is illuminated



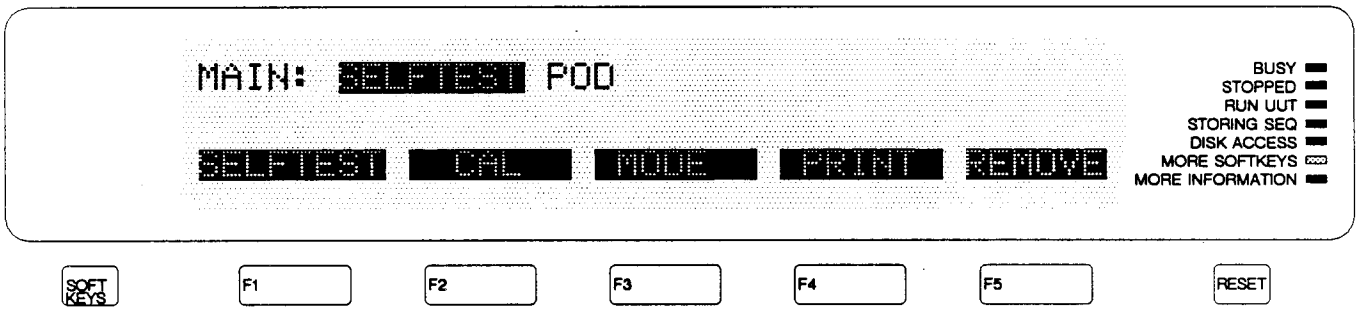
Arrow Keys
choose which Command Field to highlight. Also used to display additional information when "MORE INFORMATION" LED is illuminated

Application Keypad and Softkeys

Illustrated to the left is the Mainframe Display and a portion of the Application Keypad. To the right of the display the More Softkeys and More Information annunciators are circled. The five Function Keys, [SOFT KEYS] key, and arrow keys are located on the swing-down Application Keypad.

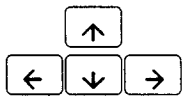
- Function keys and Softkeys
  The words in boxes on the bottom line of the Mainframe display (SELFTEST e.g.)are called *softkey labels* and correspond to the function keys [F1] through [F5] on the Application Keypad. To select a softkey function, press the function key on the keypad just below the corresponding softkey label. The chosen or default function will be displayed flashing on the top line of the display in its appropriate field. If there are more softkey options than can be displayed at one time on the bottom of the display, a red "MORE SOFTKEYS" LED located to the right of the display is illuminated. The [SOFT KEYS] button scrolls hidden softkey labels into view if the LED is on. Pressing the key again recalls prior labels.

- Arrows
  There are four arrows on the Application Keypad. These keys move the cursor on the Mainframe display to different fields on the top line of the display. A field is a word or number in a command line that can be changed. To change a field, the field must be highlighted (flashing) on the display.


MAIN: SELFTEST POD
SELFTEST
CAL
MODE
PRINT
REMOVE
BUSY
STOPPED
RUN UUT
STORING SEQ
DISK ACCESS
MORE SOFTKEYS
MORE INFORMATION

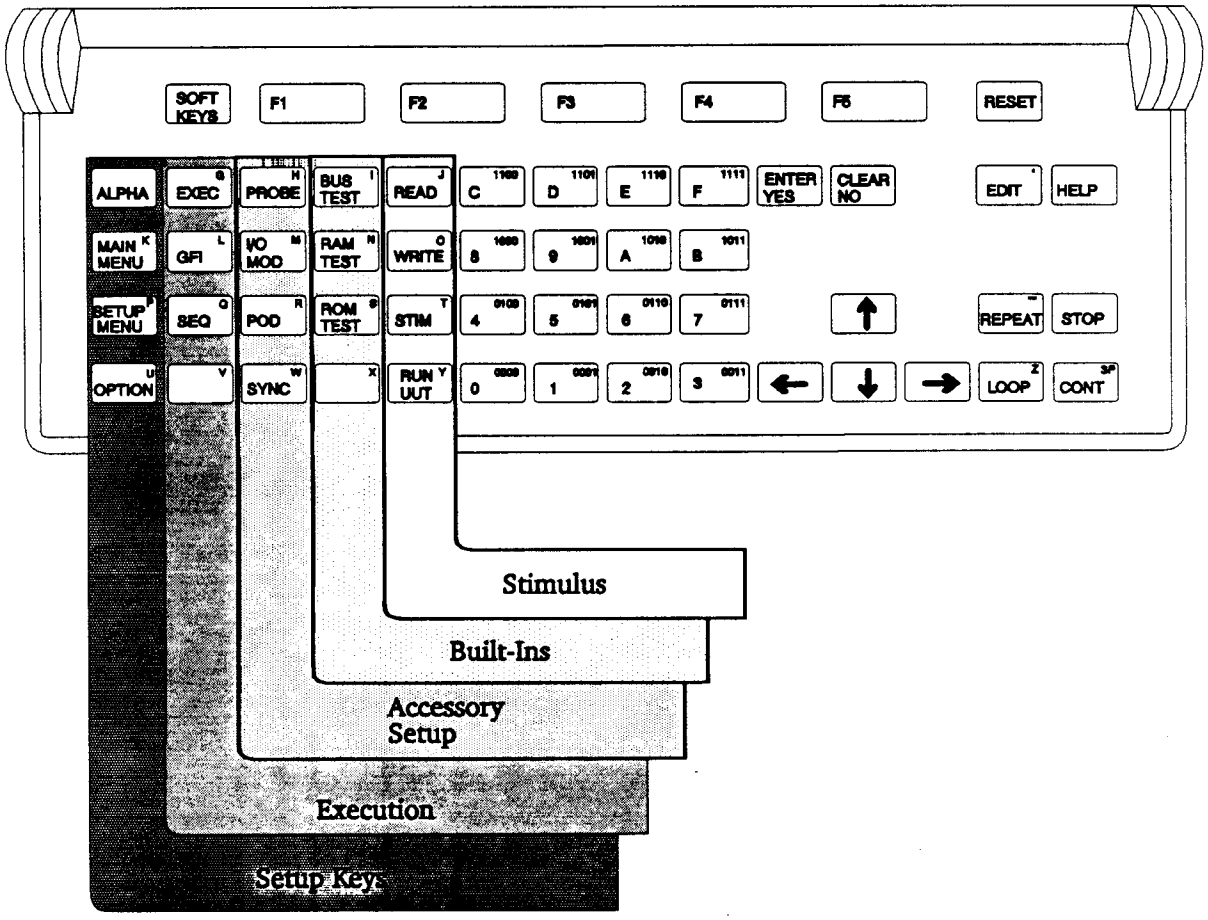
SOFT KEYS
F1
F2
F3
F4
F5
RESET


MAIN MENU



Exercise 2-2
Using Softkeys

1. Press [MAIN MENU]. Notice that the red MORE SOFTKEYS LED to the right of the fluorescent display is illuminated.
2. Press [SOFT KEYS]. Notice the different softkey labels. Press [SOFT KEYS] again to show the previously displayed softkeys. Observe how [SOFT KEYS] toggles what is displayed on the bottom line of the display between different sets of softkey labels valid for the current command level.
3. Press [→] and observe how the highlighted field switches from SELFTEST to PUL and how the softkey options change to reflect the valid command options for the new field.


SOFT KEYS
F1
F2
F3
F4
F5
RESET
ALPHA
EXEC
PROBE
BUS TEST
READ
C
D
E
F
ENTER YES
CLEAR NO
EDIT
HELP
MAIN MENU
GFI
I/O MOD
RAM TEST
WRITE
8
9
A
B
SETUP MENU
SEQ
POD
ROM TEST
STIM
4
5
6
7
REPEAT
STOP
OPTION
SYNC
RUN UUT
0
1
2
3
LOOP
CONT
Stimulus
Built-Ins
Accessory Setup
Execution
Setup Keys

Exercise 2-3
Keyboard Introduction

Setup Keys: [ALPHA] [MAIN MENU] [SETUP MENU] [OPTION]

Execution: [EXEC] [GFI] [SEQ]

Accessory setup: [PROBE] [I/O MOD] [POD] [SYNC]

Built-ins: [BUS] [RAM] [ROM] [READ] [WRITE] [STIM] [RUN UUT]

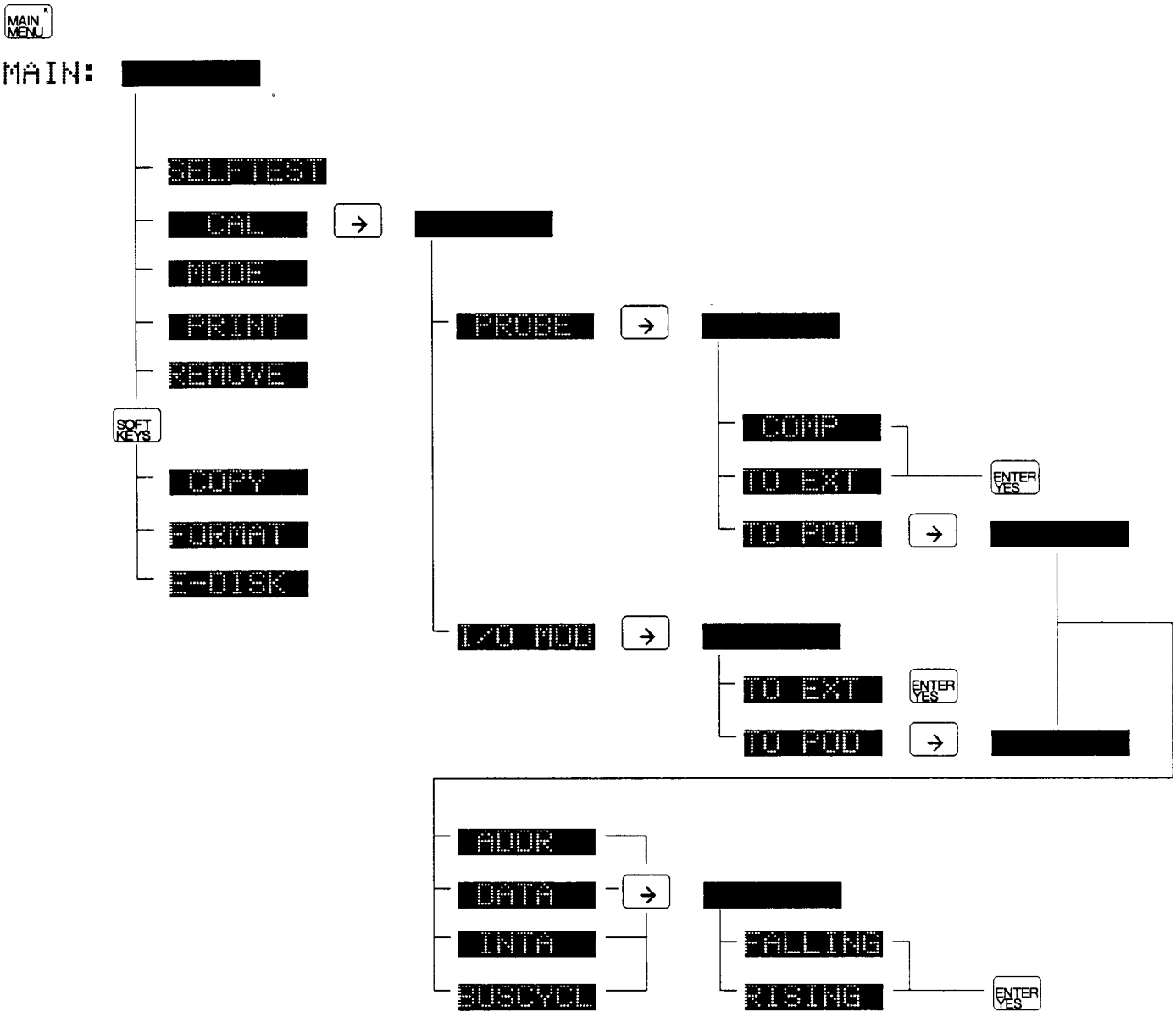
Example Test:

Example Fault: Fault switch 2-7

Bus Test : [BUS] [0] [ENTER YES]

Help Key: [HELP]

Press [↓] to read the complete message.


MAIN MENU
MAIN:
SELFTEST
CAL
MODE
PRINT
REMOVE
SOFT KEYS
COPY
FORMAT
E-DISK
PROBE
COMP
TO EXT
TO POD
ENTER YES
I/O MOD
TO EXT
TO POD
ENTER YES
ADDR
DATA
INTA
BUSCYCL
FALLING
RISING
ENTER YES

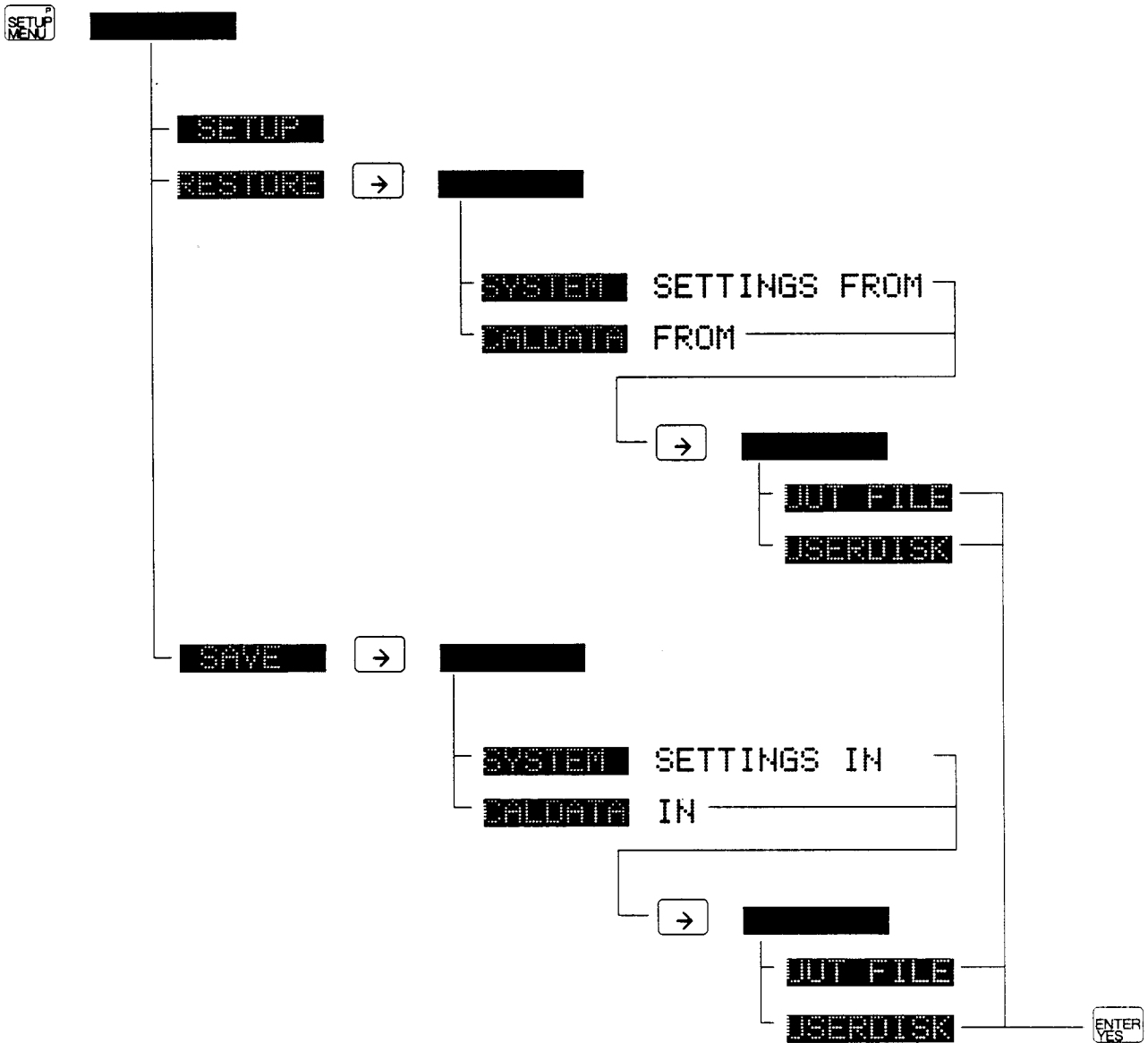
Exercise 2-4
Probe and I/O Module Calibration

To ensure proper operation when troubleshooting, the Probe and I/O Module must be calibrated with the UUT. The calibration can be stored and restored before testing begins. If the calibration data is not saved, the Probe and I/O Module must be calibrated every time the Mainframe is powered up.

Using the keystroke flow chart on the previous page and following the instructions on the Mainframe display, calibrate the probe and I/O Module to:

- pod addr
- pod data

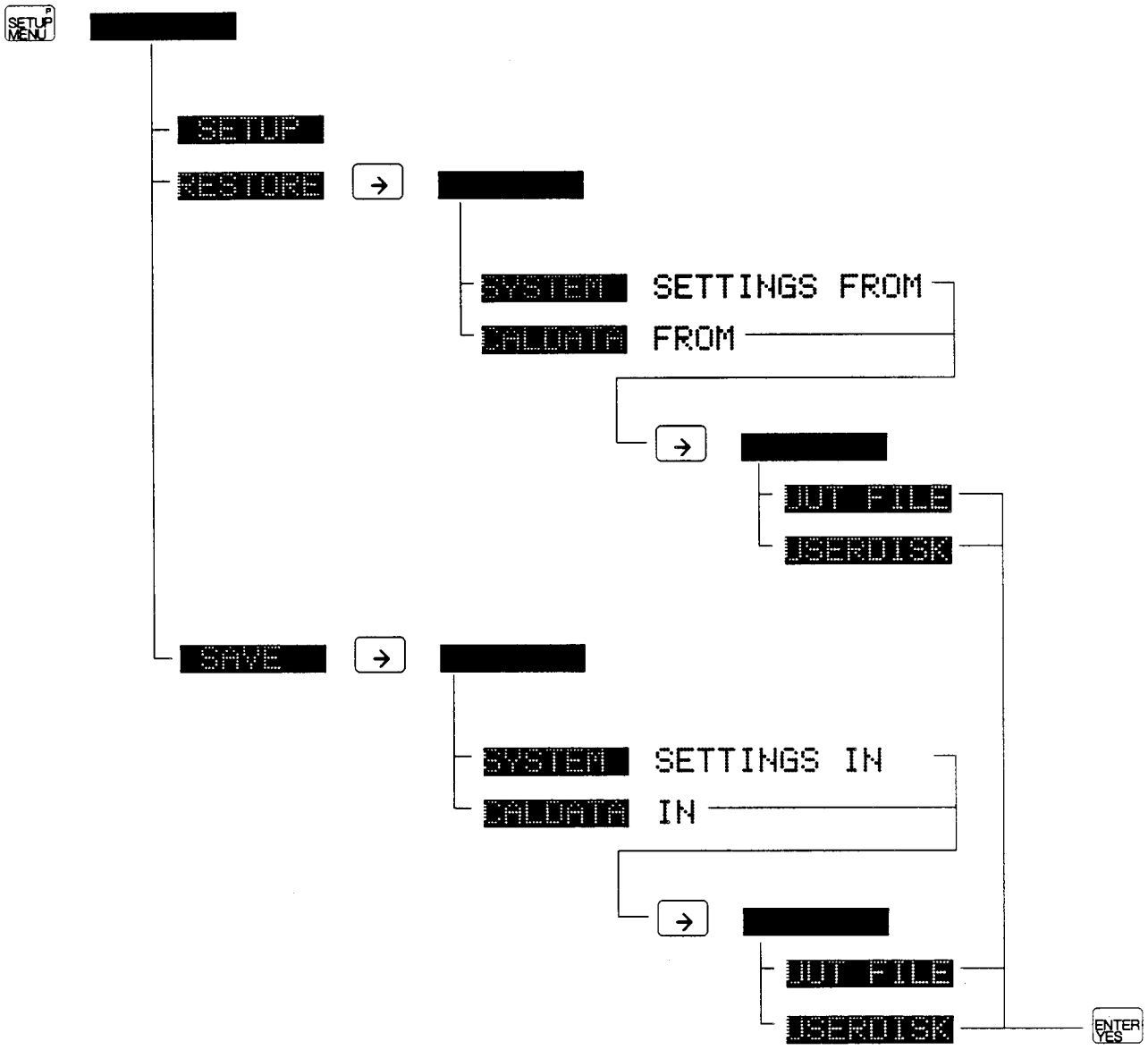
Note: Calibration of the I/O Module requires the use of the I/O Module calibration header (JF part #813980).


SETUP MENU
SETUP
RESTORE
SYSTEM
SETTINGS FROM
CALDATA
FROM
OUT FILE
USERDISK
SAVE
SYSTEM
SETTINGS IN
CALDATA
IN
OUT FILE
USERDISK
ENTER YES

Exercise 2-5
Saving Calibration Data

After performing the calibration of the Probe and I/O Module, save the calibration settings by selecting the setup menu.

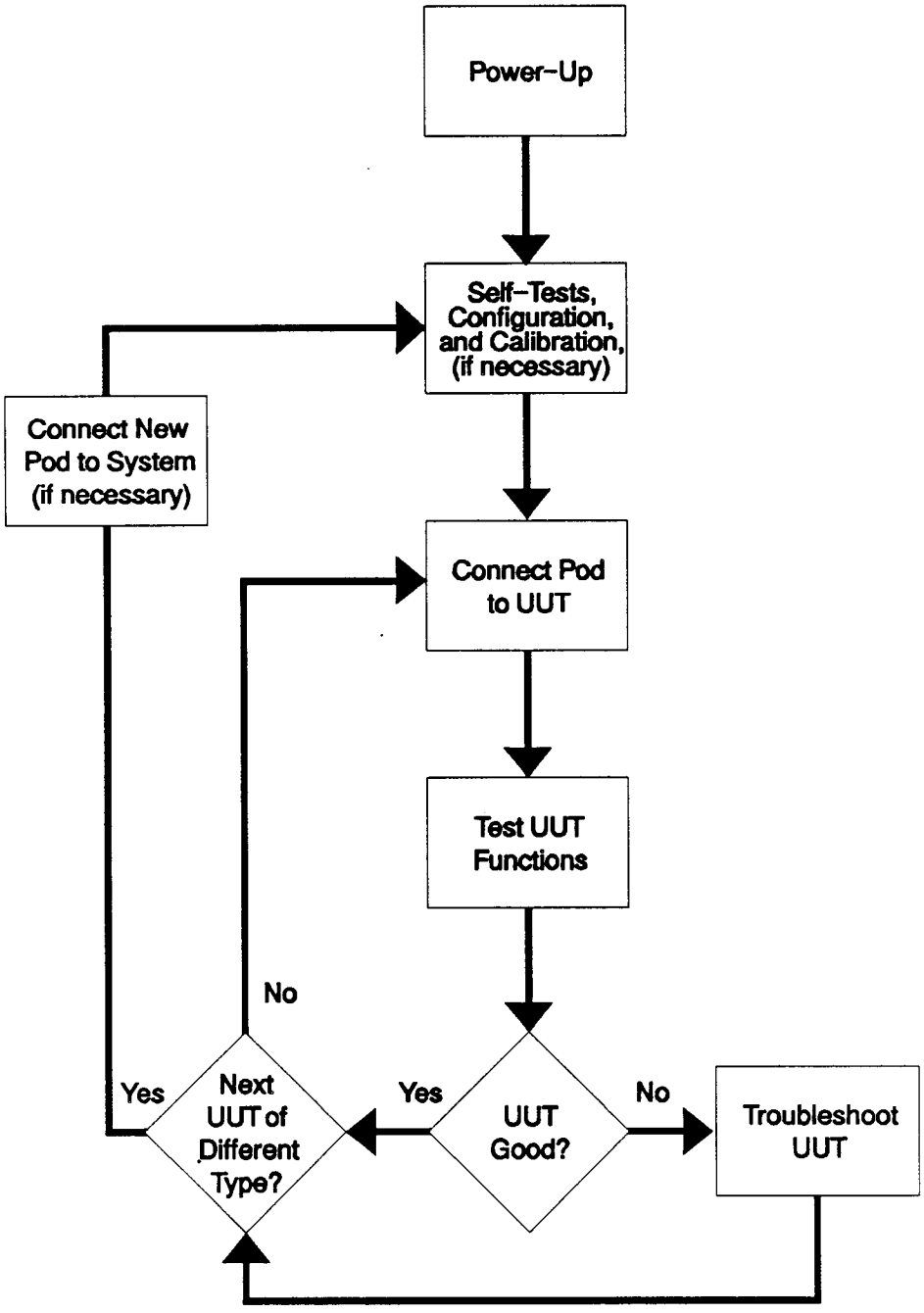
When saving either the calibration data or system settings, your choices are the USERDISK or UUT FILE. We recommend that you save your calibration data to a UUT FILE. At this time, perform the necessary steps to save your calibration data to the UUT FILE TRAINER.


SETUP MENU
SETUP
RESTORE
SYSTEM
CALDATA
SETTINGS FROM
FROM
JUT FILE
JSERDISK
SAVE
SYSTEM
CALDATA
SETTINGS IN
IN
JUT FILE
JSERDISK
ENTER YES

Exercise 2-6
Restoring Calibration Data

If the Mainframe has been turned off, you must restore your calibration data before a test or troubleshooting session begins.

Use the setup menu to restore the calibration data you saved on the UUT FILE TRAINER.


Power-Up
Self-Tests, Configuration, and Calibration, (if necessary)
Connect New Pod to System (if necessary)
Connect Pod to UUT
Test UUT Functions
No
Yes
Next UUT of Different Type?
Yes
UUT Good?
No
Troubleshoot UUT

Summary

Self-Test:

Should be performed for each device (pod, probe, I/O Module) when the device is attached to the mainframe.

Configuration:

Can be saved on and restored from user disk for each UUT. They specify SETUP MENU command settings and provide information for other commands.

Calibration:

Generates data which can be stored on user disk and restored at power-up or reset. This data is sometimes invalid. Calibration must be performed when:

- The system is first installed and every 30 days thereafter.
- The pod, probe, or I/O Module is changed.
- The system is powered up or reset.
- Change the type of UUT.

Before you test or troubleshoot a UUT, you should ensure that selftests, calibration, configuration, and disk drive verification have all been performed appropriately. Some guidelines are as follows:

After each power-up or reset, before you test or troubleshoot a UUT:

- Calibrate and configure the system for the type of UUT you are about to test or troubleshoot.
- Ensure that self-tests have been performed on the pod, and probe.
- Verify that the system mode is appropriate and that your user disk is in DR1.

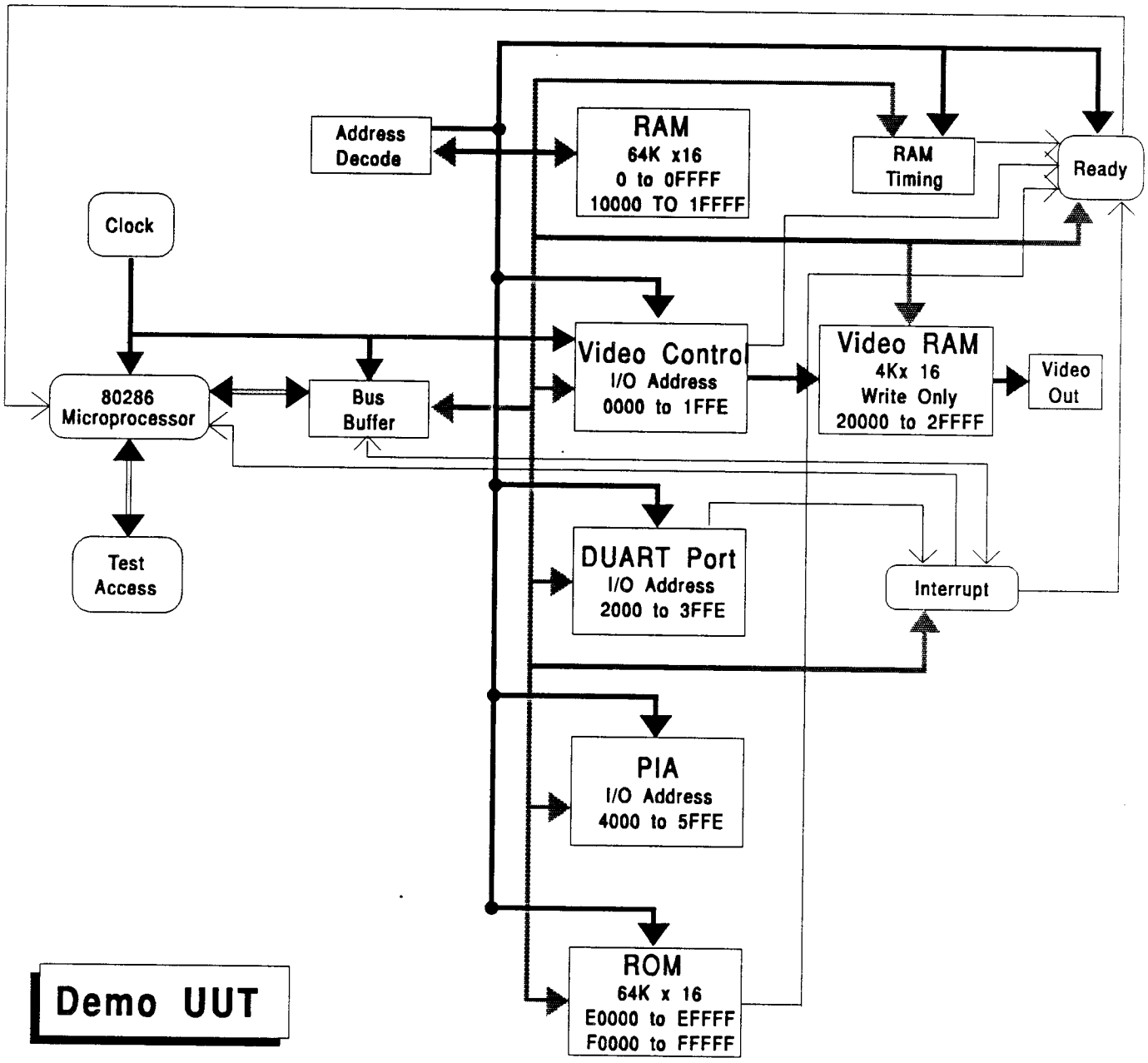
If you reset or change any part of the system:

- The self-tests will have to be repeated and the system recalibrated. You can save new calibration data on the user disk, writing over the old data.
- Test and troubleshooting sequences and the system configuration remain valid.

If you change UUTs from one type to another:

- You may have to change pods. If so, repeat the self-tests and calibrations with the new pod.
- System settings are probably different for the new UUT. Restore settings for the new UUT to reconfigure the system.

Section 3
Understanding the UUT


Address Decode
RAM
64K x16
0 to 0FFFF
10000 TO 1FFFF
RAM Timing
Ready
Clock
Video Control
I/O Address
0000 to 1FFE
Video RAM
4Kx 16
Write Only
20000 to 2FFFF
Video Out
80286 Microprocessor
Bus Buffer
Test Access
DUART Port
I/O Address
2000 to 3FFE
Interrupt
PIA
I/O Address
4000 to 5FFE
ROM
64K x 16
E0000 to EFFFF
F0000 to FFFFF
Demo UUT

Introduction

A UUT should be well understood before functional test and fault isolation begins. Taking time at the beginning to study the UUT will result in greater fault coverage, and more accurate fault detection.

