9100-SERIES SOLUTIONS

9100-SERIES SYSTEMS TRAINING - PART 1

FLUKE®

P/N 870563

Fluke Corporation

P.O. Box 9090
Everett, WA 98206
(206) 347-6100 or
(800) 44-FLUKE

Rev. June 1994

Table of Contents

SECTION 1

Test and Troubleshooting Approaches

Section Page

SECTION 2

System Configuration

Section Page

SECTION 3

Understanding the UUT

Section Page

SECTION 4

Bus Test

Section Page

SECTION 5

RAM Testing

Section Page

SECTION 6

ROM Testing

Section Page

SECTION 7

Parallel Interface Adapter

Section Page

SECTION 8

Fault Isolation

Section Page

SECTION 9

Editor Operation

Section Page

SECTION 10

Functional Test

Section Page

SECTION 11

The TL/1 Test Language

Section Page

SECTION 12

PIA Functional Test

Section Page

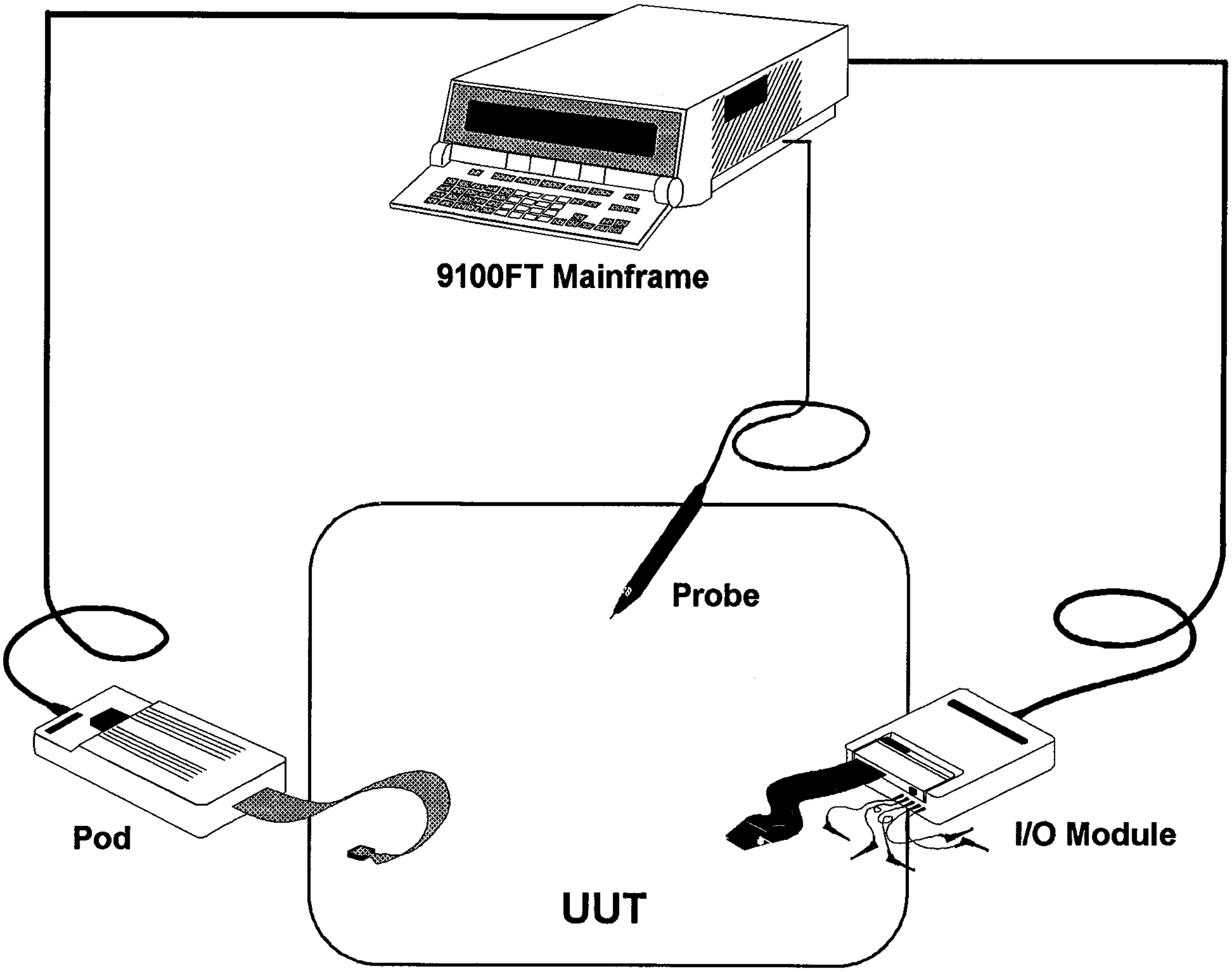
SECTION 13

Handlers

Section Page

Section 1

Test and Troubleshooting Approaches


9100FT Mainframe
Probe
Pod
I/O Module
UUT

9100FT SYSTEM

The 9100FT 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).

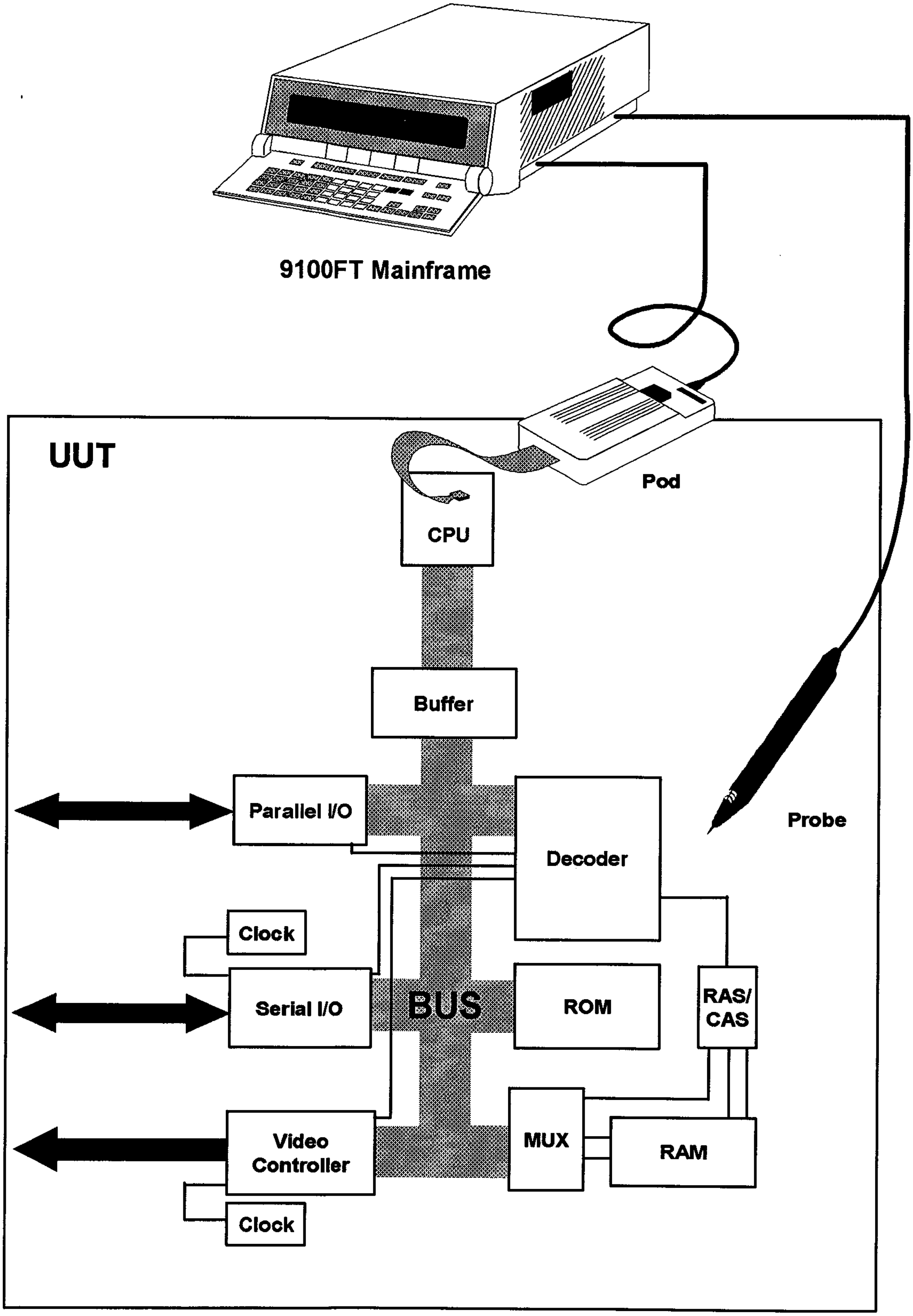
Testing and troubleshooting can be done in two ways:

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

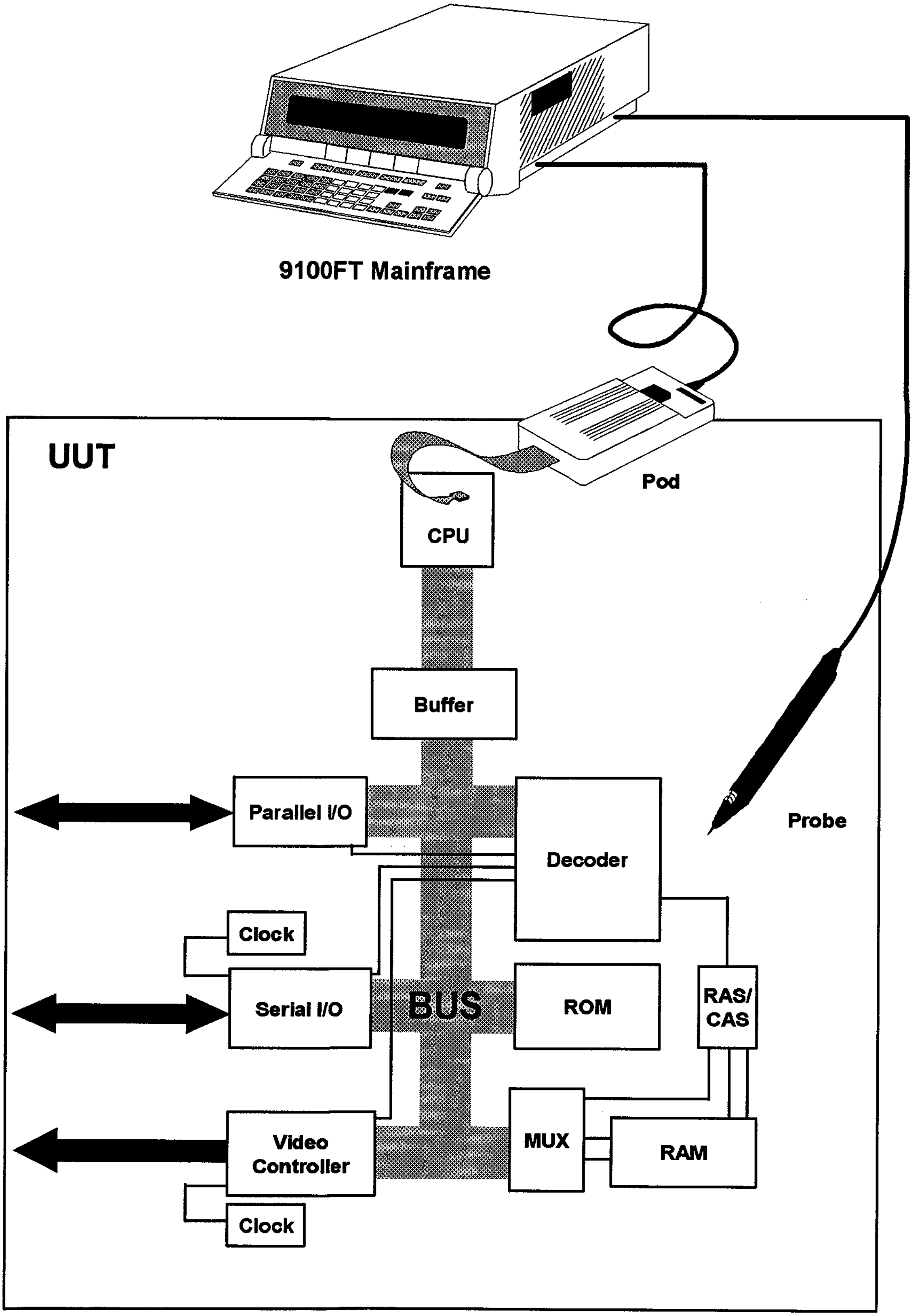
The user employs test or troubleshooting sequences that are stored on a user disk. Most often the user should be familiar with the basic methods of implementing test or fault isolation test sequences.


9100FT Mainframe
UUT
Pod
CPU
Buffer
Parallel I/O
Decoder
Probe
Clock
Serial I/O
BUS
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 through a connection at the microprocessor socket. This allows the operator to specify read and write operations anywhere in the UUT address space. All the tester's operations are derived from this fundamental ability to manipulate data at an address.

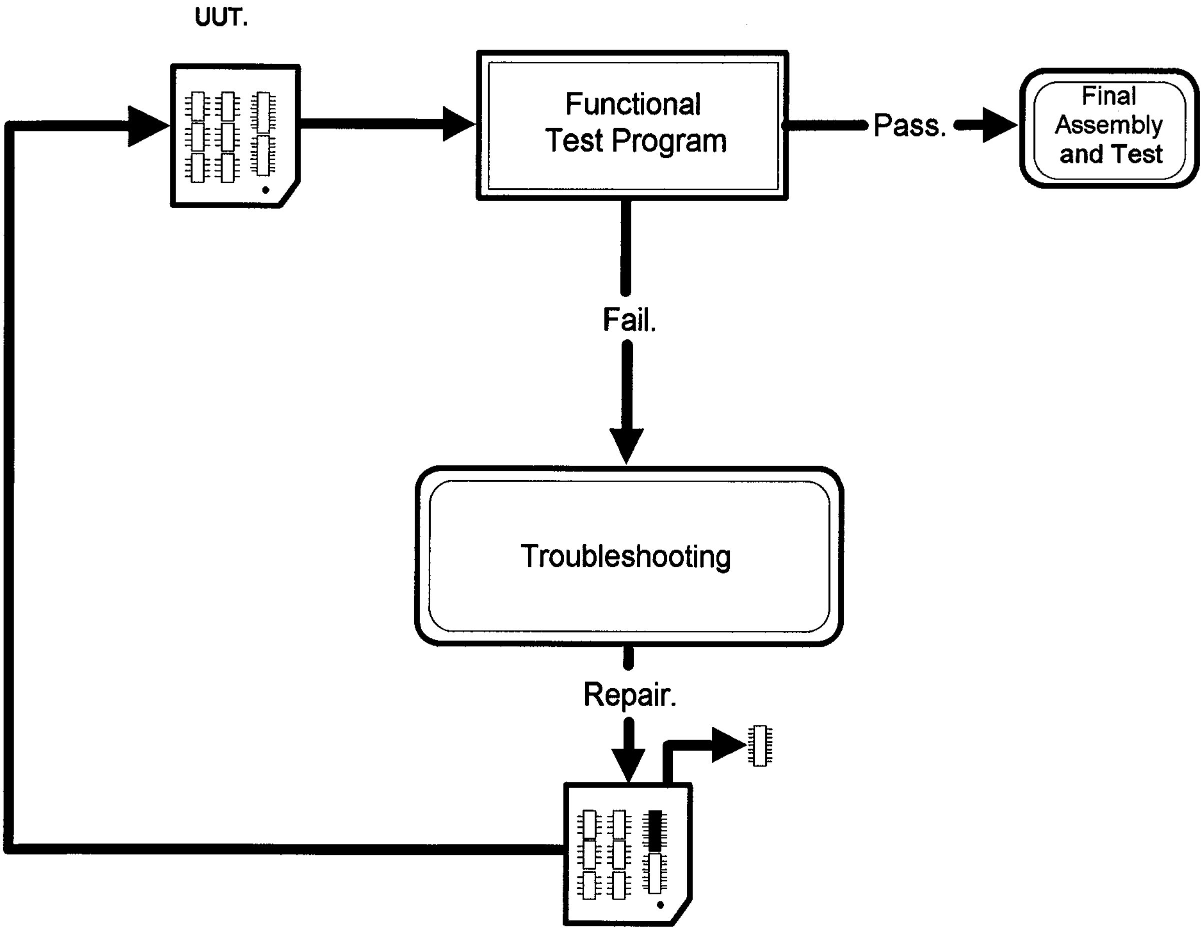
When the test system is connected to the UUT's microprocessor socket, the system takes over the digital activity normally done by the UUT's microprocessor. The test system can exercise and test ROM, RAM, I/O, or any other circuit that is related to the microprocessor bus. Some systems can also emulate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


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

9100FT 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
and Test
Fail.
Troubleshooting
Repair.

9100FT GENERAL TEST AND TROUBLESHOOTING FLOW

The* 9100-SERIES *testing strategy identifies two areas in the test process:

- ✓ FUNCTIONAL TESTING
- ✓ TROUBLESHOOTING

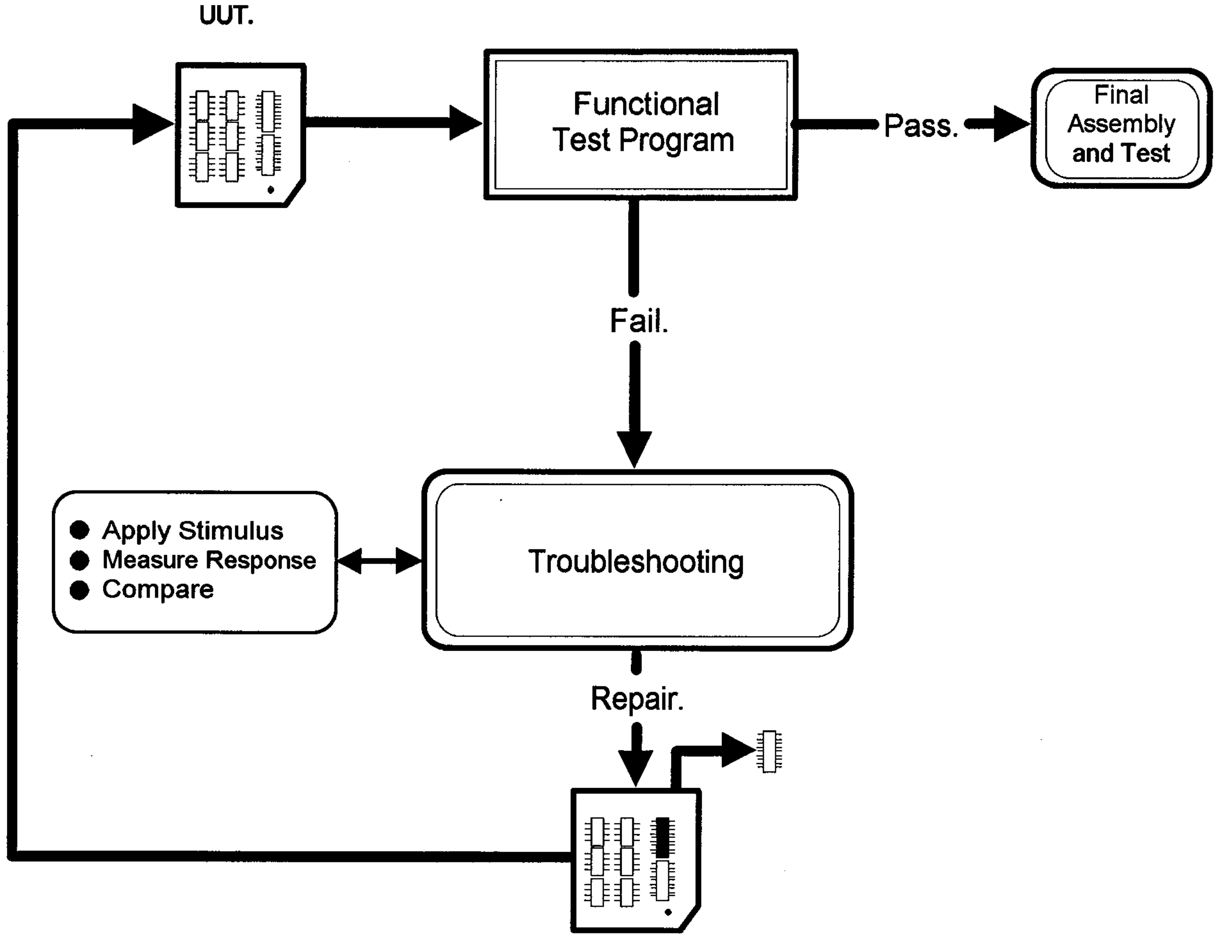
The functional test area could be subassembly, final assembly, or final test. At whatever stage the functional test is performed, the ultimate goal is to verify the proper operation of the UUT as quickly and as accurately as possible.

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 semi-automated troubleshooting while Guided Fault Isolation is fully automated troubleshooting.


UUT.
Functional Test Program
Pass.
Final Assembly and Test
Fail.
Troubleshooting
● Apply Stimulus
● Measure Response
● Compare
Repair.

CHARACTERISTICS OF TROUBLESHOOTING

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

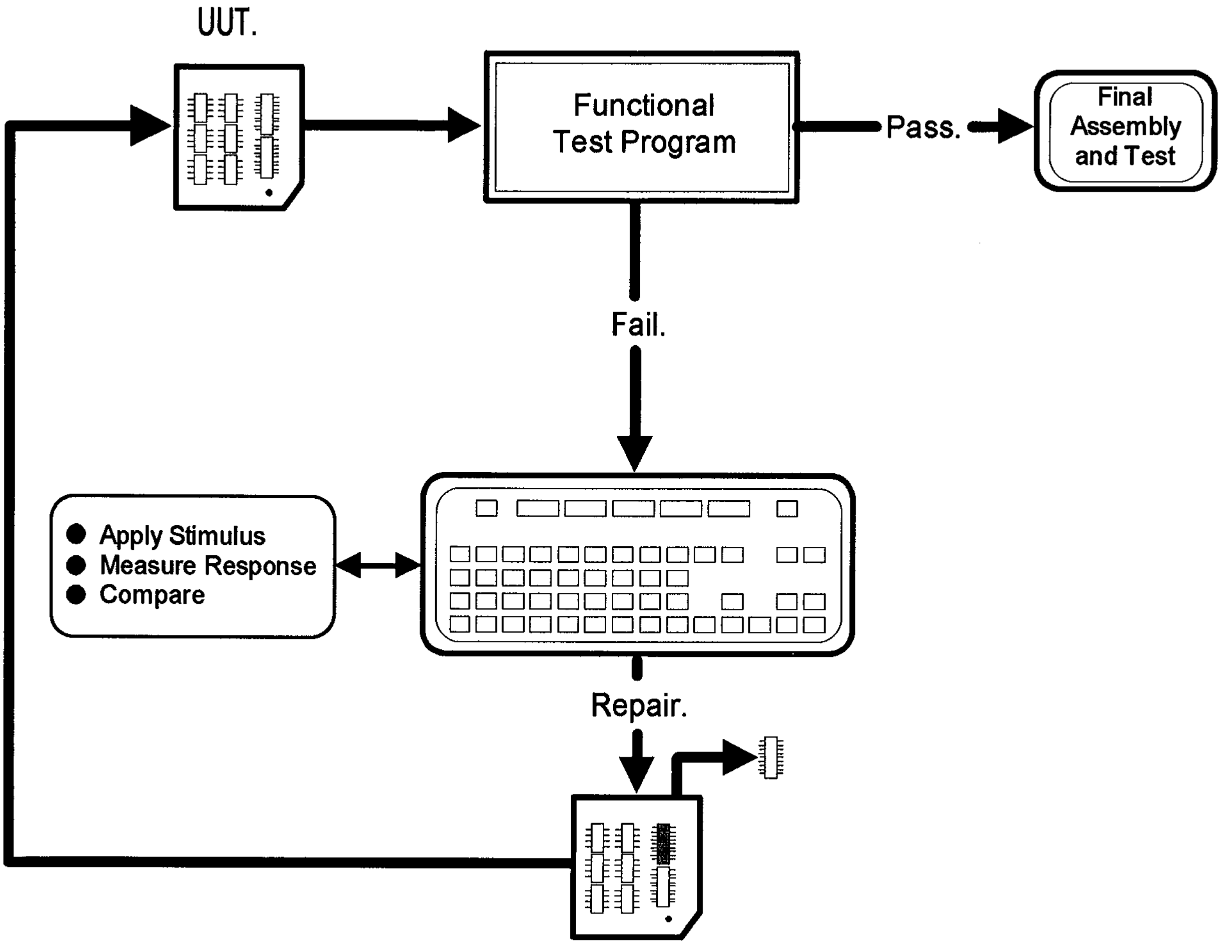
① Stimulate the circuitry being tested.