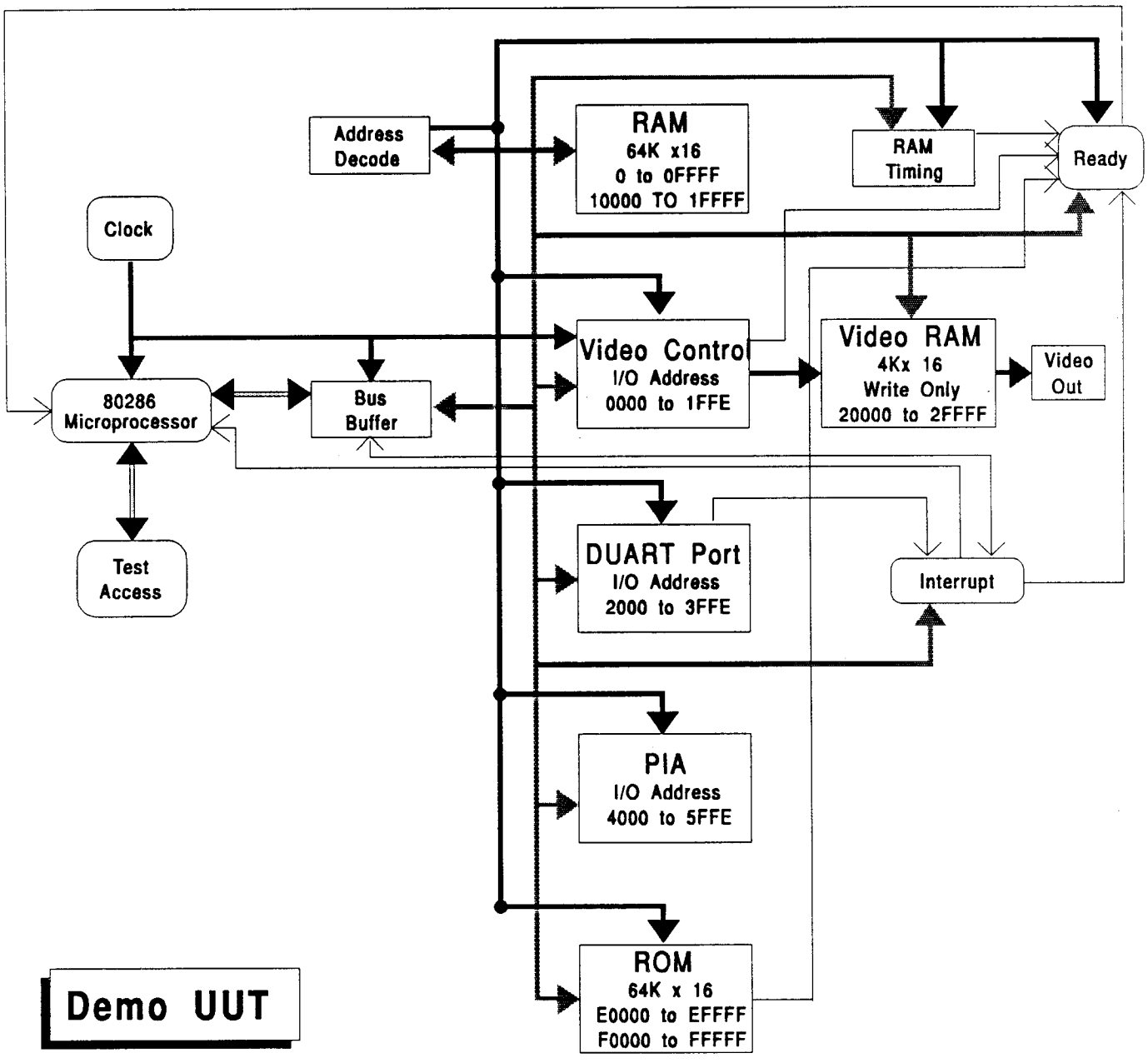
Before developing functional tests and troubleshooting routines:

- Learn what each circuit does, how it works, and how to initialize it.
- Determine the UUT memory map.
- Determine the initialization procedures for each programmable chip.

Exercise 3-1
Demo UUT

As a class discuss:

1. Demo UUT Block Diagram (computer concepts)
2. Demo UUT Schematic (chip functions)
3. Verify Map of Each Address Decoder Output (number systems)


Address Decode
RAM
64K x16
0 to 0FFFF
10000 TO 1FFFF
RAM Timing
Ready
Clock
Video Control
I/O Address
0000 to 1FFE
Video RAM
4Kx 16
Write Only
20000 to 2FFFF
Video Out
80286 Microprocessor
Bus Buffer
Test Access
DUART Port
I/O Address
2000 to 3FFE
Interrupt
PIA
I/O Address
4000 to 5FFE
ROM
64K x 16
E0000 to EFFFF
F0000 to FFFFF
Demo UUT

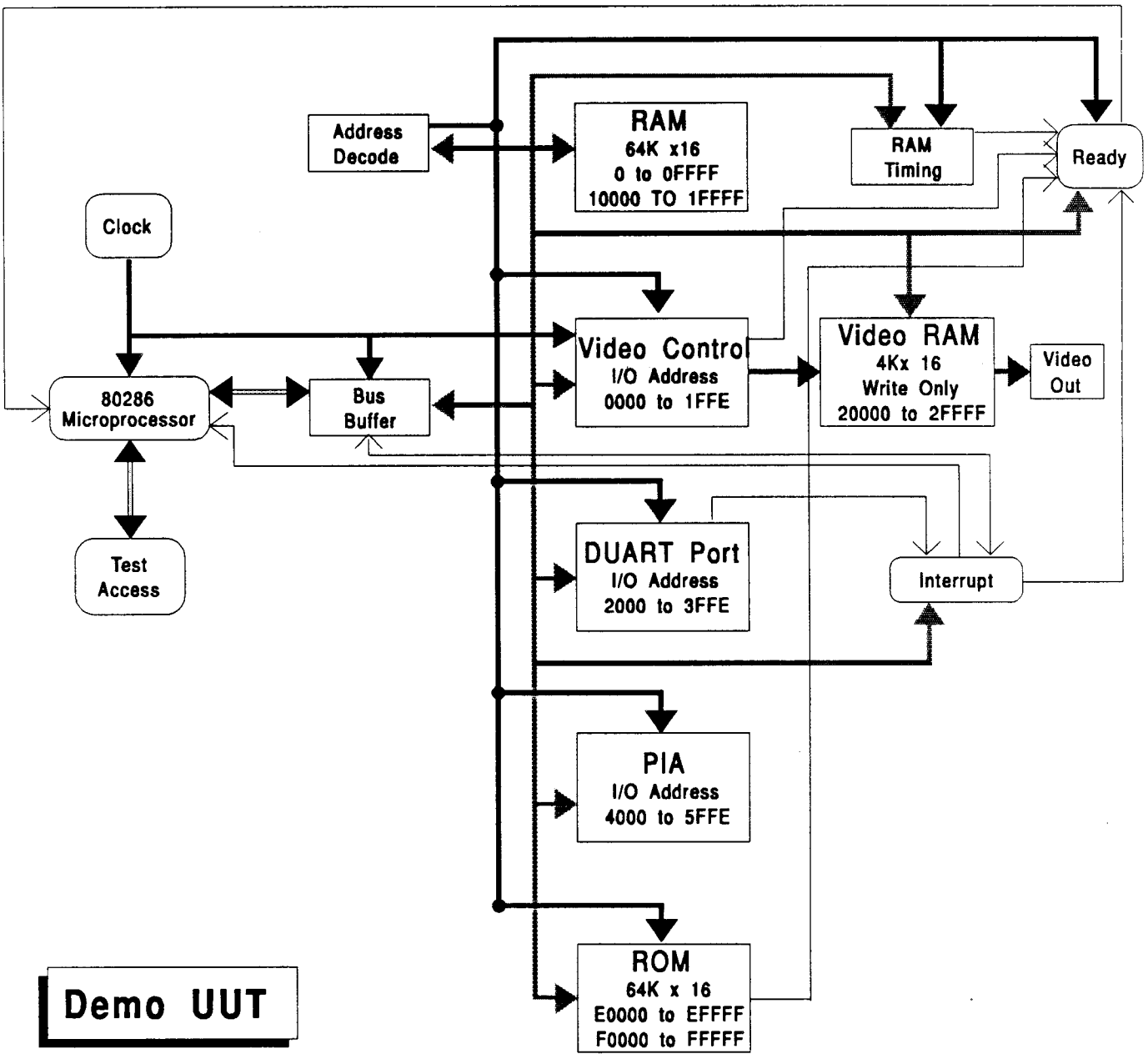
Partitioning

Partitioning divides the UUT into a collection of smaller functional blocks which are easier to understand and test. It is the first step toward a "divide-and-conquer" method of testing and troubleshooting. Each partition can then be tested separately to determine if the components are functioning as designed. A functional test only determines whether the partition passes or fails. It does not detect component failures, but suggests a starting point to begin fault isolation.

Partition Guidelines

Before you begin partitioning a circuit, you need to understand the UUT. How much understanding is necessary?

- Group circuits by function, making the functional blocks well-defined pieces of the UUT block diagram and as logically distinct as possible.
- If a functional block is large, subdivide it. This will improve troubleshooting efficiency.
- If failure of a circuit can cause failures to appear in many other parts of the UUT, make that circuit a functional block.
- If a circuit requires a unique test setup, make it a functional block.


Address Decode
RAM
64K x16
0 to 0FFFF
10000 TO 1FFFF
RAM Timing
Ready
Clock
Video Contro
I/O Address
0000 to 1FFE
Video RAM
4Kx 16
Write Only
20000 to 2FFFF
Video Out
80286 Microprocessor
Bus Buffer
Test Access
DUART Port
I/O Address
2000 to 3FFE
Interrupt
PIA
I/O Address
4000 to 5FFE
ROM
64K x 16
E0000 to EFFFF
F0000 to FFFFF
Demo UUT

Partitioning's Big Payoff

After the partitioning is done, step back and look at the resulting detailed block diagram. Imagine that a functional test has been developed for each individual block. If a novice user has nothing but this block diagram and the collection of individual block tests, he can make a fair degree of progress toward troubleshooting and repairing a complex system.

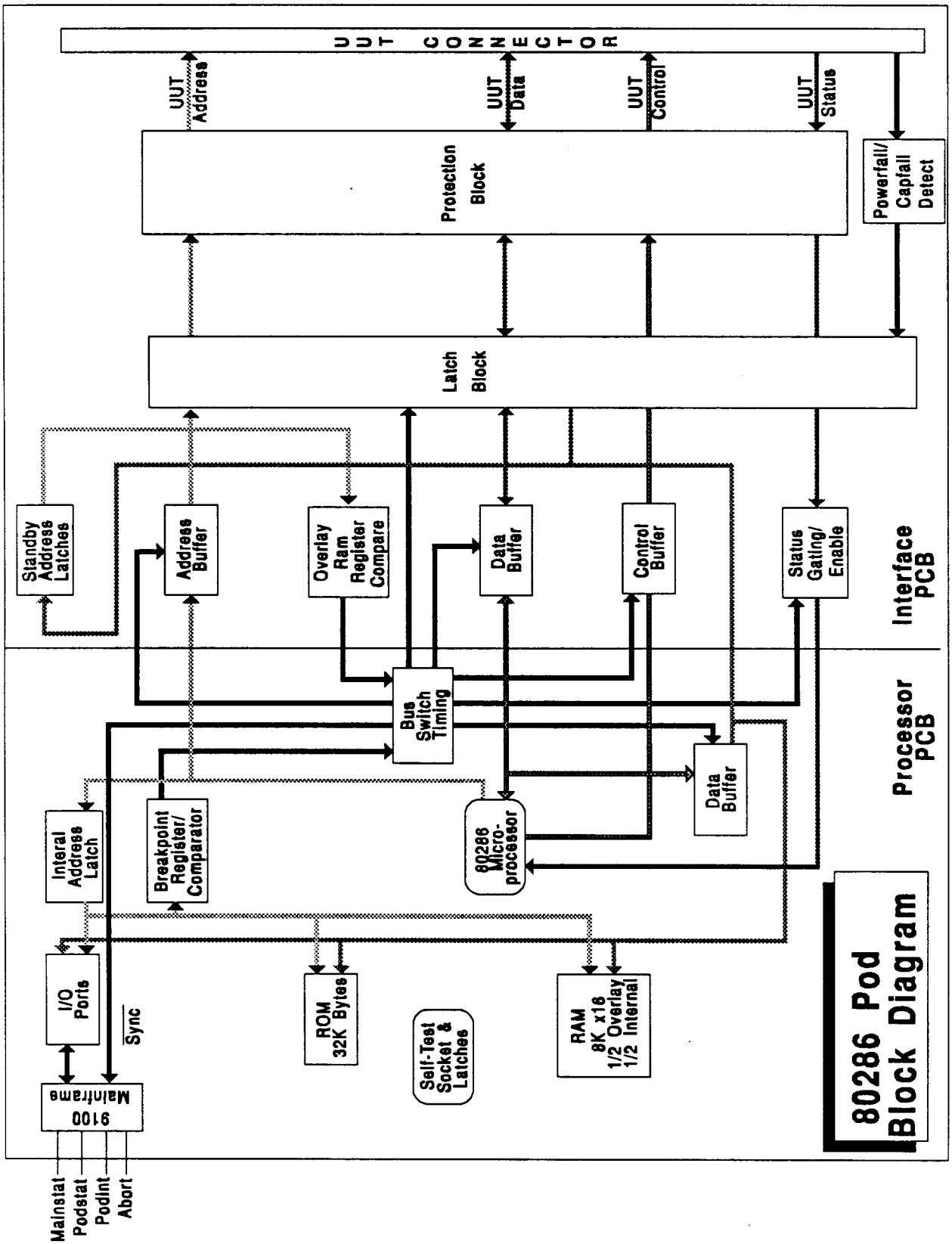
With thoughtful partitioning, a board may be determined to be good without running all of its individual functional block tests; some functional blocks can be assumed to be good if tests for other functional blocks that depend on them are good.

Through partitioning, the large problem of testing and troubleshooting a complex system can be divided into smaller, more easily handled problems.

Functional testing of a partitioned circuit can be divided into the following steps:

- Stimulate the inputs to the partitioned circuit.
- Measure the outputs of the partitioned circuit.
- Evaluate each output of the partitioned circuit and decide whether it passes or fails.
- If all outputs passed, declare the partitioned circuit to be good, otherwise declare that the partitioned circuit has failed.

Section 4
Bus Test


UUT CONNECTOR
UUT Address
UUT Data
UUT Control
UUT Status
Protection Block
Powerfail/ Capfail Detect
Latch Block
Standby Address Latches
Address Buffer
Overlay Ram Register Compare
Data Buffer
Control Buffer
Status Gating/ Enable
Interface PCB
Bus Switch Timing
Data Buffer
Processor PCB
Interal Address Latch
Breakpoint Register/ Comparator
80286 Micro-processor
I/O Ports
Sync
ROM 32K Bytes
Self-Test Socket & Latches
RAM 8K x16 1/2 Overlay 1/2 Internal
9100 Mainframe
Mainstat
Podstat
Podint
Abort
80286 Pod Block Diagram

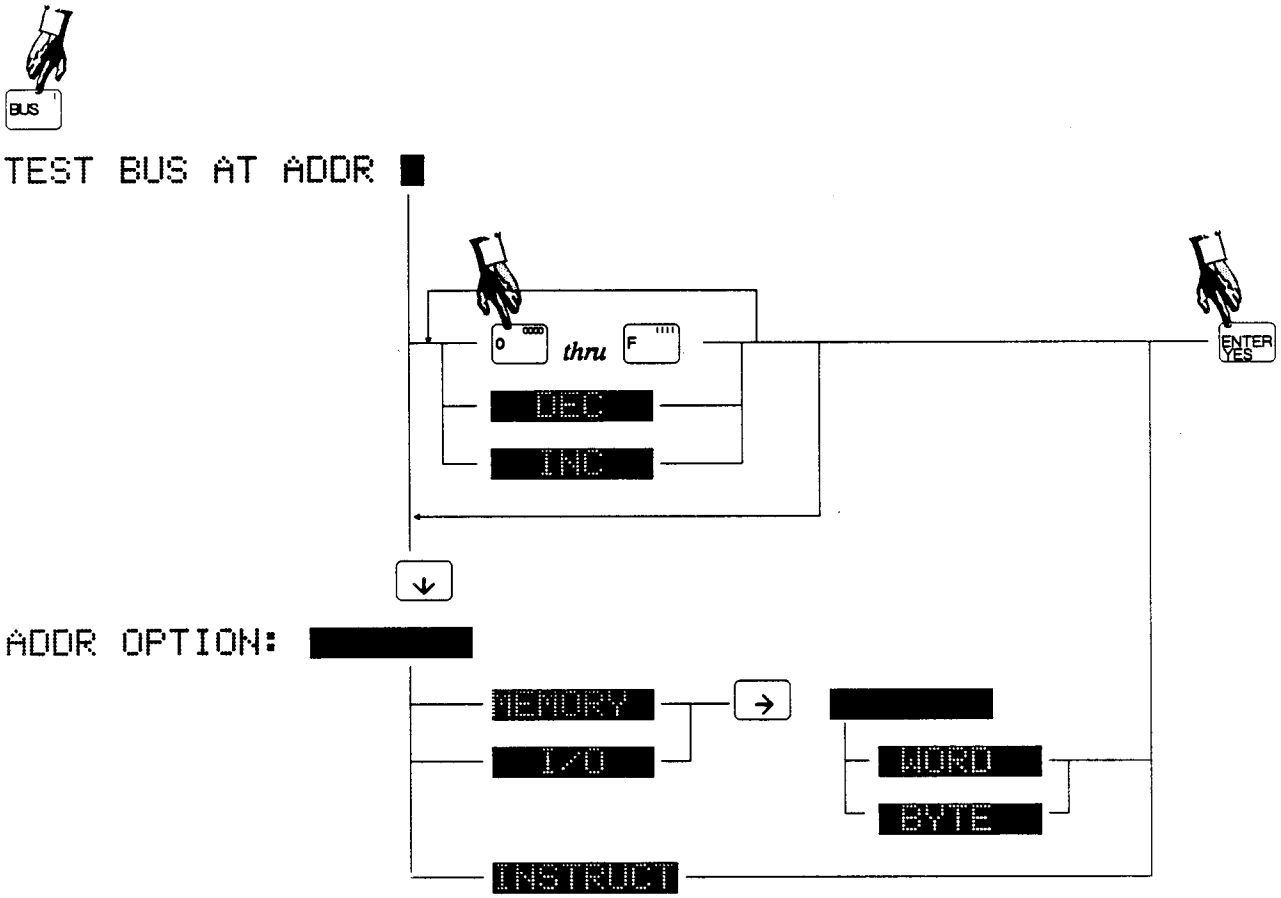
Built-In Bus Test

The 9100A simplifies the test of microprocessor buses through its built-in Bus Test. The built-in Bus Test has a number of test functions which provides a quick analysis of the bus and reduces operator interaction to a couple of button presses.

The Bus Test checks for stuck data, address or control lines by driving each line high and low and measuring the level that actually was present on the line. The measurement is done by level detectors or latches that are built into each pod on each of the drivable lines. When the level is put on the lines, a trigger signal is sent to the latches to store the level present on the line. After the cycle is completed the latches are read to see what level was detected and a comparison is made to the level that was supposed to be on the line and an error reported if a difference exists.

Not only is the checking of lines done during a Bus Test but it happens during all pod operations. This helps detect dynamic faults that occur while a UUT access is taking place.

Using this technique makes it possible to check the microprocessor lines for proper level operation, which gives the ability to check for stuck lines and tied lines. With the level detection inside the Interface Pod we cannot measure levels along the bus lines, making it impossible to check for open bus lines using the Bus Test. As we will see later, other built-in tests can be used to check for open bus lines.


BUS
TEST BUS AT ADDR
0
thru
F
DEC
INC
ENTER
YES
ADDR OPTION:
MEMORY
I/O
WORD
BYTE
INSTRUCT

How to Use the Bus Test

The keystrokes necessary to begin the test are illustrated to the left. The example uses the address of 0. Address 0 is the first address of RAM space on the UUT.

The address specified in the Bus Test should be in memory that is readable and writable. Other locations should only be specified if they physically exist and are not written to by other devices.

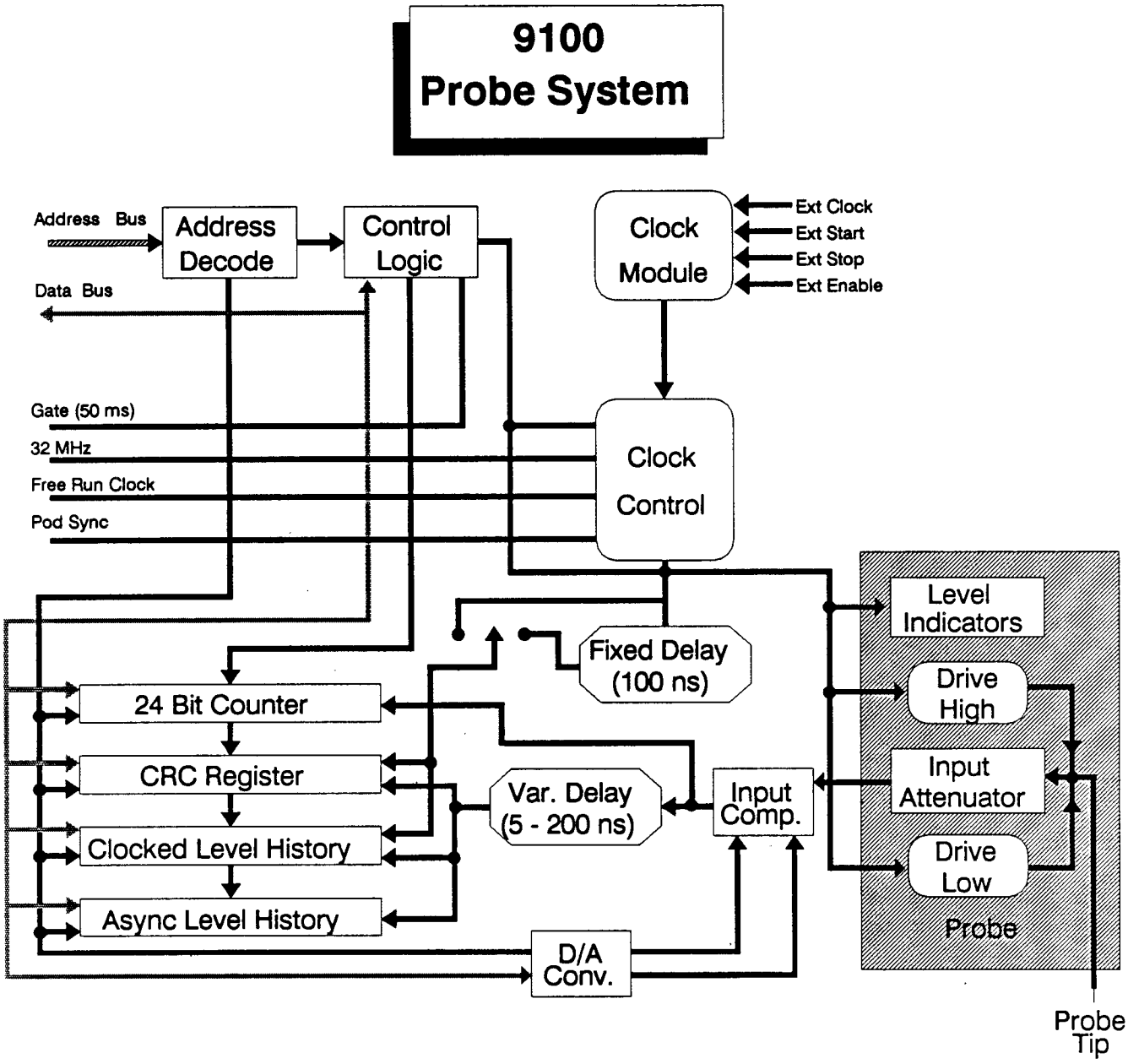
Exercise 4-1
Perform a Bus Test

Press:

Faults for address, data, or control that are displayed on the Mainframe identify the line name, and reference that line back to the microprocessor pin number.

The Bus Test identifies lines that are stuck high, stuck low, or tied together. Bus Test uncovers all drivability problems that occur at the microprocessor socket. Usually, faults are likely to be detected during execution of other tests, but the Bus Test diagnostics are best here, since the bus is isolated to a very small area that affects most all other circuitry.

As a Class:
Refer to Appendix F in the Technical User's Manual and review the Bus Test Fault messages and Pod Related Fault messages.


9100
Probe System
Address Bus
Address Decode
Control Logic
Clock Module
Ext Clock
Ext Start
Ext Stop
Ext Enable
Data Bus
Gate (50 ms)
32 MHz
Free Run Clock
Pod Sync
Clock Control
Level Indicators
Fixed Delay (100 ns)
Drive High
24 Bit Counter
CRC Register
Var. Delay (5 - 200 ns)
Input Comp.
Input Attenuator
Clocked Level History
Drive Low
Async Level History
Probe
D/A Conv.
Probe Tip

Probe Operations

The following table contains the electrical specifications for the 9100-Series probe.

Input Threshold

TTL VOLTAGE	CMOS VOLTAGE	RS-232 VOLTAGE	ECL VOLTAGE	LOGIC LEVEL
2.6 to 5.0V	3.7 to 5.0V	3.2 to 30V	-1.15 to 30.0V	1
2.2 to 2.6V	3.3 to 3.7V	2.8 to 3.2V	-1.35 to -1.15V	1 or X
1.0 to 2.2V	1.2 to 3.3V	-2.8 to 2.8V	-	X
0.6 to 1.0V	0.8 to 1.2V	-3.2 to -2.8V	-1.55 to -1.35V	X or 0

Input Impedance

70K ohm shunted by less than 33 pF

Data Timing For Synchronous Measurements

Maximum Frequency: 40 MHz

Minimum Pulse Width (H, L): 12.5 nsec

Minimum Pulse Width(3-state): 20 nsec

Setup Times

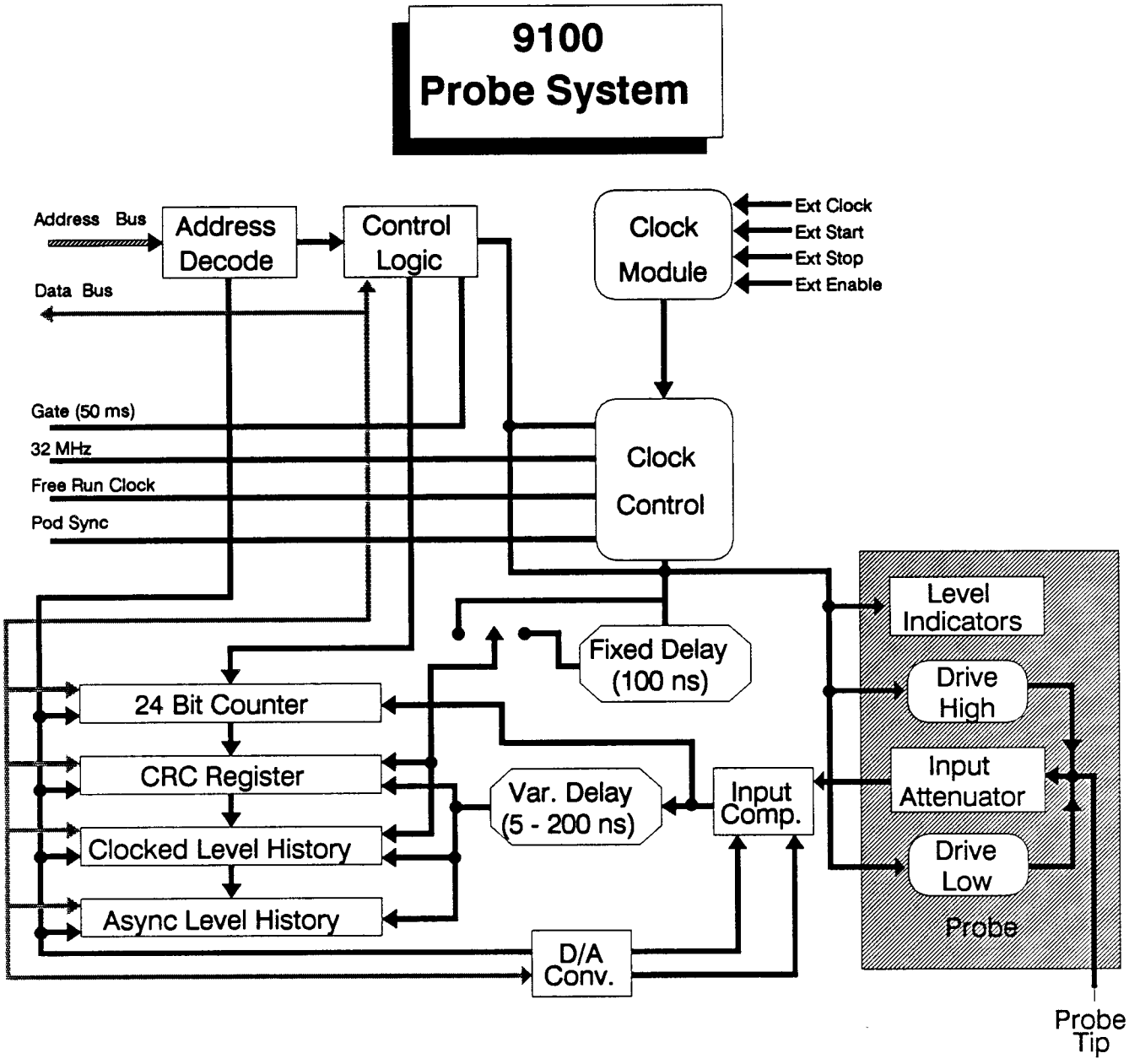
Data to Clock: 5 nsec

Start, Stop, or Enable to Clock: 10 nsec

Hold Time

Clock to Enable: 10 nsec

Clock to Start or Stop: 0.0 nsec


9100
Probe System
Address Bus
Address Decode
Control Logic
Clock Module
Ext Clock
Ext Start
Ext Stop
Ext Enable
Data Bus
Gate (50 ms)
32 MHz
Free Run Clock
Pod Sync
Clock Control
Level Indicators
Fixed Delay (100 ns)
Drive High
24 Bit Counter
CRC Register
Var. Delay (5 - 200 ns)
Input Comp.
Input Attenuator
Clocked Level History
Drive Low
Async Level History
Probe
D/A Conv.
Probe Tip

Data Timing For Asynchronous Measurements

Maximum Frequency:	40 MHz
Minimum Pulse Width (H or L):	12.5 nsec
Minimum Pulse Width - Invalid (X)	
TTL or CMOS:	100 nsec +/- 20 nsec
RS-232:	2000 nsec +/- 400 nsec

Transition Counting

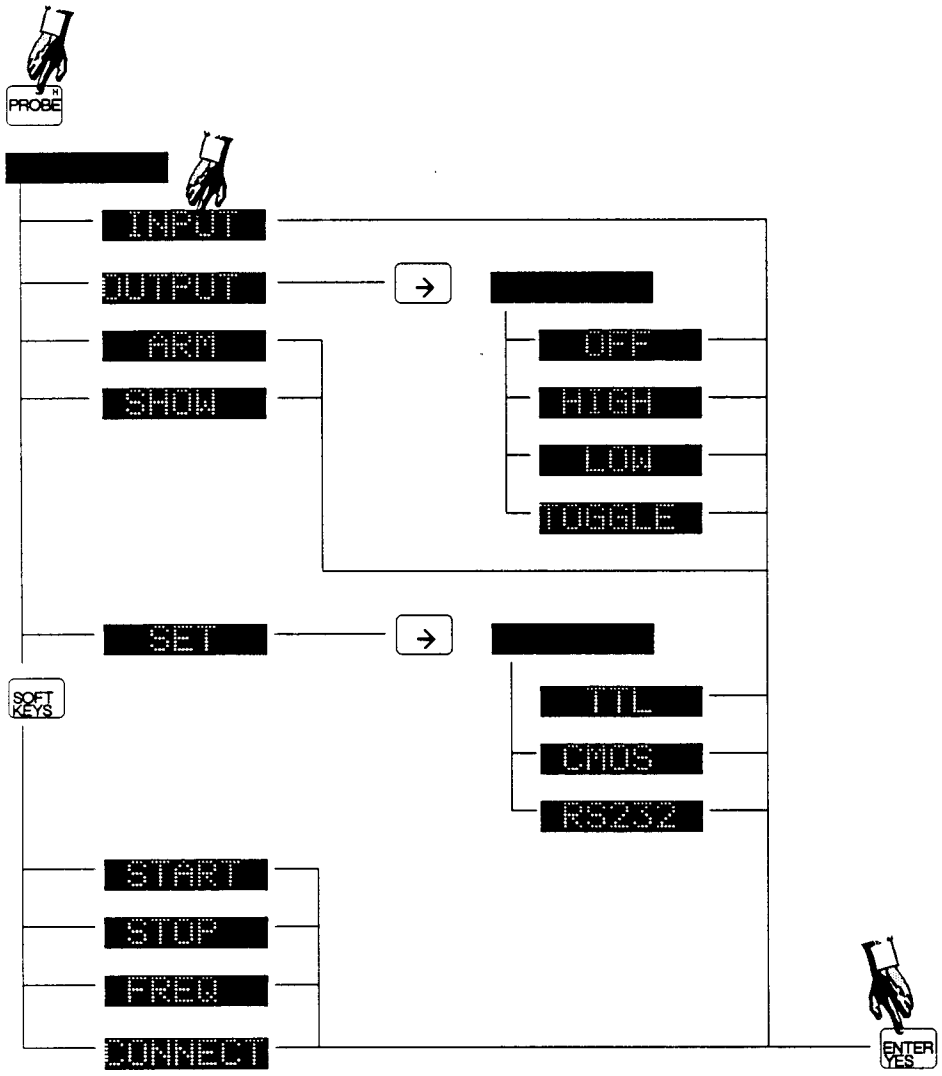
Maximum Frequency:	40 MHz
Maximum Count:	16777216 + overflow
Maximum Stop Count:	65536 clocks

Frequency Measurement

Maximum Frequency:	40 MHz
Resolution:	20 Hz
Accuracy:	+/- 250 ppm X Freq. +/- 20 Hz

Output Pulser

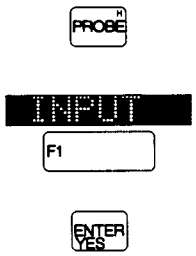
High:	>3.5V @ 200 mA for less than 10 μsec >4.5V @ 1 mA continuously


PROBE
INPUT
OUTPUT
ARM
SHOW
OFF
HIGH
LOW
TOGGLE
SET
TTL
CMOS
RS232
SOFT KEYS
START
STOP
FREQ
CONNECT
ENTER YES

Exercise 4-2
Probe Input Sync Freerun

This exercise illustrates the PROBE INPUT function, with the probe synchronized to freerun. Two pieces of information are provided when this function is selected: the Input Probe Level (current) and the Level History (since last input).