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.

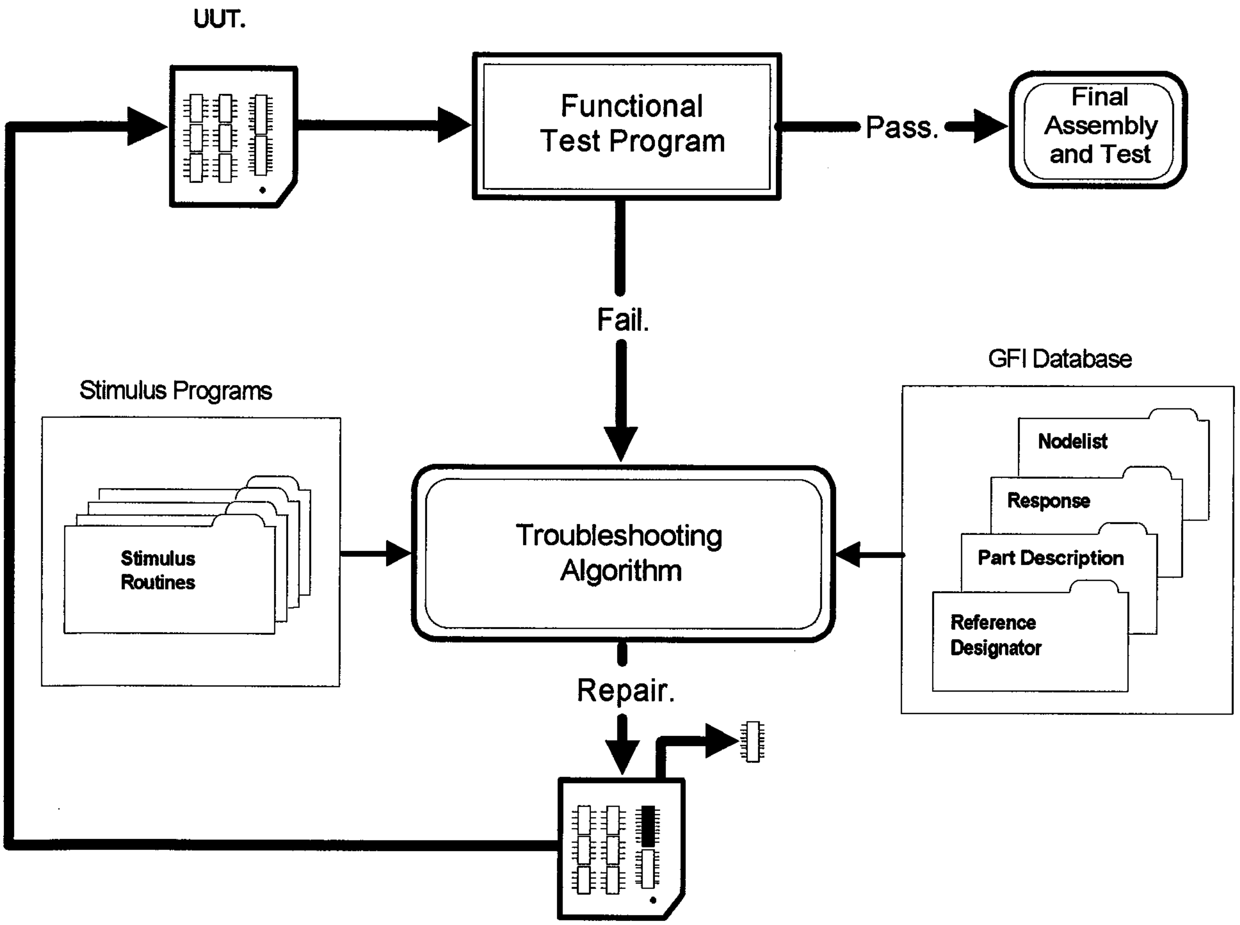

UUT.
Functional Test Program
Pass.
Final Assembly and Test
Fail.
● Apply Stimulus
● Measure Response
● Compare
Repair.

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 and Test
Fail.
GFI Database
Stimulus Programs
Stimulus Routines
Troubleshooting Algorithm
Nodelist
Response
Part Description
Reference Designator
Repair.

GUIDED FAULT ISOLATION METHOD

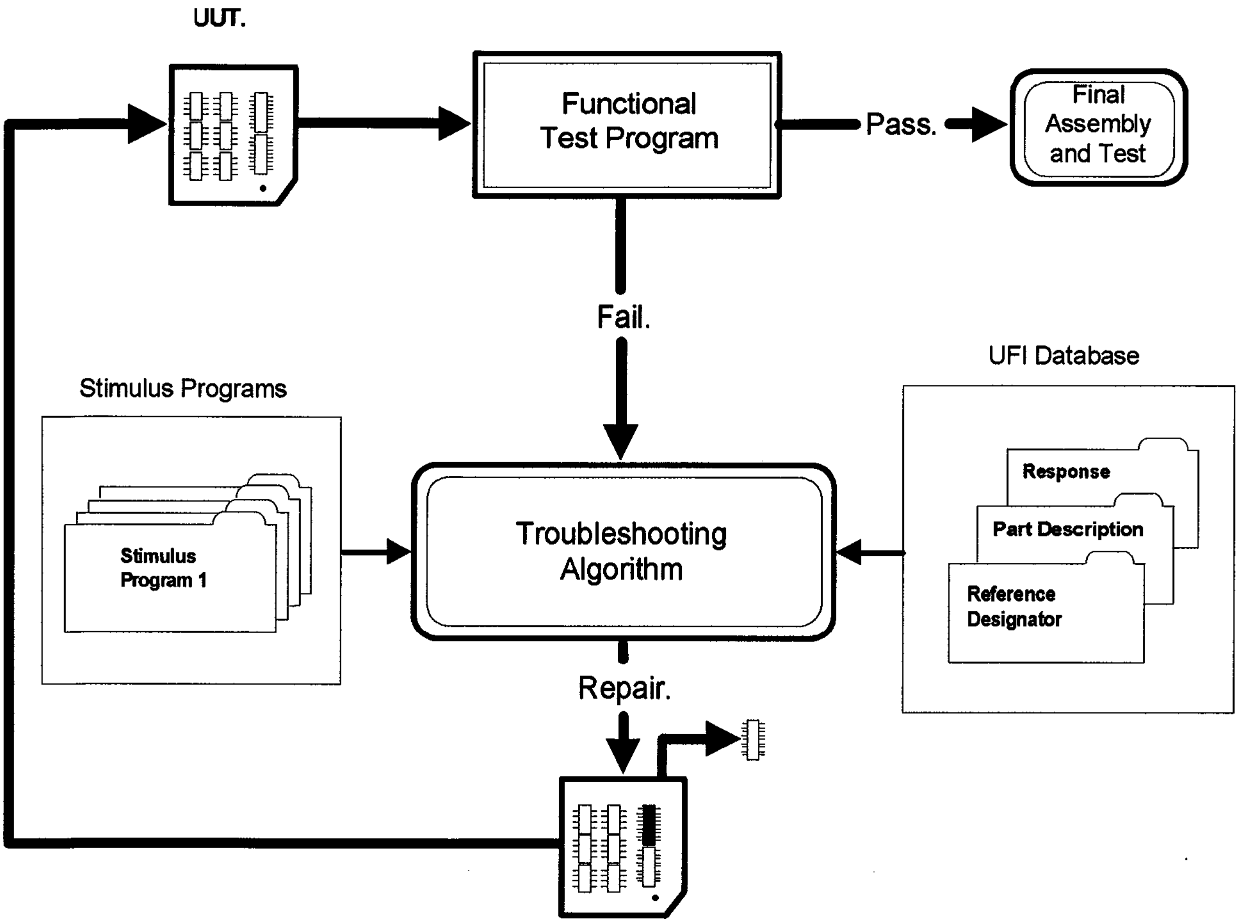
The Guided Fault Isolation GFI technique is the ultimate method in the troubleshooting process. All the elements of troubleshooting are contained in files.

The test equipment knows:

- ✓ the types of devices.
- ✓ how the various devices are connected.
- ✓ which nodes are *inputs.*
- ✓ which nodes are *outputs.*
- ✓ which nodes are *bi-directional.*
- ✓ how to stimulate each of the nodes.
- ✓ which measurement tool is used to determine if the response is correct.
- ✓ what the correct response is.
- ✓ 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
and Test
Fail.
UFI Database
Stimulus Programs
Stimulus
Program 1
Troubleshooting
Algorithm
Response
Part Description
Reference
Designator
Repair.

UNGUIDED FAULT ISOLATION METHOD

The Unguided Fault Isolation UFI method of troubleshooting incorporates semi-automation. 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 simply tells the test equipment which node to test.

For example...

① The operator tells the test equipment to test U1 pin 1.

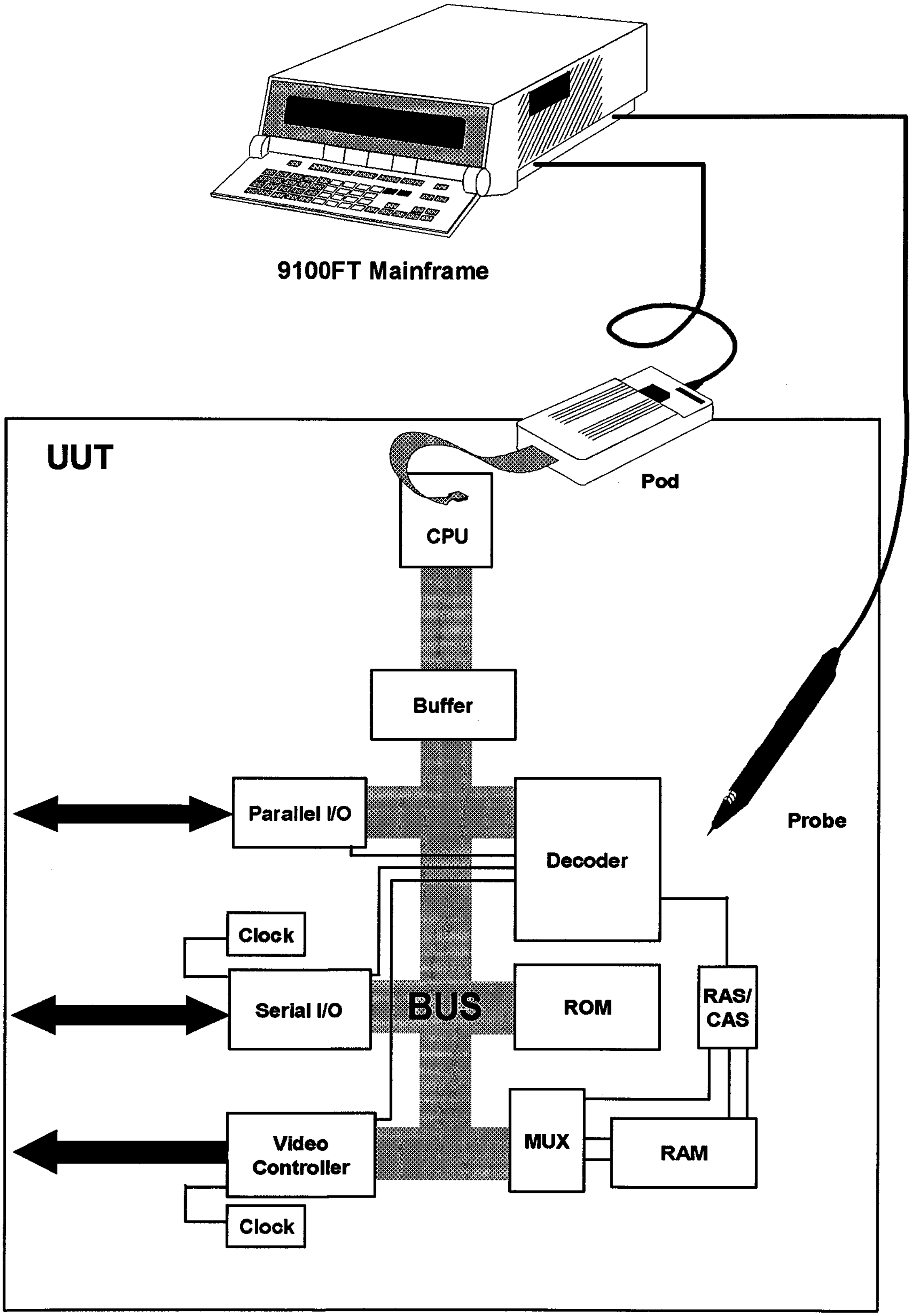
② The equipment goes to the stimulus program and pulls out the proper stimulus for that node.

③ The file tells the operator which measurement device to use (such as a probe or an I/O module).

④ 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.

⑤ 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.


9100FT Mainframe
UUT
Pod
CPU
Buffer
Parallel I/O
Decoder
Probe
Clock
Serial I/O
BUS
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 9100FT Mainframe and be able to choose the characteristics that best fit your application.

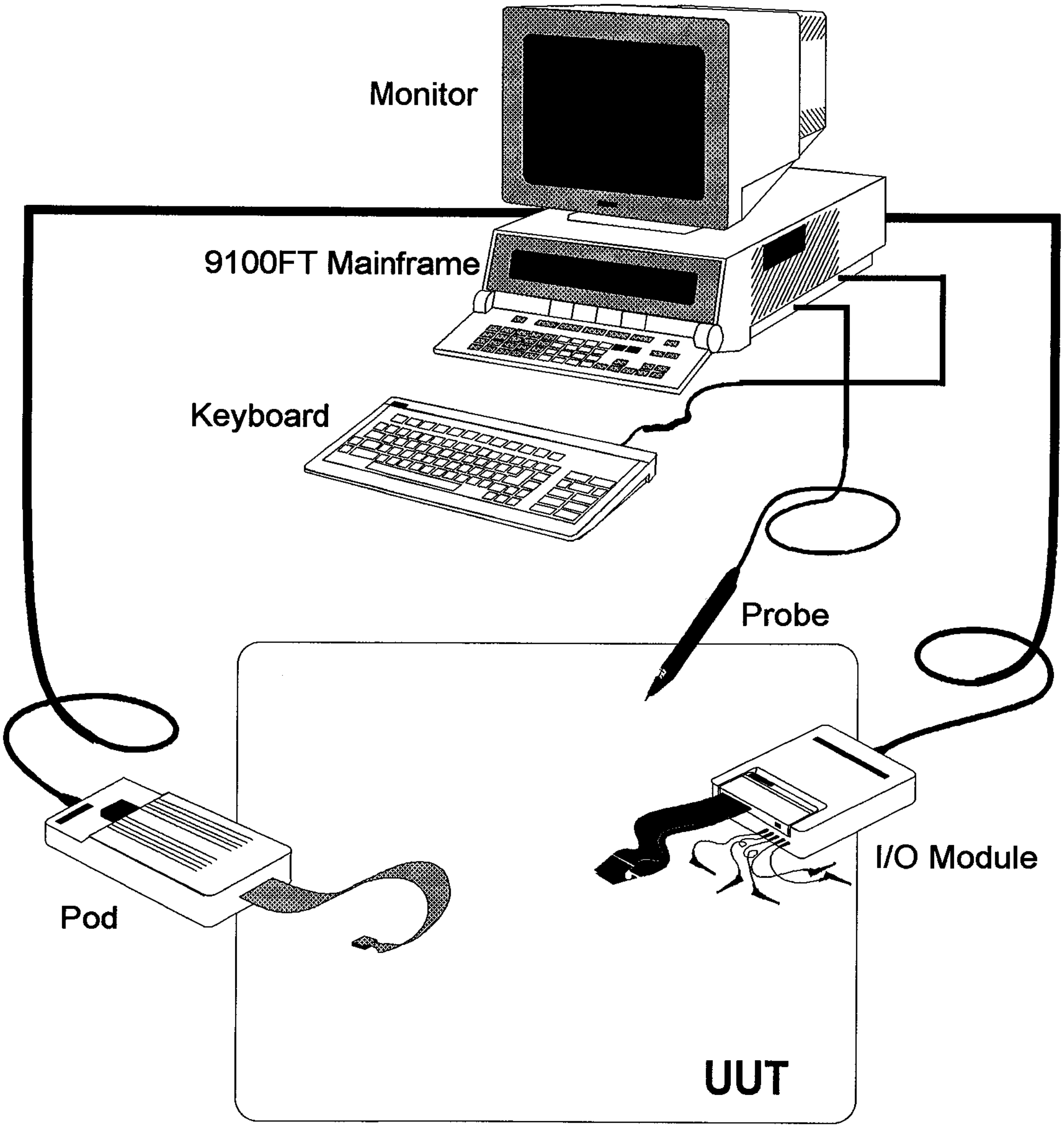
Section 2

System Configuration

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.

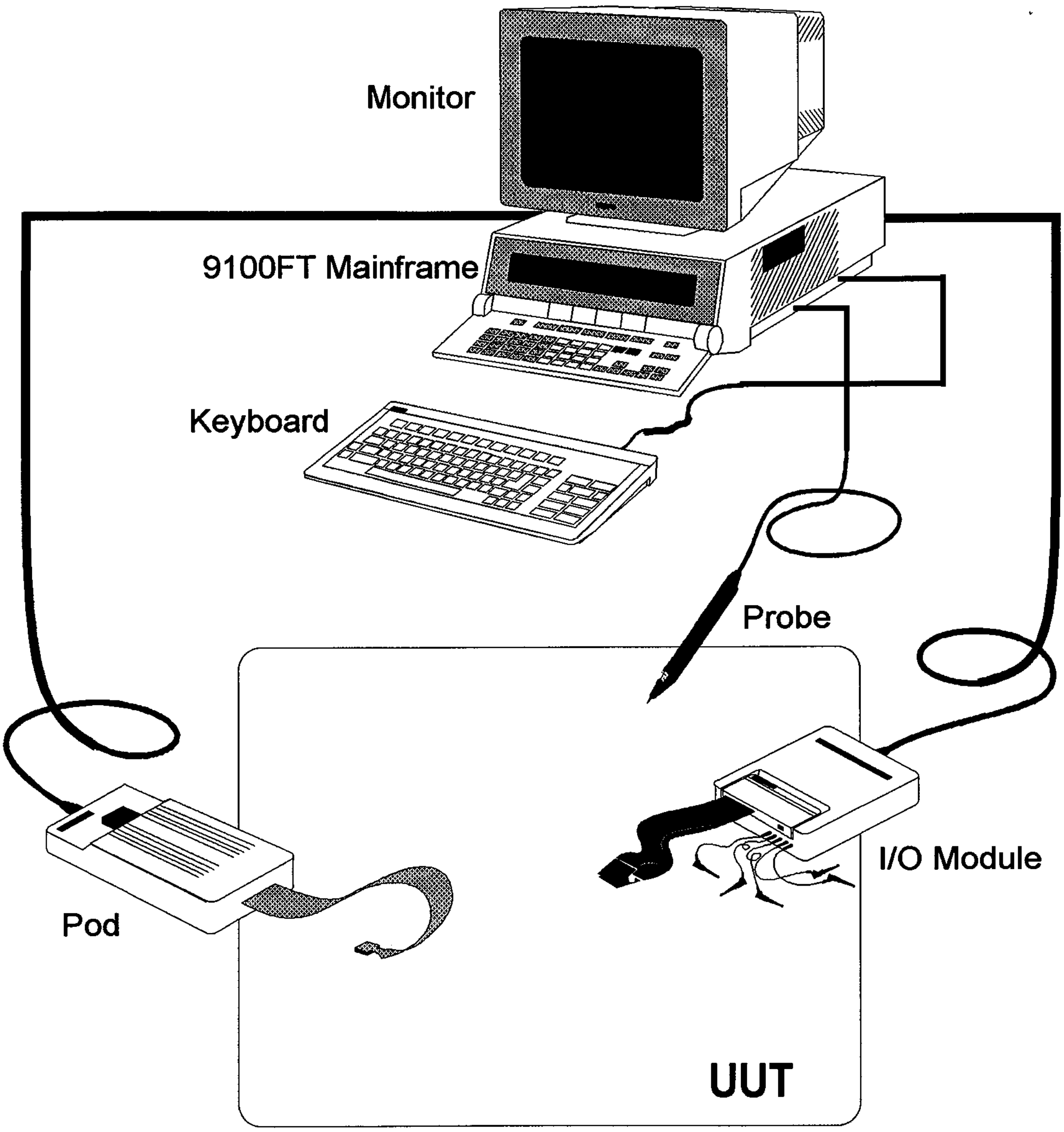
You will also perform the calibration procedure for the probe and the I/O *Module. The calibration procedures ensure accurate and repeatable measurements. These procedures must be done to ensure correct testing and test development.*


Monitor
9100FT Mainframe
Keyboard
Probe
I/O Module
Pod
UUT

THE 9100FT SYSTEM

The 9100FT 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.


Monitor
9100FT Mainframe
Keyboard
Probe
I/O Module
Pod
UUT

EXERCISE 2-1

PREPARING THE 9100FT FOR TESTING - EXERCISE 2-1

① Cable connections.

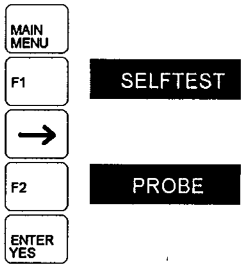
② Description of each part.

③ Power-up the 9100FT before UUT to activate
Pod protection circuits.

④ Selftests:

a. *Probe*

press the following sequence of keys...



MAIN MENU

F1 SELFTEST

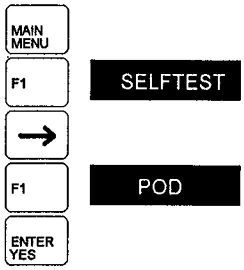
→

F2 PROBE

ENTER YES

b. *Pod:*

press the following seequence of keys...