1. To begin the exercise, sync the Probe to freerun. Press the following sequence of keys:

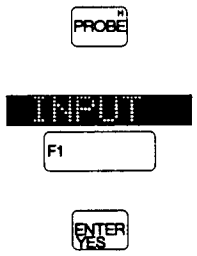


The following results are displayed:

```
INPUT PROBE LEVEL = X
    LEVEL HISTORY = X
```

The results indicate that the current level on the probe is in the invalid region. The probe has only seen an invalid condition since the last input statement.

2. Place the probe on the UUT GND and press the following sequence of keys:



The following results are displayed:

```
INPUT PROBE LEVEL = 0
    LEVEL HISTORY = X0
```

3. With the Probe still in place, press ENTER YES. The following results are displayed:

```
INPUT PROBE LEVEL = 0
    LEVEL HISTORY = 0
```

Since the Probe was never removed from UUT GND, the level history indication is now only 0 (low).

4. Place the PROBE on the UUT +5V power source. Press ENTER YES.

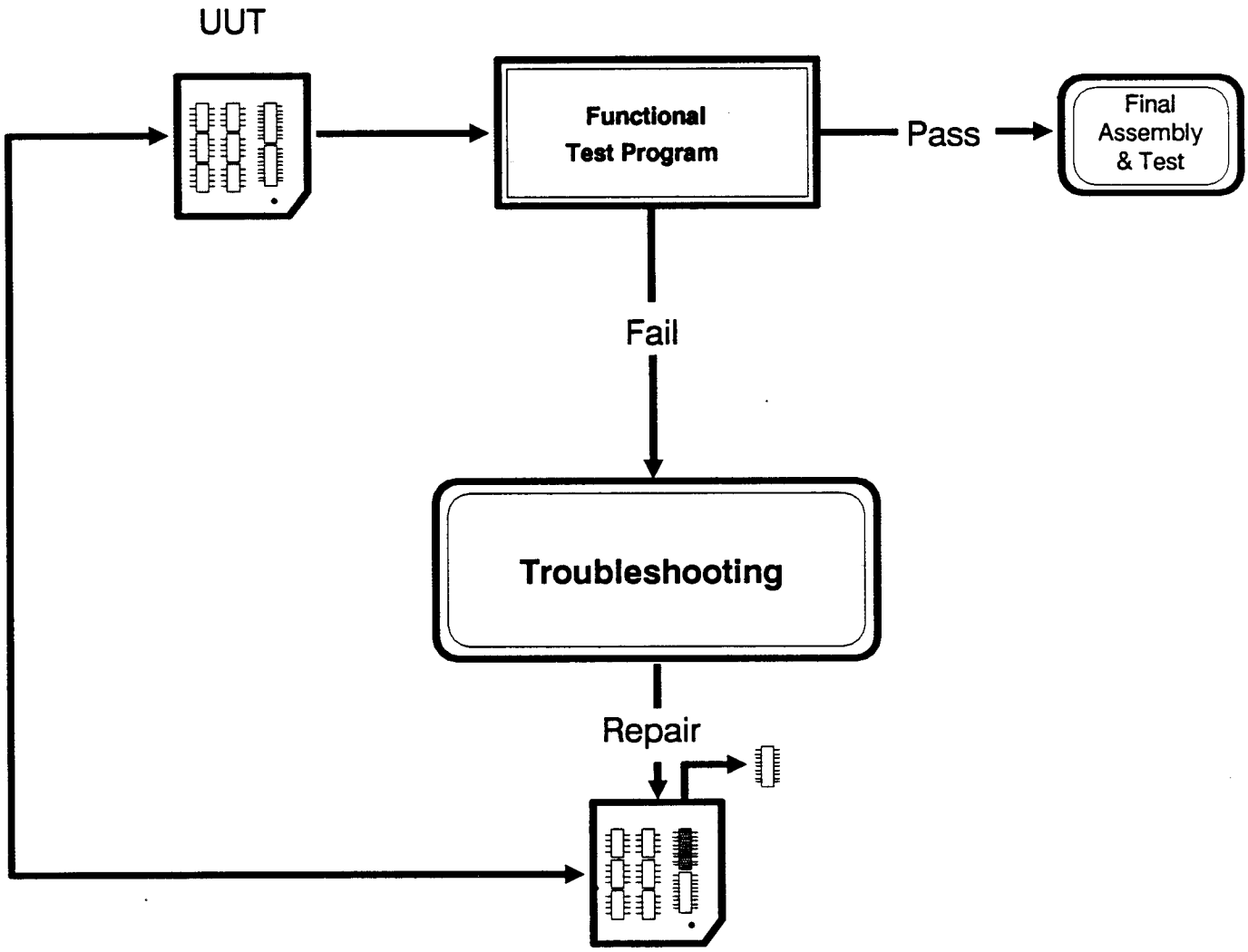
   The following results are displayed:

```
INPUT PROBE LEVEL = 1
    LEVEL HISTORY = 1X0
```

The current level is 1 (high), but note the level history. Since the last input function, the probe detected a low, invalid, and high condition.

5. Sync the Probe to Freerun

   a. Loop on bus test and probe a few data, address and control lines.
   b. Set fault switch 2-7 and repeat step 5-a.


UUT
Functional
Test Program
Pass
Final
Assembly
& Test
Fail
Troubleshooting
Repair

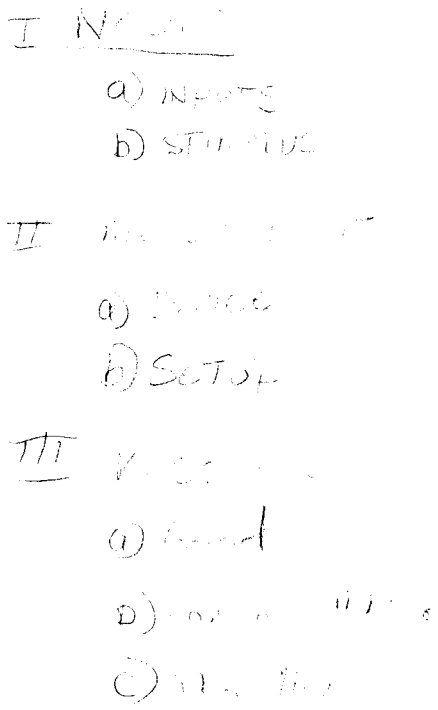
General Test and Troubleshooting Flow

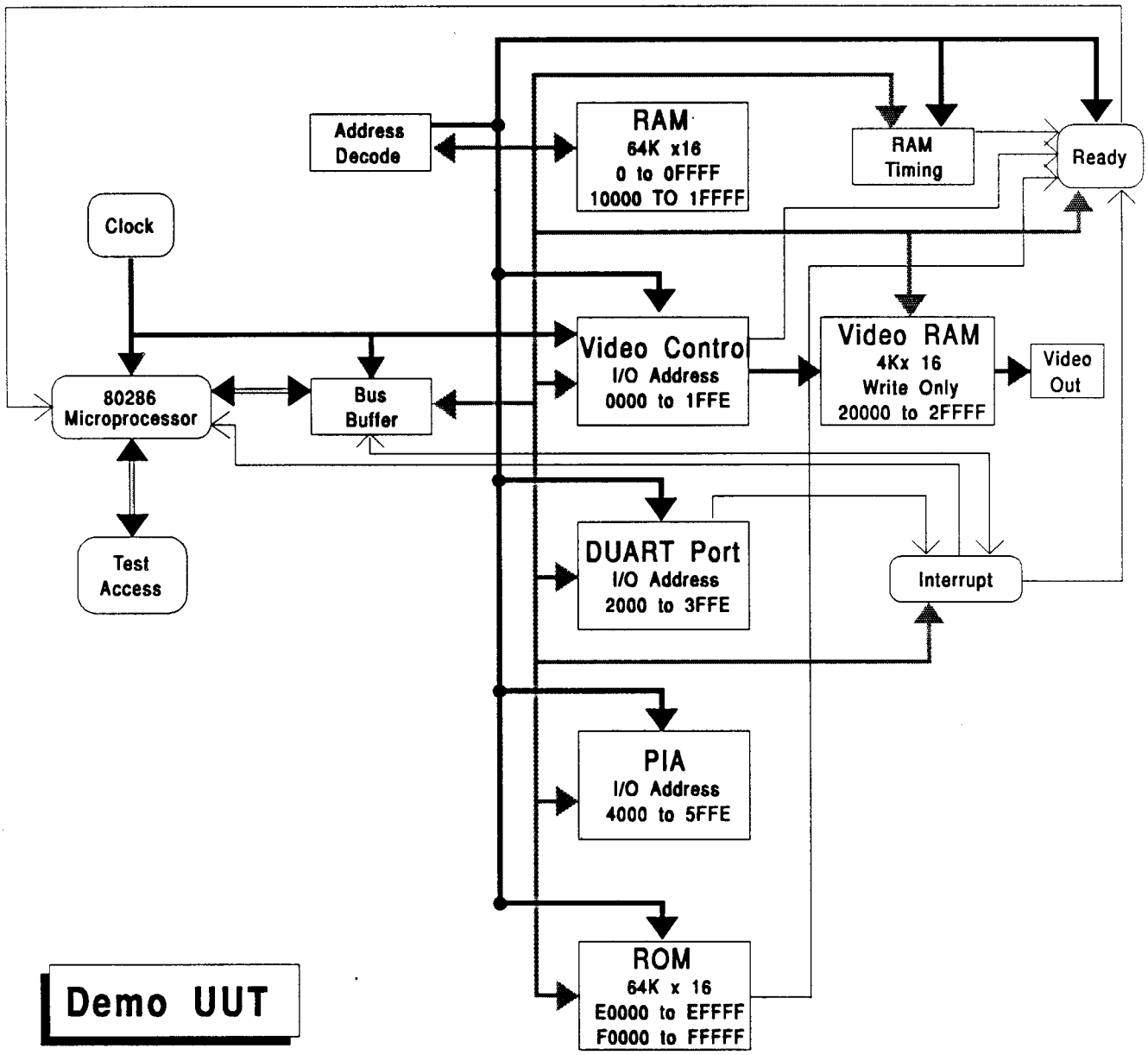
The 9100 series of instruments have two modes of operation:

- Functional Test
- Fault Isolation

The basic functions of a functional test system and fault isolation system are similar. During either task, the system must emulate bus cycles and measure levels and signal patterns. But the two tasks have different goals. During functional testing, the goal is to determine whether a UUT is good or bad; it is not necessary to know where the faults are. However, in fault isolation the goal is to determine what component is bad or what node is bad so that the UUT can be repaired.

Both functional testing and fault isolation require some form of stimulus with an appropriate way of measuring the stimulus results. We have to pass stimulus through the circuitry and then evaluate or measure the response and make a decision based on how that response compares with what we know to be a good response.




Address Decode
RAM
64K x16
0 to 0FFFF
10000 TO 1FFFF
RAM Timing
Ready
Clock
Video Contro
I/O Address
0000 to 1FFE
Video RAM
4Kx 16
Write Only
20000 to 2FFFF
Video Out
80286 Microprocessor
Bus Buffer
Test Access
DUART Port
I/O Address
2000 to 3FFE
Interrupt
PIA
I/O Address
4000 to 5FFE
ROM
64K x 16
E0000 to EFFFF
F0000 to FFFFF
Demo UUT

Whether functional testing or performing fault isolation you must understand THE NODE to be tested, how to make THE MEASUREMENT, and what to do with THE RESPONSE.

The Node

Testing the node requires a thorough understanding of its operation. It is therefore necessary to begin by finding all of the related inputs to the node.

Once all of the related inputs have been found, determine to what level each of these lines must be driven so the node in question "toggles". Once the related inputs have been identified and the proper driving levels have been determined, decide what or if any microprocessor commands can be used to drive those inputs. Then associate the microprocessor action to available 9100 commands.

The Measurement

Make your test equipment talk to you. Before a measurement can be made, a troubleshooter must set up the measurement conditions. These may include certain functions, ranges, and modes that apply specifically to that piece of test equipment. Once all of the device setup has been established look at the node in question. Does the node require any circuitry specific setups? This becomes especially important on circuitry controlled by programmable devices.

The Response

The last step in testing a node when performing fault isolation is to measure the response. The microprocessor must be capable of driving the node's associated input lines. The stimulus will then drive the node so that all possible problems that may occur on the node will be identified.

Once the results have been gathered the troubleshooter must perform one of the most difficult tasks, the evaluation of the results.

Did a fault occur? If so, what type? Do the results indicate where the next test should occur?

In this course we will explore each of the elements involved in functional test and fault isolation using the immediate mode of operation.

Each exercise will introduce different stimulus and measurement capabilities of your test equipment which will allow you to start developing test and fault isolation strategies for your own UUT.

Hands-On Training 4-1

1. Set fault switch 2-5 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL
LOOP TEST	FAIL

4. Fault Message (if there is one)

   H4 STUCK LOW PIN 27

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault?
7. Have instructor check your results.

Hands-On Training 4-2

1. Set fault switch 2-7 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL
[illegible]	Fail

4. Fault Message (if there is one)

D12 [illegible] Stuck High

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault?
7. Have instructor check your results.

Hands-On Training 4-3

1. Hold the RESET switch on the Demo UUT and do a Bus Test.
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL

4. Fault Message (if there is one)

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault?
7. Have instructor check your results.

Hands-On Training 4-4

1. Set fault switch 2-4 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL

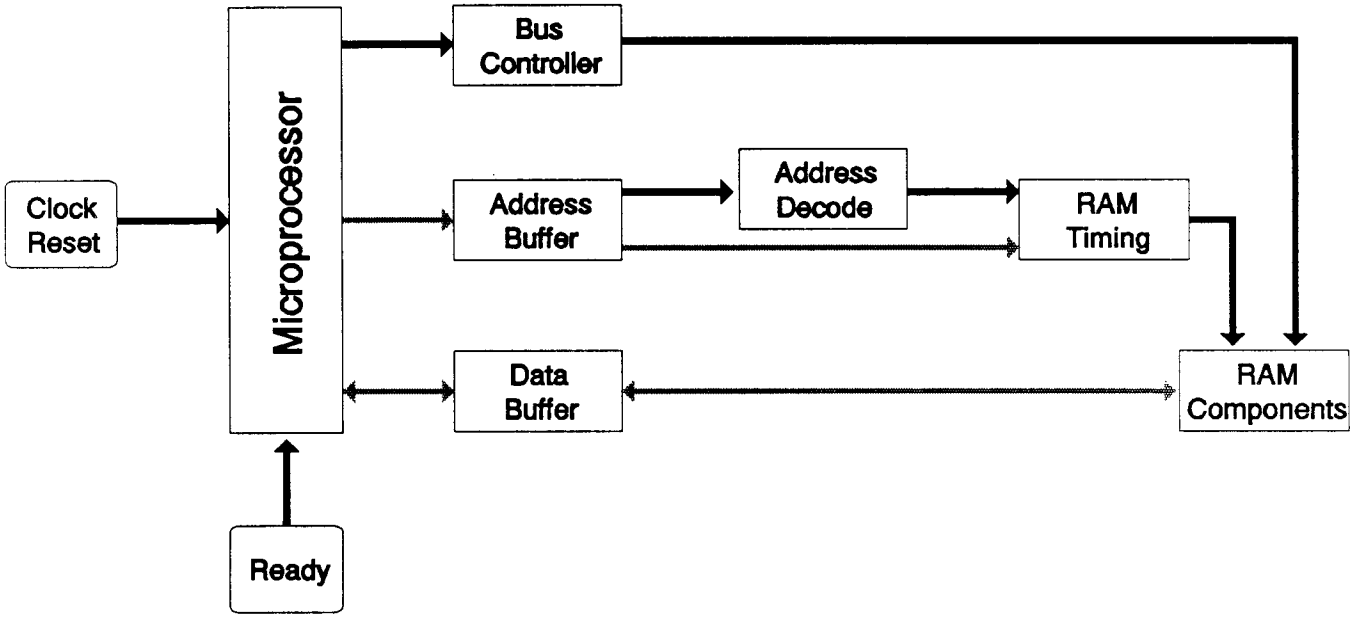
4. Fault Message (if there is one)

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault?
7. Have instructor check your results.

Section 5
RAM Testing


Clock
Reset
Microprocessor
Bus
Controller
Address
Buffer
Address
Decode
RAM
Timing
Data
Buffer
RAM
Components
Ready

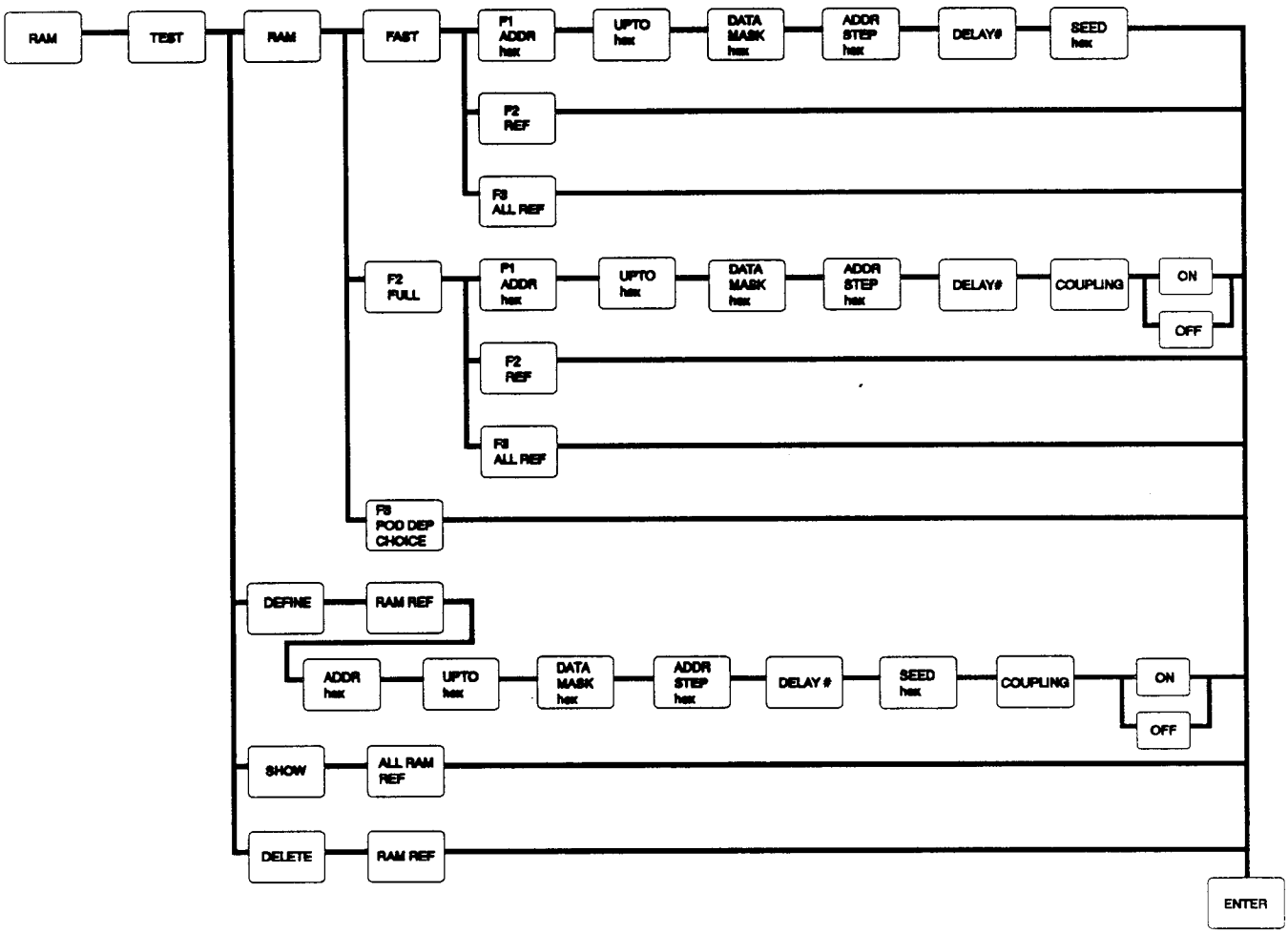
Introduction

The RAM Functional Block stores user programs in dynamic RAM memory. The block uses 4164-type chips to store 64K words of data. The data is 16 bits wide. It can be addressed as a byte (8 bits) or a word (16 bits) at a time. The RAM memory can be selected as the higher order byte, lower order byte, or the whole word.

A refresh cycle occurs every 15 microseconds if no Read or Write cycle is under way. Refresh has priority over Read or Write cycles, which are postponed until refresh is complete.

The address range of the RAM is 0-1FFFF.

As a class:
Review the UUT schematic.


RAM
TEST
RAM
FAST
F1 ADDR hex
UPTO hex
DATA MASK hex
ADDR STEP hex
DELAY#
SEED hex
F2 REF
F8 ALL REF
F2 FULL
F1 ADDR hex
UPTO hex
DATA MASK hex
ADDR STEP hex
DELAY#
COUPLING
ON
OFF
F2 REF
F8 ALL REF
F8 POD DEP CHOICE
DEFINE
RAM REF
ADDR hex
UPTO hex
DATA MASK hex
ADDR STEP hex
DELAY #
SEED hex
COUPLING
ON
OFF
SHOW
ALL RAM REF
DELETE
RAM REF
ENTER

Using the RAM Test

Use the Mainframe RAM Fast test to check the RAM functional block. RAM Fast is designed to quickly identify common RAM failures such as address decoding errors or bits that are not readable or writable. RAM Fast functionally tests all the components and support circuitry that are essential for proper RAM operation. The RAM test allows you to name each chip by name, address range, and other parameters.

RAM Fast identifies the following faults:

- Stuck cells
- Open or stuck address lines
- Open or stuck data lines
- Internal faults that affect an entire row or column
- Aliasing between data bits at the same address
- Dynamically coupled cells

When defining the area to be tested, begin by entering the first and last address.

The mask, a hex number, defines the data bits tested over the specified address range.

Lower byte mask - 00FF
Upper byte mask - FF00
Word mask - FFFF

After defining the mask, enter the step. The step specifies the address increment size.

Delay sets the number of milliseconds between memory accesses. Delay slows the RAM test to ensure that refresh to the UUT RAMs is occurring.

Seed sets the random number generator. The Seed is discussed in greater detail further on in this section.

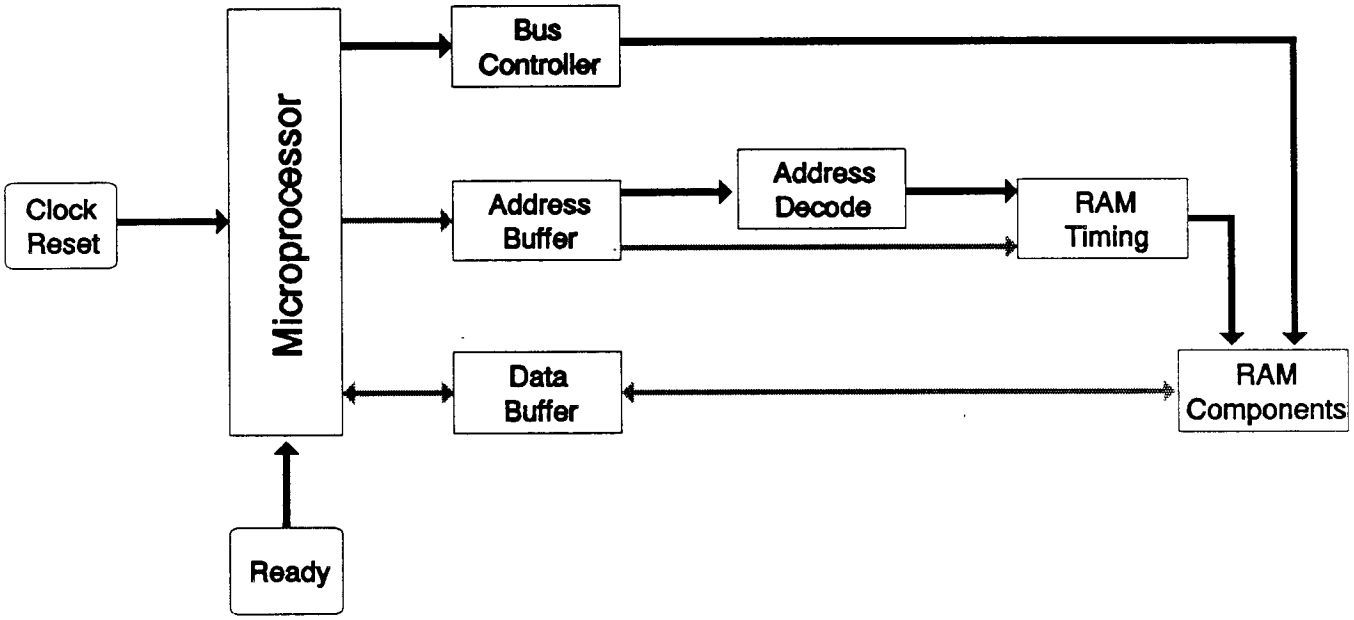
<table>
<tr><th rowspan="2">Fault Condition</th><th rowspan="2">Testramfast</th><th colspan="2">Testramfull</th></tr>
<tr><th>Coupling enabled</th><th>Coupling disabled</th></tr>
<tr><td>stuck cells</td><td colspan="3" rowspan="5">Always found</td></tr>
<tr><td>aliased cells</td></tr>
<tr><td>stuck address lines</td></tr>
<tr><td>stuck data liness</td></tr>
<tr><td>shorted address lines</td></tr>
<tr><td>multiple selection decoder</td><td rowspan="5">Maybe found</td><td colspan="2" rowspan="2">Always found</td></tr>
<tr><td>dynamic coupling</td></tr>
<tr><td>shorted data lines</td><td rowspan="2">Always found</td><td rowspan="2">Maybe found</td></tr>
<tr><td>aliasing between bits (in same word)</td></tr>
<tr><td>pattern–sensitive faults</td><td colspan="2">Not found</td></tr>
<tr><td>refresh problems</td><td colspan="3">Always found if delay is sufficiently long and standby reads do not mask the problem</td></tr>
</table>

Exercise 5-1
Performing a RAM Fast Test

Perform a RAM Fast test over the address range of 0-1FFFE. When performing the test, check all 16 data bits, use the default delay, and set seed to 1. Write the steps performed in the space below.

As a class:

Refer to the Technical User's Manual Appendix F and review the RAM Test Fault messages.


Clock
Reset
Microprocessor
Bus
Controller
Address
Buffer
Address
Decode
RAM
Timing
Data
Buffer
RAM
Components
Ready

Using Seed in the RAM Test

The last exercise used a seed of 1. If the seed is always set to 1, the random number for each individual address location is always the same every time the RAM test is run. If a seed of 567 were chosen, the number in an address location is different than a seed of 1, but is always the same for a seed of 567. However, if a seed of 0 is chosen, the numbers in each address location are different each time the test is performed.

No matter what value is specified for the seed, the probability of finding a fault is the same. However, the value of the seed may change the first error reported by the Mainframe (though this is a remote possibility). Test engineers concerned with repeatability on a particular UUT should always use the RAM Fast functional test with the same seed.

To demonstrate how the RAM Test Seed function works, perform the following exercise.

Exercise 5-2
Using the RAM Test Seed

Execute the following operations. Write the data that was read at address AA in the space provided. Compare the data at AA for the seed that was chosen.

RAM FAST 0-FE MASK FFFF STEP 2 DELAY 250 SEED 1
READ ADDR AA _____

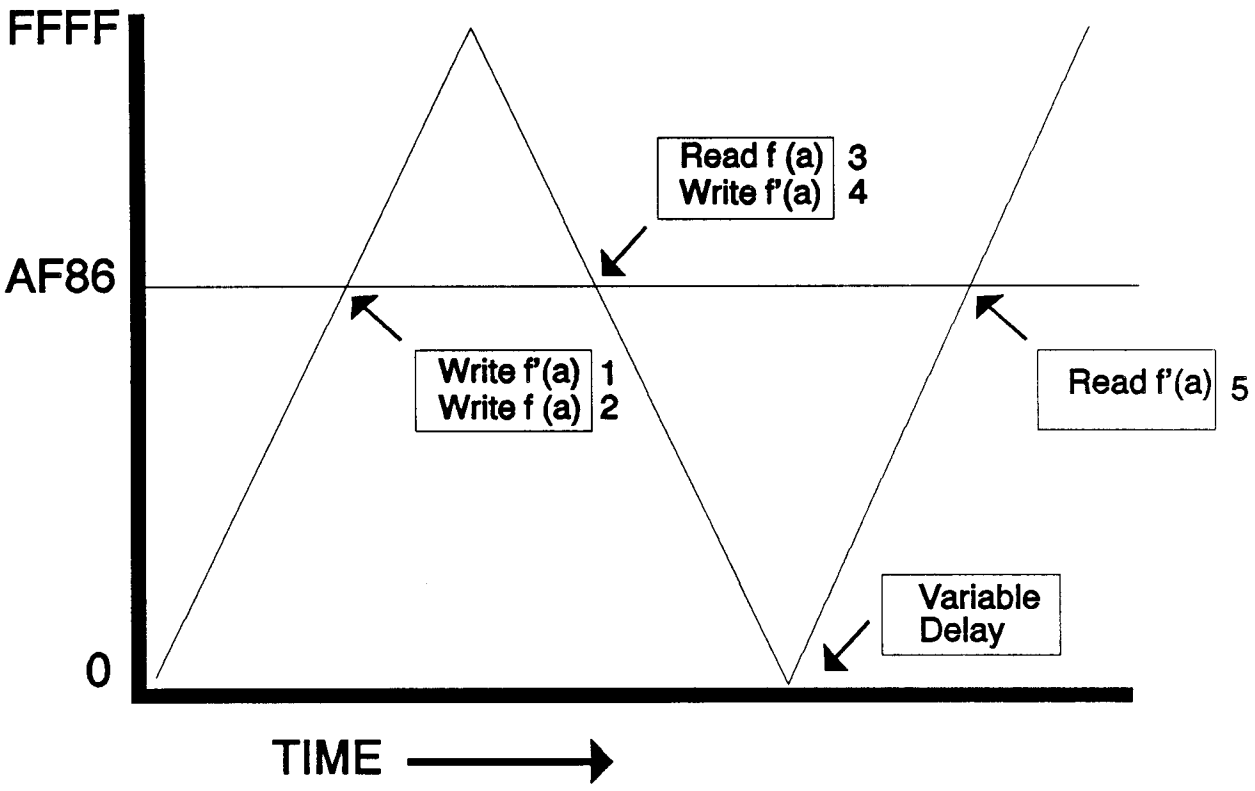
RAM FAST 0-FE MASK FFFF STEP 2 DELAY 250 SEED 1
READ ADDR AA _____

RAM FAST 0-FE MASK FFFF STEP 2 DELAY 250 SEED 567
READ ADDR AA _____

RAM FAST 0-FE MASK FFFF STEP 2 DELAY 250 SEED 567
READ ADDR AA _____

RAM FAST 0-FE MASK FFFF STEP 2 DELAY 250 SEED 0
READ ADDR AA _____

RAM FAST 0-FE MASK FFFF STEP 2 DELAY 250 SEED 0
READ ADDR AA _____


FFFF
AF86
0
TIME
Read f (a) 3
Write f'(a) 4
Write f'(a) 1
Write f (a) 2
Read f'(a) 5
Variable
Delay

How the RAM Fast Test Works

The RAM Fast test is a "Probabilistic Test" that completely covers a wide range of common faults and thoroughly covers nearly every possible RAM fault. The basic idea of the RAM Fast test is to do three marches through memory. But instead of writing all ones or all zeros as in traditional march tests, the 9100-Series RAM Fast test writes random data. The test is extremely fast, performing only five accesses to each location in the tested range of RAM.

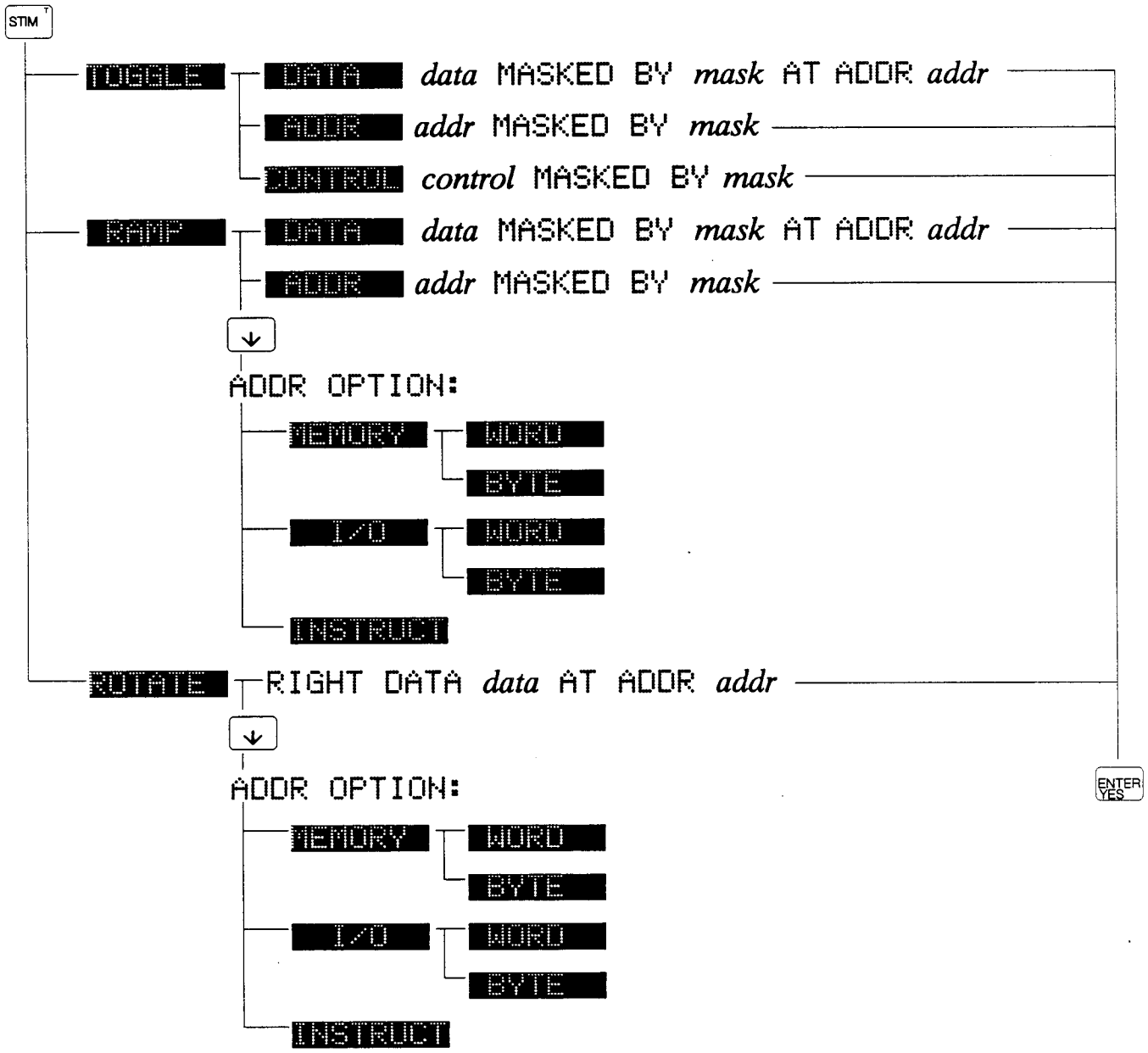
To explain the RAM Fast test, refer to the figure located on the previous page. This example demonstrates the specific address locations that are tested. The test passes over the entire address range three times while accessing each address five times.

The following steps explain how RAM is tested by the RAM Fast test:

1. The RAM Fast test begins with a brief pre-test that rapidly detects gross faults in RAM functionality.
2. A seed of 1 creates the random number D886 for address location AF86.
3. Access 1 - Write the compliment of D886 (2779).
   Access 2 - Write D886.

   Access 1 and 2 verify the operation of Static devices. A static device must be driven to both a high and a low state to determine if it is good.
4. Access 3 - Read and verify data is D886.
   Access 4 - Write the compliment of D886 (2779).

   Before the third pass begins, the RAM test performs the variable delay. This delay verifies whether the RAM circuitry needs to be refreshed, or whether refresh has occurred and RAM still contains the proper data (which was read during access 5).
5. Access 5 - Read and verify data is 2779.


STIM
TOGGLE
DATA
data MASKED BY mask AT ADDR addr
ADDR
addr MASKED BY mask
CONTROL
control MASKED BY mask
RAMP
DATA
data MASKED BY mask AT ADDR addr
ADDR
addr MASKED BY mask
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ROTATE
RIGHT DATA data AT ADDR addr
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ENTER YES

Stimulus Functions

The functions available from the stimulus menu are:

- RAMP
- TOGGLE
- ROTATE

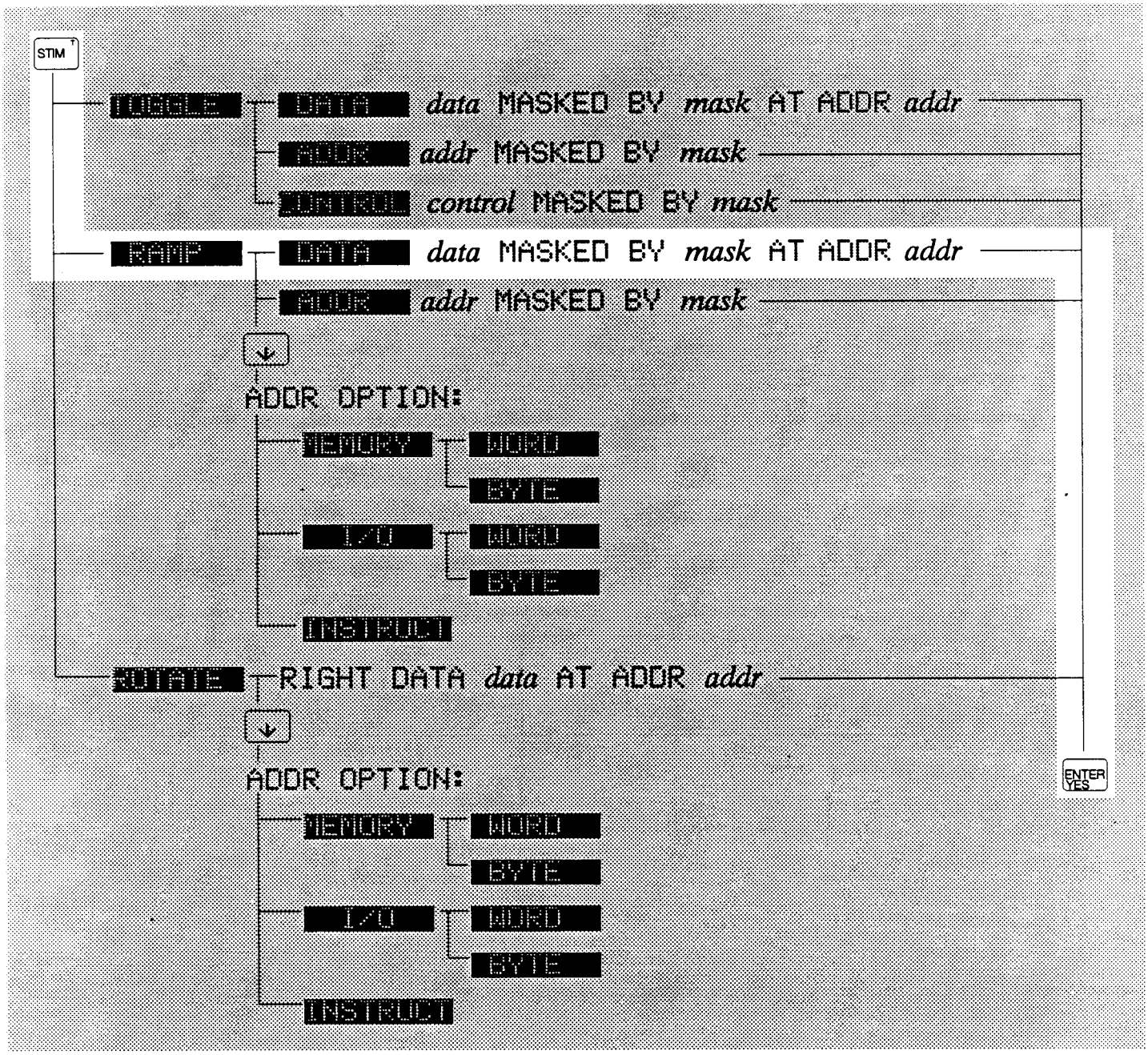
These functions perform a series of Read or Write commands and provide stimuli to locate faults or troubleshoot a fault.

Each function uses a mask to identify the bit positions that are to be stimulated, as shown in the following example.

Example

To stimulate bits 9, 8, 5, 2 and 0, use the following mask:

Bit Position	98	7654	3210
Binary Hex	11 3	0010 2	0101 5


STIM
TOGGLE
DATA
data MASKED BY mask AT ADDR addr
ADDR
addr MASKED BY mask
CONTROL
control MASKED BY mask
RAMP
DATA
data MASKED BY mask AT ADDR addr
ADDR
addr MASKED BY mask
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ROTATE
RIGHT DATA data AT ADDR addr
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ENTER
YES

Ramp Data

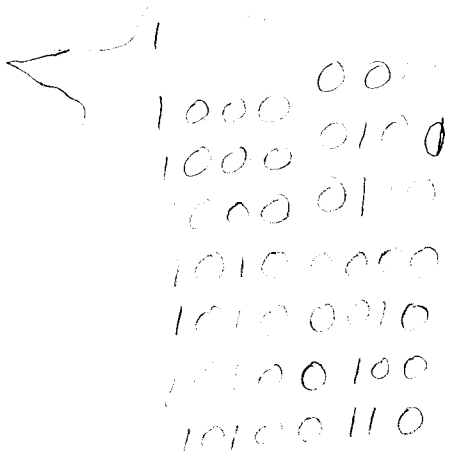
The Ramp Data command performs a series of WRITE DATA commands to a specified address. During the Ramp Data function, the masked data bits change with each write operation. The unmasked bits do not change.

When the Ramp Data function is executed, the data values are determined from the original data by setting the values of all masked bits to all combinations of ones and zeros, starting with all zeros and ending with all ones.

Example

Ramp Data 80 masked by 26 at address 0.

Mask = 0010 0110
Data = 1000 0000



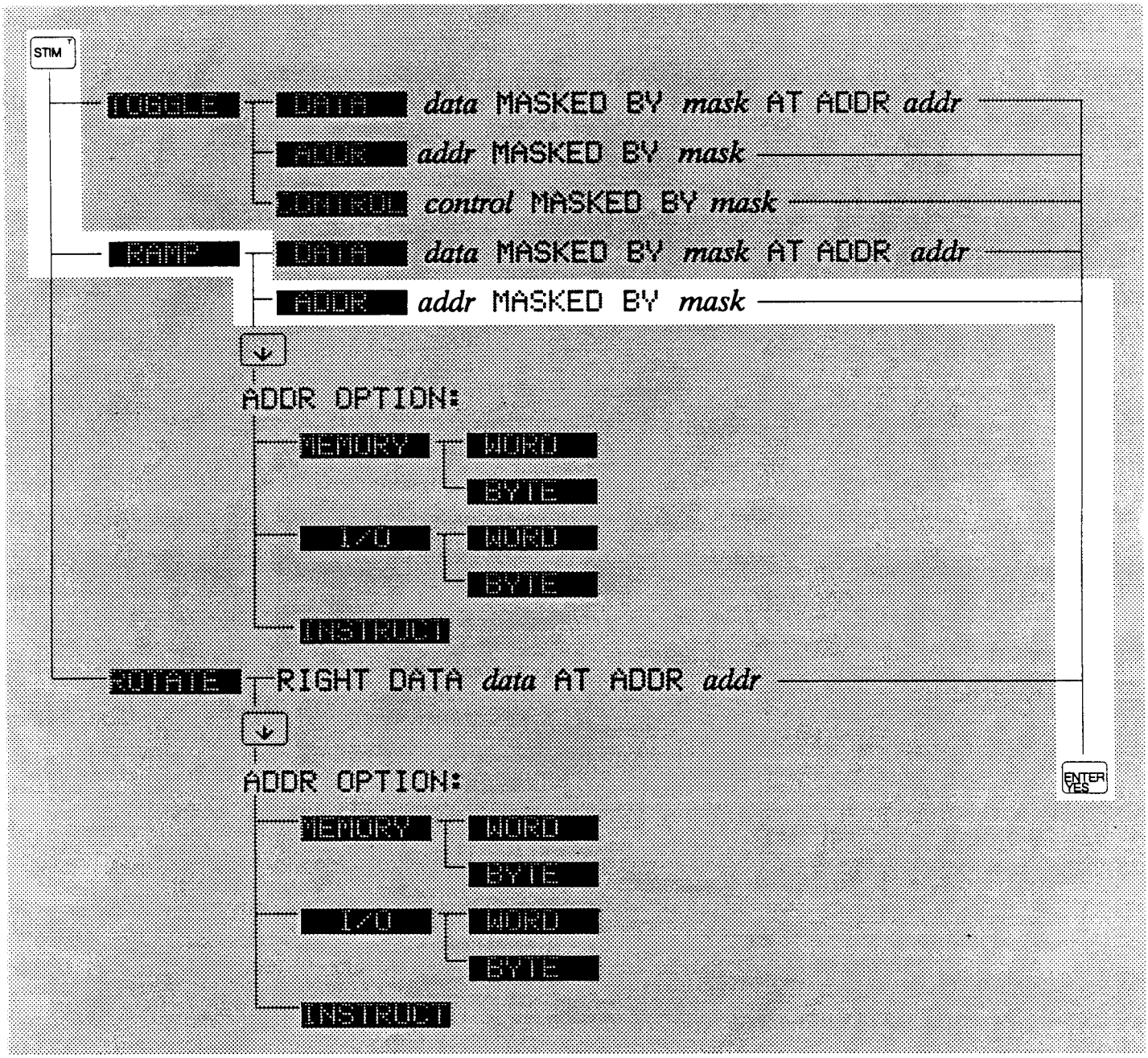
Operation performed:

Write 80	1000 0000
Write 82	1000 0010
Write 84	1000 0100
Write 86	1000 0110
Write A0	1010 0000
Write A2	1010 0010
Write A4	1010 0100
Write A6	1010 0110

Which data bits were stimulated with the previous example?

Exercise 5-3

Sync the Probe to pod data and loop on the example. Probe the data lines.


STIM
TOGGLE
DATA
data MASKED BY mask AT ADDR addr
ADDR
addr MASKED BY mask
CONTROL
control MASKED BY mask
RAMP
DATA
data MASKED BY mask AT ADDR addr
ADDR
addr MASKED BY mask
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
WALK
RIGHT DATA data AT ADDR addr
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ENTER
YES

Ramp Address

The Ramp Address command performs a series of READ ADDR commands at a specified address. During the Ramp Address function, the masked address bits change with each read operation. The unmasked bits do not change.

When the Ramp Address function is executed, the address values are determined from the original address by setting the values of the masked bits to all combinations of ones and zeros, starting with all zeros and ending with all ones.

Example

Ramp Address 1F000 masked by 700

Mask = 0 0000 0111 0000 0000
Addr = 1 1111 0000 0000 0000

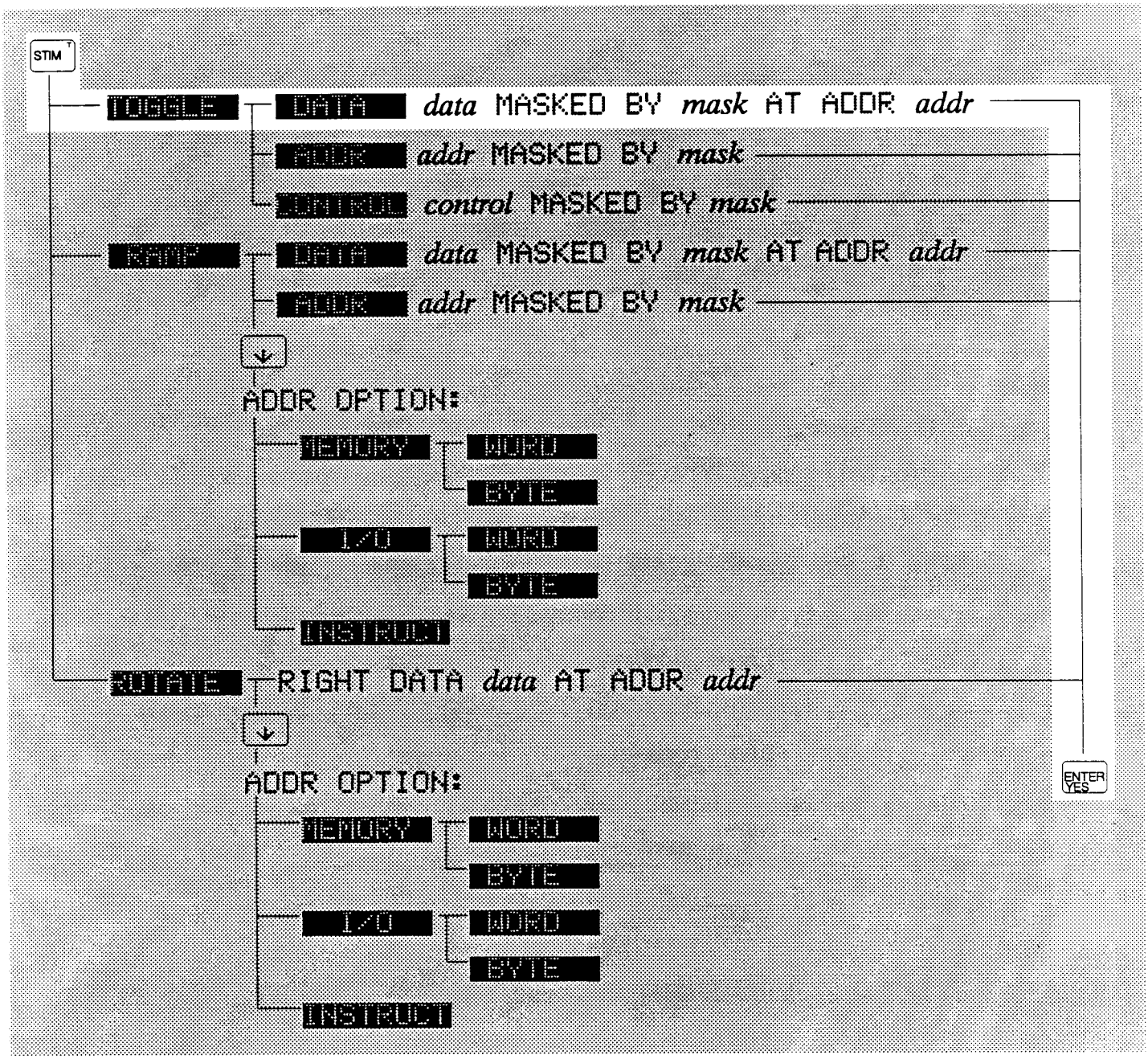
Operations performed:

Read 1F000	1 1111 0000 0000 0000
Read 1F100	1 1111 0001 0000 0000
Read 1F200	1 1111 0010 0000 0000
Read 1F300	1 1111 0011 0000 0000
Read 1F400	1 1111 0100 0000 0000
Read 1F500	1 1111 0101 0000 0000
Read 1F600	1 1111 0110 0000 0000
Read 1F700	1 1111 0111 0000 0000

Which address bits were stimulated with the previous example?

Exercise 5-4

Sync the Probe to pod address and loop on the example. Probe the address lines.


STIM
TOGGLE
DATA
data MASKED BY mask AT ADDR addr
ADDR
addr MASKED BY mask
CONTROL
control MASKED BY mask
RAMP
DATA
data MASKED BY mask AT ADDR addr
ADDR
addr MASKED BY mask
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
UNSTRUCT
ROTATE
RIGHT DATA data AT ADDR addr
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
UNSTRUCT
ENTER
YES

Toggle Data

The Toggle Data function performs two WRITE DATA commands for each bit set in the mask.

Example

Toggle Data 80 masked by 26 at address 0.

Data specified = 80 (1000 0000)
Mask = 26 (0010 0110)

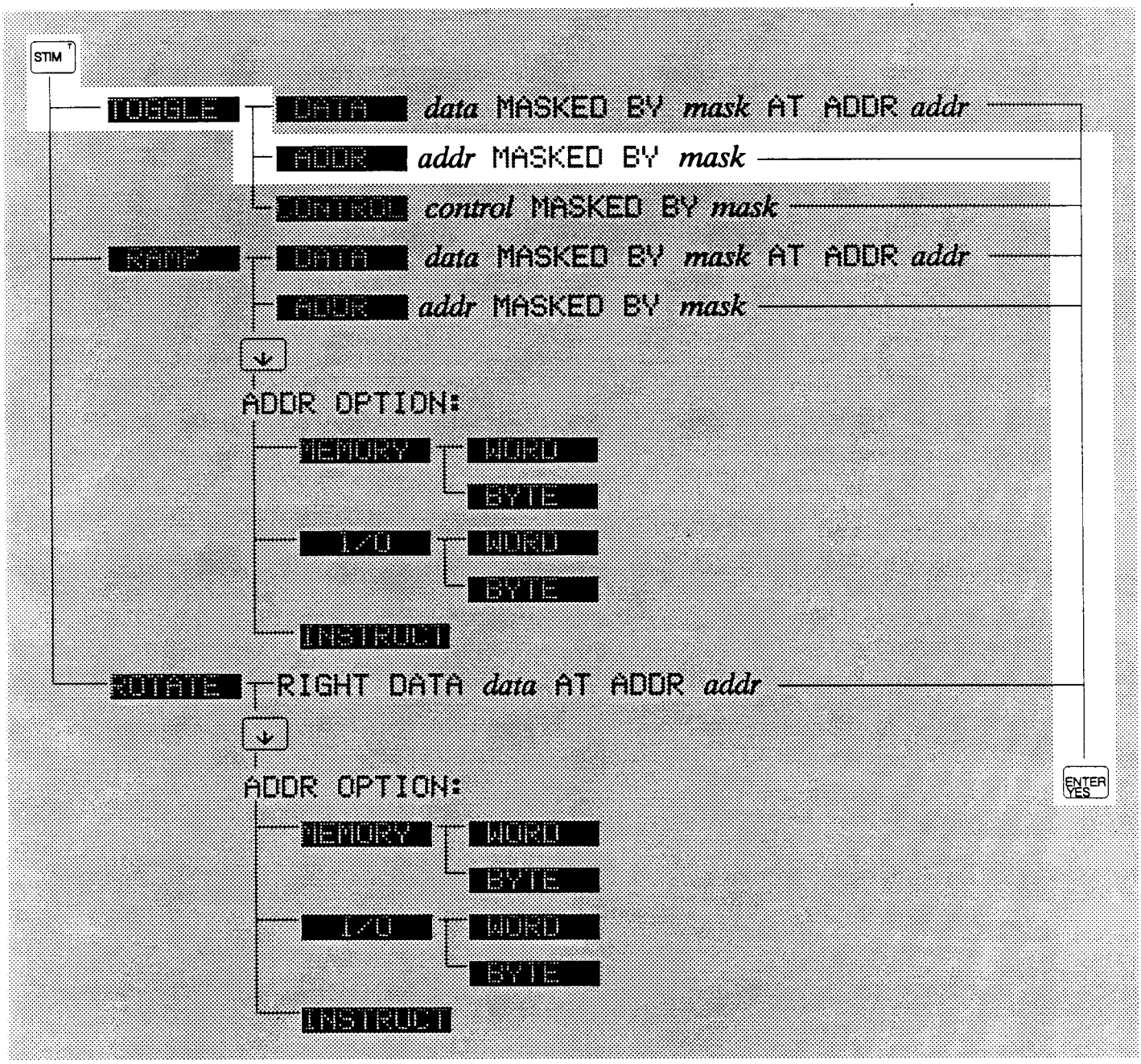
Operations:

Write 80 = 1000 0000
Write 82 = 1000 0010
Write 80 = 1000 0000
Write 84 = 1000 0100
Write 80 = 1000 0000
Write A0 = 1010 0000

How does the Toggle Data function differ from the Ramp Data function?

Exercise 5-5

Sync the Probe to pod data and loop on the example. Probe the data lines.


STIM
TOGGLE
DATA data MASKED BY mask AT ADDR addr
ADDR addr MASKED BY mask
CONTROL control MASKED BY mask
RAMP
DATA data MASKED BY mask AT ADDR addr
ADDR addr MASKED BY mask
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ROTATE
RIGHT DATA data AT ADDR addr
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ENTER YES

Toggle Address

The Toggle Address function performs two READ ADDR commands for each bit set in the mask.