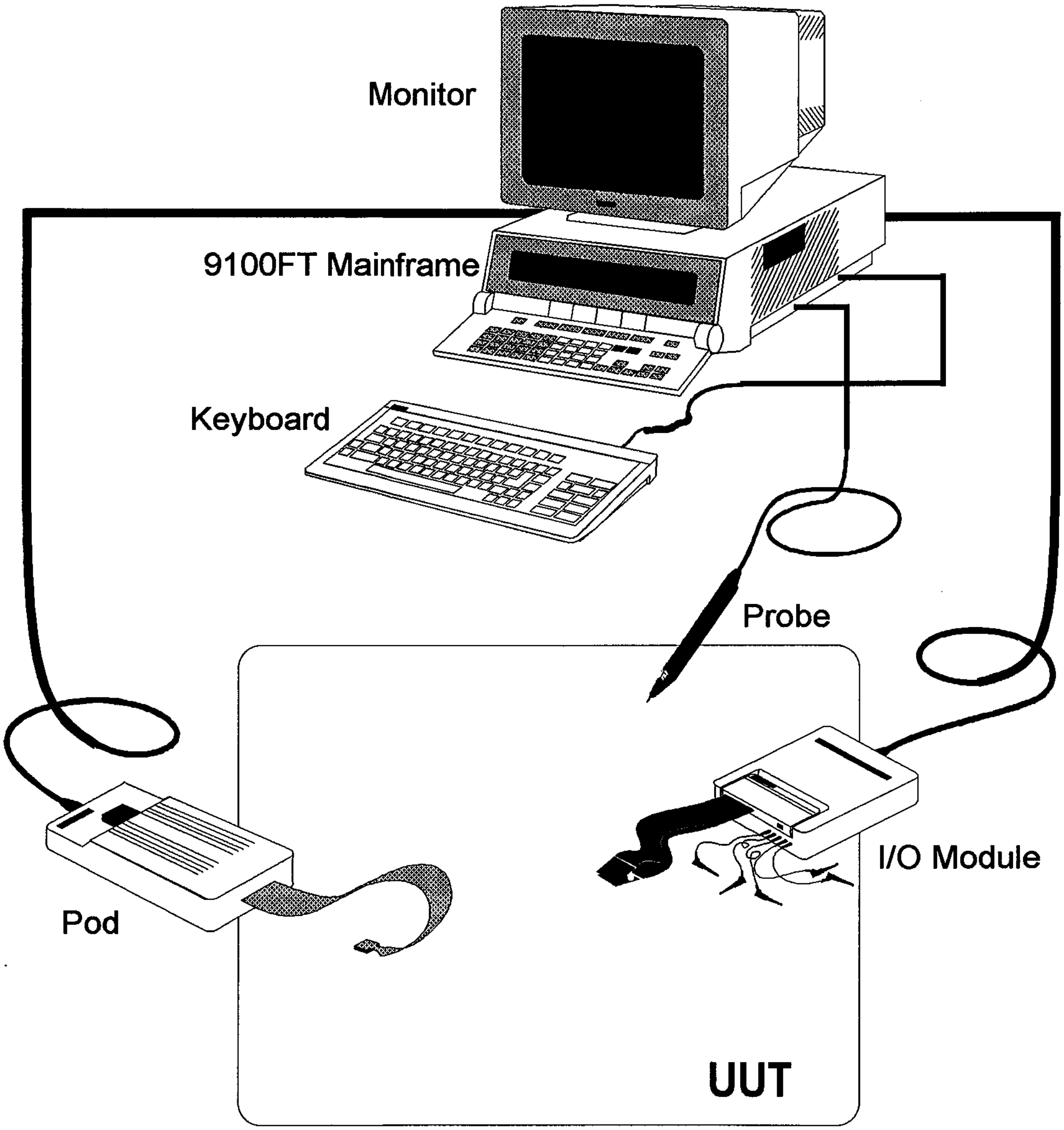
MAIN MENU

F1 SELFTEST

→

F1 POD

ENTER YES


Monitor
9100FT Mainframe
Keyboard
Probe
I/O Module
Pod
UUT

c. *I/O Module:*

press the following sequence of keys...

MAIN MENU

F1 SELFTEST

→

F3 I/O MOD

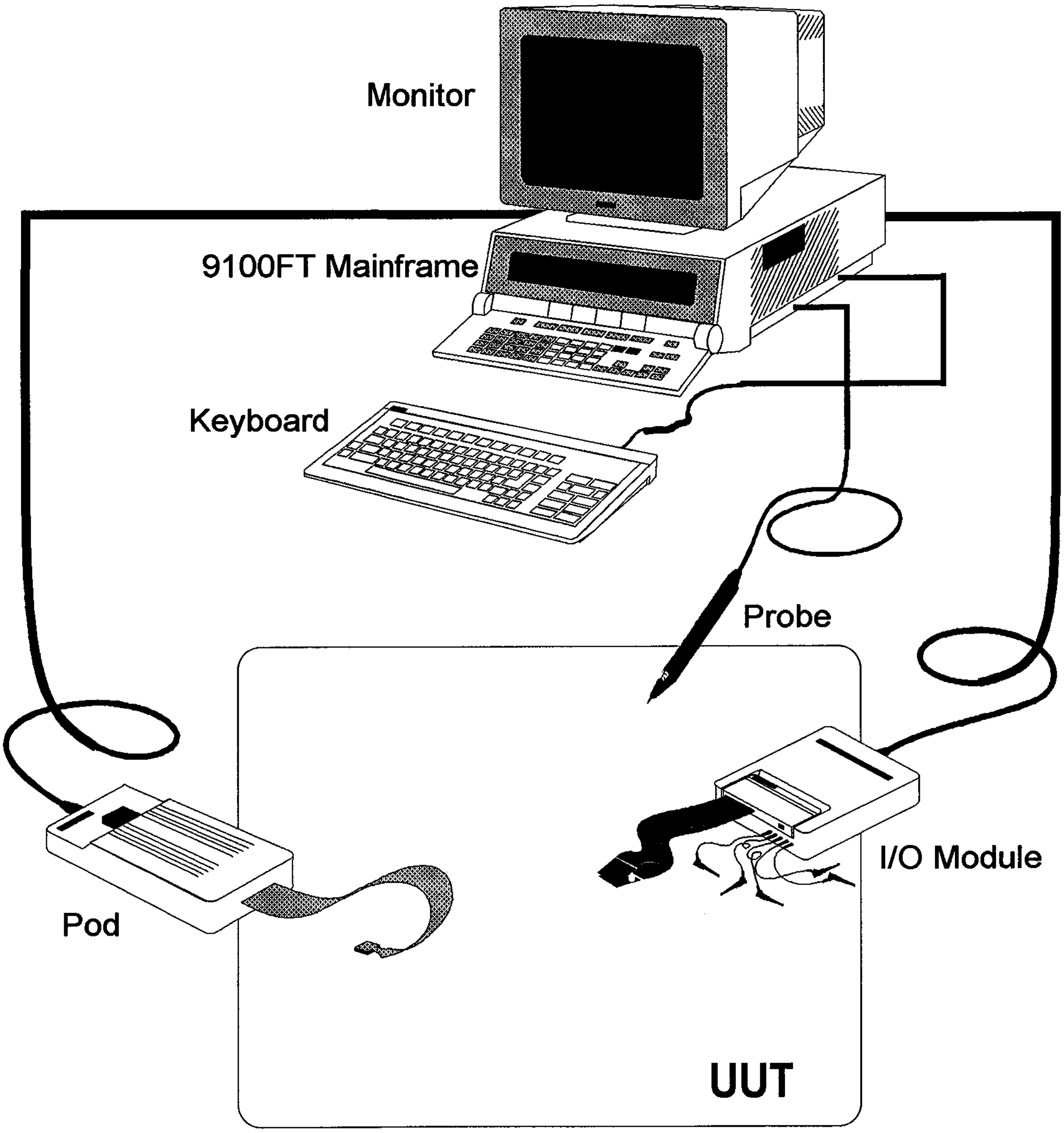
→

F4 INC or F5 DEC until 1 is displayed

ENTER YES

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

⑤ UUT Connection.

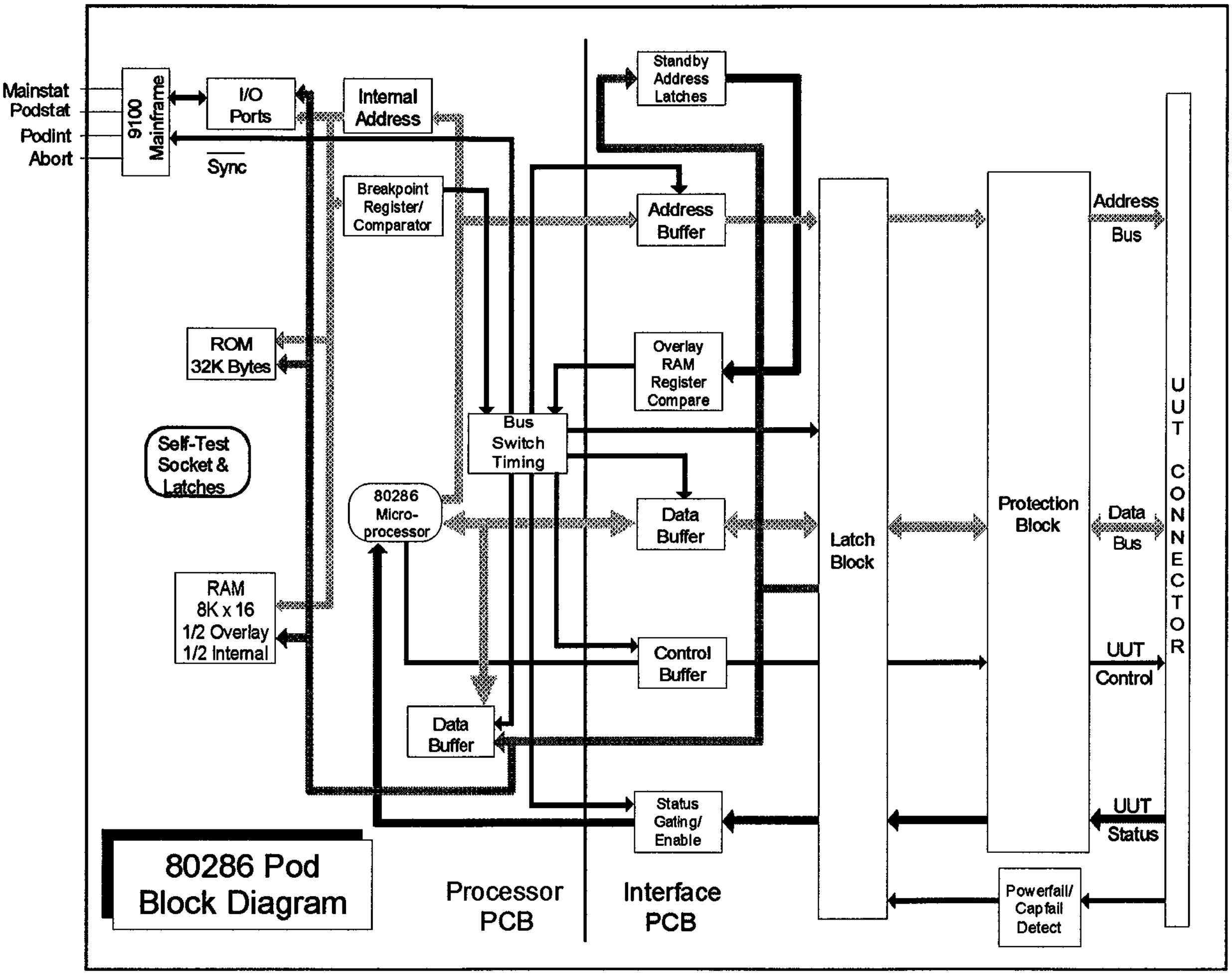

Monitor
9100FT Mainframe
Keyboard
Probe
I/O Module
Pod
UUT

9100FT OVERVIEW

The 9100FT system consists of a 9100FT Digital System without the programming option, version 6.1 System Software Disks, version 6.1 Master User Disks (Note that the System Disk and Master User Disk software is preloaded onto the hard disk).

The Master User Disks include 32 pod databases to support standard microprocessor interface pods.

Testing and troubleshooting digital boards with the 9100FT 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.



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 microprocessor emulation pods is essentially the same. Each pod uses the same method to communicate with the mainframe, contains interface 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 the UUT busses 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.

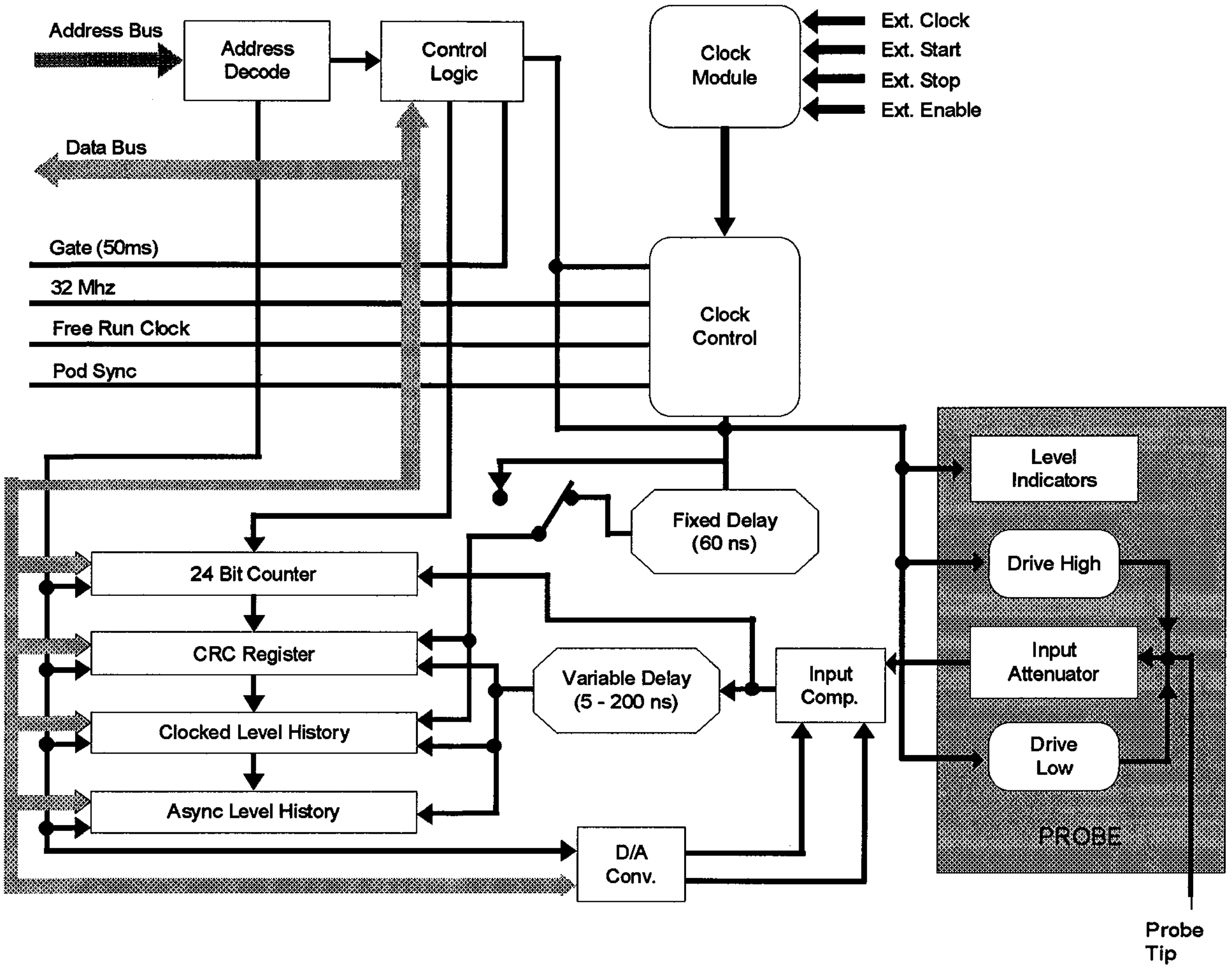
Status lines are control inputs to the microprocessor. Sometimes these lines can stop test execution entirely. The mainframe therefore, is capable of measuring these lines or, ignoring these lines and continuing with execution.

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

- **MAINSTAT**
- **PODINT**
- **ABORT**
- **PODSTAT**

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

9100FT Probe System



THE PURPOSE OF THE PROBE AND CLOCK MODULE

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

As a measurement tool , the probe detects:

- ✓ level activity (whether the activity is synchronous or asynchronous to the microprocessor)
- ✓ transition counts
- ✓ frequency
- ✓ CRC signatures.

The activity measured by the probe can be synchronized to the pod, the clock module, or a free-running clock.

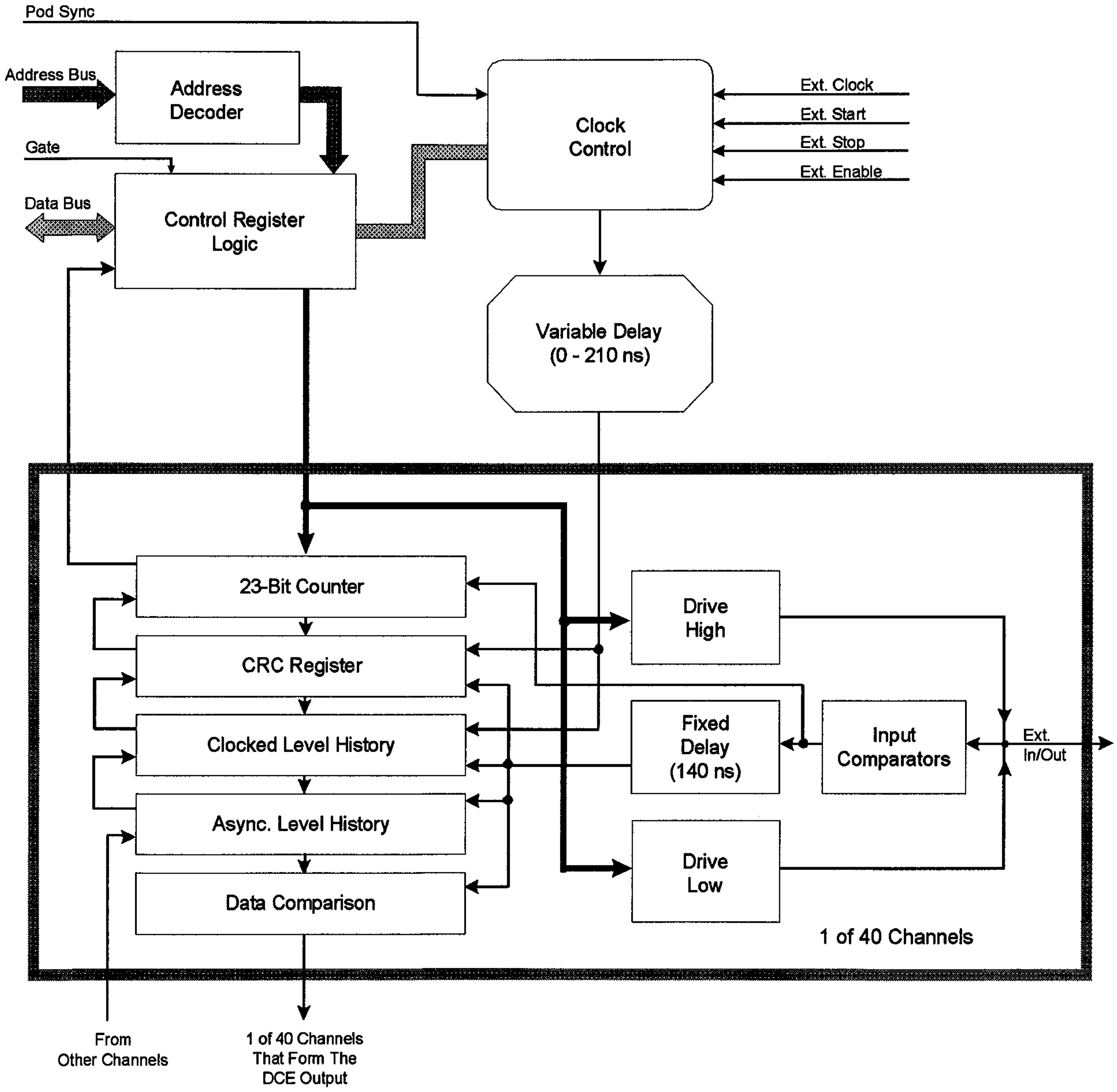
As a stimulus device, the probe provides pulses that are High, Low, or toggling. The pulses are preset to 0-5 volts independent of set thresholds TTL, CMOS, RS-232, or ECL.

The probe contains input circuitry consisting of an attenuator and level indicators. The indicators have circuitry that stretches the pulses allowing the operator to see the levels of very fast pulses.

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

The probe circuitry communicates with the mainframe through the mainframe's address and data bus.

9100FT I/O Module Block Diagram



THE FUNCTION OF THE I/O MODULE

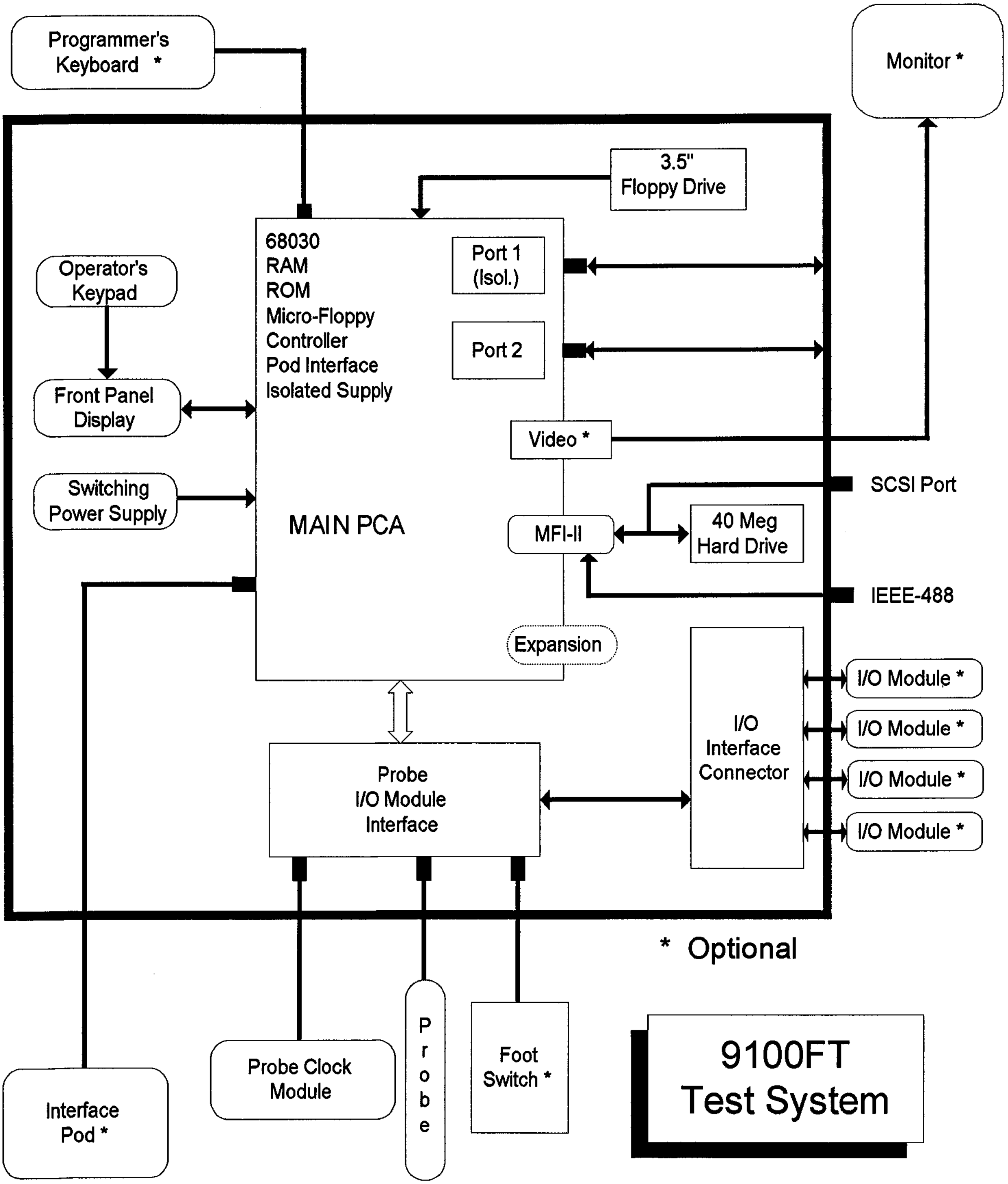
The function of the I/O module is essentially the same as that of the probe. The I/O module acts as a measurement device that senses activity on the circuitry and as a stimulus device that provides activity to the circuitry. The difference between the two is that where the I/O module has 40 channels,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

As a measurement tool, the I/O module can detect:

- ✓ level activity (both synchronous and asynchronous)
- ✓ transition counts
- ✓ frequency
- ✓ CRC Signatures
- ✓ pattern recognition

As a stimulus device, the I/O module not only can pulse as the probe but can also drive/overdrive outputs.