Example

Toggle Address 1F000 masked by 700

Address = 1F000 (1 1111 0000 0000 0000)
Mask = 700 (0111 0000 0000)

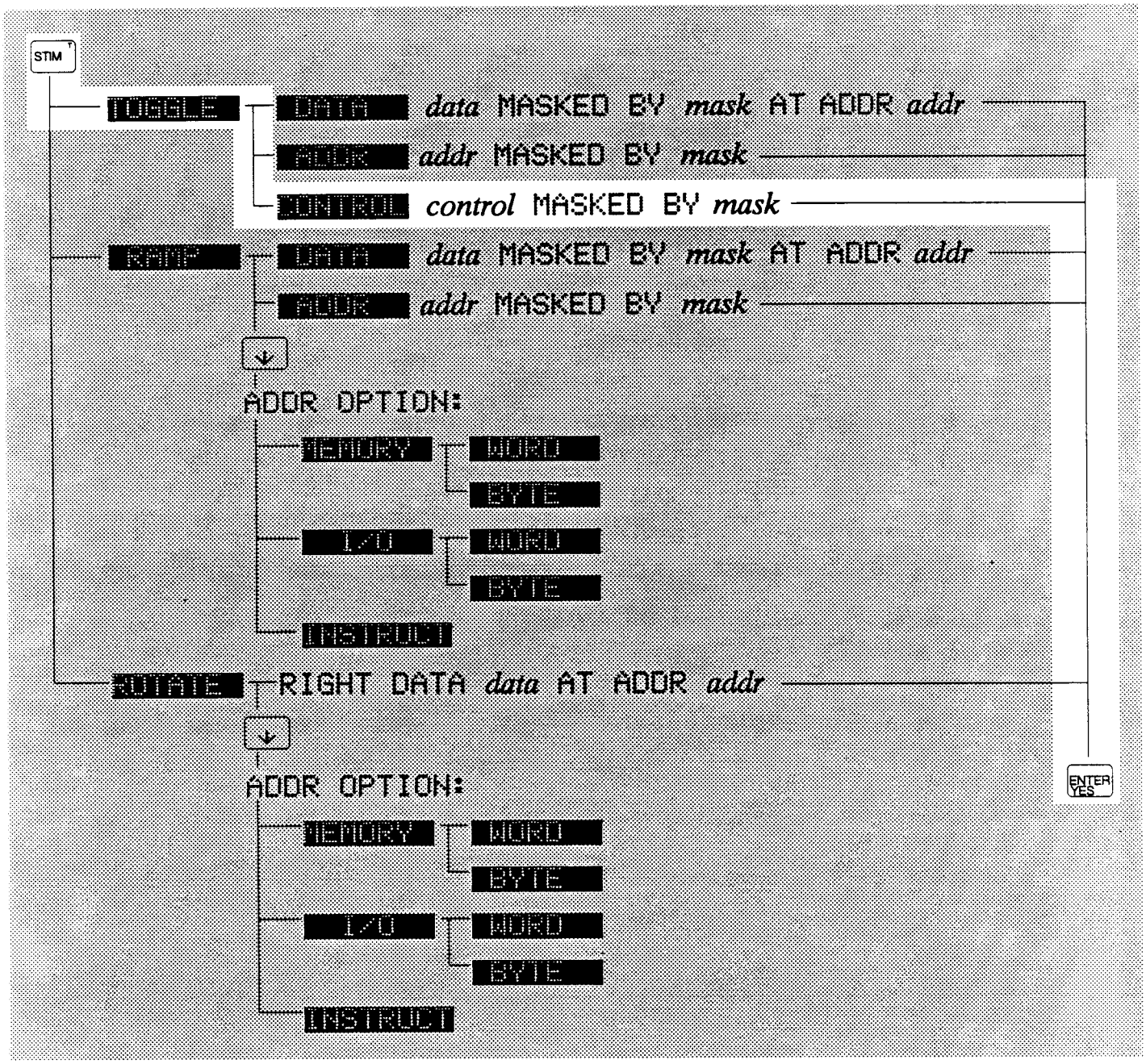
Operations:

Read Addr = 1F000	1 1111 0000 0000 0000
Read Addr = 1F100	1 1111 0001 0000 0000
Read Addr = 1F000	1 1111 0000 0000 0000
Read Addr = 1F200	1 1111 0010 0000 0000
Read Addr = 1F000	1 1111 0000 0000 0000
Read Addr = 1F400	1 1111 0100 0000 0000

How does the Toggle Address function differ from the Ramp Address function?

Exercise 5-6

Sync the Probe to pod address and loop on the example. Probe the data lines.


STIM
TOGGLE
DATA
data MASKED BY mask AT ADDR addr
ADDR
addr MASKED BY mask
CONTROL
control MASKED BY mask
RAMP
DATA
data MASKED BY mask AT ADDR addr
ADDR
addr MASKED BY mask
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ROTATE
RIGHT DATA data AT ADDR addr
ADDR OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ENTER
YES

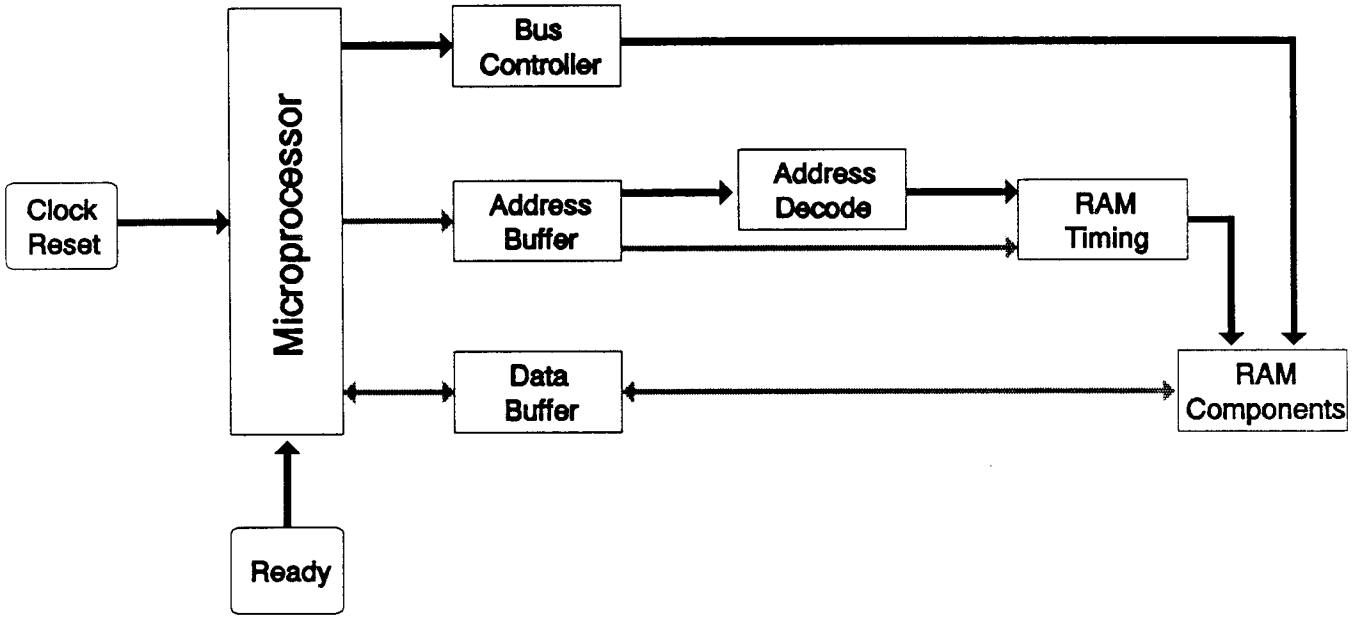
Toggle Control

The Toggle Control function performs two WRITE CONTROL operations for each bit set.

To identify the control lines that are writable for the pod, look at the decal on the back of the pod case. Any control line that can be written can also be toggled.

Rotate Data

The Rotate Data performs a series of WRITE DATA commands. First, the data is written to the specified address. Then the data is rotated right, the right-most bit becoming the left-most one. The process is repeated as many times as there are data bits.


Clock
Reset
Microprocessor
Bus
Controller
Address
Buffer
Address
Decode
RAM
Timing
Data
Buffer
RAM
Components
Ready

Hands-On Training 5-1

1. Set fault switch 1-1 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL

4. Fault Message (if there is one)

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault?
7. Have instructor check your results.

Hands-On Training 5-2

1. Set fault switch 1-5 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL

4. Fault Message (if there is one)

D7 stuck high

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault?
7. Have instructor check your results.

Hands-On Training 5-3

1. Set fault switch 1-4 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL

4. Fault Message (if there is one)

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault?
7. Have instructor check your results.

Hands-On Training 5-4

1. Set fault switch 4-8 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL

4. Fault Message (if there is one)

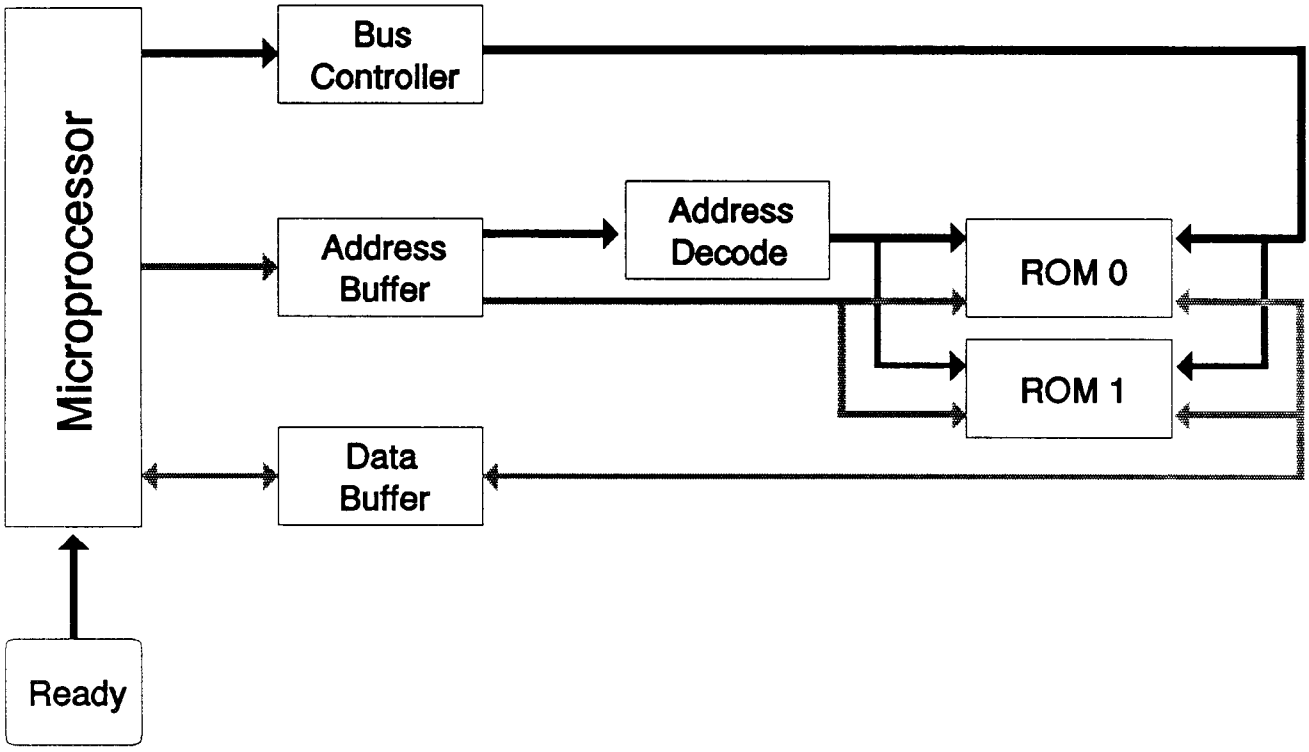
A 3 1

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault?
7. Have instructor check your results.

Section 6
ROM Testing

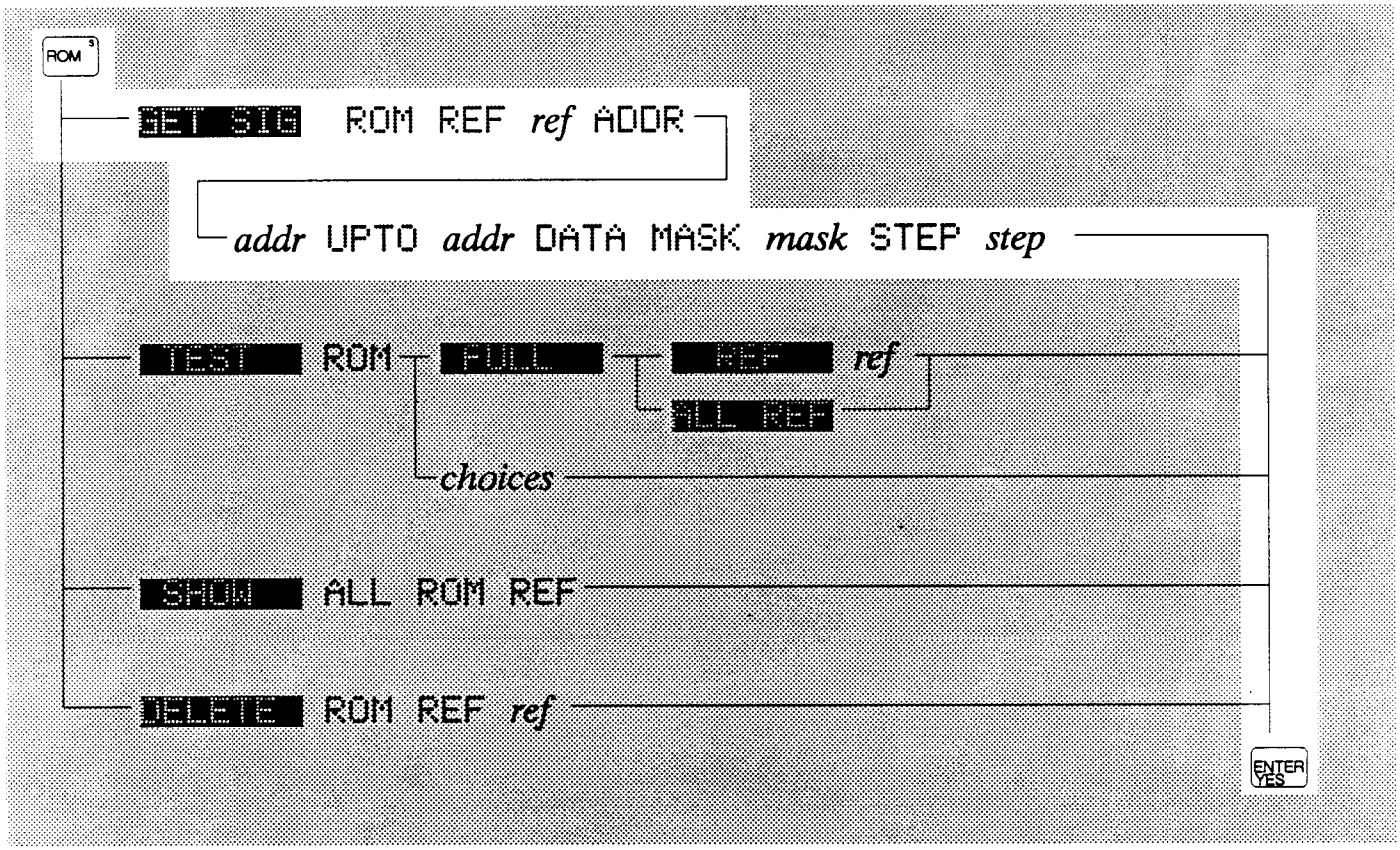

Microprocessor
Bus Controller
Address Buffer
Address Decode
ROM 0
ROM 1
Data Buffer
Ready

Introduction

The Trainer UUT ROM is made up of four 32K byte devices (27256). ROM0 (U29 and U30) is addressed from E0000 to EFFFF. ROM1 (U27 and U28) is addressed from F0000 to FFFFF.

As a class:

Review the UUT schematic.



Faults Identified

- Incorrect Signature
- Address Bit Fault
- Addrss Tied to Address Fault
- All Data Bits Tied High
- All Data Bits Tied Low
- Data Bit Fault
- Data Tied to Data Fault

Exercise 6-1
Obtaining a **ROM** Signature

Use the following procedure to obtain a ROM signature from a known good UUT:

Press	Comments
ROM	
GET SIG	GET SIG (get signature)
→	
U27	Reference designators for devices (enter this field using the alpha characters on the application keypad)
→	
F0000	First address of ROM
→	
FFFFE	Last address of ROM
→	
FF	Mask that determines which data bits are used for the signature
→	
2	Address step. Read words or bytes
ENTER YES	

Continued...

After you press ENTER YES, the following signature is shown on the Mainframe display:

SIGNATURE = F387

Repeat the previous steps to gather signatures for U28, U29 and U30. Record the signature that was obtained.

Using the ROM Test

The ROM test obtains a CRC signature (data compression technique based on the CRC-16 algorithm) from the ROM under test, and compares it to a previously stored CRC signature from an identical ROM that is known to work correctly.

If the ROM signature is found to be incorrect, a diagnostic routine uses all signature information to look for address and data lines that are open, tied to each other, stuck high or stuck low.

Exercise 6-2
Performing a ROM Test

1. Perform a ROM Test over U27, U28, U29 and U30.

2. Write the steps performed in the space provided.

As a class:

1. Refer to the technical User's Manual Appendix F and review the ROM Test Fault Messages.

Hands-On Training 6-1

1. Set fault switch 4-2 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL

4. Fault Message (if there is one)

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault? I 1 5 stuck low
7. Have the instructor check your results.

Hands-On Training 6-2

1. Set fault switch 4-1 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL

4. Fault Message (if there is one)

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault?
7. Have the instructor check your results.

Hands-On Training 6-3

1. Set fault switch 3-7 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL

4. Fault Message (if there is one)

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault?
7. Have the instructor check your results.

Hands-On Training 6-4

1. Set fault switch 1-2 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL

4. Fault Message (if there is one)

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault?
7. Have the instructor check your results.

Hands-On Training 6-5

1. Set fault switch 4-3 to its fault position
2. Record each step of your procedure using the following format.
3. Functional Test

FUNCTIONAL TEST	PASS OR FAIL

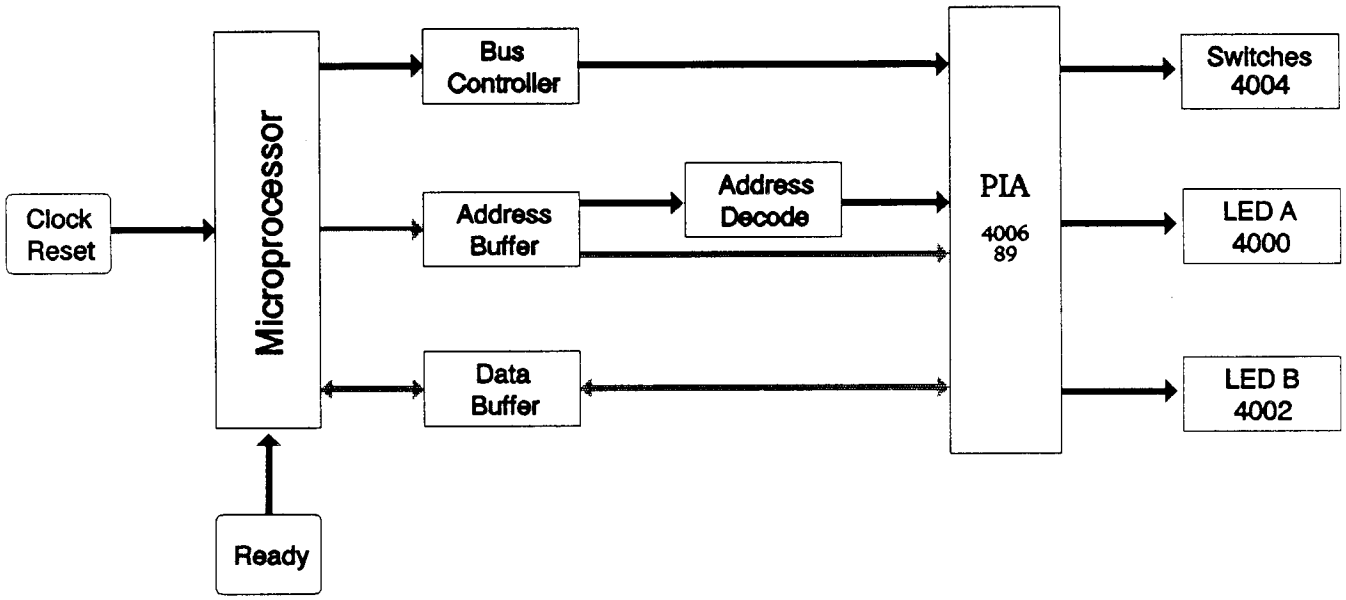
4. Fault Message (if there is one)

5. Fault Isolation

STIMULUS COMMANDS	NODE	RESPONSE

6. Evaluate
   Where's the fault? U9 PIN 1 Stuck Low
7. Have instructor check your results.

Section 7
Parallel Interface Adapter


Clock
Reset
Microprocessor
Bus
Controller
Address
Buffer
Address
Decode
Data
Buffer
Ready
PIA
4006
89
Switches
4004
LED A
4000
LED B
4002

Introduction

The parallel interface functional block will be used as an example for developing a test in an area not covered by a 9100A built-in test.

Understanding the Parallel Interface Adapter Block

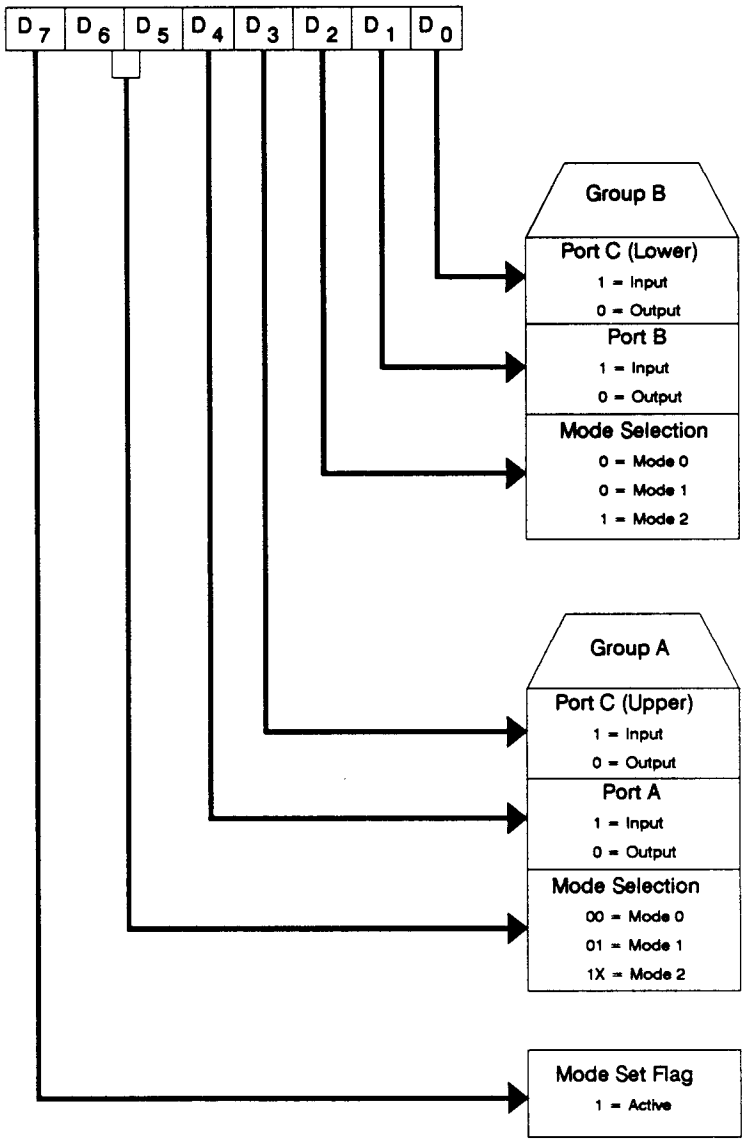
As a class:

Describe the basic operation of the functional block named "PIA" (Parallel Interface Adapter*). On the previous page is an illustration that will help you organize your thoughts. Remember to functionally test a block and declare it "good" all operations of the block must be exercised.

As a class:

Review the UUT schematic.

* This device may also be known as a *peripheral interface adapter*, a *programmable peripheral interface*, or a *programmable communications adapter.*


D7 D6 D5 D4 D3 D2 D1 D0
Group B
Port C (Lower)
1 = Input
0 = Output
Port B
1 = Input
0 = Output
Mode Selection
0 = Mode 0
0 = Mode 1
1 = Mode 2
Group A
Port C (Upper)
1 = Input
0 = Output
Port A
1 = Input
0 = Output
Mode Selection
00 = Mode 0
01 = Mode 1
1X = Mode 2
Mode Set Flag
1 = Active

Exercise 7-1
Configuring the PIA

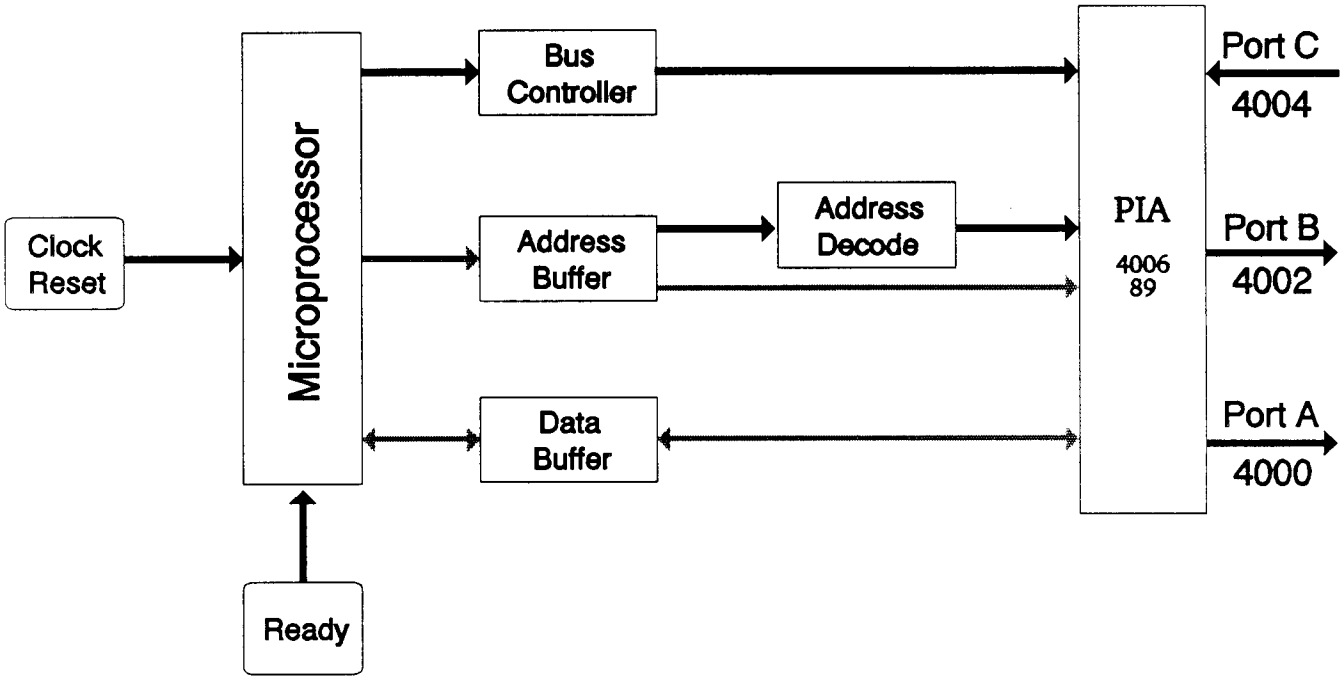
1. Determine the control word to activate:

   PORT A = output
   PORT B = output
   PORT C = input
   CONTROL = 9

2. Determine the PIA Addresses:

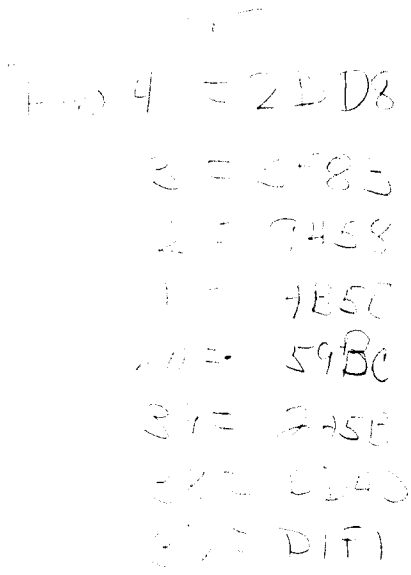
   PORT A = [illegible]
   PORT B = 11002
   PORT C = 4004
   CONTROL = 11003

3. Configure the PIA.
4. Verify configuration by displaying a 7 to LED A.

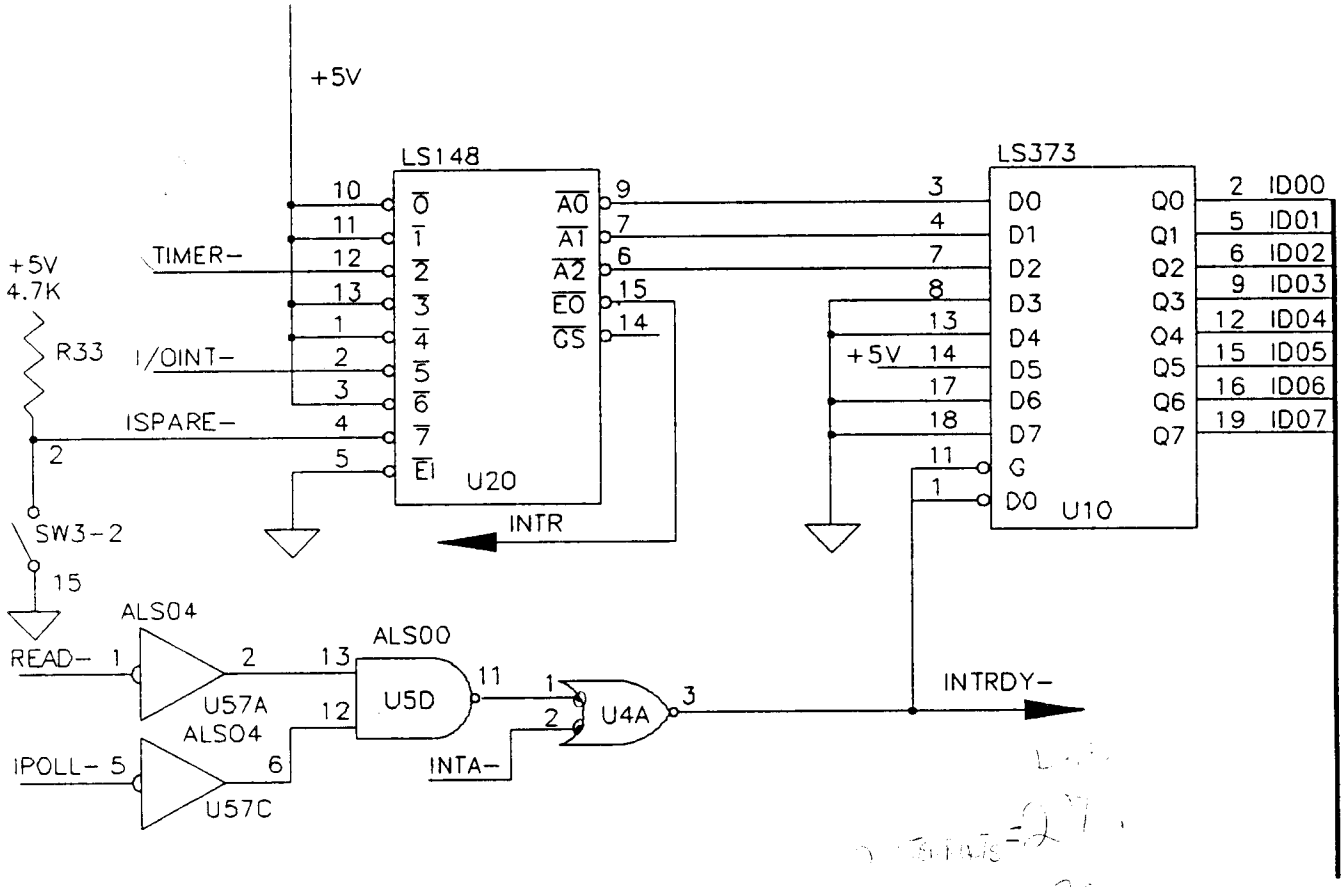

Clock
Reset
Microprocessor
Bus
Controller
Address
Buffer
Address
Decode
Data
Buffer
Ready
PIA
4006
89
Port C
4004
Port B
4002
Port A
4000

Exercise 7-2
LED Functional Test for the PIA

1. Configure the PIA for the proper input/output setting of each port.
2. Use the I/O module (pin mode) with the 40 pin clip to functionally test the PIA chip
3. As a class discuss the immediate mode capabilities of the I/O module before beginning the test.