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



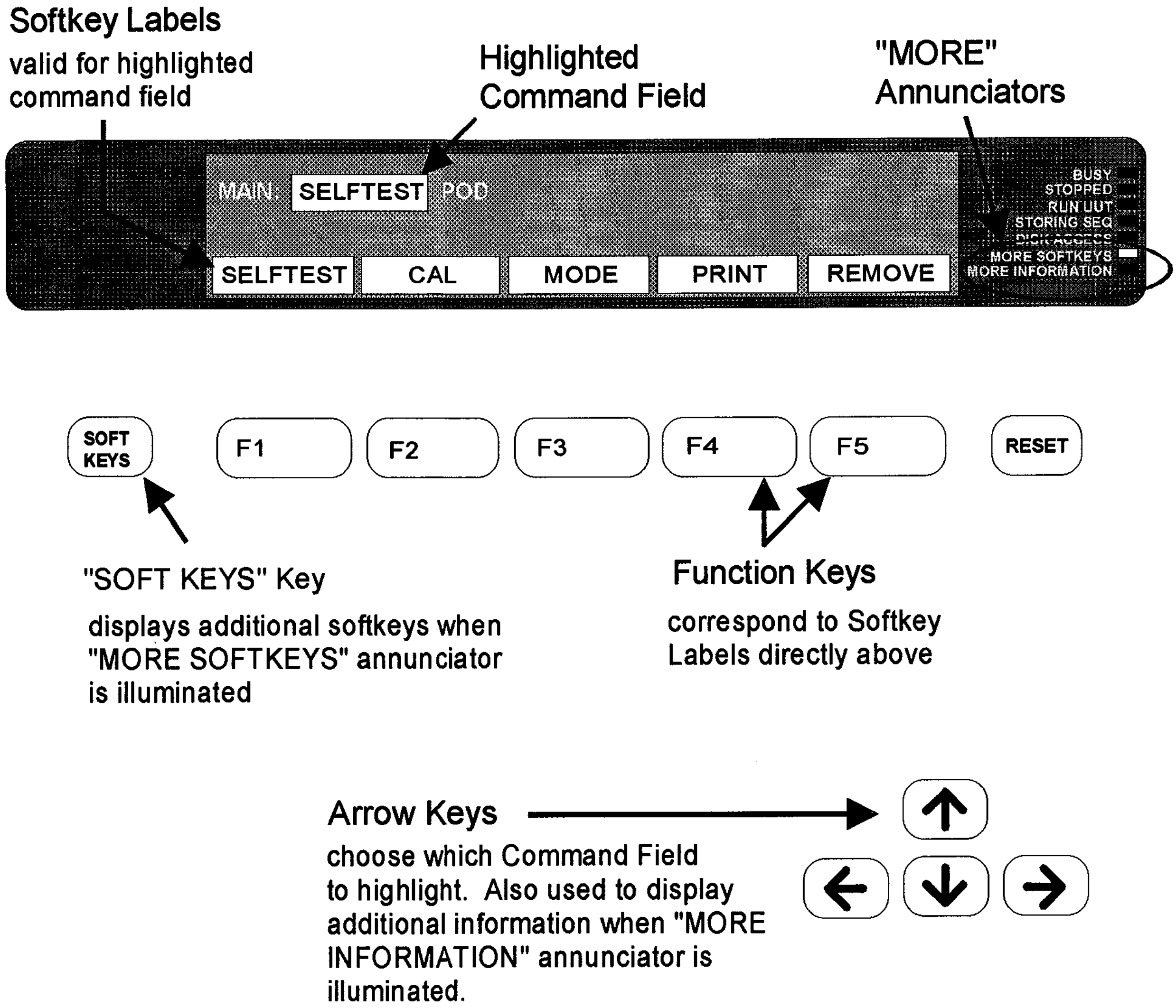
* Optional

9100FT 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, etc.

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



Softkey Labels

valid for highlighted command field

Highlighted Command Field

"MORE" Annunciators

"SOFT KEYS" Key

displays additional softkeys when "MORE SOFTKEYS" annunciator is illuminated

Function Keys

correspond to Softkey Labels directly above

Arrow Keys

choose which Command Field to highlight. Also used to display additional information when "MORE INFORMATION" annunciator 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

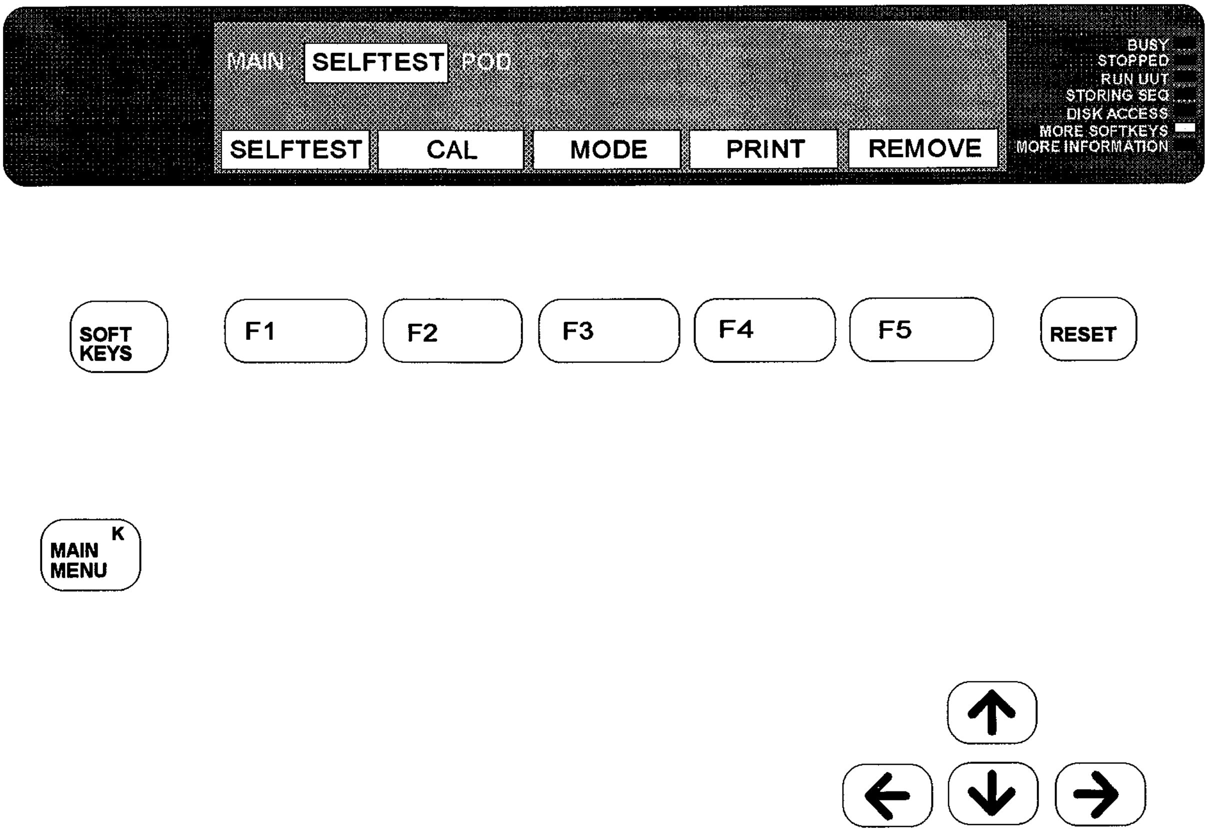
The words in boxes on the bottom line of the mainframe display (e.g. SELFTEST) are called *softkey labels* and correspond to the function keys F1 through F5 on the Application Keypad.

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 it's appropriate field. If there are more softkey options than can be displayed at one time, a red "MORE SOFTKEYS" annunciator (located to the right of the display) is illuminated.

The SOFT KEYS button scrolls hidden softkey labels into view if the annunciator is on. Pressing the key repeatedly eventually 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 display. A *field* is a word, number, or alpha input in a command line that can be changed. To change a field, the field must be flashing or have a flashing cursor.


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
K
MAIN MENU

EXERCISE 2-2

USING SOFTKEYS -EXERCISE 2-2

① *press...* the MAIN MENU key.

Notice that the red **MORE SOFTKEYS** annunciator to the right of the display is illuminated.

② *press...* the SOFT KEYS key

Notice the different softkey labels.

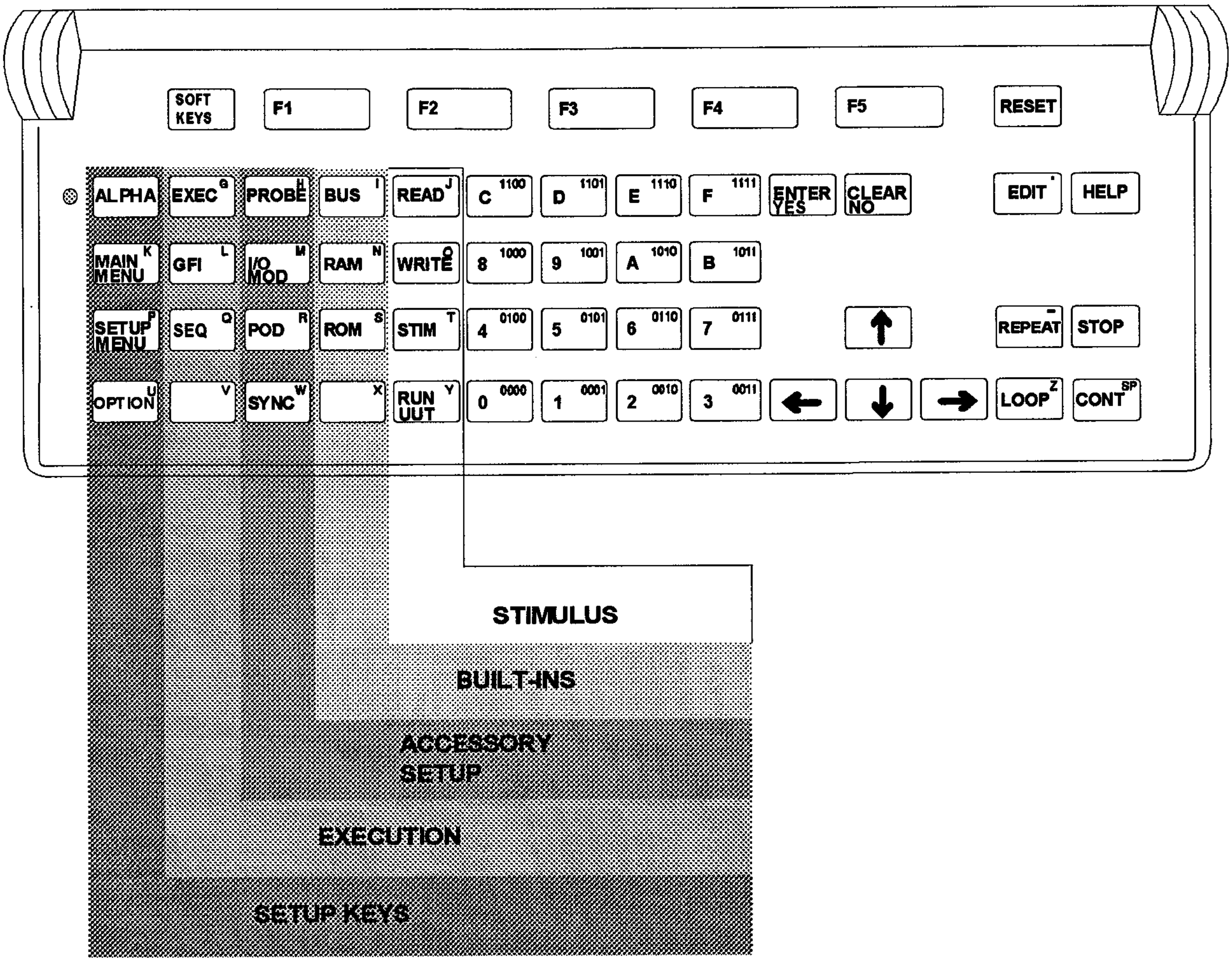
③ *press...* the SOFT KEYS key a second time

Previously displayed softkeys are shown

Observe how the SOFT KEYS key toggles between both softkey label sets valid for the current command level.

④ *press...* the → key

Observe how the flashing field switches from SELFTEST to POD and how the softkey options change to reflect the valid command options for the field.


SOFT KEYS
F1
F2
F3
F4
F5
RESET
ALPHA
EXEC G
PROBE H
BUS I
READ J
C 1100
D 1101
E 1110
F 1111
ENTER YES
CLEAR NO
EDIT '
HELP
MAIN MENU K
GFI L
I/O MOD M
RAM N
WRITE O
8 1000
9 1001
A 1010
B 1011
SETUP MENU P
SEQ Q
POD R
ROM S
STIM T
4 0100
5 0101
6 0110
7 0111
↑
REPEAT -
STOP
OPTION U
V
SYNC W
X
RUN UUT Y
0 0000
1 0001
2 0010
3 0011
←
↓
→
LOOP Z
CONT SP
STIMULUS
BUILT-INS
ACCESSORY SETUP
EXECUTION
SETUP KEYS

EXERCISE 2-3

KEYPAD INTRODUCTION - EXERCISE 2-3

Setup Keys:

ALPHA | MAIN MENU K | SETUP MENU P | OPTION U

Accessory Setup:

EXEC G | GFI L | SEQ Q

Accessory Setup:

PROBE H | I/O MODE M | POD R | SYNC W

Built-ins:

BUS I | RAM N | ROM S

Stimulus:

READ J | WRITE O | STIM T | RUN UUT Y

EXAMPLE TEST:

EXAMPLE FAULT - FAULT SWITCH 2-7

Bus Test:

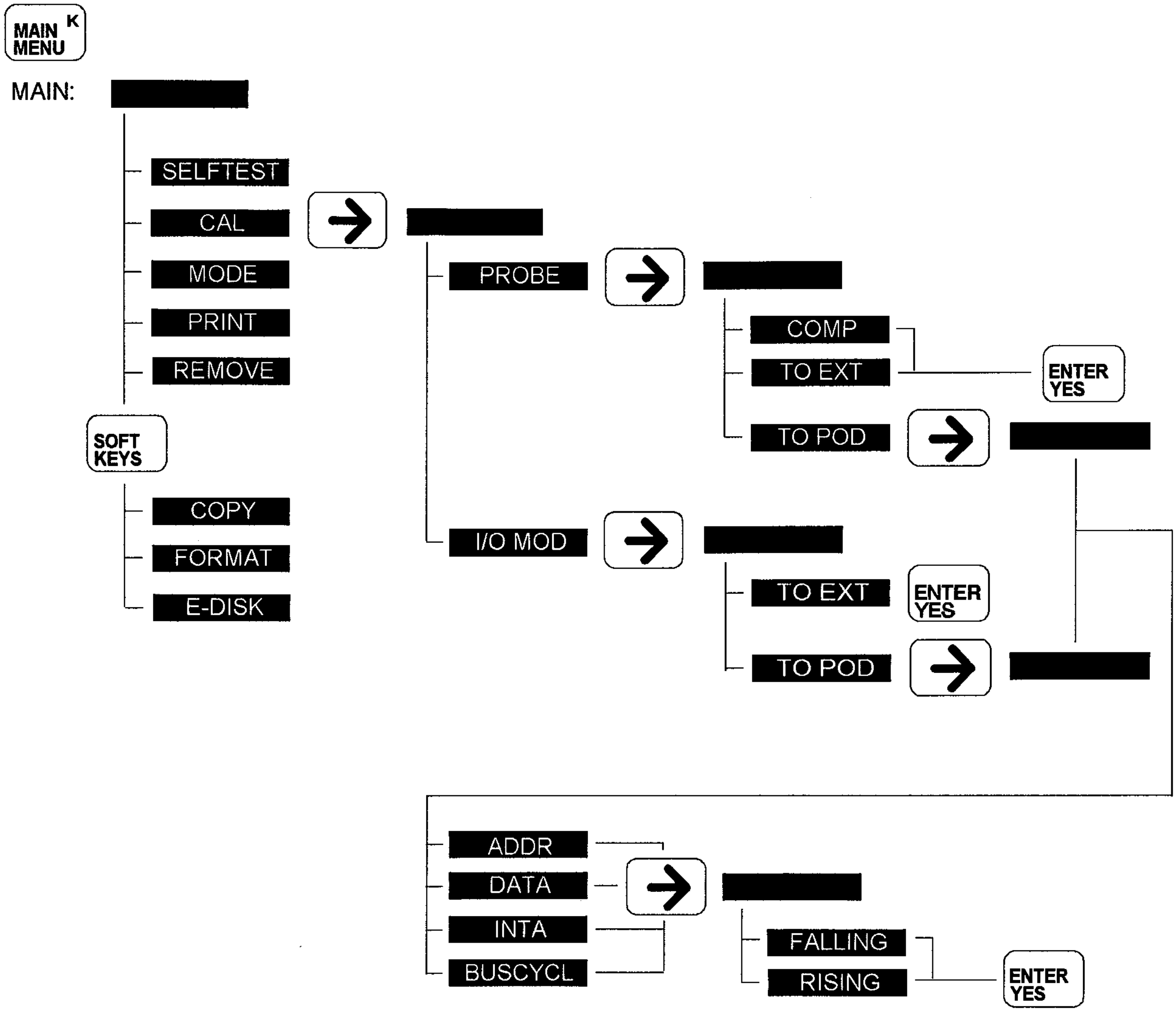
press... BUS I | 0 0000 | ENTER YES

Help Key:

press... HELP

to read the complete message:

press... ↓


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

EXERCISE 2-4

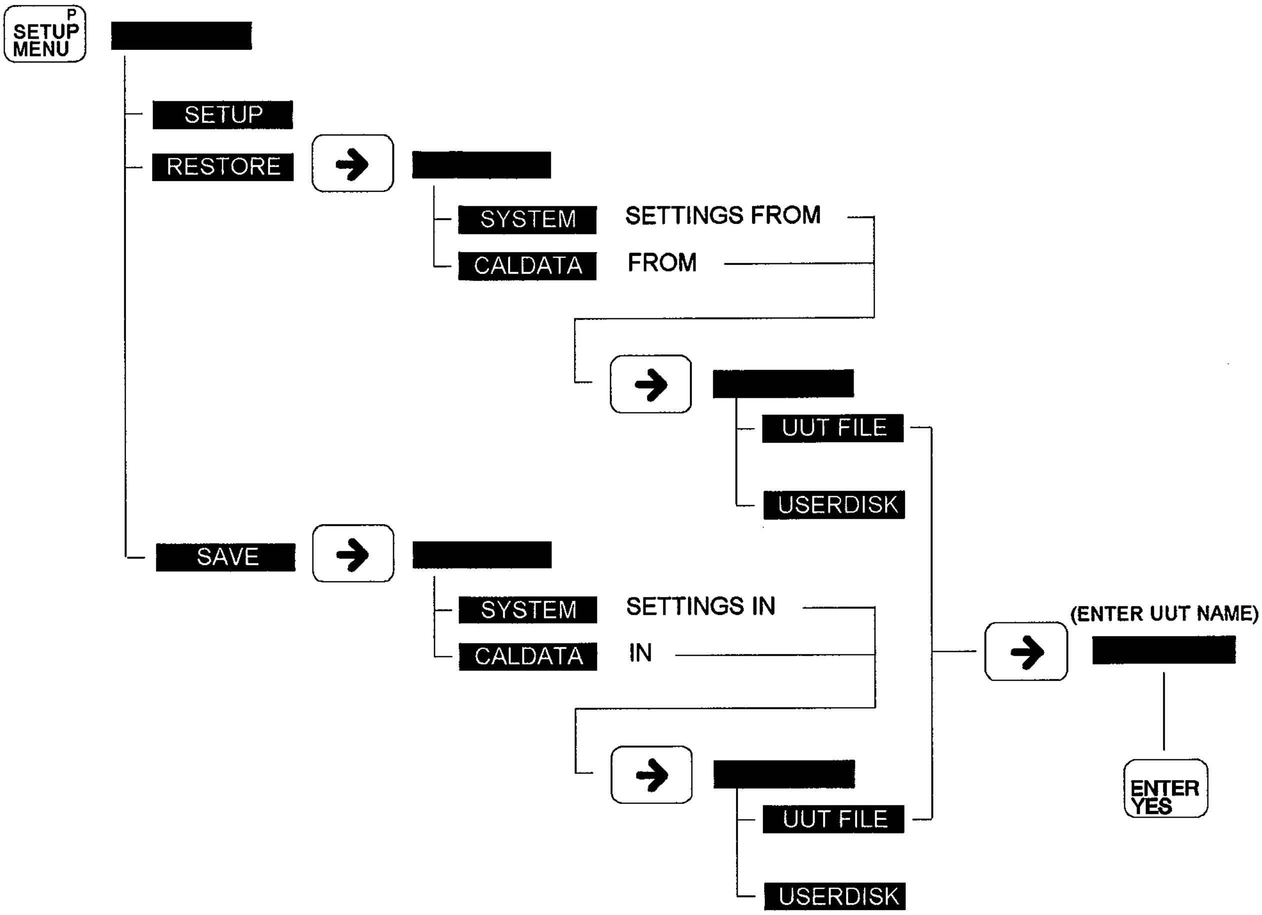
PROBE AND I/O MODULE CALIBRATION - EXERCISE 2-4

To ensure proper operation when troubleshooting, the Probe and I/O module must be calibrated with the UUT. The calibration can be saved 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 or reset.

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 Address**
- ✓ **Pod Data**

Note: *Calibration of the I/O module requires the use of the I/O module calibration header* (FPN 813980).


P
SETUP
MENU
SETUP
RESTORE
SYSTEM
SETTINGS FROM
CALDATA
FROM
UUT FILE
USERDISK
SAVE
SYSTEM
SETTINGS IN
CALDATA
IN
UUT FILE
USERDISK
(ENTER UUT NAME)
ENTER
YES

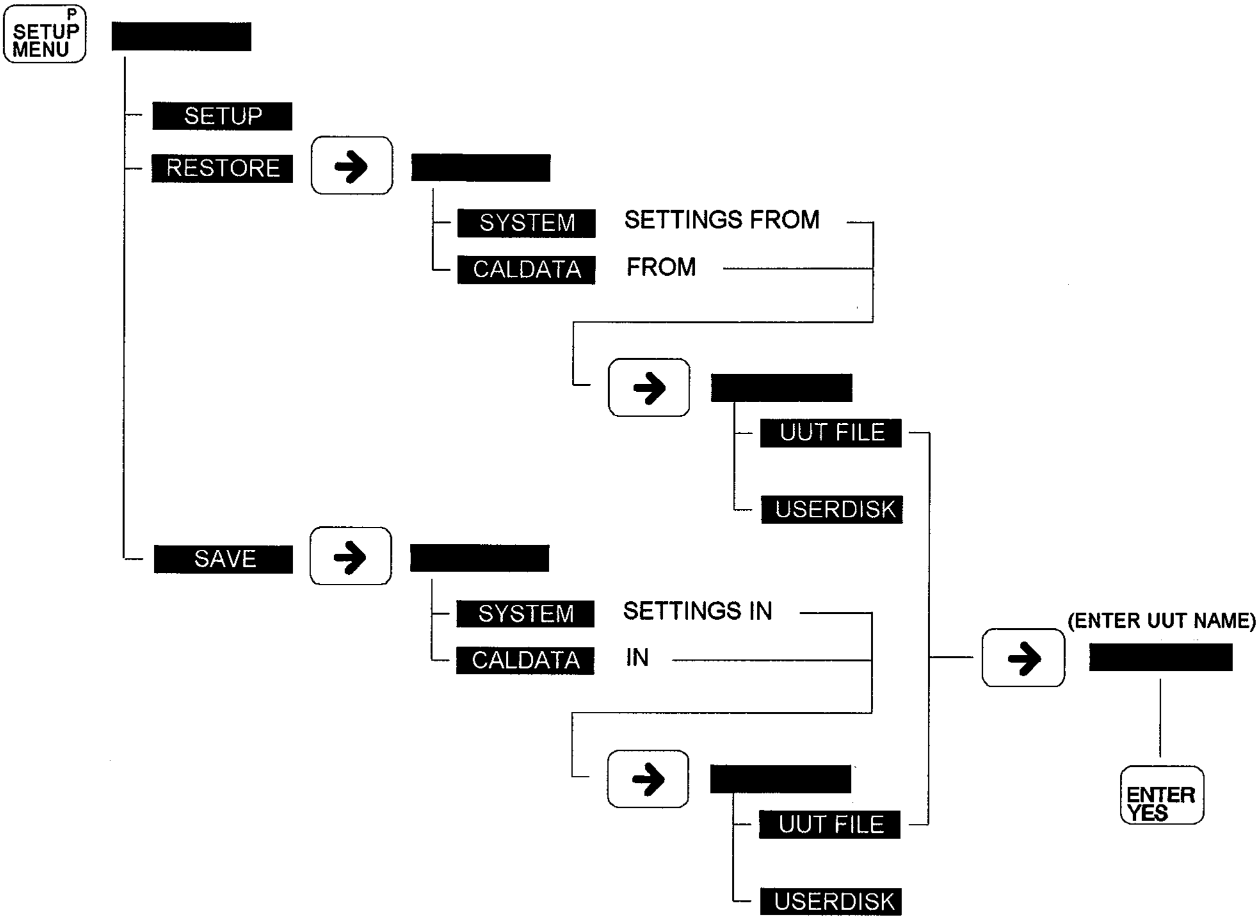
EXERCISE 2-5

SAVING CALIBRATION DATA - EXERCISE 2-5

After performing the calibration of the probe and I/O module, save the calibration data by selecting the Setup Menu.