Section 8
Fault Isolation


+5V
LS148
10
11
12
13
1
2
3
4
5
Ō
Ī
2̄
3̄
4̄
5̄
6̄
7̄
Ēi
U20
Ā0
Ā1
Ā2
ĒO
GS
9
7
6
15
14
LS373
3
4
7
8
13
+5V 14
17
18
11
1
D0
D1
D2
D3
D4
D5
D6
D7
G
D0
U10
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
2 ID00
5 ID01
6 ID02
9 ID03
12 ID04
15 ID05
16 ID06
19 ID07
+5V
4.7K
R33
TIMER–
I/OINT–
ISPARE–
2
SW3-2
15
INTR
ALS04
READ– 1
2
U57A
ALS00
13
12
U5D
11
1
2
U4A
3
INTRDY–
ALS04
IPOLL– 5
6
U57C
INTA–

Exercise 8-1
Interrupt Circuitry Stimulus

Develop a procedure to verify the interrupt circuitry is operating correctly.

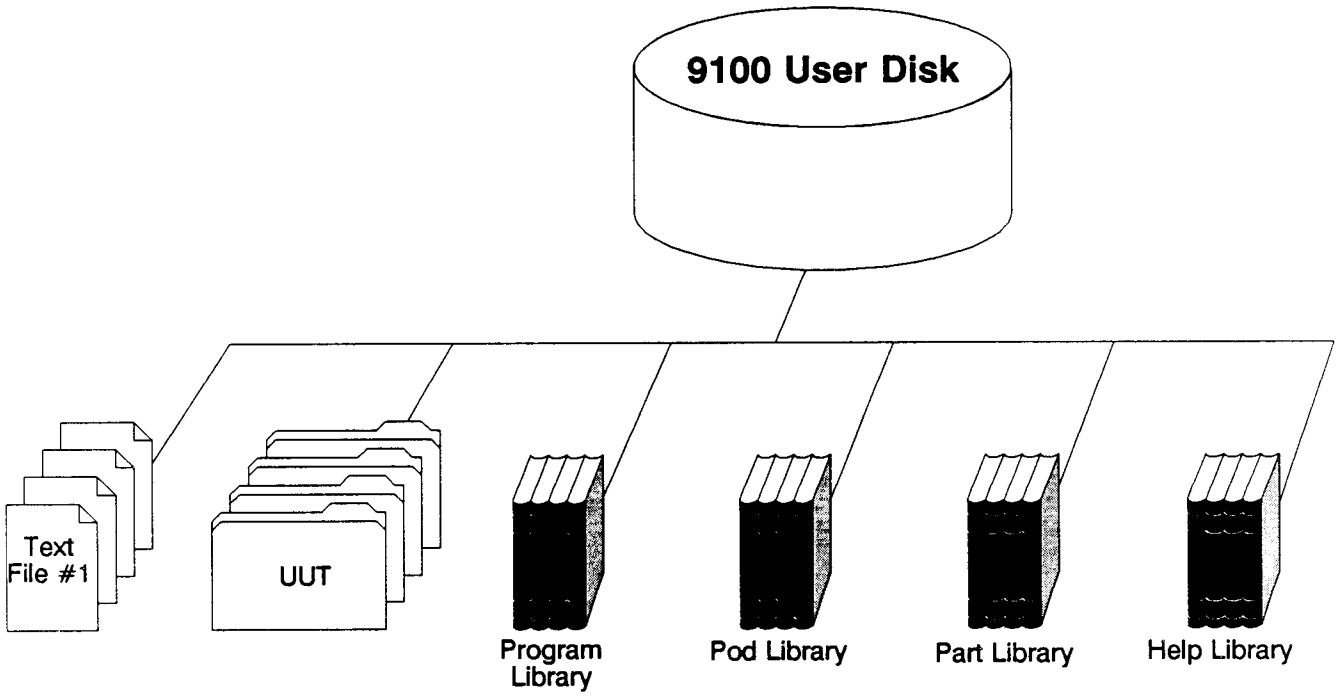
As a class:
Review test strategy.

As a class:
Review exercise when complete.

Section 9
Editor Operation

The editor is an integral part of the 9100-Series system environment. The editor allows you to create, store, or change the data and programs required for testing and troubleshooting with the 9100-Series Mainframe.

A userdisk is the formatted storage space on a disk allocated for user-accessible information. Each physical disk incorporates a userdisk that contains data and programs for one or more units under test. To provide additional userdisks, you add more floppy disks.



User Disk Level (/HDR or /DR1)

Userdisk Organization

The disk system of the 9100-Series Mainframe is structured much the same as most small computers on the market today. A userdisk consists of the following:

- Text Files

Operator's instructions and program documentation are stored in text documents.

- Part Library

The Part Library consists solely of part descriptions. The Part Library is shared by all the UUT descriptions on the same userdisk, so that the part description does not have to be duplicated for each UUT that uses the part.

- Program Library

The Program Library contains programs that perform frequently used operations that are not UUT specific. These programs are called by any other program on the userdisk.

- Pod Library

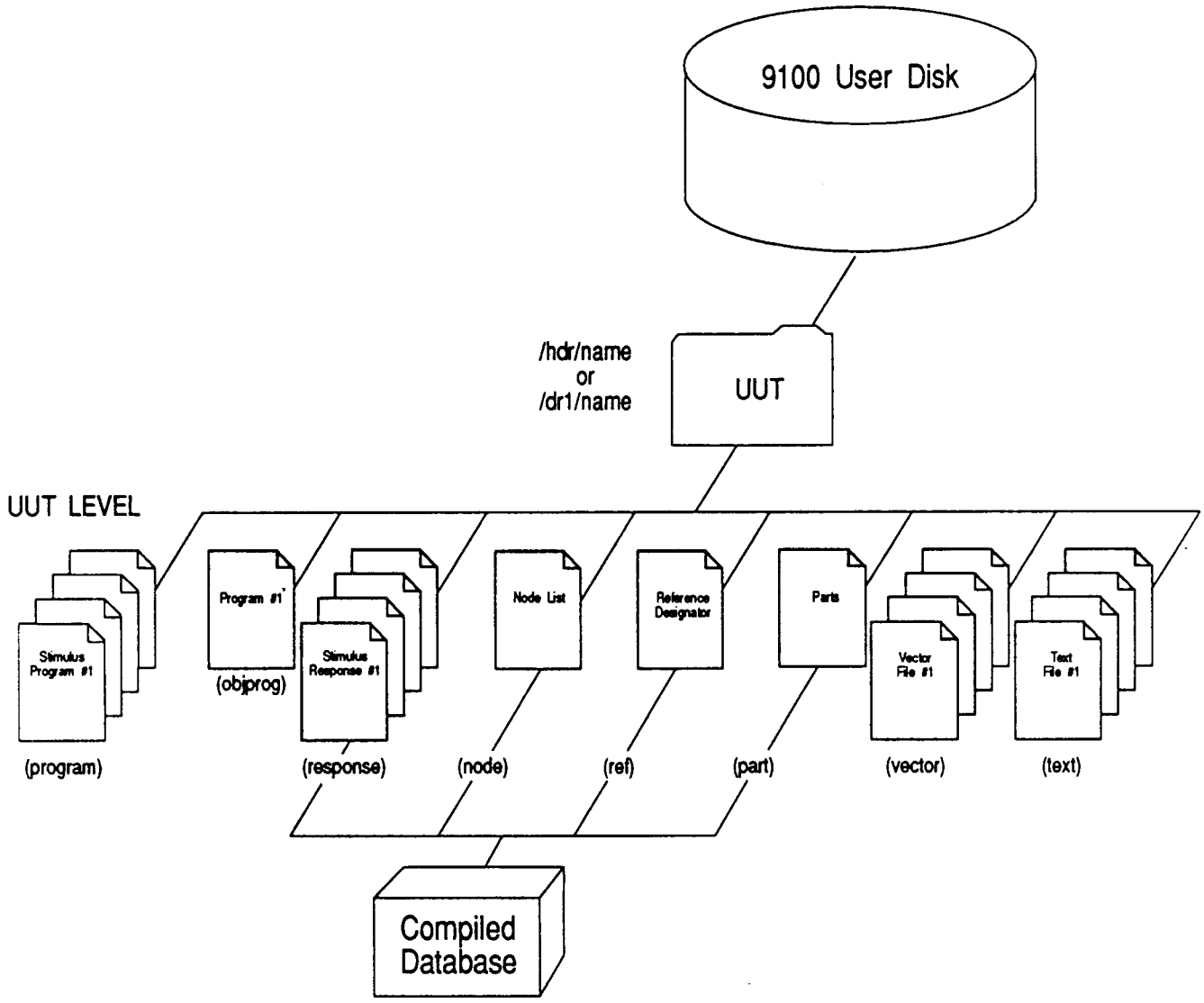
The Pod Library contains files known as Pod Databases that contain information necessary for the Mainframe to know the Pod in use. There are Pod Databases for all interface pods manufactured by Fluke.

- Help Library

The Help Library contains files specific to 9100 fault messages.

- UUT Directories

Each userdisk may contain one or more UUT directory.


9100 User Disk
/hdr/name
or
/dr1/name
UUT
UUT LEVEL
Stimulus Program #1
(program)
Program #1
(objprog)
Stimulus Response #1
(response)
Node List
(node)
Reference Designator
(ref)
Parts
(part)
Vector File #1
(vector)
Text File #1
(text)
Compiled Database

Directories

Each UUT directory includes the following files:

- Programs

Programs are either functional tests or stimulus routines.

- Stimulus Responses

Stimulus responses contain the correct data measurements that result from the application of a stimulus routine.

- Node List

The node list describes all the interconnections of the UUT.

- Reference Designators

The reference designator list contains names that represent devices on the UUT. With this list you assign a unique name and part description to every device on the UUT.

- Compiled Data Base

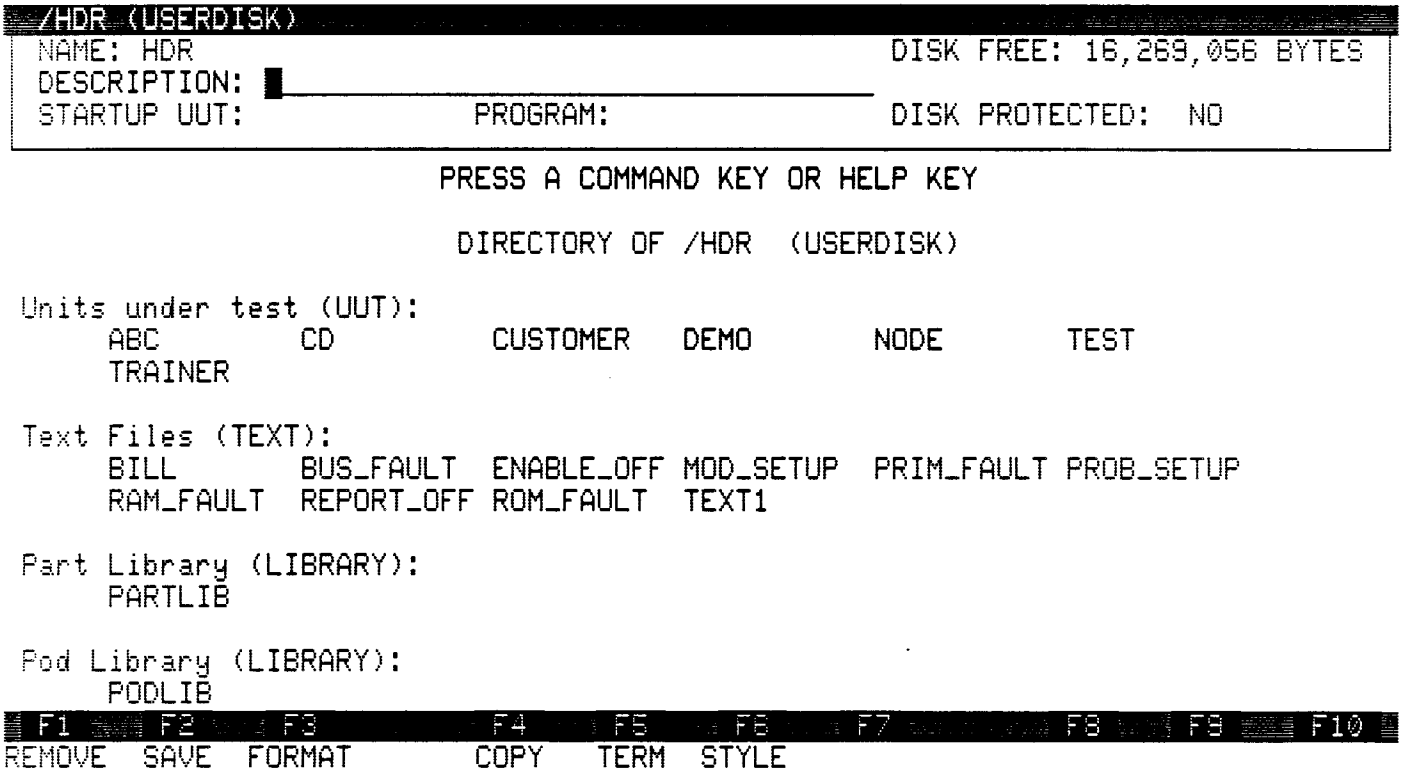
A compiled database contains responses, reference designators, parts escriptions, and node list converted to a form that the GFI program can use for isolating faults. You cannot edit a compiled database. Stimulus programs are not compiled into the database.

- UUT Text

Text files are documents that normally describe the UUT or the tests.

- Vector Data File

Vector files contain data used with the Vector I/O Module.


/HDR (USERDISK)
NAME: HDR
DISK FREE: 16,269,056 BYTES
DESCRIPTION:
STARTUP UUT:
PROGRAM:
DISK PROTECTED: NO
PRESS A COMMAND KEY OR HELP KEY
DIRECTORY OF /HDR (USERDISK)
Units under test (UUT):
ABC CD CUSTOMER DEMO NODE TEST
TRAINER
Text Files (TEXT):
BILL BUS_FAULT ENABLE_OFF MOD_SETUP PRIM_FAULT PROB_SETUP
RAM_FAULT REPORT_OFF ROM_FAULT TEXT1
Part Library (LIBRARY):
PARTLIB
Pod Library (LIBRARY):
PODLIB
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10
REMOVE SAVE FORMAT COPY TERM STYLE

Environment

The programmer's monitor and keyboard provide the communications interface to the Mainframe editor. When you use the editor, you cannot troubleshoot with the Mainframe since the operator's keypad and display are not active for the duration of your editing session.

To help explain the programming environment, press [EDIT] on the Mainframe application keypad.

The information window and edit window for the HDR userdisk appears in the display. From this point on, you enter commands from the programmer's keyboard.

Once the editor is invoked, the cursor appears in he information window. The following definitions explain all the fields that reside in the information window and how to move from the information window to the first command line.

- DESCRIPTION

Contains pertinent data about the disk being edited, such as the owner's name, or the date the disk was created and for what application. This window does not require any information and can be skipped over by pressing [Return] or [Edit] on the programmer's keyboard.

- STARTUP UUT

Allows the Mainframe to execute a test immediately after booting up on disk. (This window is good for dedicated test stations with a low skilled operator.) This window does not require any information and can be skipped over by pressing [Return] or [Edit] on the programmer's keyboard.

- STARTUP PROGRAM

Specifies the first program that is executed once the Mainframe is booted up. This window does not require any information and can be skipped over by pressing [Return] or [Edit] on the programmer's keyboard.

At the bottom of the programming monitor screen are a series of softkey numbers with functions listed below them. These functions are used for top level disk manipulating processes (except for F5 (TERM) which is the terminal emulator). Definitions of all the function keys are listed below.

- F1 (REMOVE)

Removes any type of file from the disk being edited.

- F2 (SAVE)

Saves all the information that you have changed or entered into the information window.

- F3 (FORMAT)

Formats a disk that has been inserted into the microfloppy disk drive.

- F4 (COPY)

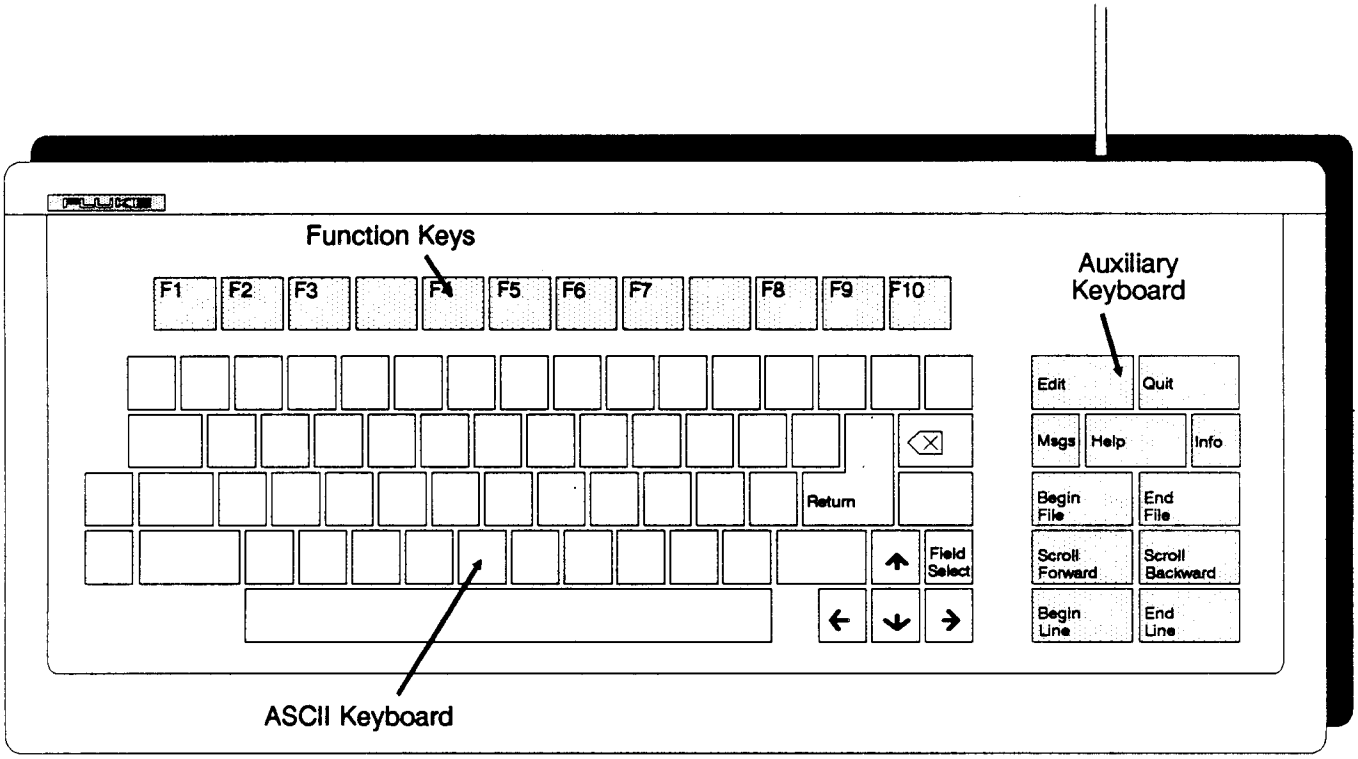
Copies an existing file to a different disk using the same name or to the same disk using a different name.

- F5 (TERM)

Enters the terminal emulator.

- F6 (STYLE)

Allows selection of brief or long directory lists.


Function Keys
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
Auxiliary Keyboard
Edit
Quit
Msgs
Help
Info
Begin File
End File
Scroll Forward
Scroll Backward
Begin Line
End Line
Return
Field Select
ASCII Keyboard

Programmer's Keyboard

The programming station keyboard is laid out into three basic groups:

- ASCII Keyboard
- Function Keys
- Auxiliary Keyboard

To begin an editing session, press the [Edit] key on Auxilary Keyboard. The editor is redirected to either DR1 (userdisk) or any item displayed in the HDR user-disk window. A prompt appears to let you enter the name and type of what you want to edit.

Type (not case sensative)

```
/hdr/trainer
```

or

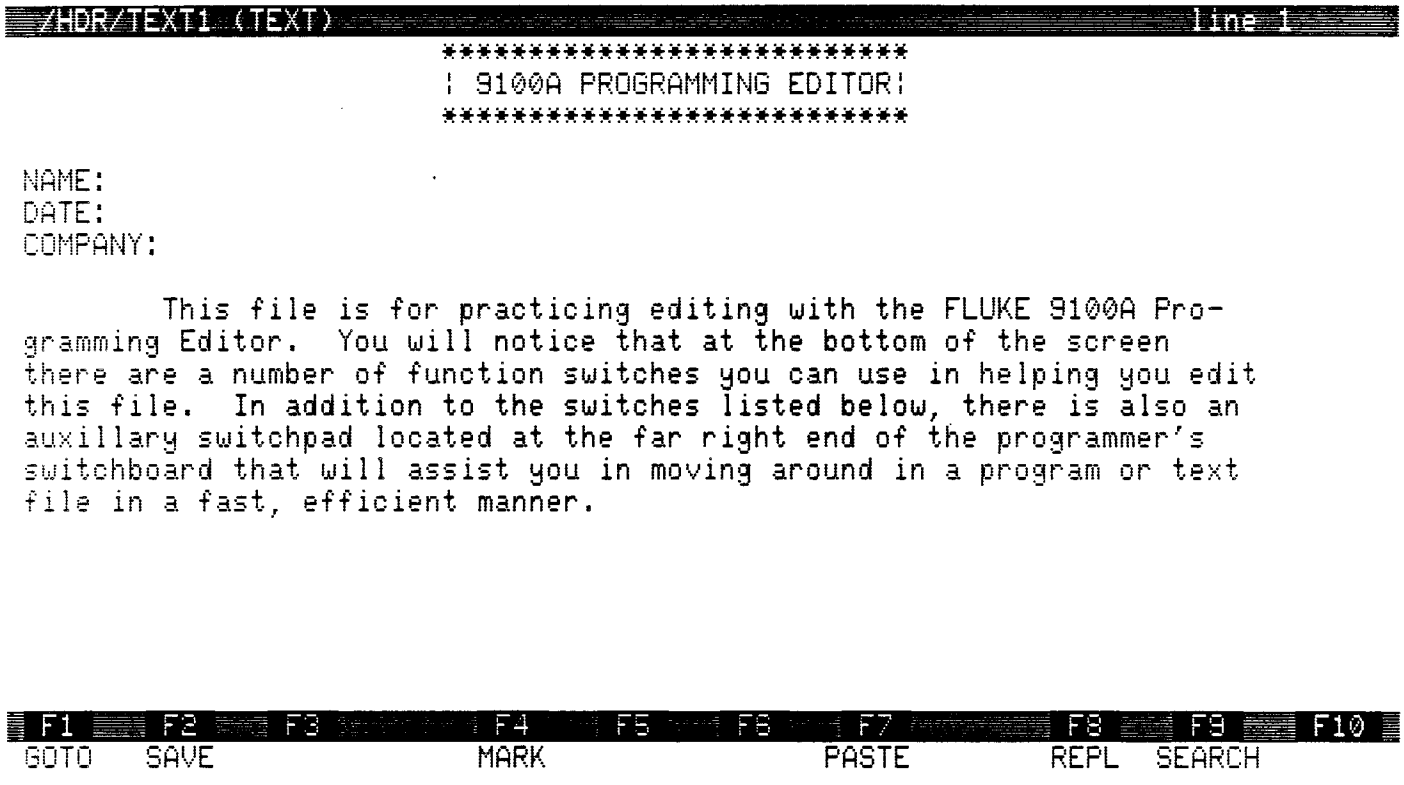
```
trainer
```

The monitor now requests the type of file. Locate the [Field Select] key on the keyboard. This key allows you to select various types of files and directories. Pressing [Field Select] scrolls the selections forward. To reverse the scroll (go backwards) press the [Shift] and [Field Select] keys simultaneously.

Select the type as UUT using the [Field Select] key and [Return]. You are now at the UUT window. Press [Return] again. You will be at the UUT Directory level.

Later in this section we will look at the various files in this window. However, this section concentrates on the editor by showing how to edit a text file. TL/1 syntax is discussed in a later section.

Press [Quit]. The USERDISK Directory is now displayed.


/HDR/TEXT1 (TEXT) line 1
****
! 9100A PROGRAMMING EDITOR!
****
NAME:
DATE:
COMPANY:
This file is for practicing editing with the FLUKE 9100A Programming Editor. You will notice that at the bottom of the screen there are a number of function switches you can use in helping you edit this file. In addition to the switches listed below, there is also an auxillary switchpad located at the far right end of the programmer's switchboard that will assist you in moving around in a program or text file in a fast, efficient manner.
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10
GOTO SAVE MARK PASTE REPL SEARCH

Text Entry

Use the text file called TEXT1 to practice the editor functions. Using this text file allows you to concentrate on editor functions rather than TL/1 syntax.

The following steps demonstrate how to use the editor to edit a text file.

1. On the programmer's keyboard, press the [EDIT] key.

   Edit file `text1` on `/hdr`.

   Select the proper type using the [FIELD SELECT] key.

2. Use the arrow keys to position the cursor to the proper place.

   Enter your name, date, and company into the appropriate spaces.

3. SAVE your changes using softkey F2 corresponding to the function key [F2].

4. F1 (GOTO Function)

   The GOTO function allows you to go to a specific line number. With GOTO, you may jump to an area of a long program quickly instead of scrolling to it.

   GOTO Line 14.

5. F4 (MARK function)

   Pressing F4 causes the Mainframe to use the cursor position as a reference point.

   Position the curser at the first character of the line and press [↓] twice to mark the next two lines. Each time you press the arrow and pass a line, the line is marked in reverse video. To mark words or characters, press the right or left arrows.

6. F5 (CUT Function)

   Pressing F5 cuts the lines that you previously marked with the MARK function. The cut text is stored in a buffer and is used in the next step.

7. F7 (PASTE Function)

   Pressing F7 pastes on the screen everything that was stored in the cut buffer. The PASTE function is useful in programs where a line of test or various commands are used more than once.

8. GOTO the line that your name appears on.

9. MARK your name, date, and company.

   Press F6 (YANK).

   What is the difference between YANK and CUT?

10. F9 (SEARCH Function)

    Pressing F9 allows you to search for a string of text. When you press F9, the editor asks you what you want to search for. Enter "switch". Notice that the cursor advances to the first instance of the word "switch".

11. F8 (REPLACE Function)

    Pressing F8 replaces a specified string of characters with another string of characters. Press F8; the editor asks for the characters to be replaced. Enter the word "switch" [Return]. The editor now asks for the characters that you want in the place of the word switch. Enter the word "key" and a [Return]. Notice that the word "switch" was replaced with the word "key".

12. [Shift] F8 (Multiple Replace)

    To continue replacing the word "switch" with "key", you can simultaneously press Shift and F8. Perform this operation throughout the entire file.

13. F2 (SAVE)

    To save the file, press F2 (SAVE).

14. Press the [Info] key on the auxiliary keypad. The following information is provided at the top of the screen for the current edit file and the userdisk:

    - Amount of userdisk space remaining
    - Size of the program file
    - Write protection

    To leave the Info window, press [Info] again.

15. Exit

    To exit the editor, press [Quit] on the auxiliary keypad. Each time [Quit] is pressed you return one level up in the editor. You are now at the /HDR userdisk level. If you press [Quit] again, you would leave the editor and return to the Mainframe's application keypad.

    To leave the editor from any level and return to the application keypad, press [Shift] and [Quit] simultaneously.

16. Exit the editor to the application keypad.

```
/HDR/CUSTOMER/MICRO_DATA (PROGRAM)                              line 1
program micro_data
    devname = "/mod1"
    option = getspace space "memory",width "word"
    setspace( option )
    sync device devname, mode "pod"
    sync device "pod", mode "data"
    threshold device devname, level "ttl"
    counter device devname, mode "transition"
    arm device devname
      a = 0
        rampdata addr 0, data 0, mask $1F
      a = read addr 0
        rampdata addr 0, data 0, mask $1F0
        rampdata addr 0, data 0, mask $1F00
        rampdata addr 0, data 0, mask $F000
    readout device devname
end program

 F1     F2     F3          F4     F5     F6     F7          F8     F9     F10
 GOTO   SAVE   DEBUG       MARK               PASTE       REPL   SEARCH CHECK
```

The Debugger

The debugger is an interactive tool that locates logical problems in TL/1 programs. With the debugger, you can execute and control programs or any function within a program. You can also view and alter the values of variables at any stage of program execution. By using the debugger to view the values of variables during program execution, you can determine if the program performs as intended.

To demonstrate the debugger capabilities, perform the following exercise.

Using the Debugger

1. Edit `/hdr/customer/main` Type: `Program`.

   You should now be viewing the program `main`. At this point, it is not necessary to concentrate on the TL/1 statements.

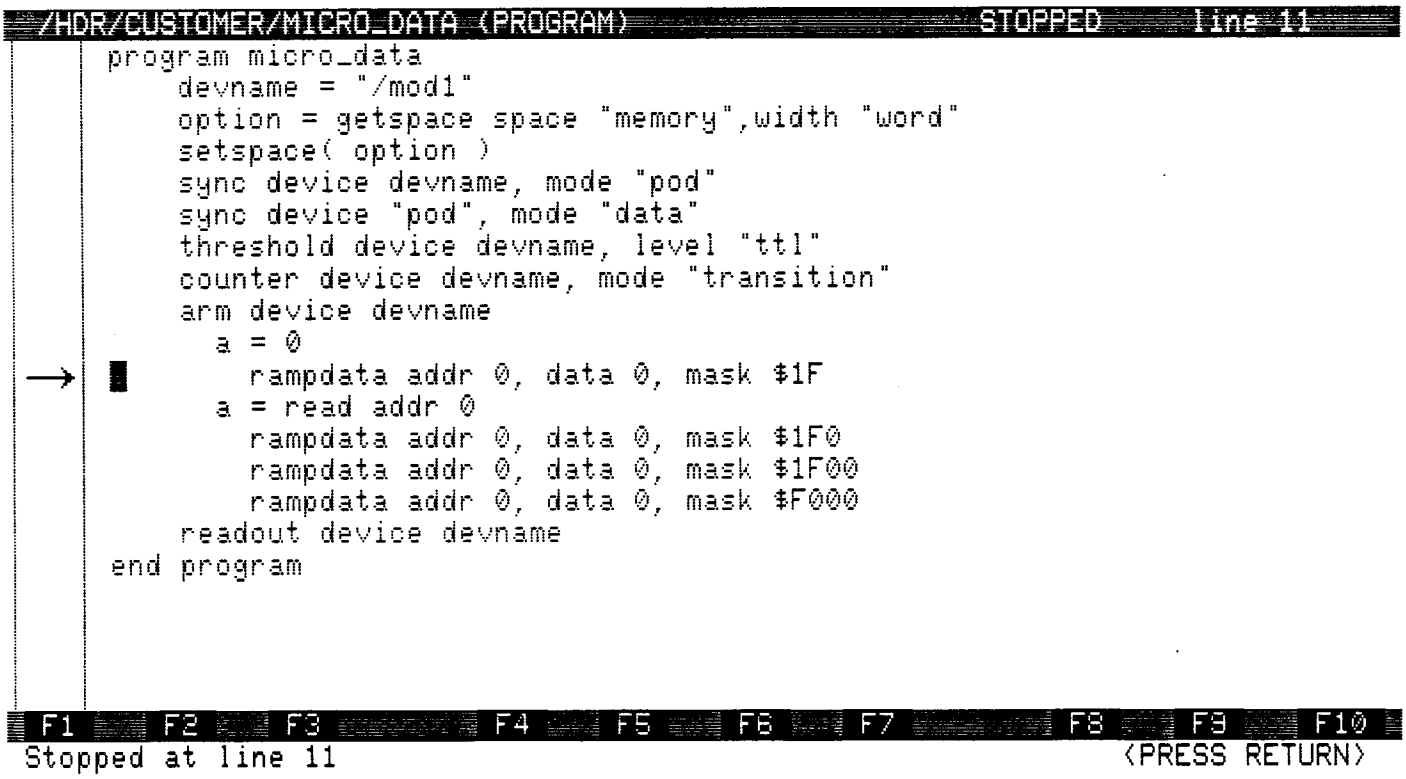
2. **F3** (DEBUG)
   Press **F3** to load the Debugger. The bottom of the screen will display `LOADING DEBUGGER...`. When loaded, a status column will be seen on the left side of the screen and new function key labels appear.

3. **F9** (SEARCH)

   Press **F9**. The `SEARCH` function scrolls forward to the characters you specify. This function is case sensitive. Search for the word `print`. The cursor should now be on line 14

4. **F6** (BREAK)

   Press **F6** (`BREAK`) to set a breakpoint at line 14. A breakpoint can be set for any line that performs an action. If a breakpoint has already been set, pressing **F6** removes the breakpoint. When a breakpoint is encountered, program execution will halt prior to the execution of the command(s) on that line.


/HDR/CUSTOMER/MICRO_DATA (PROGRAM) STOPPED line 11
program micro_data
devname = "/mod1"
option = getspace space "memory",width "word"
setspace(option)
sync device devname, mode "pod"
sync device "pod", mode "data"
threshold device devname, level "ttl"
counter device devname, mode "transition"
arm device devname
a = 0
rampdata addr 0, data 0, mask $1F
a = read addr 0
rampdata addr 0, data 0, mask $1F0
rampdata addr 0, data 0, mask $1F00
rampdata addr 0, data 0, mask $F000
readout device devname
end program
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10
Stopped at line 11
<PRESS RETURN>

5. (VIEW)
   The view softkey prompts for a program name and then displays the program.
   Press the VIEW function key located between the F7 and F8 function keys.
   At the VIEW PROGRAM prompt, enter: read_write and press Return . You should be looking at the program read_write.
   Search for print and set a breakpoint.
   Press the VIEW function key and enter main at the prompt.
   Press Return. You should be back to the MAIN program.

6. F4 (EXECUTE)

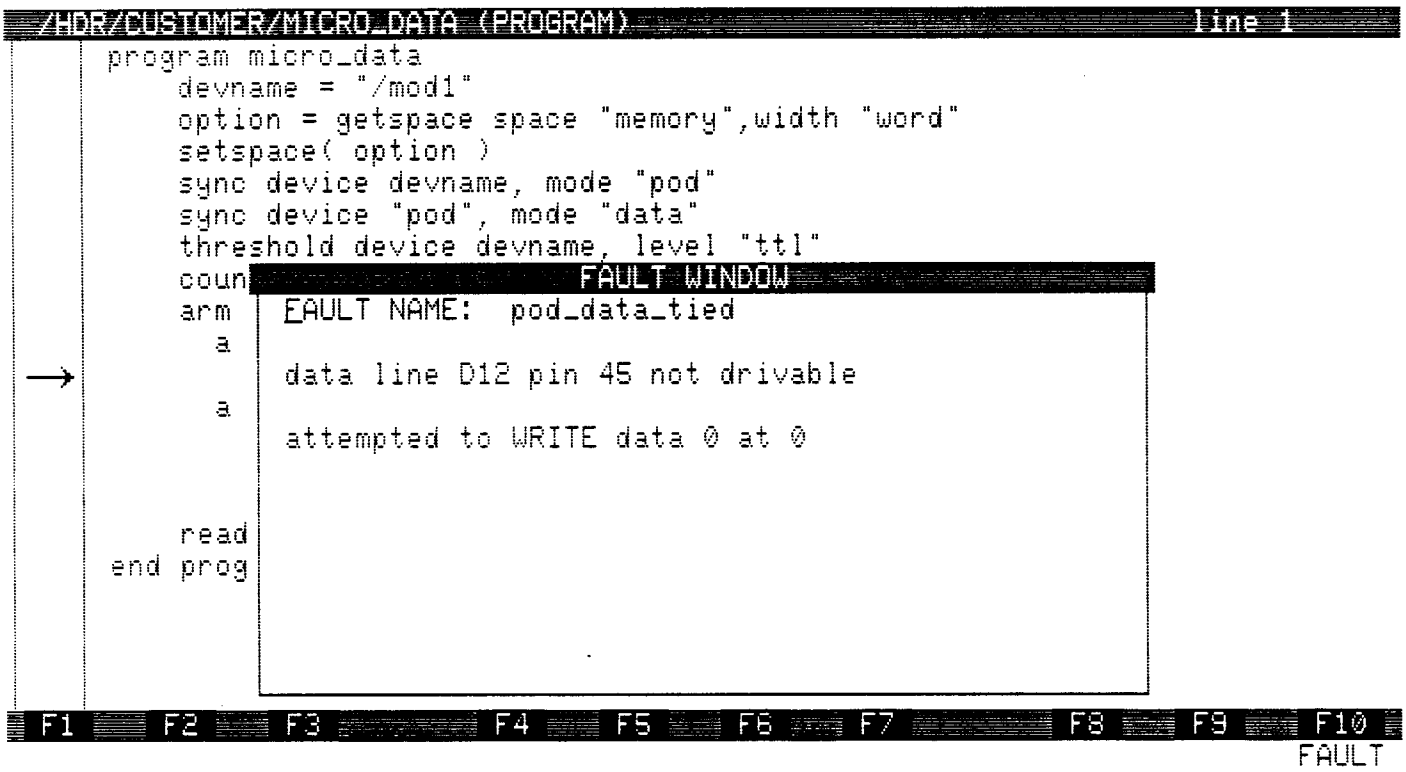
   Press F4 (EXECUTE) and then Return (for the default filename MAIN) to begin program execution. At line 14, a breakpoint is encountered and program execution is halted.
   Press Return to view the function key labels.

7. F7 (SHOW)

   Press F7 (SHOW). At the SHOW VARIABLE prompt, enter the variable "a" then press Return. You should see: a=43605 (decimal). To see a value in hex, press the Shift and F7 simultaneously. The variable "a" should still be there so press Return. You should now see: a=AA55 (hex).

8. F8 (SET)

   Pressing F8 (SET) allows you to assign a value to a variable.
   Set the variable "a" to the value $55AA (the dollar sign ($) indicates the number is a hex value.
   Use the down arrow key to move to line 17 and set a breakpoint.
   Do the same at line 18.
   NOTE: Breakpoints may be set or cleared while program execution is halted.


/HDR/CUSTOMER/MICRO_DATA (PROGRAM) line 1
program micro_data
devname = "/mod1"
option = getspace space "memory",width "word"
setspace(option)
sync device devname, mode "pod"
sync device "pod", mode "data"
threshold device devname, level "ttl"
coun
arm
a
a
read
end prog
FAULT WINDOW
FAULT NAME: pod_data_tied
data line D12 pin 45 not drivable
attempted to WRITE data 0 at 0
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10
FAULT

9. F3 (CONT)
Pressing F3 allows the program to continue execution until either:

- program execution is complete,
- a breakpoint is reached,
- an error occurs,
- a UUT fault is detected.

Press F3 (CONT). The Applications Display will read: MAIN DATA =$55AA. The Programmers screen will momentarily display the word LOADING.... until execution halts at line 9 of the program read_write.
Press F3 again. This time the Applications display prints read_write data=$A5A5 and program execution halts at line 17 of the program main.
Press Return.

10. F1 (STEP)
Pressing F1 (STEP) causes the current TL/1 command to be executed, steps to the next command, then halts prior to it's execution. If multiple commands are on the same line, the arrow will not move to the next TL/1 command line until all commands on that line are executed.
Press F1 (STEP) . The program will halt on the first command line (line 3) of the program bus_test.
Press F3 (CONT) . Execution continues and halts at line 18 of program main.

11. F2 (NEXT)
Pressing F2 (NEXT) causes the current TL/1 command line to be executed, steps to the next command line, then halts prior to it's execution. If multiple commands are on the same line, all commands will be executed and the arrow moves to the next TL/1 command line. Press F2 (NEXT). Notice that unlike STEP, the NEXT function did not allow the program to halt inside the called program.
Press F3 (CONT) to end the program.

12. F5 (INIT)
    Pressing the F5 (INIT) function key causes all breakpoints to be cleared and all variables to reset.
    Press F5 (INIT) . Observe that all breakpoints are gone and you can no longer SHOW the variable "a". VIEW the program read_write to verify the breakpoint set there has also been cleared. VIEW main.

13. Set fault SW1-1 to the fault position then execute program main.
    The program encounters a fault that is displayed in a fault window.

14. F10 (FAULT)
    Press F10 (FAULT) to toggle the fault window on and off.

15. Reset fault switch SW1-1 to the no-fault position.

16. Continue program execution by pressing F10 (FAULT) .
    NOTE: The completion status of the program is FAIL.

17. Leave the debugger and return to the Applications keypad.

Section 10
Functional Test

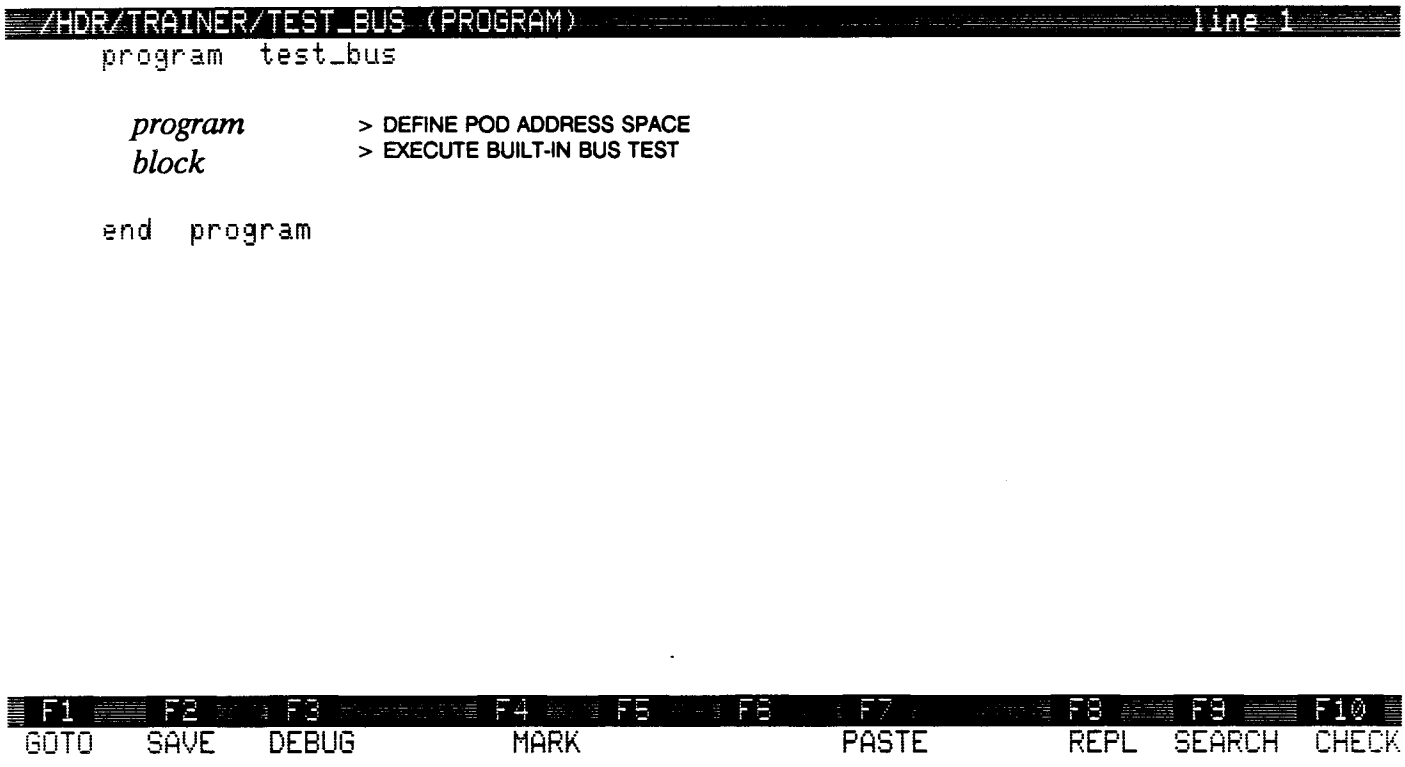
This section is a series of exercises designed to introduce you to the programming environment of the 9100A. The exercises will familiarize you with the operation of the program debugger, built-in functions, measurement devices, and the TL/1 language.

Bus Test

The built-in Bus Test is an easy, fast method of testing the UUT microprocessor bus and provides excellent diagnostic information to the user. Bus Test detects faults that can cause almost all other functional tests to fail.

Bus Test checks each address, data, and control line to be sure each can be individually driven high and low. It also verifies that no address or data lines are tied together or stuck at a fixed level.

In most instances, Bus Test will be the first major functional test performed on the UUT.


/HDR/TRAINER/TEST_BUS (PROGRAM) line 1
program test_bus
program
block
> DEFINE POD ADDRESS SPACE
> EXECUTE BUILT-IN BUS TEST
end program
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10
GOTO SAVE DEBUG MARK PASTE REPL SEARCH CHECK

Bus Test Structure

The basic structure of the program used to test the bus functional area of the UUT is illustrated on the previous page.

Exercise 10-1
Design the "test_bus" Program

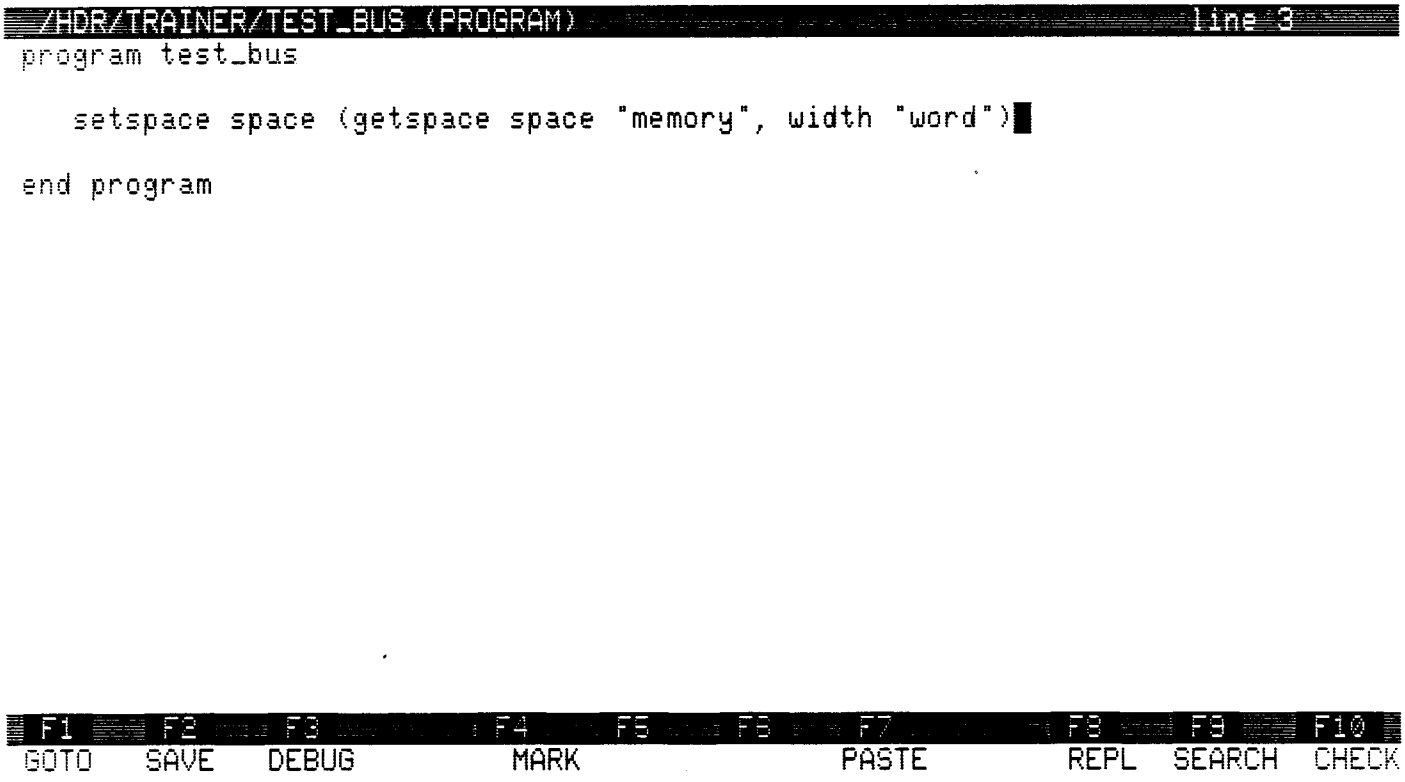
1. Edit the program "`test_bus`" for the UUT trainer.

   Prior to the entering the bus test command, you should define the proper pod addressing space. The address space option is pod dependent. This information can be found in the supplemental pod manual. For the purpose of these exercises, we will consider only the following 80286 pod address space options:

   ```
   memory byte
   memory word
   I/O byte
   I/O word
   ```

   To set the space, the program must first access the pod data base to retrieve the appropriate hex number sent to the pod. The TL/1 command used to retrieve this number is `getspace`.

   What would be the proper form of the `getspace` command if you wanted the pod to perform memory word accesses?


/HDR/TRAINER/TEST_BUS (PROGRAM) line 3
program test_bus
setspace space (getspace space "memory", width "word")
end program
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10
GOTO SAVE DEBUG MARK PASTE REPL SEARCH CHECK

Once the proper value is retrieved from the pod library, the 9100A then must send this number to the pod. The command used for this is `setspace`

2. Using the `getspace` and `setspace` commands, enter the command line used to set the pod to perform memory word accesses.

   Your program should look something like the figure illustrated on the previous page.

3. Using positional notation modify the `setspace` command line.

   The next step would be to perform the actual Bus Test.

4. Enter the command that will perform the built-in Bus Test on the trainer UUT.

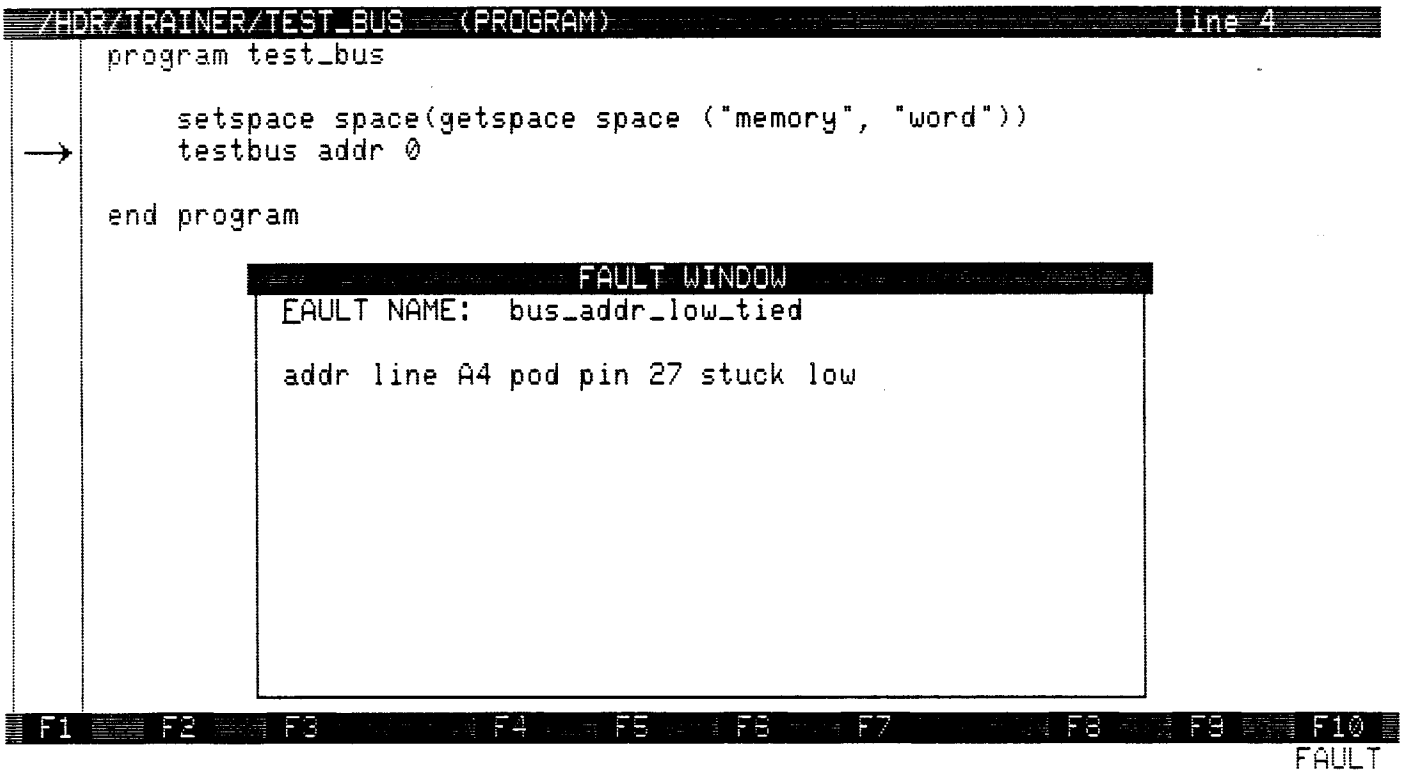
   The command asks for an address which the test will use to perform the reads and writes. The address specified in the bus test should be in a memory location that is readable and writeable. Other locations should only be specified if they physically exist and are not written to by other devices.

   Use address 0 in your `testbus` command line. Address 0 is the first address of RAM space on the UUT.

5. Use the F10 (CHECK) function to check your program for errors

6. Enter the debugger mode of operation and execute your program.

   Execution should have completed with `status = PASS`.


/HDR/TRAINER/TEST_BUS (PROGRAM) line 4
program test_bus
setspace space(getspace space ("memory", "word"))
testbus addr 0
end program
FAULT WINDOW
FAULT NAME: bus_addr_low_tied
addr line A4 pod pin 27 stuck low
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10
FAULT

Bus Test Faults

Exercise 10-2
Stuck Address

1. Select fault switch SW2-5 (Set to ON).
2. Execute the "test_bus" program and record the name and description of the fault.

Exercise 10-3
Stuck Data

1. Select fault switch SW2-7 (Set to ON).
2. Execute the "test_bus" program and record the name and decription of the fault.

Exercise 10-4
Fault

Input lines to the microprocessor are defined as STATUS LINES. Every time the pod performs an operation on the UUT these input lines are monitored.

1. Press and hold the [RESET] button down on the Trainer UUT.
2. Execute the "test_bus" program and record the name and description of the fault.

RAM Test

Exercise 10-5
Design the "test_ram" program

1. Design a program "test_ram" under directory UUT/TRAINER, to test UUT RAM over the address range of 0–1FFFE using the testramfast command.
2. When performing the test, check all 16 data bits, use the default delay, and set seed to 1. (Don't forget the Address Option)

Exercise 10-6
RAM Data Fault

1. Select fault SW1-5 (Set to OFF).
2. Execute "test_ram" and record the name and description of the fault.

Exercise 10-7
RAM Address Fault

1. Select fault SW4-8 (Set to ON).
2. Execute "test_ram" and record the name and description of the fault.

Exercise 10-8
RAM Cannot Modify Fault

1. Select Fault SW1-4 (Set to OFF).
2. Execute "test_ram" and record the name and description of the fault.

ROM Test

Exercise 10-9
Design the "test_rom" program.

1. Design a program "test-rom" to test ROM by checking each ROM chip one at a time using the testromfull command.

Exercise 10-10
Data Stuck

1. Select SW3-1 (Set to ON).
2. Execute the "test-rom" program and record the name and description of the fault.

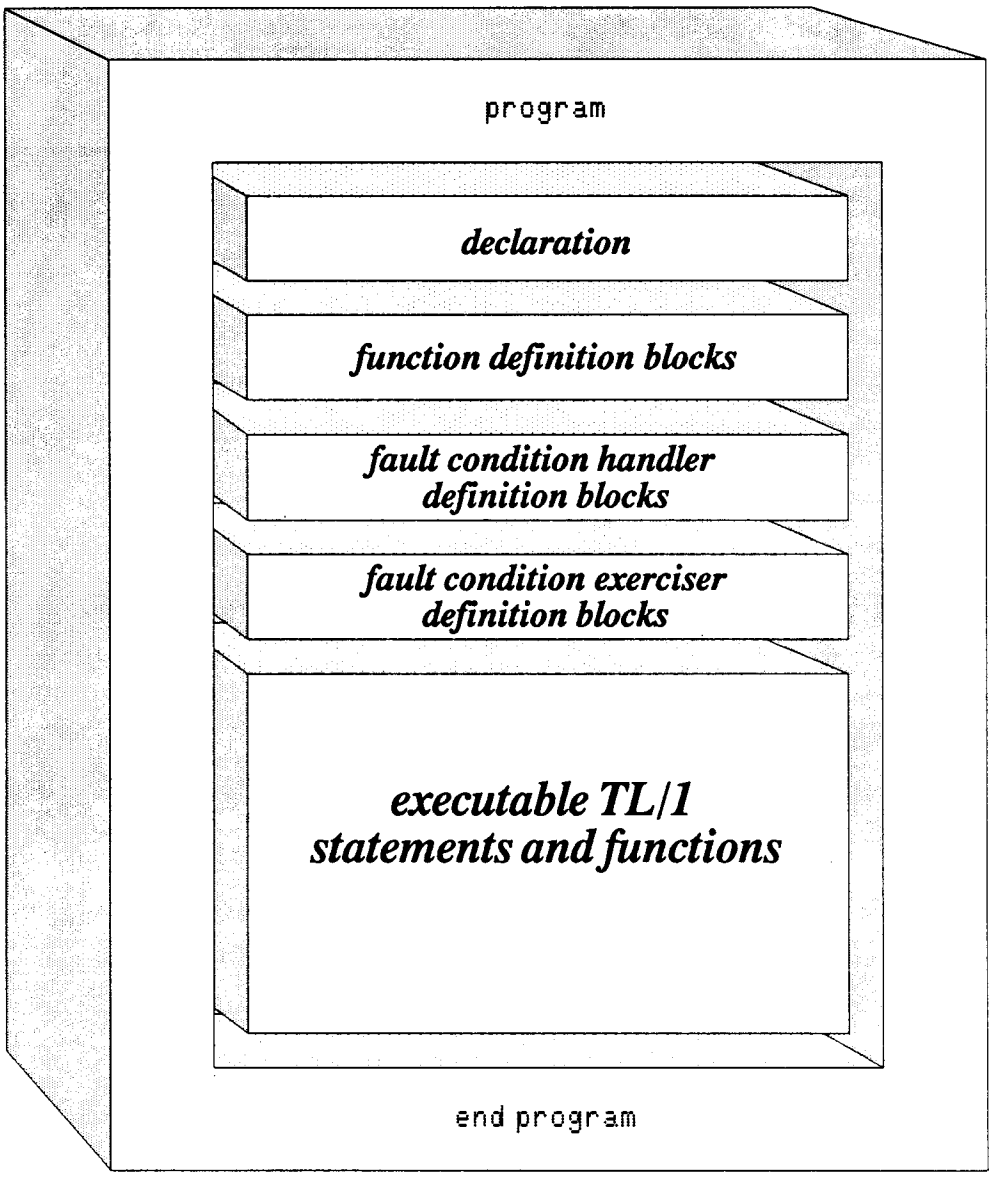
Exercise 10-11
All Data Stuck

This exercise illustrates the fault messages received when testing ROM that has a fault that prevents a block from being selected from the decoder.

1. Begin by selecting SW3-7 (Set to ON).
2. Execute the "test_rom" program and record the description of the fault.

Section 11
The TL/1 Test Language

TL/1 is a structured programming language specifically designed for developing test and troubleshooting routines. Command vocabulary is based on the test environment and minimizes language learning time. On most commands, default entries are available that simplify the process of writing test and troubleshooting programs.


program
declaration
function definition blocks
fault condition handler definition blocks
fault condition exerciser definition blocks
executable TL/1 statements and functions
end program

Structure of a TL/1 Program

TL/1 defines two types of statements: simple statements, and block statements. A simple statement performs a single action. Block statements delimit the beginning and end of blocks and control the execution of the statements they enclose.