When saving either the calibration data or system settings, your choices are the USERDISK or UUT FILE. It is recommended that you save your calibration data to a **UUT FILE**.

For this exercise, save your calibration data to a **UUT FILE** in the **TRAINER** directory.

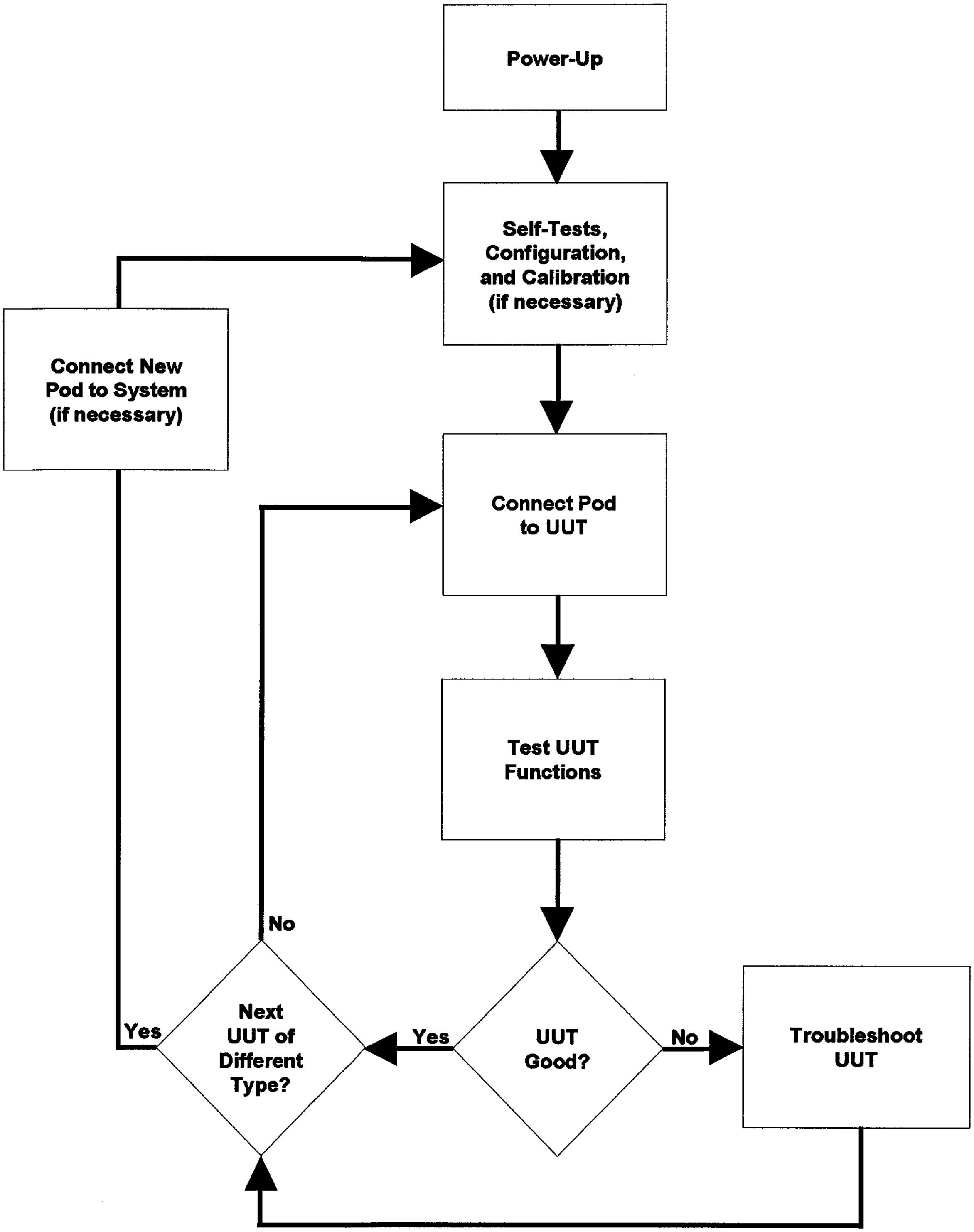

P
SETUP
MENU
SETUP
RESTORE
SYSTEM
SETTINGS FROM
CALDATA
FROM
UUT FILE
USERDISK
SAVE
SYSTEM
SETTINGS IN
CALDATA
IN
UUT FILE
USERDISK
(ENTER UUT NAME)
ENTER
YES

EXERCISE 2-6

RESTORING CALIBRATION DATA - EXERCISE 2-6

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

Use the Setup Menu to restore the calibration data you saved in the **TRAINER** directory, **UUT FILE**.


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-Tests:

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

Configuration:

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

Calibration:

Generates data which can be saved on the user disk and restored at power-up or reset. This data is sometimes invalid.

Calibration should be performed when:

- ✓ The system is first installed and every 30 days thereafter.
- ✓ The Pod, Probe, or I/O Module is changed.

If the stored calibration is valid, restore the calibration data whenever:

- ✓ The system is powered up or reset.
- ✓ The type of UUT is changed.

This page intentionally left blank.

Before you test or troubleshoot a UUT, you should ensure that selftests, calibrations, and configurations have been performed appropriately. Some guidelines are as follows:

After each power-up or reset, before testing or troubleshooting a UUT:

- ✓ Configure and calibrate the system for the type of UUT you are about to test or troubleshoot.
- ✓ Ensure that the Self-tests have been performed on the Pod, Probe, and I/O Modules.
- ✓ Verify that the system settings are set or restored for the type of UUT.

If mainframe was reset or any part of system was changed:

- ✓ Self-tests and calibrations may need to be repeated. Save new calibration data to the user disk, writing over the old data.
- ✓ Test and troubleshooting sequences and system configuration remain valid.

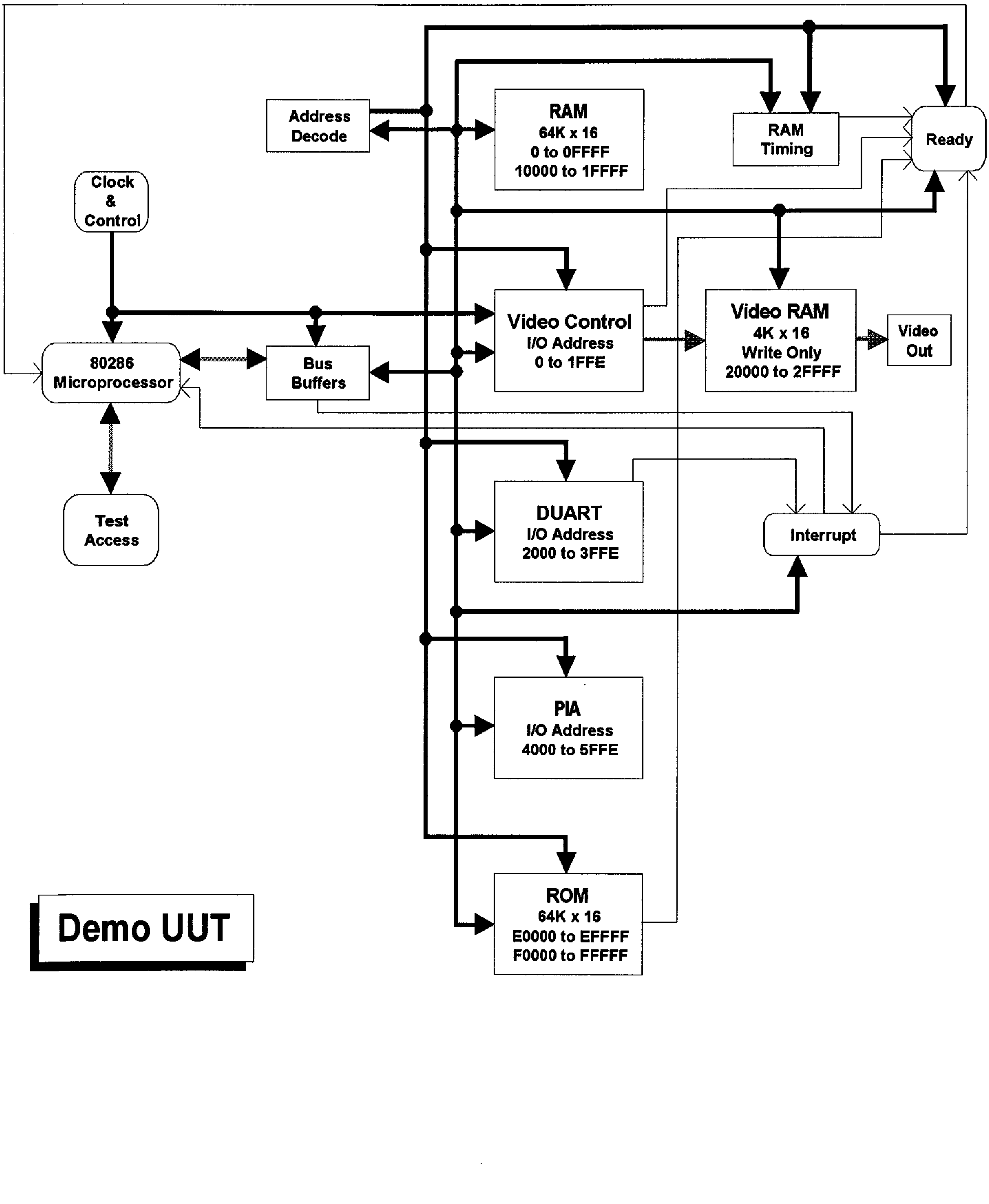
If the UUT is changed to another type.

- ✓ A different Pod may be required. If so, self-test needs to be performed. Calibration needs to be performed or restored. Save new calibration data to the user disk, writing over the old data.
- ✓ System settings are most likely different for the new UUT. Setup or restore system settings for the new UUT to configure the system.

Section 3

Understanding the UUT

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.*

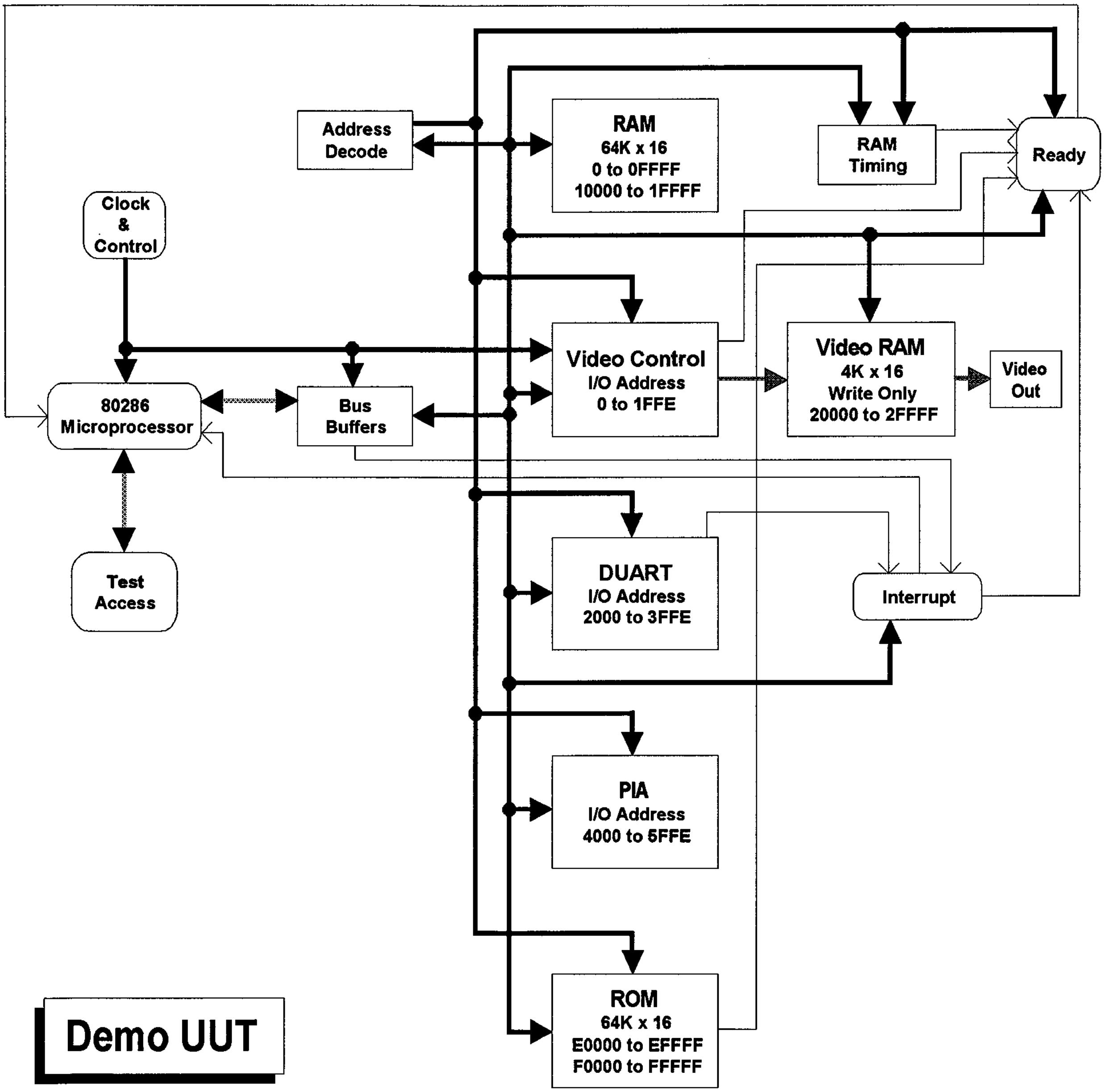

Address Decode
RAM
64K x 16
0 to 0FFFF
10000 to 1FFFF
RAM Timing
Ready
Clock & Control
80286 Microprocessor
Bus Buffers
Video Control
I/O Address
0 to 1FFE
Video RAM
4K x 16
Write Only
20000 to 2FFFF
Video Out
Test Access
DUART
I/O Address
2000 to 3FFE
Interrupt
PIA
I/O Address
4000 to 5FFE
ROM
64K x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

Demo UUT

GUIDELINES TO UNDERSTANDING

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.


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

Demo UUT

DEMO UUT

Exercise 3-1

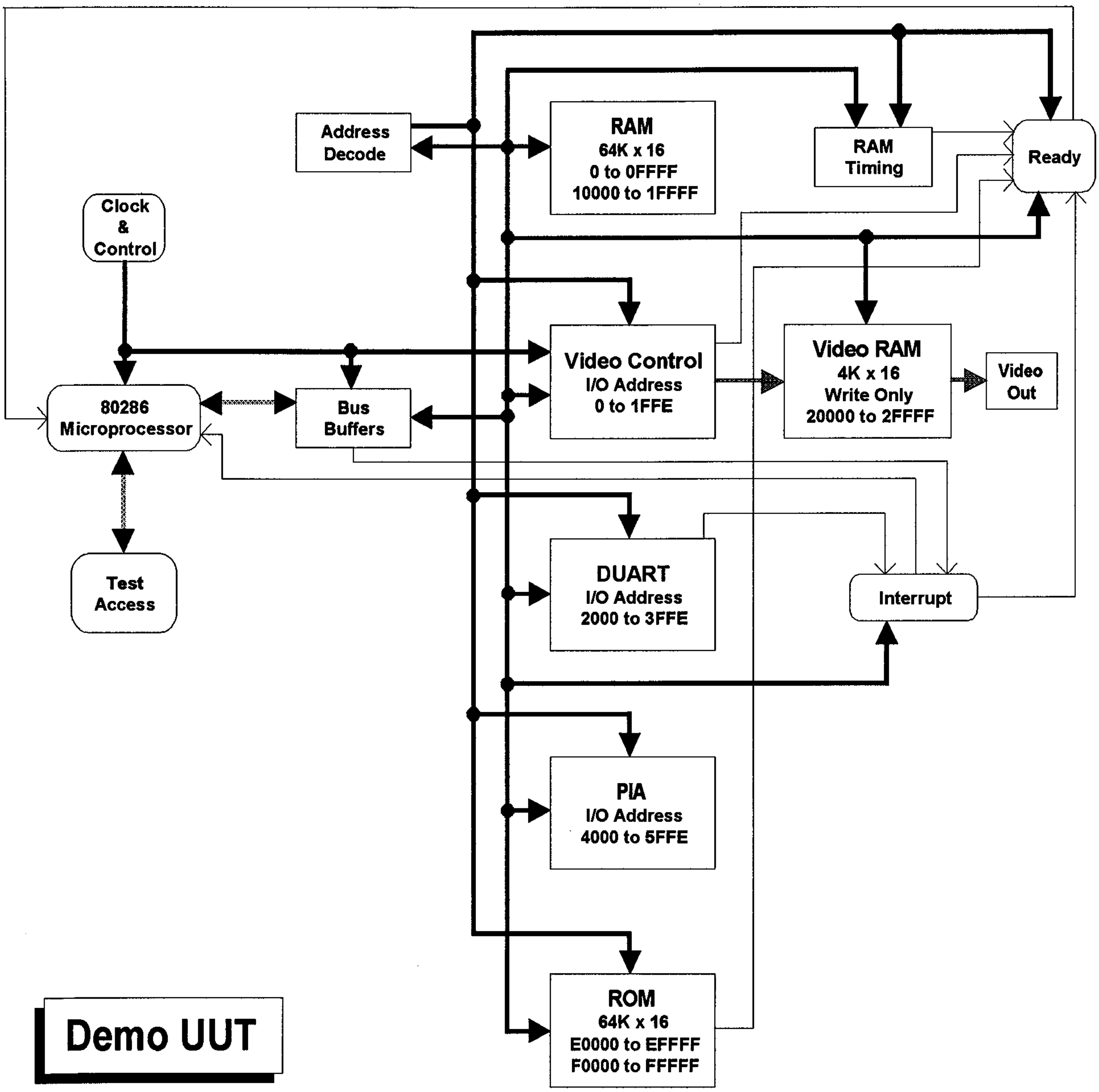
DEMO UUT - EXERCISE 3-1

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 x 16
0 to 0FFFF
10000 to 1FFFF
RAM
Timing
Ready
Clock
&
Control
Video Control
I/O Address
0 to 1FFE
Video RAM
4K x 16
Write Only
20000 to 2FFFF
Video
Out
80286
Microprocessor
Bus
Buffers
Test
Access
DUART
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

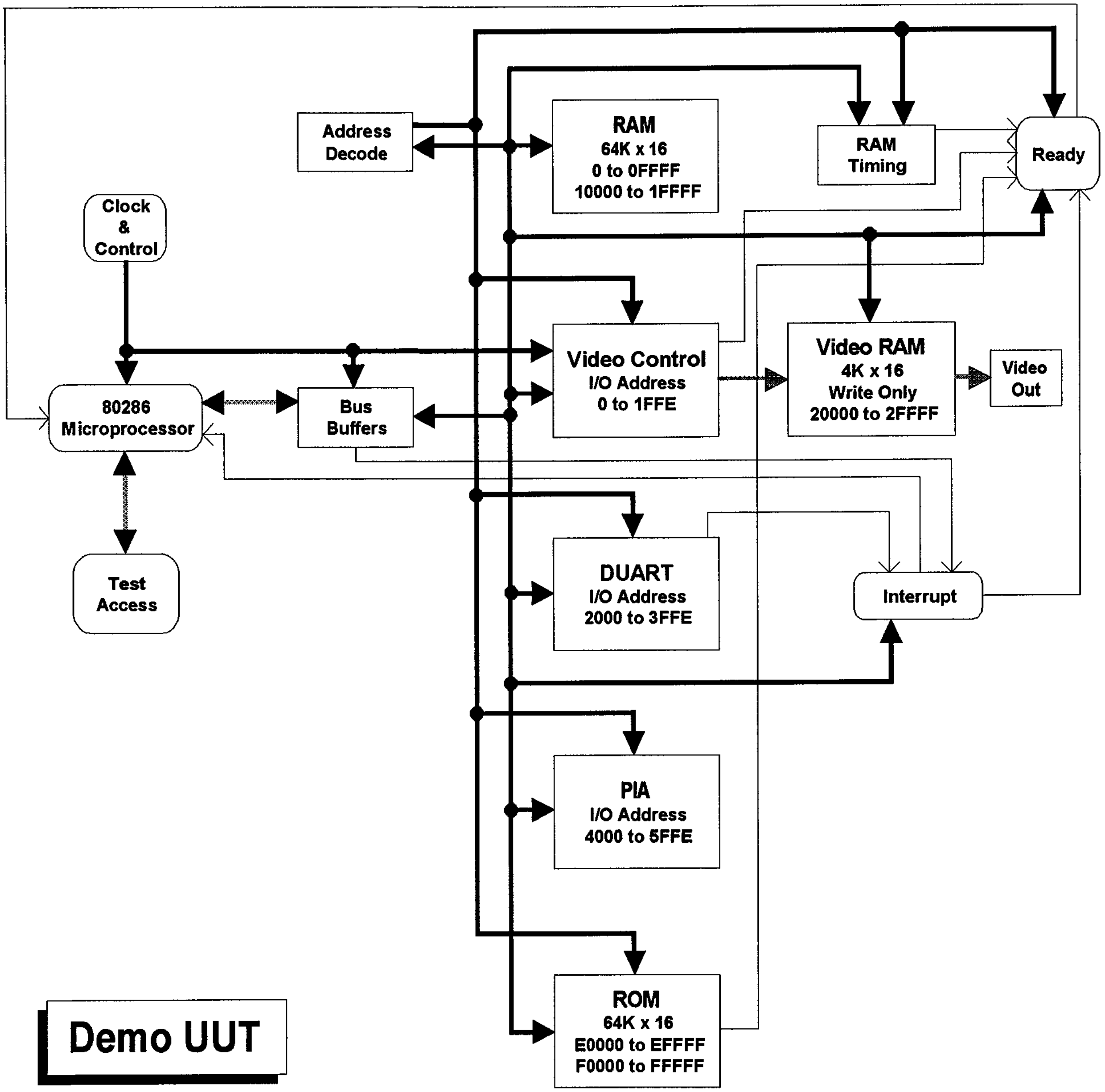
PARTITIONING

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 x 16
0 to 0FFFF
10000 to 1FFFF
RAM Timing
Ready
Clock & Control
Video Control
I/O Address
0 to 1FFE
Video RAM
4K x 16
Write Only
20000 to 2FFFF
Video Out
80286 Microprocessor
Bus Buffers
DUART
I/O Address
2000 to 3FFE
Interrupt
Test Access
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:

① Stimulate the inputs to the partitioned circuit.

② Measure the output of the partitioned circuit.