There are four types of definition blocks that can be placed within a TL/1 program block:

```
program

   declare
      Defines the characteristics and initial value
      of variables.These definition blocks begin
      with a TL/1 declare command.
   end declare

   function
      Defines functions that can be called
      from any place within the program that
      defines it. These definition blocks
      begin with a TL/1 function command.
   end function

   handle
      Defines a TL/1 block to be called when a UUT
      fault condition is detected. These definition blocks
      begin with a TL/1 handle command.
   end handle

   exercise
      Defines a TL/1 block to be called when a
      UUT  fault condition is detected and the
      LOOP key on the operator's keypad is pressed.
      These  definition blocks begin with a TL/1
      exercise command.
   end exercise

   ! The main program would now begin here!

end program
```

Variable Declarations

The TL/1 language supports three kinds of data:

Integer Numbers

data type: `numeric`

Integer numbers are 32 bit positive integers which can have any value from 0 to 4,294,967,295 (base 10) or from 0 to FFFFFFFF (base 16). Numeric values may be written in either decimal or hexidecimal notation. Hexidecimal numbers must be prefixed with the "$" character.

```
declare numeric x = $2B
declare numeric y = 43
```

Floating Point Numbers

data type: `floating`

Floating point numbers use the IEEE standard for double-precision floating-point numbers, except that Infinity and NAN (Not A Number) are not supported.

```
declare floating z = +1.23E-03
```

Character Strings

data type: `string`

Strings in TL/1 are sequences which contain from 0 to 255 ASCII (8-bit) characters.

```
declare string message = "Hi"
```

ASCII characters which do not have a printable representation in strings can be placed in strings by using the backslash character:

`\"`	is the string quote character
`\nl`	is the newline character (defines the end of a line and does not necessarily include a line feed)
`\\`	is the backslash character
`\1B`	is the ESC character (the hexadecimal number corresponding to the ASCII ESC character is 1B)

```
program declare_1      !follow the directions below to see how declares work

   declare
      global numeric x = 32
   end declare

   function ww
      declare
         !global numeric x          !Run this program in debug with both variables
         !numeric x                 !commented out then run it with only one
      end declare                   !commented out and then run it again with
                                    !only the other one commented out and then with
                                    !neither commented out

   if x = 32 then
      print "global=32"
   else
      print "global=x"
   end if
   end function

   print x
   wait (2000)
   ww()
   end program
```

Local and Global Variables

Variables declared within an block (ie. function/end function, handle/end handle, etc.) are local to that block unless they are specifically declared as "`global`". Variables declared as "`global`" in the declaration block must also be declared as "`global`" within the block which uses them.

Persistent Variables (Ver 5.0)

Unlike global variables, "`persistent`" variables retain their values even between program executions. `Persistent` variables are useful for communicating UUT status between GFI stimulus programs.

Associated Commands:

`resetpersvars` resets the persistent variable set, to the empty set.

`clearpersvars` clears the values of currently active persistent variables.

Variable Arrays

Types:	`numeric`
	`string`
	`floating`

Dimensions:	Up to three dimension are supported and may have any number of subscripts, limited only by available memory. (Local and global only!)

Math Functions

`fabs`	returns the absolute value of a floating point number.
`log`	returns the logarithm of the a floating point number in a specified base.
`natural`	returns the floating point value of a selected natural constant.
`pow`	returns the value of one argument raised to the power of the other argument.
`sqrt`	returns the square root of a floating point number.
`sin`	returns the sine of a floating point number.
`asi`	returns the inverse sine of a floating point number.
`cos`	returns the cosine of a floating point number.
`acos`	returns the inverse cosine of a floating point number.
`tan`	returns the tangent of a floating point number.
`atan`	returns the inverse tangent of a floating point number.

Arithmetic Operations

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
-	Multiplies the operand by -1.0 (floating point only)

Relational Operators

=	Equal to
<>	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Logical Operators

& and	Logical AND
| or	Logical OR
^ xor	Logical exclusive OR
~ opl	One's complement
not	Logical negation

String Operators

+	appends a string expression to the end of another string expression
instr	returns the index number of the position at which a sub-string appears in a string, or zero if substring does not appear in the string
len	returns the number of characters in a string
mid	extracts a new string of a specified length from the given string beginning with the character at the specified index
token	extracts lexical tokens from strings

Type Conversion Operators

ascii	returns the ascii code of a character.
chr	returns a string of a single character corresponding to a numeric value.
str	returns the string representation of a numeric operand.
val	returns the numeric value of a string using the specified radix.
fstr	returns a string respresentation of a floating-point number.
fval	returns the floating-point value of a string.
cflt	returns the floating-point value of an integer.
cnum	returns the integer value (rounded) of a floating-point number.
isval	pre-test an expression for validity as arguments to the val command.
isflt	pre-test an expression for validity as arguments to the fval command.

Bit Shifting Operators

`shl` `<<`	returns new value of operand after shifting left by one (or more) bits.
`shr` `>>`	returns new value of operand after shifting right by one (or more) bits.

Bit Mask Operators

`bitmask`	returns a number in which all bits from bit 0 through the specified bit are set.
`setbit`	returns a number in which the bit specified is set.
`lsb`	returns the index of the least-significant set bit in the operand.
`msb`	returns the index of the most sigificant set bit in the operand.

System Functions

`systime`	returns the number of seconds since 00:00:00 on January 1, 1980.
`readdate`	converts the number returned by `systime` into a string usable as the current date.
`readtime`	format time information from `systime`.
`sysdata`	last data that was read or written.
`sysaddr`	last address that was read from or written to.

Loop Constructs

```
loop / end loop
loop until
loop while
loop for
for / next
```

Conditional Statements

```
if then
else
else if
end if
```

Transfer of Control

```
execute
wait
return
abort
goto / label
```

Devices and Files

File naming convention
Delete
Path name
List of device names

User Defined Functions

Passing Parameters
Returning a Value
Scope of Declared Variables

Communicating with Devices and Files

The TL/1 language includes commands which provide access to the 9100A's communication devices (eg. operator display/keypad, RS-232 ports, etc.) as well as text files stored on the floppy or hard disk. Below is a list of the 9100A's devices which can be directly accessed:

`/term1`	Mainframe's three-line Display / Keypad
`/term2`	Programmer's Monitor / Keyboard
`/port1`	Isolated RS-232 port
`/port2`	Non-isolated RS-232 port

Text file names must conform to the normal 9100A file naming convention (ie. up to 10 alpha-numeric characters) and can include the name of the userdisk and UUT where the file is stored, as shown below:

```
video_data

/hdr/turbo_xt/video_data
```

To get information from a device (or file), the `input` statement is used. The following program fragment demonstrates the use of the `input` command to get a numeric value from the default device (`/term1`) and place it in the variable called "`end_address`":

```
declare numeric end_address

input end_address
```

Information can also be sent to a device by using the `print` statement. The following is an example of a `print` command which will send the character string "`a_message`" to the default device (`/term1`):

```
declare string a_message

print a_message
```

Redirecting Input/Output

It is sometimes desirable to print (or input) to a device other than the default. To do this, a "channel" must be opened to the device and all print (or input) statements must then reference this "channel", as shown in the examples below:

```
declare numeric in_chan
declare numeric out_chan
declare string a_string
declare numeric file_chan

in_chan = open device "/term2", as "input"
out_chan = open device "/term2", as "output"
file_chan = open device
            "/hdr/cpu1/test_log", as "output"

input on in_chan, a_string
print on out_chan, a_string
print on file_chan, a_string

close (in_chan)
close (out_chan)
close (file_chan)
```

The open command returned a numeric value that represented the "channel" to be input from (in this example the channel pointed to /term2 ... the programmers keyboard). The numeric variable in_chan must be used in each input command that is to get it's data from /term2. Once the program is done using /term2, the channel should be closed.

I/O Modes

By default, the `input` command uses what is called the "`buffered`" mode. This means it will not input characters until the [RETURN] key (carriage return) has been pressed, at which time it will retrieve all characters up to the carriage return. It is sometimes desirable to have `input` return one character at a time. This can be accomplished by `opening` the device in "`unbuffered`" mode. There are other *mode* options used for the RS-232 ports and in specifying window parameters (refer to the TL/1 Reference Manual).

```
declare numeric in_chan
declare string x

    in_chan = open device "/term1", as "input",mode "unbuffered"

   input on in_chan, x
```

In the previous example, if no character had been pressed, `input` would have returned a null. It's a good idea, therefore, to see if a key has been pressed before attempting an `input` when using unbuffered mode. The `poll` function can be used to check an open channel for input characters. (The `poll` command can check for other events as well ... refer to the TL/1 Reference Manual).

```
declare numeric in_chan
declare numeric status
declare string a_character

in_chan = open device "/term2", as "input", mode "unbuffered"

! See if there is a character to be "input"

status = poll channel in_chan, event "input"

! Stay in loop until there is a character

loop until status = 1
status = poll channel in_chan, event "input"
end loop

   ! Get the character

input on in_chan, a_character

!!!! another way to do the same thing !!!!

declare numeric in_chan
declare numeric status
declare string a_character

open device "/term1", as "input", mode "unbuffered"

loop until ( poll (in_chan, "input") = 1)

   ! do nothing

end loop

input on in_chan, a_character
```

Only single-character strings can be `input` using "`unbuffered`" mode, whereas "`buffered`" mode allows for `input`ing multi-character strings as well as numeric values. Refer to the TL1 Reference Manual to see what the differences are between "`buffered`" and "`unbuffered`" modes when using the `print` statement.

Specifying the Address Option in TL/1

```
getspace / setspace
sysspace
podinfo
```

Functional Test Commands

`pretestram`	performs a very fast pretest of RAM to find any simple faults such as totally dead memory chip, stuck address lines, or stuck data lines.
`testbus`	bus test for micro-processor emulation pods

```
testramfast
testramfull
testromfull
diagnoseram
diagnoserom
getromsig
```

Stimulus Commands

```
read /readstatus /readvirtual
write /writecontrol /writefill /writevirtual
rampaddr
rampdata
toggleaddr
toggledata
rotate
writepatt
pulse
writepin
readblock
writeblock
writeword
readword
loadblock
```

Device Setup Commands

```
reset
sync
threshold
counter
enable
edge
stopcount
podsetup
setoffset / getoffset
restorecal
```

Measurement Commands

```
arm
readout
checkstatus
count
level
sig
pollbutton
readbutton
```

I/O Module to Write Data Patterns

```
storepatt / writepatt
clearpatt
clearoutputs
strobeclock
```

I/O Module to Recognize a Data Pattern

```
compare / iomod_dce
```

Run UUT Functions

`runuut`	Causes the UUT to begin executing instructions from its own memory.
`haltuut`	Terminates normal `runuut` operation, if it is active, and displays any fault conditions that occurred during `runuut` execution.
`waituut`	Suspends TL/1 program execution until one of the following events occurs: 1) The pod encounters a breakpoint 2) A DCE condition occurs 3) The number of milliseconds specified expires
`polluut`	returns a value indicating whether the pod is exectuting instructions in the `runuut` mode.
`runuutvirtual`	Causes the UUT to begin executing instructions from it's own memory, asyncronously to the system.

Prompting the Operator

```
connect
clip
assign
probe
assoc
```

Termination Status

```
fault
passes/fails
refault
handle / end handle
```

GFI Functions

`gfi hint`	Add a node to the end of the GFI hint list
`gfi suggest`	Get the next node in the GFI hint list
`gfi clear`	Erases GFI summary and GFI suggestion list
`gfi control`	Determine if program is being executed under GFI control
`gfi device`	Get name of GFI measurement device
`gfi test`	Executes all stimulus routines associated with a pin
`gfi status`	Gets the status of a node
`gfi accuse`	Returns string containing GFI acusation of failure
`gfi pass`	Forces GFI to pass a node
`gfi fail`	Forces GFI to fail a node
`gfi autostart`	Starts GFI automatically if GFI hints exist
`gfi ref`	Gets the name of the node being tested
`dbquery`	Retrieve informaton from GFI database

Pod Specific Support

```
readvirtual
writevirtual
```

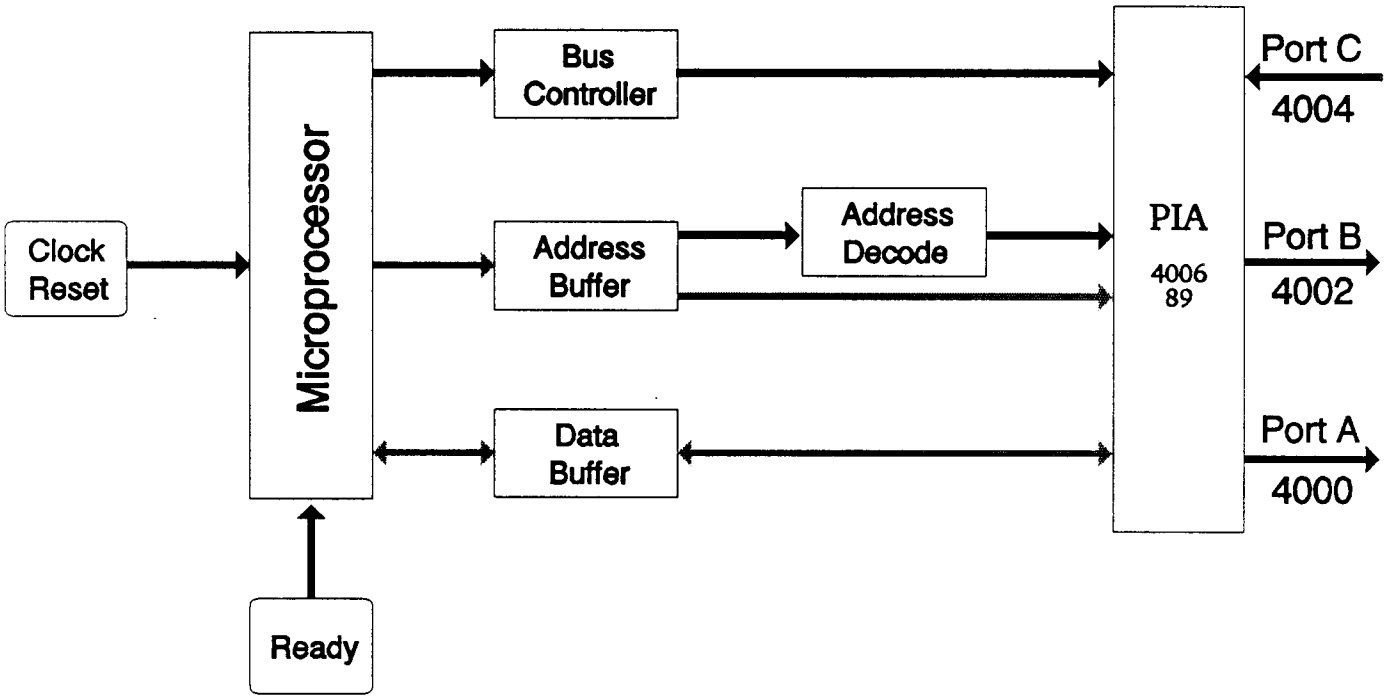
Vector Output I/O Module

`clockfreq`	external clock frequency (1,5,10, or 20MHz).
`drivepoll`	returns information about vector driving status when driving vectors
`edgeoutput`	sets the signal edges for triggering the START, STOP, and DR CLK lines
`enableoutput`	controls enabling (or qualifying) of the vector drive clock
`strobeoutclock`	provides a software controlled strobe for single-step vector driving when the syncoutput mode is set to "int"
`syncoutput`	determines the clocking source for driving out vector patterns
`vectordrive`	enables the vector drive function
`vectorload`	loads the named vector file into the declared I/O module(s)

Window / Menu Commands

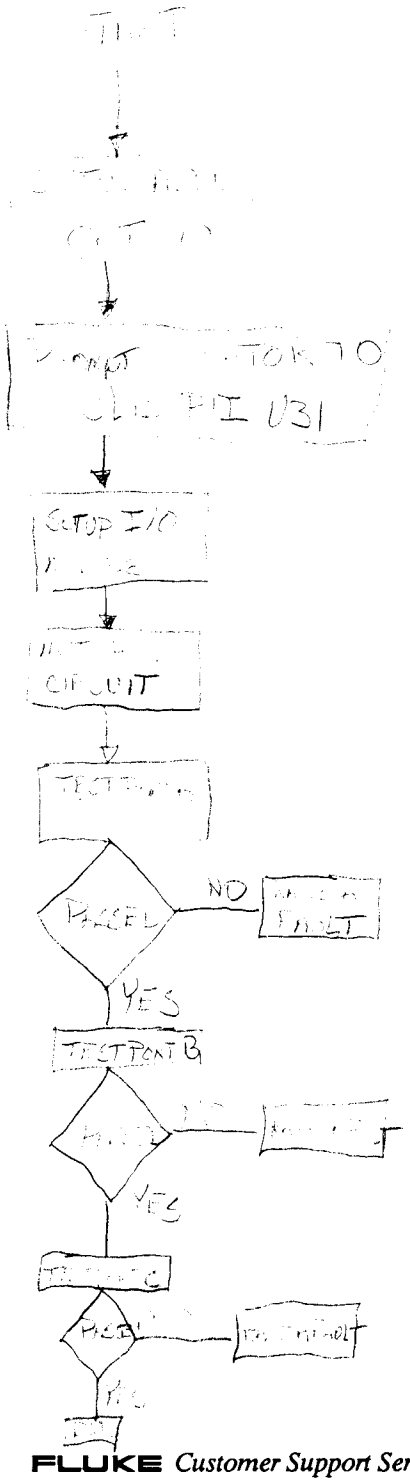
`define menu`	defines a menu or menu item.
`define mode`	defines the way a reference mode designator is displayed in a window.
`define part`	defines the shape and size for a `draw` command.
`define ref`	used to define a reference designator. It associates a reference designator with a location in a window and a part shape previously defined by a `define part` command. To display the reference designator, use a `draw` command.
`define text`	used to define text to be displayed in a window using the scaled location coordinates of a `draw` command. A piece of text is associated with attributes and a location in a window.
`draw`	draws all the previously defined UUT components and then all of the defined text on a window in the monitor.
`draw ref`	draw previously defined UUT components on a window in a monitor. Can also be used to change the display mode of a component or list of components.
`draw text`	displays the text at the scaled location indicated.
`readmenu`	used to display and read from a menu defined by the `define menu` command.
`remove`	removes definitions made by a `define` command.

Section 12
PIA Functional Test

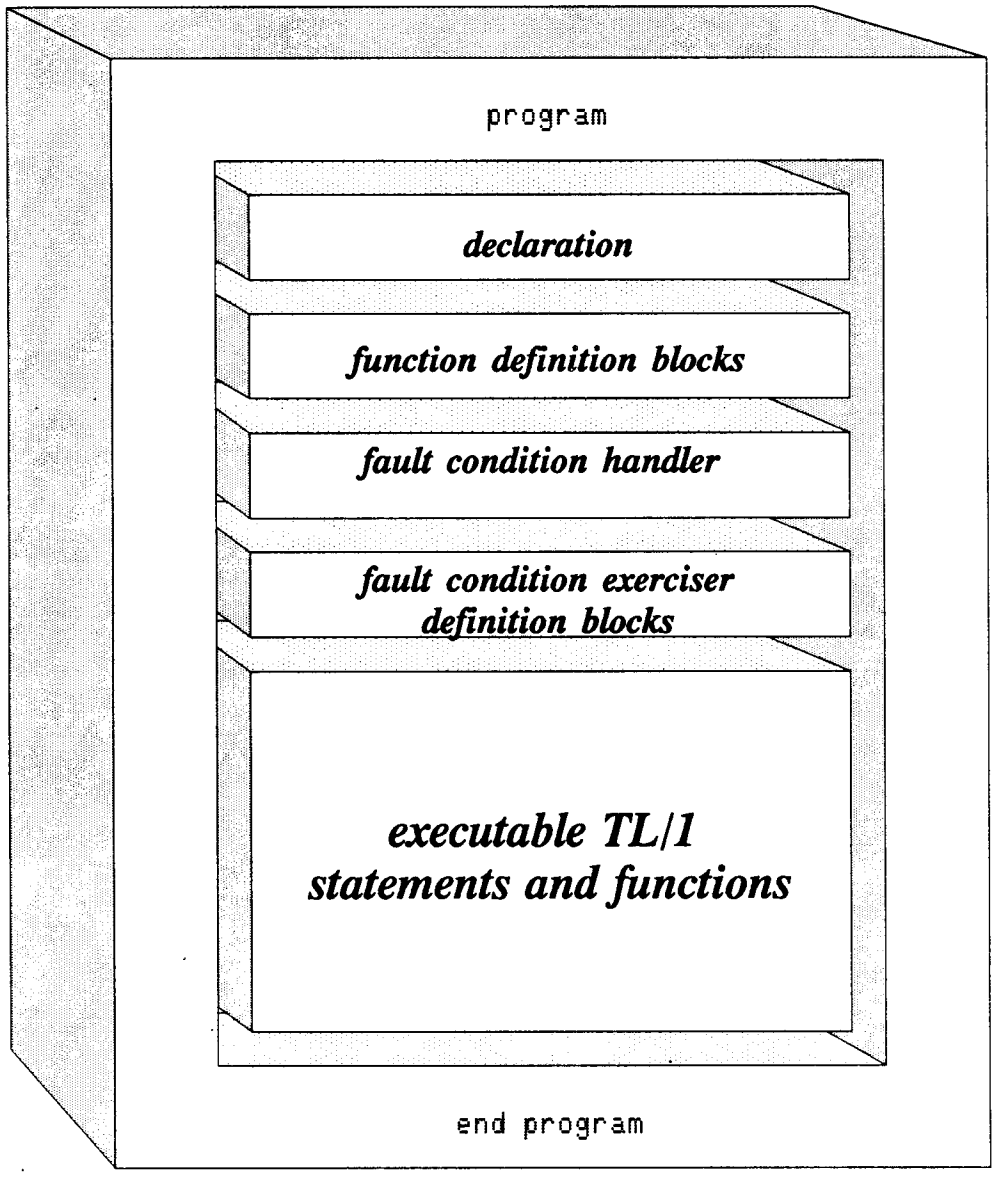

Clock
Reset
Microprocessor
Bus
Controller
Address
Buffer
Address
Decode
Data
Buffer
Ready
PIA
4006
89
Port C
4004
Port B
4002
Port A
4000

Exercise 12-1

1. As a class, develop a flow diagram that tests the PIA using the I/O Module with the 40 pin clip.
2. As a class write the program.



Section 13
Handlers


program
declaration
function definition blocks
fault condition handler
fault condition exerciser definition blocks
executable TL/1 statements and functions
end program

Exercise 13-1

1. As a class write a Handler to handle fault 3-1.
2. As a class write a Handler to handle fault 4-2.
3. As a class write a Handler to handle a faulty ROM chip.
4. As a class review the faults that were raised in the PIA Functional Test.

Appendix A
Glossary

ADDRESS BUS

A number of lines used by the microprocessor for locating data in RAM, ROM, or I/O devices. DMA controllers may also use the address bus for transferring large blocks of data to or from memory. Each line is referred to individually as an address line.

ALGORITHM

A prescribed set of rules or procedures for solving a complex problem.

ASCII

American Standard Code for Information Interchange as defined by ANSI document X3.64-1077. ASCII is a standardized code set for 128 characters including full alphabet (upper and lower case), numerics, punctuation, and a set of control codes.

ASYNCHRONOUS DATA/CIRCUITS

Data or other signals that function independently of microprocessor bus cycles.

ASYNCHRONOUS MEASUREMENT

A technique used by the 9100A to measure digital signals not synchronized to the microprocessor bus timing.

BACKTRACING

A procedure for locating the source of a fault on a UUT by taking digital measurements along a signal path from bad outputs to bad inputs. Backtracing stops at the point where a bad output has all good inputs.

BLOCK ADDRESS

A set of contiguous addresses defined by a beginning and ending address.

BLOCK READ

TL/1 command that reads a specified block of UUT addresses to a text file using a specified format (Intel or Motorola).

BLOCK WRITE

TL/1 command that writes a text file using a specified format (Intel or Motorola) to a specified block of UUT addresses.

BREAKPOINT

Stops program execution when a previously defined condition occurs. The TL/1 debugger uses a line oriented breakpoint that stops program execution just prior to executing the specified program line. The new interface pods have a breakpoint feature that stops RUN UUT execution when a specified UUT address appears on the address bus.

BUFFER

1: A device used to isolate one circuit from another (i.e. a bus buffer prevents a bus failure in one circuit from interfering with the same bus in another circuit). **2:** Used to boost the drive capability of the circuit being buffered. **3:** Provides temporary data storage for data transfers, usually between memory and I/O devices.

BUS

A series of bi-directional lines connected from the microprocessor to each device that interchanges data with it. Only one device can transmit at a time while any number of other devices can receive or be placed in a tri-state condition.

CAD (Computer-Aided Design)

Electronic CAD systems let the user create, manipulate, and store designs on a computer. Used mainly for laying out complex, high density, multi-layered printed circuit boards.

CAE (Computer-Aided Engineering)

Very similar to CAD systems except they are used more for design simulation, verification, and optimization.

CAS (Column Address Strobe)

A line used by dynamic RAM to latch the column address into the RAM device.

CAPTURE SYNCHRONIZATION

Synchronizes the response gathering hardware with vector output timing generated by the Vector Output I/O Module.

CONTROL LINE

An output line from the microprocessor that controls a particular function such as Read or Write. Also an input to a device which enables, disables, or controls the device.

CRC (Cyclic Redundancy Check)

A CRC (often referred to as signature) is a four digit hexadecimal number representing a serial stream of binary data. The binary data is shifted through a special 16 bit register which when complete, results in a 16 bit signature remaining in the register.

DATA BUS

A bus used to move parallel data between the CPU, memory, and I/O devices.

DEBUG

Locating and removing hardware and software errors.

DEBUGGER (9100A)

A TL/1 Editor tool used to test the functionality of TL/1 programs and debug execution errors.

DIRECTORY

A collection of related data sets (i.e. program files, text files) on a disk.

DMA (Direct Memory Access)

A technique for quickly transferring large amounts of data directly between I/O devices and memory or between memory and memory. The DMA controller supplies address and control signals required to accomplish the transfer directly rather than going through the CPU.

EDITOR

The programming environment that provides accessibility to files and automated test development.

EXTERNAL SYNCHRONIZATION

This technique uses the external Start, Stop, Clock, and Enable lines on either the I/O or Probe clock modules to provide measurement timing.

FAULT

A defect in a UUT that causes the circuitry to operate incorrectly.

FAULT COVERAGE

Normally expressed as a percentage of faults that a board test can detect of all possible faults.

FAULT DETECTION

The process of determining if a UUT has faults. See FUNCTIONAL TEST.

FAULT ISOLATION

The process of locating the component or defect causing the failure.

FUNCTIONAL TEST

A test that provides a pass/fail status on a specific section of the UUT circuitry.

GFI (Guided Fault Isolation)

An automated backtracing algorithm used to locate the source of a failure.

HANDSHAKING

Incorporates the use of special control lines or control codes to coordinate the transfer of data between two or more devices.

ICT (In-Circuit Test)

A test performed on one component at a time to verify component functionality; does not check for dynamic interaction between components.

IMMEDIATE MODE

The operating mode in which the operator interacts with the 9100A via the Applications keypad and display. Actions specified are performed as the proper keys are pressed.

INTERFACE POD

Translates generic mainframe commands (read,write,bus test, etc.) into microprocessor machine code to accomplish the desired action at the UUT.

INTERRUPT

An input to the microprocessor that is activated by an I/O device when it is ready to perform a function.

INTERNAL SYNCHRONIZATION

Used when the I/O Module is the source of the stimulus. The clock edge used by the measurement device is generated internally by the program.

I/O (Input/Output)

Generally refers to external input/output devices (keyboard, modem, or display) and the interface (PIAs, UARTs, DUARTs, etc.) required to communicate with the microprocessor.

I/O MODULE

A part of the 9100A system that can both stimulate and measure digital circuitry with up to 40 separate lines.

KERNEL

The heart of a microprocessor-based system which includes the microprocessor bus, RAM, ROM.

MASK

A value where each logic 1 represents a bit that is to be acted on.

NODE

A set of component pins on a UUT that are connected together.

PIA (Parallel Peripheral Interface)

A popular I/O interface device that has three, 8 bit I/O registers, or ports that can be configured as input, output, or bi-directional.

POD SYNCHRONIZATION

Synchronizes the response gathering hardware with an internal pod signal called podsync. Podsync generation can be made to depend on valid address, data, or other (pod dependent) timing cycles.

RAM (Random Access Memory)

Semi-conductor memory that can be read or written to much faster than tape or disk drive memories. There are two basic forms of RAM; static and dynamic. Dynamic RAM requires a periodic refresh cycle and special circuitry to perform, but uses considerably less power and space than static RAM and generally comes in higher density packages.

RAS (Row Address Strobe)

A line used by dynamic RAM to latch the row address into the RAM device.

REGISTER

A temporary storage device for holding data.

RESPONSE

The measurement of a node characterized by the stimulus applied.

ROM (Read Only Memory)

Used mostly for storing programs not intended to be altered. Often referred to as firmware. Some ROMs can only be programmed once (PROMs), but due to the need to make software changes, reprogrammable ROMs (EPROMs, EEPROMs) are also available.

ROM SIGNATURE

A four digit hexadecimal number (CRC) which represents the data stored in a specified section of ROM. The signature is obtained by shifting all the binary bits in the specified ROM space through a special 16 bit register.

RUN UUT

A 9100A command that causes the microprocessor in the pod to fetch instructions from the UUT memory.

SERIAL DATA

Binary data transferred one bit at a time on a single line or channel.

SIGNATURE

See CRC and ROM Signature.

SOFTKEY

A key whose function is determined by software and is changeable.

STATUS LINE

Input lines to the microprocessor used for reporting UUT conditions.

STIMULUS

Signals generated by the interface pod, I/O module, probe or externally such as an operator, and is used to exercise a node in a predictable and repeatable manner.

STIMULUS ROUTINE

A sequence of commands that begins a measurement, provides a stimulus, then ends the measurement.

SYNCHRONOUS DATA

Concurrent with microprocessor bus timing cycles.

SUB-ROUTINE

A set of software instructions that may be used over again in a program.

TL/1 (Test Language One)

The programming language used by the 9100A to perform tests on a UUT.

TRI-STATE

Being in a high impedence state.

UUT (Unit Under Test)

The circuit board or system being tested by the 9100A.

USERDISK

In the 9100A, Userdisk is the highest level in the disk structure. Userdisk can be either the hard disk (HDR or /hdr) or the floppy disk drive (DR1 or /dr1).

VARIABLE

Provides the means for storing and changing data values in a program. For example, if x is the variable name, it's value would be stored and then changed as follows:

```
x = 5              ! assign the value of 5 to x
print x            ! print the value of x
x = x + 3          ! add 3 to the value of x (x = 8)
print x            ! print the new value of x
```

VECTOR OUTPUT I/O MODULE

A 9100A option that allows test vectors to be applied to a UUT at speeds up to 25 MHz.

VIRTUAL ADDRESS

An address that extends beyond 32 bits. Used when addressing special addresses in some pods.

WAIT STATE

This causes a delay in the microprocessor bus cycle to give slow devices time to be accessed, or a RAM refresh circuit time to complete it's refresh cycle.

WINDOW

An area of the programmer's display that is reserved for certain information. The programmer can make the window appear or go away by pressing the proper key, depending on whether the information is needed.