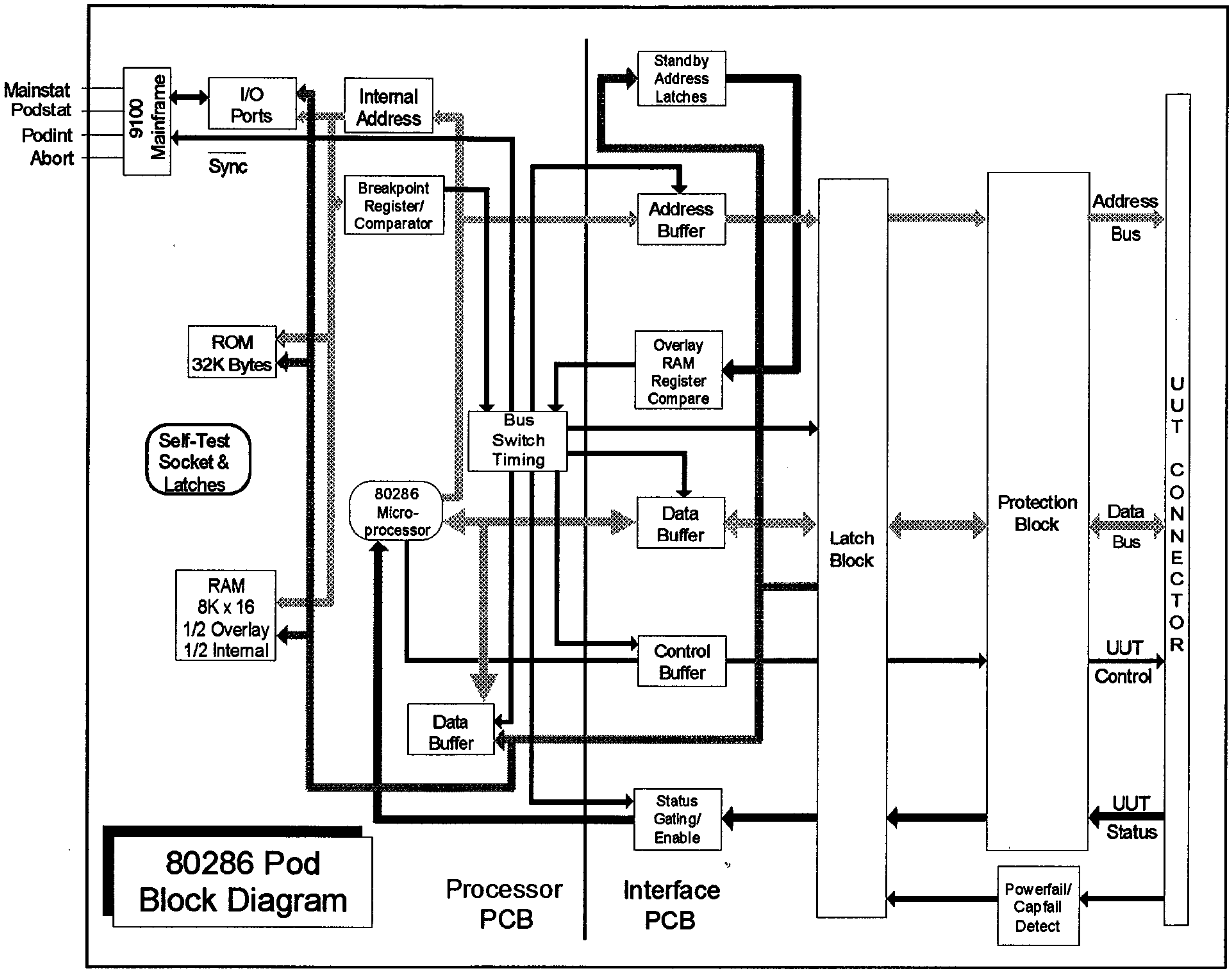
③ Evaluate each output of the partitioned circuit and decide whether it passes or fails.

④ If all outputs passed, declare the partitioned circuit to be good, otherwise declare that the partitioned circuit has failed.

Section 4

Bus Test

The 9100FT *simplifies the test of microprocessor buses through its built-in Bus Test. The built-in Bus Test has a number of test functions that provide quick analysis of the bus and reduce operator interaction to just a few button presses.*


Mainstat
Podstat
Podint
Abort
9100 Mainframe
I/O Ports
Sync
Internal Address
Breakpoint Register/ Comparator
ROM 32K Bytes
Self-Test Socket & Latches
80286 Micro-processor
RAM 8K x 16 1/2 Overlay 1/2 Internal
Bus Switch Timing
Data Buffer
Standby Address Latches
Address Buffer
Overlay RAM Register Compare
Data Buffer
Control Buffer
Status Gating/ Enable
Latch Block
Protection Block
Powerfail/ Capfail Detect
Address Bus
Data Bus
UUT Control
UUT Status
UUT CONNECTOR
80286 Pod Block Diagram
Processor PCB
Interface PCB

BUILT-IN BUS TEST

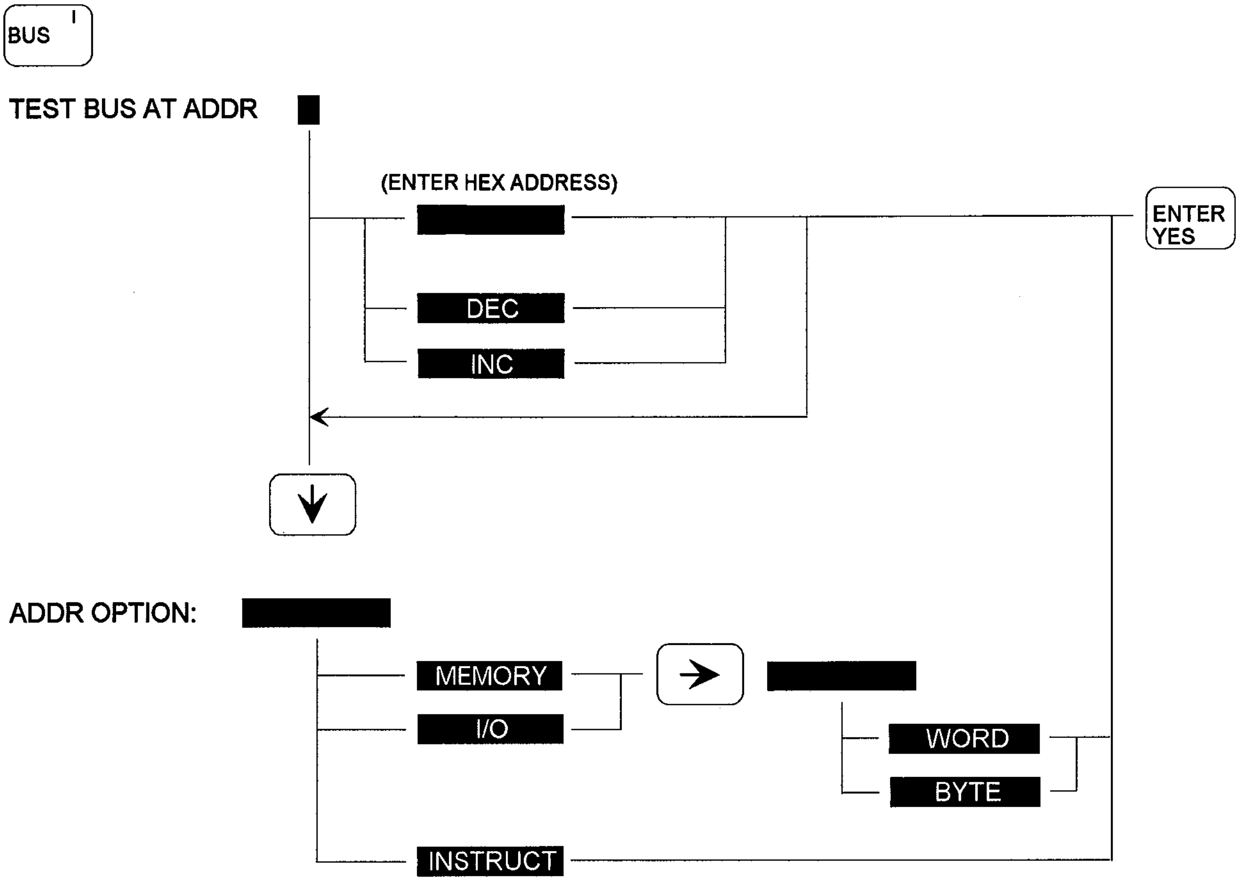
The Bus Test checks for **stuck data, address** or **control lines.** It does this by driving each line high and low and measuring the level that is actually present on the line. The measurement is done by level detectors or latches that are built into each pod on each of the drivable lines. When the 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 supposed to be on the line and an error is reported if a difference exists.

Not only is the checking of line 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 enables the operator to check for stuck and tied lines.

The interface pod cannot detect open bus lines since all the level detect circuitry is internal to the pod.

As we will see later, other built-in tests can be used to check for open bus lines.


BUS
TEST BUS AT ADDR
(ENTER HEX ADDRESS)
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 - EXERCISE 4-1

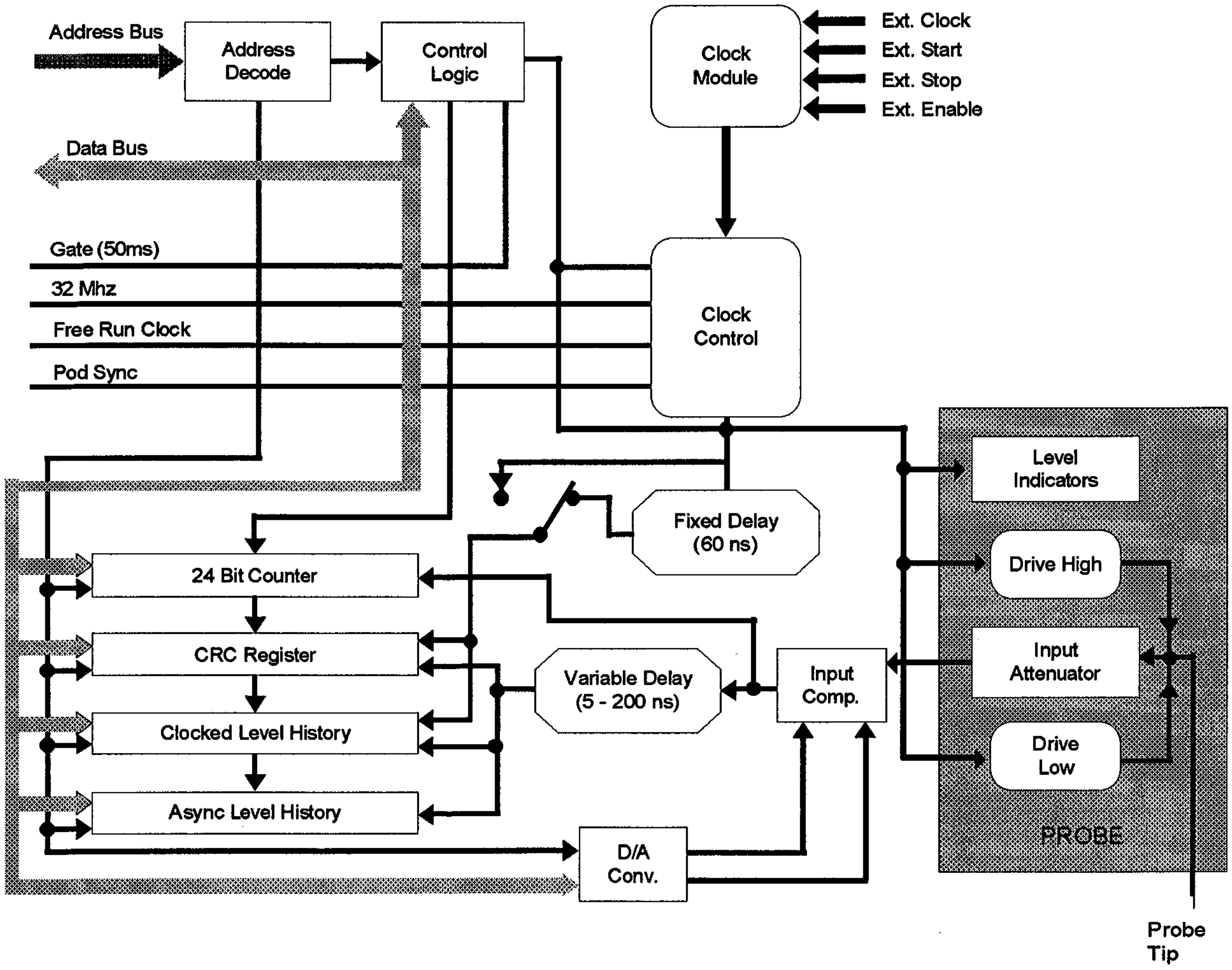
Press: [BUS] [0 0000] [ENTER YES]

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 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.

9100FT Probe System



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 3.2M	-1.35 to -1.15V	1 or X
1.0 to 2.2M	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
0.0 to 0.6V	0.0 to 0.8V	-30.0 to -3.2V	-30.0 to -1.55V	0

Input Impedance

70K ohm shunted by less than 33pF

Data Timing For Synchronous Measurements

Maximum Frequency:	40 MHz
Minimum Pulse Width (H, L):	12.5 nsec
Minimum Pulse Width (3-state):	20 msec

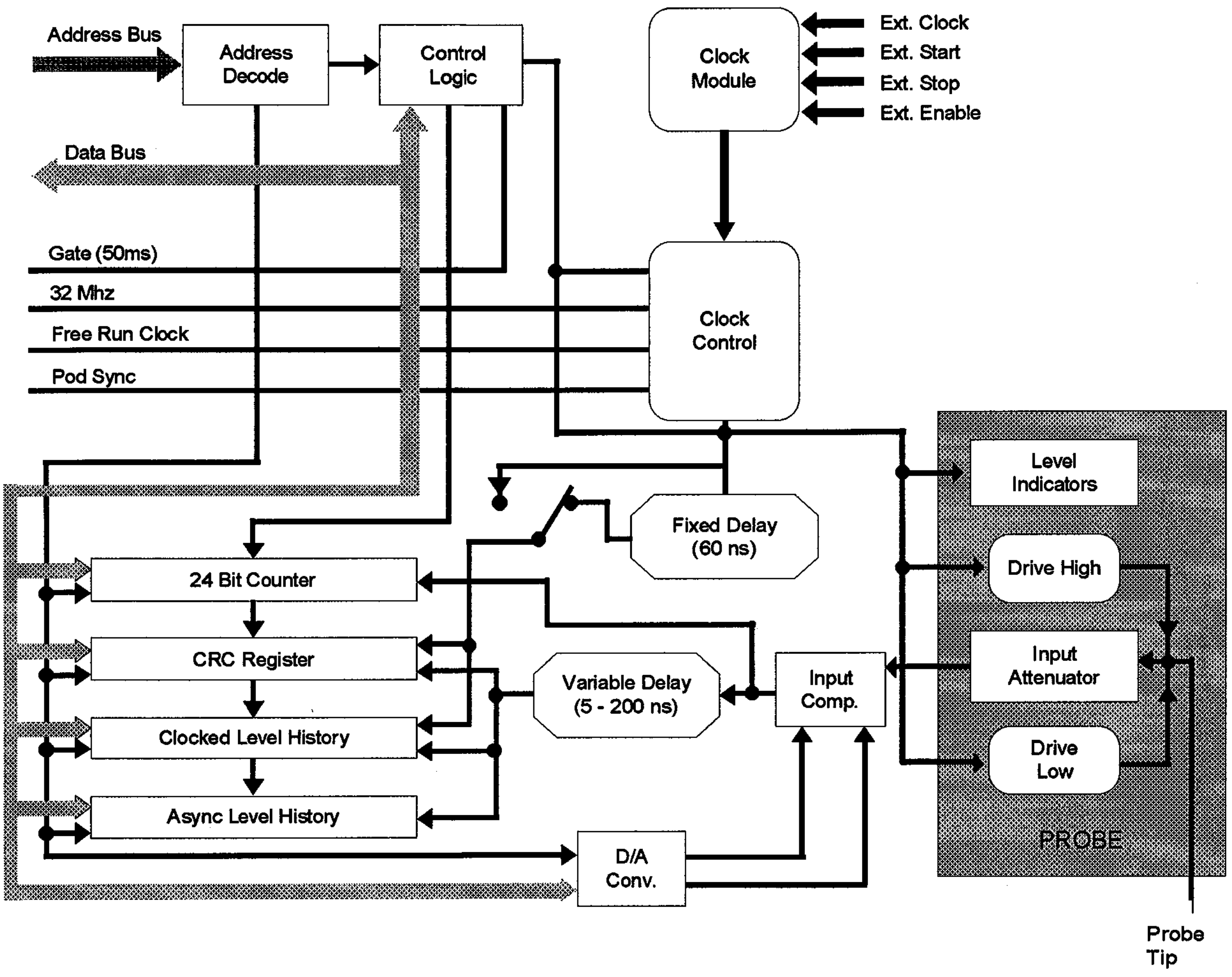
Setup Times

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

9100FT Probe System



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

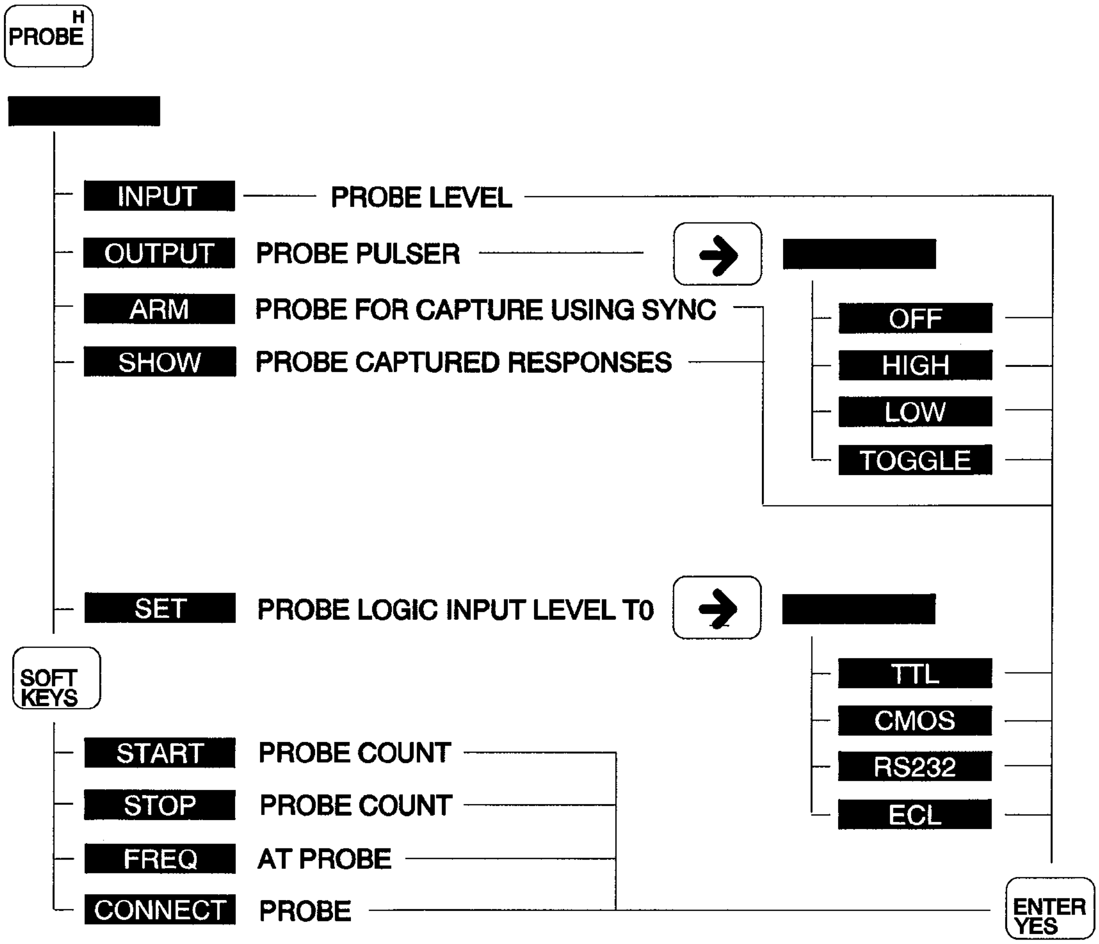
Maximum Frequency:	10 MHz
Maximum Count	16777216 + overflow
Maximum Stop Count:	65536 clocks

Frequency Measurement

Maximum Frequency:	40 MHz
Resolution:	20 Hz
Accuracy:	± 250 ppm ± 20 Hz

Output Pulser

High:	**>3.5V @ 200mA** for less than 10 µsec @ 1% duty cycle **>4.0V @ 5 mA** continuously
Low:	**<0.8V @ 200mA** for less than 10 µsec @ 1% duty cycle **<0.4V @ 5mA** continuously


H
PROBE
INPUT
PROBE LEVEL
OUTPUT
PROBE PULSER
ARM
PROBE FOR CAPTURE USING SYNC
SHOW
PROBE CAPTURED RESPONSES
OFF
HIGH
LOW
TOGGLE
SET
PROBE LOGIC INPUT LEVEL T0
TTL
CMOS
RS232
ECL
SOFT KEYS
START
PROBE COUNT
STOP
PROBE COUNT
FREQ
AT PROBE
CONNECT
PROBE
ENTER YES

Exercise 4-2

PROBE INPUT SYNC FREERUN - EXERCISE 4-2

This exercise illustrates the PROBE INPUT function, with the probe synchronized to freerun.

Two pieces of information are provided when this function is selected:

- ✓ Input Probe Level (current)
- ✓ Level History (since last input)

① To begin the exercise, sync the Probe to freerun.

Press the following sequence of keys:

PROBE ...
F1 INPUT ...
ENTER YES

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

Press the following sequence of keys:

PROBE ...
F1 INPUT ...
ENTER YES

The following results are displayed:

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

This page intentionally left blank.

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
```

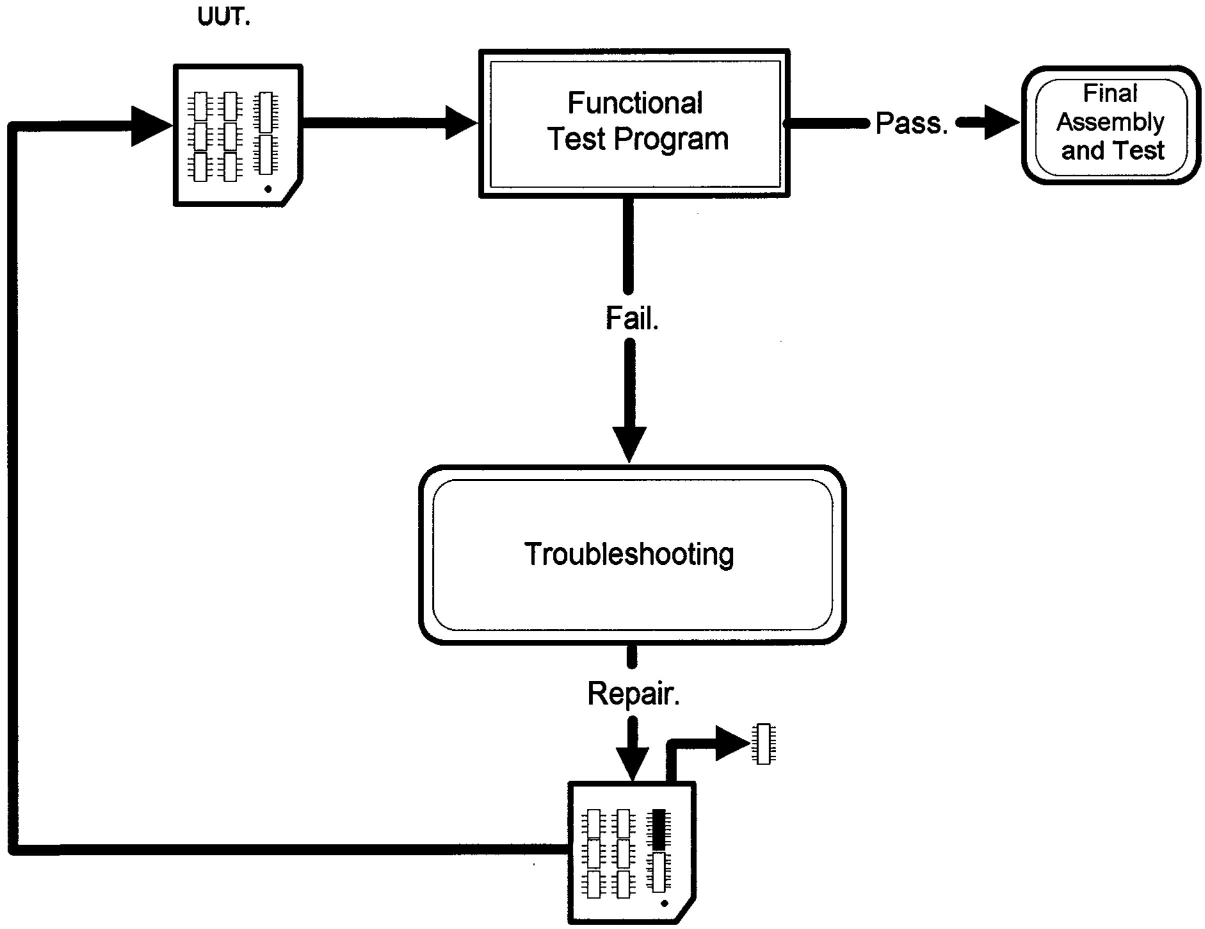
The current level is 1 (high), but note the level history.

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 5a.


UUT.
Functional Test Program
Pass.
Final Assembly and Test
Fail.
Troubleshooting
Repair.

GENERAL TEST AND TROUBLESHOOTING FLOW

The 9100-SERIES of instruments have two modes of operation:

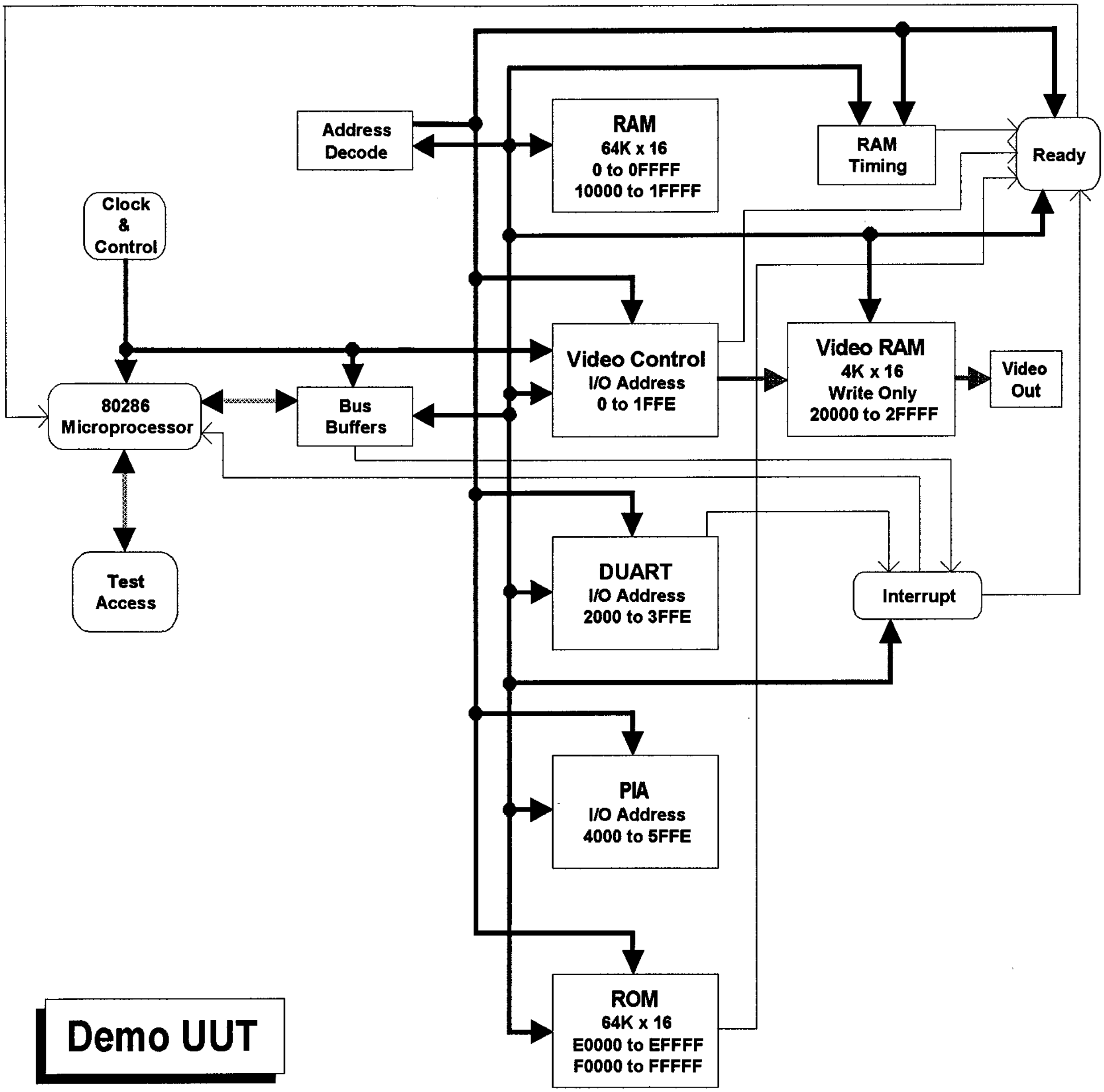
- ✓ FUNCTIONAL TEST
- ✓ FAULT ISOLATION

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.

In fault isolation, the goal is to determine which components are bad or which nodes are 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. Based on how that response compares with what we know to be a good response, a decision must be made.

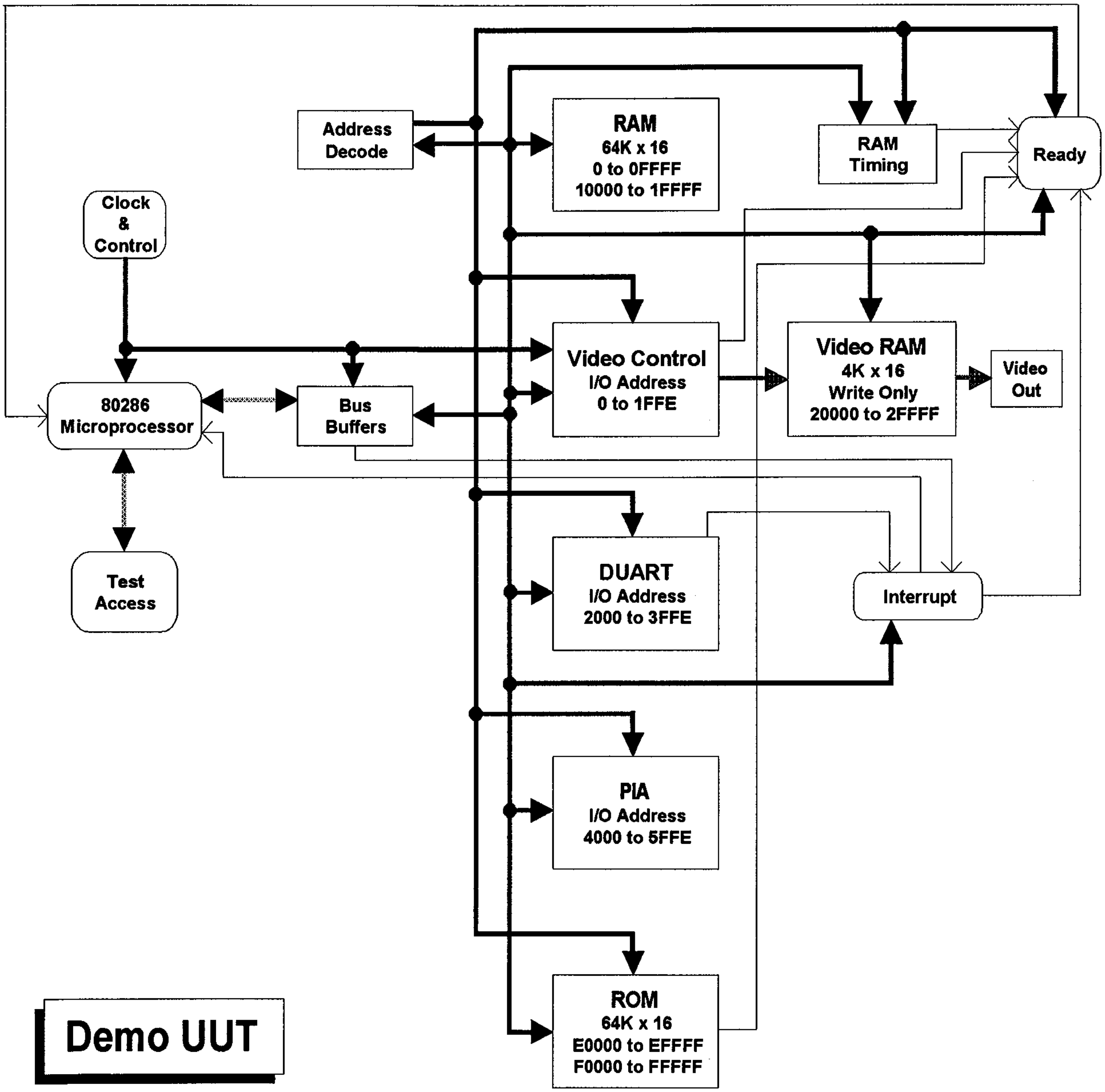

Address Decode
RAM
64K x 16
0 to 0FFFF
10000 to 1FFFF
RAM Timing
Ready
Clock & Control
80286 Microprocessor
Bus Buffers
Video Control
I/O Address
0 to 1FFE
Video RAM
4K x 16
Write Only
20000 to 2FFFF
Video Out
Test Access
DUART
I/O Address
2000 to 3FFE
Interrupt
PIA
I/O Address
4000 to 5FFE
ROM
64K x 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 identified, determine to what level each of these lines must be driven so that 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.


Address Decode
RAM
64K x 16
0 to 0FFFF
10000 to 1FFFF
RAM Timing
Ready
Clock & Control
80286 Microprocessor
Bus Buffers
Video Control
I/O Address
0 to 1FFE
Video RAM
4K x 16
Write Only
20000 to 2FFFF
Video Out
Test Access
DUART
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

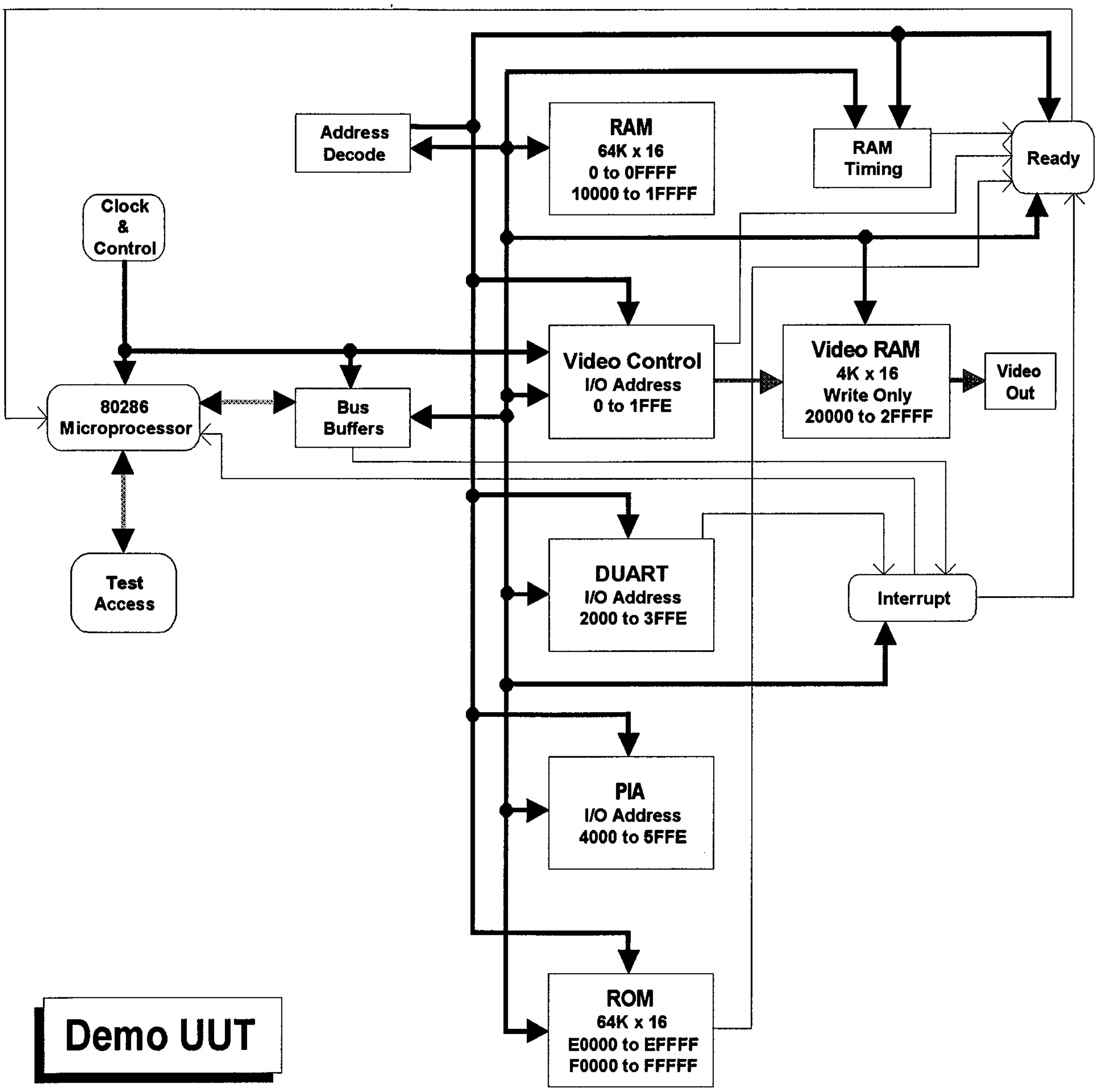
The Measurement

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 setups have been established, look at the node in question.

Does the node require any specific circuitry setups?

This becomes especially important on circuitry controlled by programmable devices.


Address Decode
RAM
64K x 16
0 to 0FFFF
10000 to 1FFFF
RAM Timing
Ready
Clock & Control
Video Control
I/O Address
0 to 1FFE
Video RAM
4K x 16
Write Only
20000 to 2FFFF
Video Out
80286 Microprocessor
Bus Buffers
Test Access
DUART
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

The Response

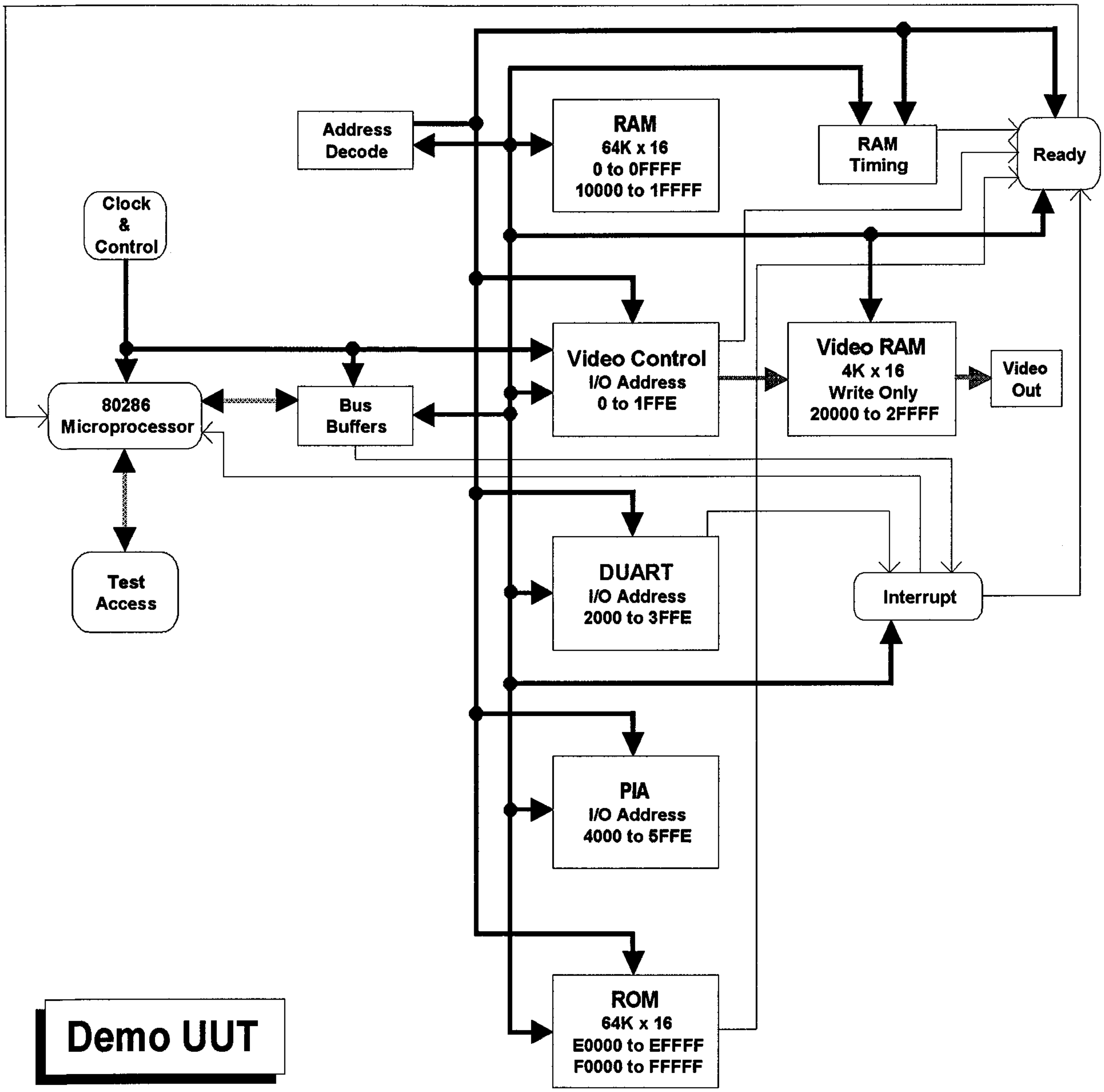
The last step in testing a node when performing fault isolation is to measure the response. The microprocessor must be capable of driving the node's associated input lines. The stimulus will then drive the node so that all problems that could occur on the node will be identified.

After the results have been gathered, the troubleshooter performs 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 testing and fault isolation using the immediate mode of operation.

Each exercise will introduce a different stimulus and measurement capability of your test equipment. This will empower you to develop test and fault isolation strategies for your own UUT.


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

④ Fault Message: (if there is one).

____

____

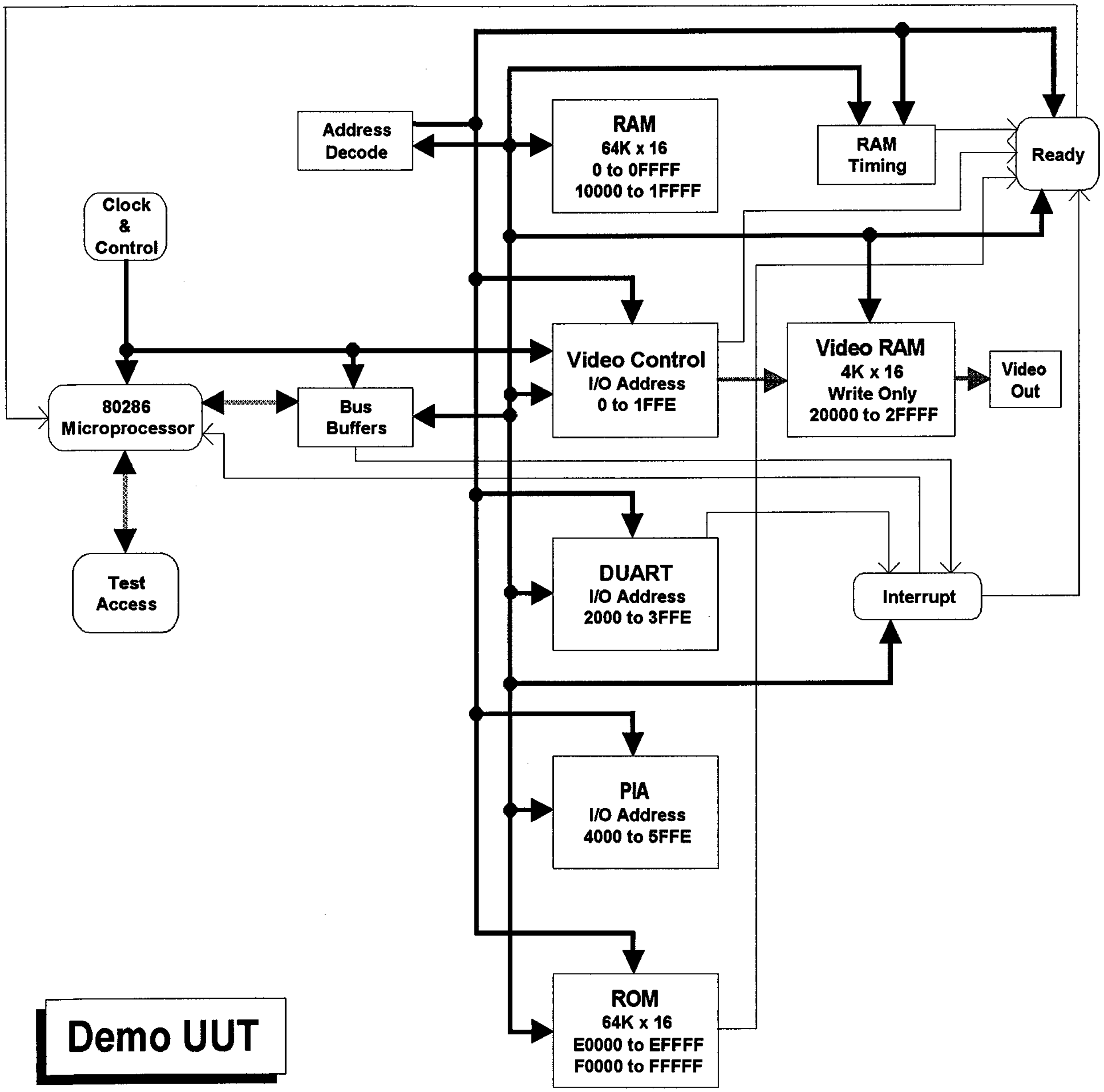
____

⑤ Fault Isolation:

STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.


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

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

④ Fault Message: (if there is one).

____

____

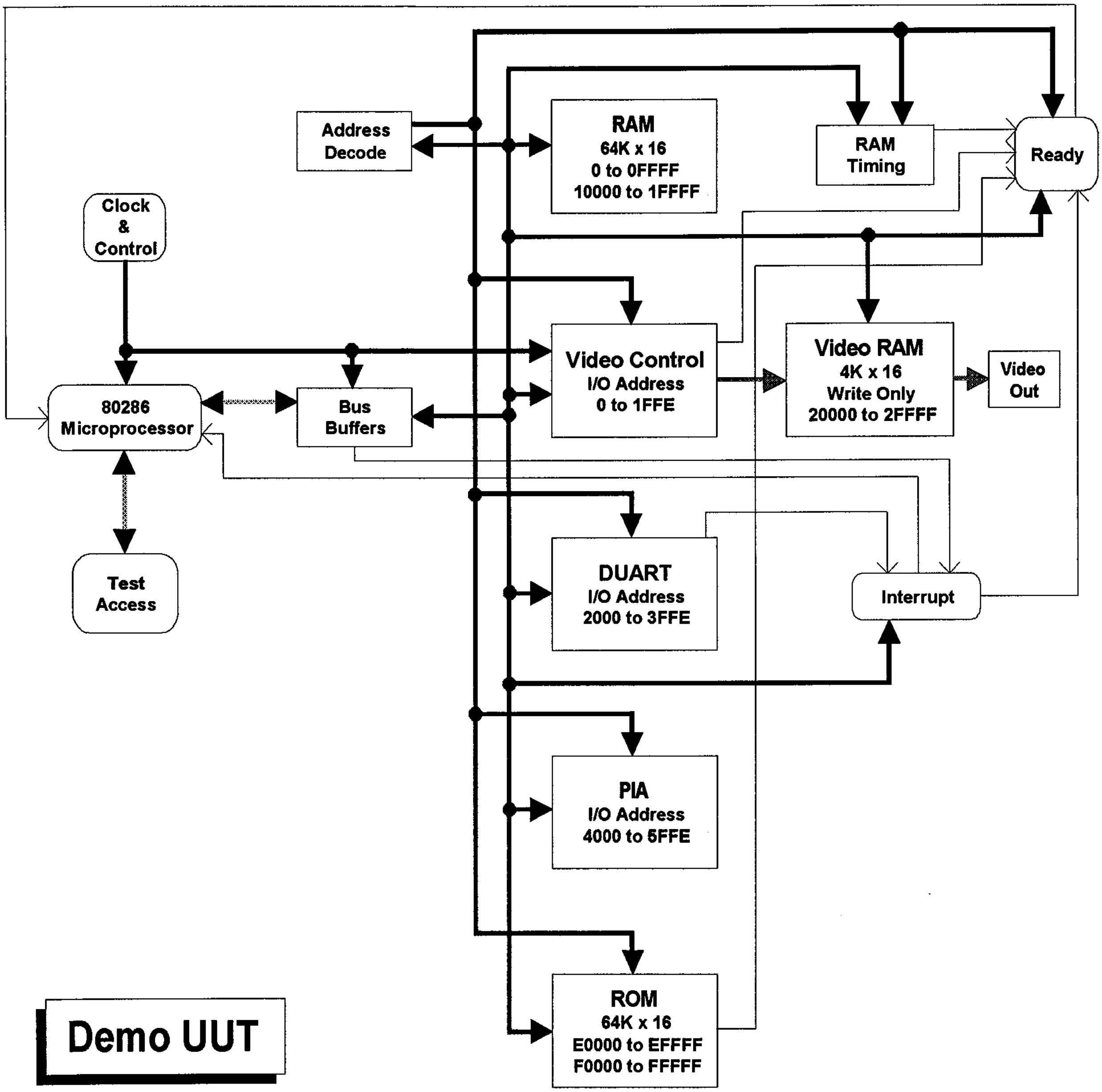
____

⑤ Fault Isolation:

STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.


Address
Decode
RAM
64K x 16
0 to 0FFFF
10000 to 1FFFF
RAM
Timing
Ready
Clock
&
Control
Video Control
I/O Address
0 to 1FFE
Video RAM
4K x 16
Write Only
20000 to 2FFFF
Video
Out
80286
Microprocessor
Bus
Buffers
Test
Access
DUART
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

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).

____

____

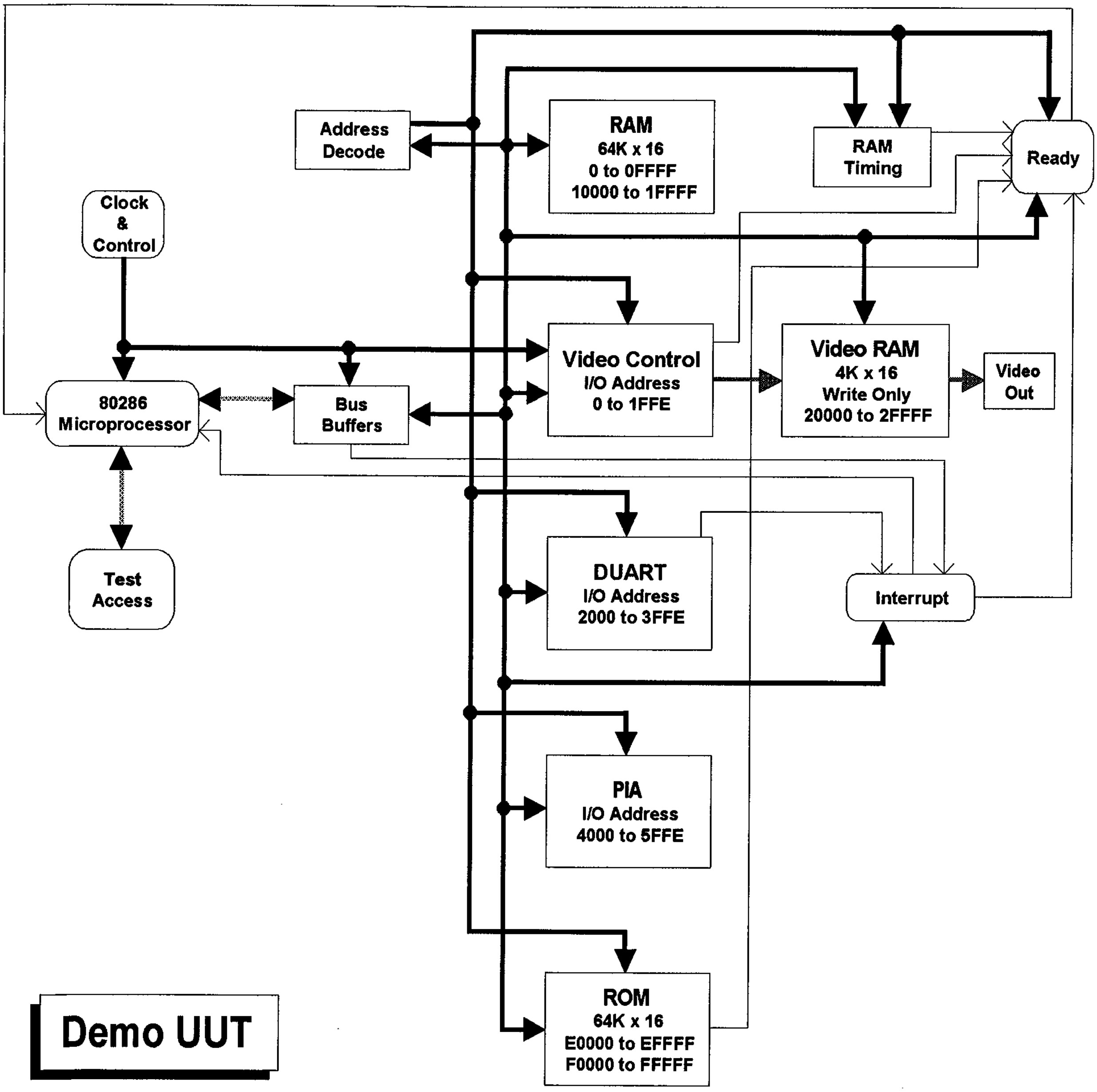
____

⑤ Fault Isolation:

STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.


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

Demo UUT

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:

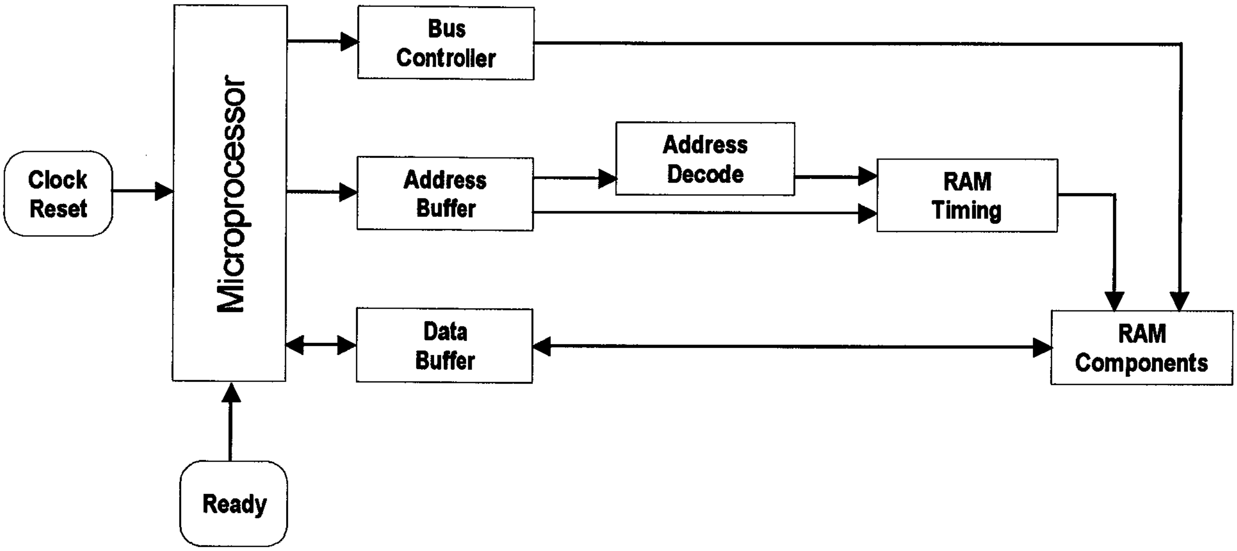
STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.

Section 5

RAM Testing


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

RAM FUNCTIONAL BLOCK

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:

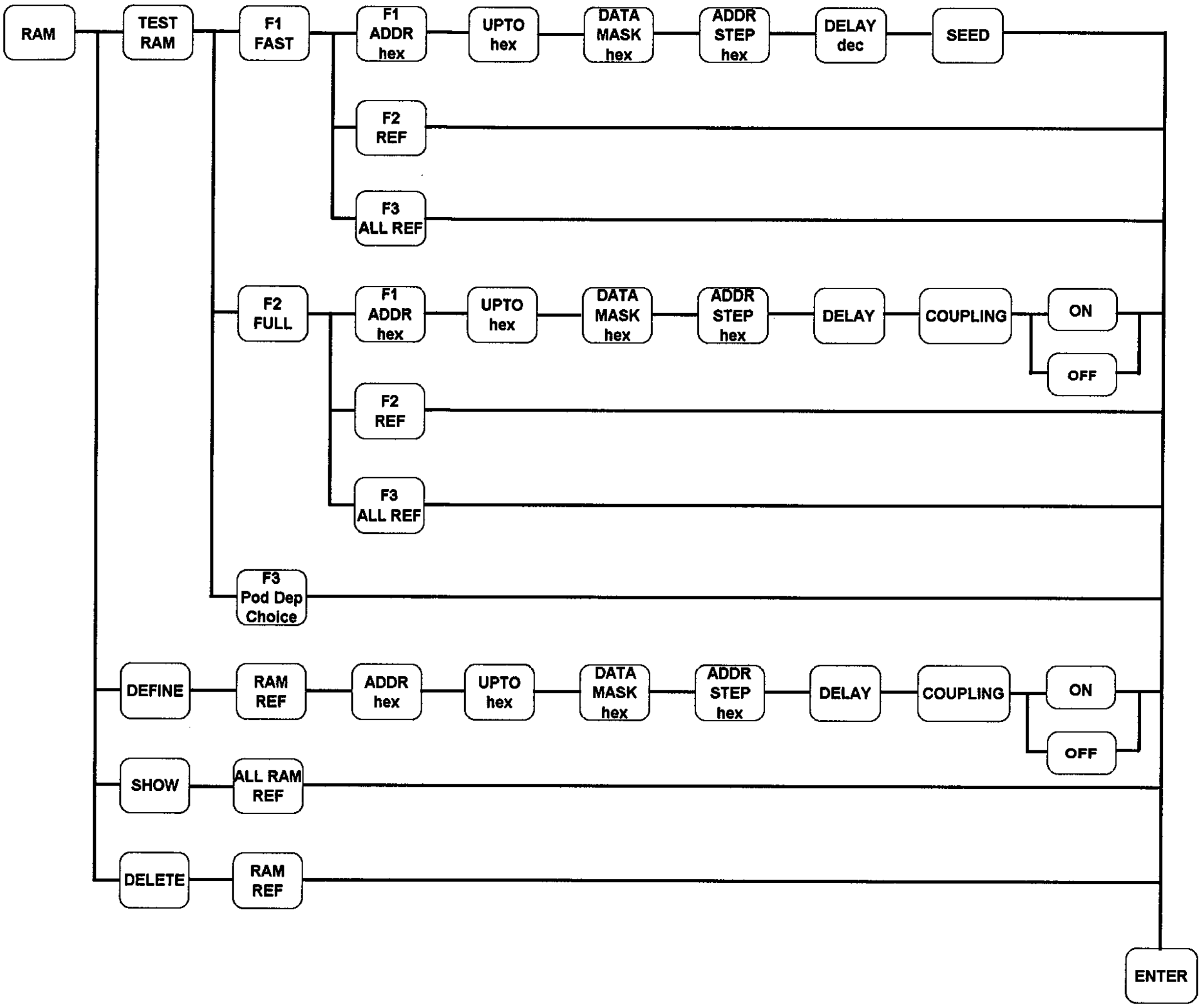
- ✓ the higher order byte
- ✓ lower order byte
- ✓ the whole word.

A refresh cycle occurs every 16 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
F1 FAST
F1 ADDR hex
UPTO hex
DATA MASK hex
ADDR STEP hex
DELAY dec
SEED
F2 REF
F3 ALL REF
F2 FULL
F1 ADDR hex
UPTO hex
DATA MASK hex
ADDR STEP hex
DELAY
COUPLING
ON
OFF
F2 REF
F3 ALL REF
F3 Pod Dep Choice
DEFINE
RAM REF
ADDR hex
UPTO hex
DATA MASK hex
ADDR STEP hex
DELAY
COUPLING
ON
OFF
SHOW
ALL RAM REF
DELETE
RAM REF
ENTER

USING THE RAM TEST

We will 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 read or writeable. RAM Fast functionally tests all the components and support circuitry that are essential for proper RAM operation. The RAM test allows you to identify each chip by name, address range, and other parameters.

RAM F*ast 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.

Fault Condition	Testramfast	Testramfull	
		Coupling enabled	Coupling disabled
Stuck Cells	Always found		
Aliased Cells			
Stuck Address Lines			
Stuck Data Lines			
Shorted Address Lines			
Multiple Selection Decoder	Maybe found	Always found	
Dynamic Coupling			
Shorted Data Lines		Always found	Maybe found
Aliasing Between Bits (in same word)			
Pattern-sensitive Faults		Not found	
Refresh Problems	Always found if delay is sufficiently long and standby reads do not mask the problem.		

Exercise 5-1

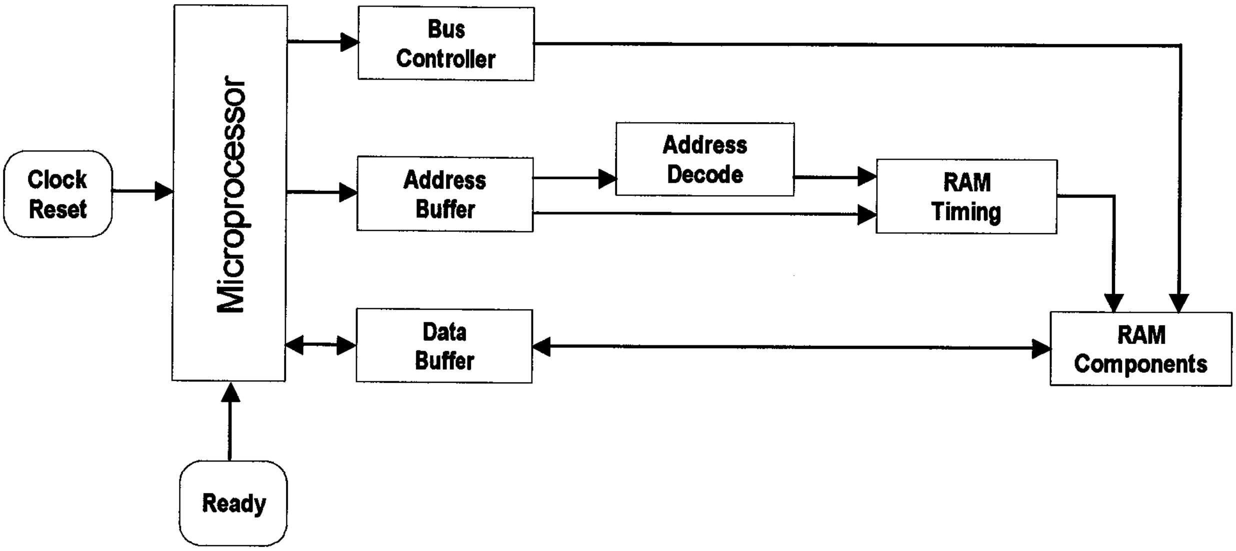
PERFORMING A RAM FAST TEST - EXERCISE 5-1

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.


Bus
Controller
Microprocessor
Clock
Reset
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. The value of the seed, however, may change the first error reported by the Mainframe (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 - EXERCISE 5-2

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 ______

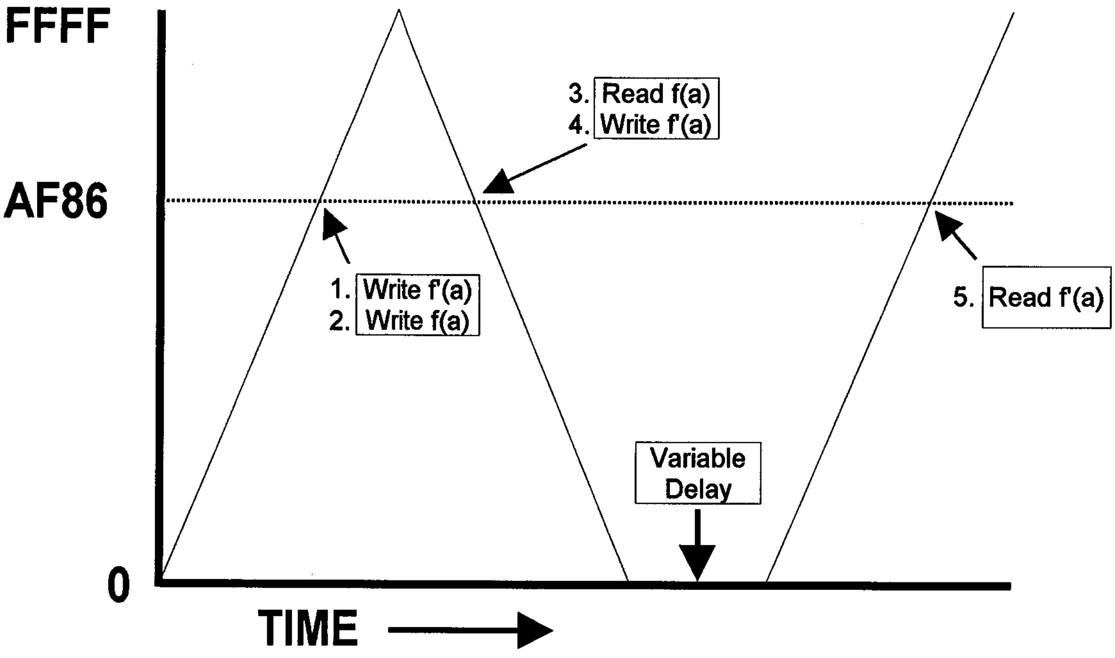
RAM FAST 0-FE MASK FFFF STEP 2 DELAY 250 SEED 1
READ ADDR AA ______

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
3. Read f(a)
4. Write f'(a)
1. Write f'(a)
2. Write f(a)
5. Read f'(a)
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 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 to the left (page 5-10). 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.

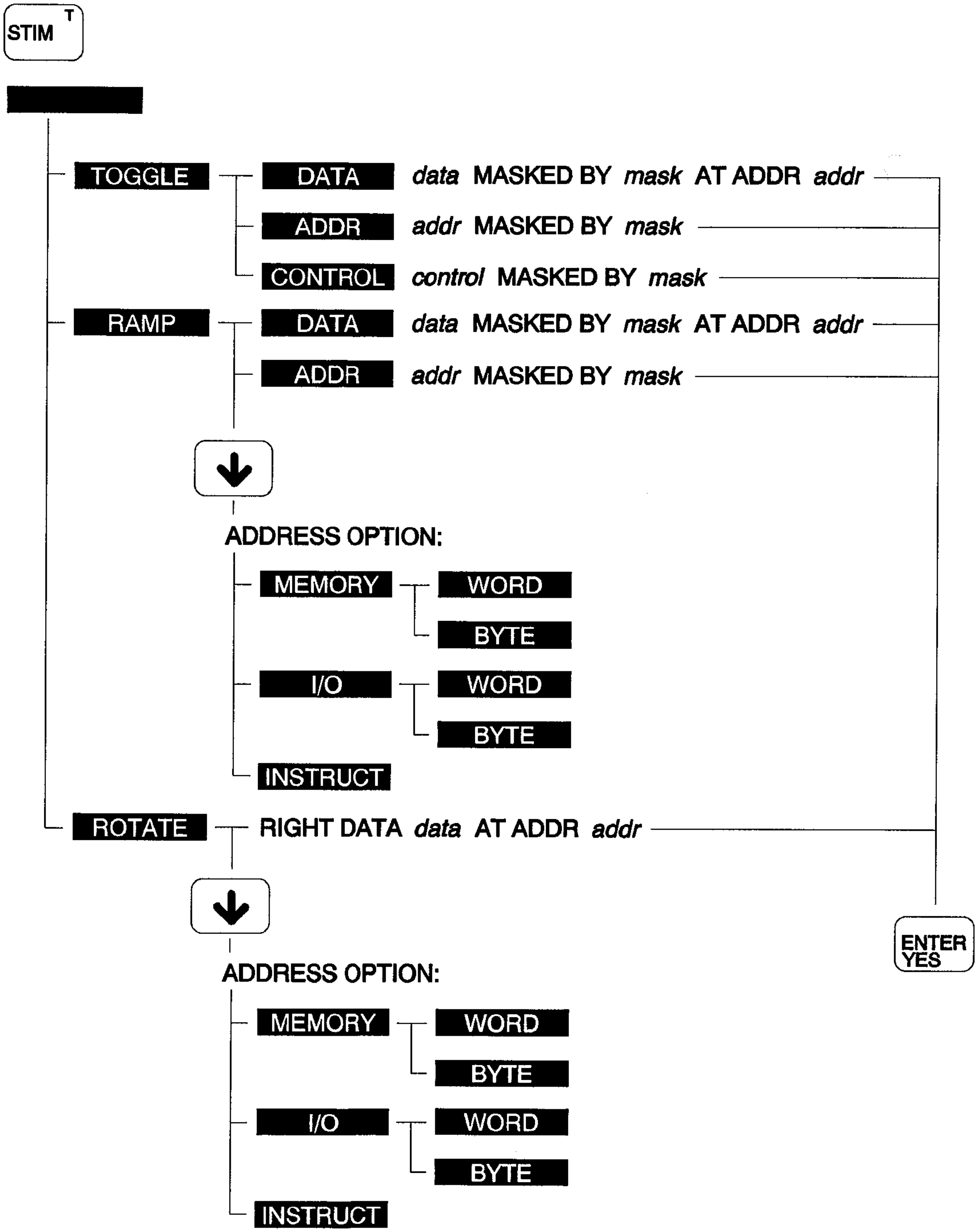
③ Access 1 - Write the compliment of D886 (2779).
Access 2 - Write D886.

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 T

- 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*
  - ↓ ADDRESS OPTION:
    - MEMORY
      - WORD
      - BYTE
    - I/O
      - WORD
      - BYTE
    - INSTRUCT
- ROTATE
  - RIGHT DATA *data* AT ADDR *addr*
  - ↓ ADDRESS OPTION:
    - MEMORY
      - WORD
      - BYTE
    - I/O
      - WORD
      - BYTE
    - INSTRUCT

ENTER YES

STIMULUS FUNCTIONS

The functions available from the stimulus menu are:

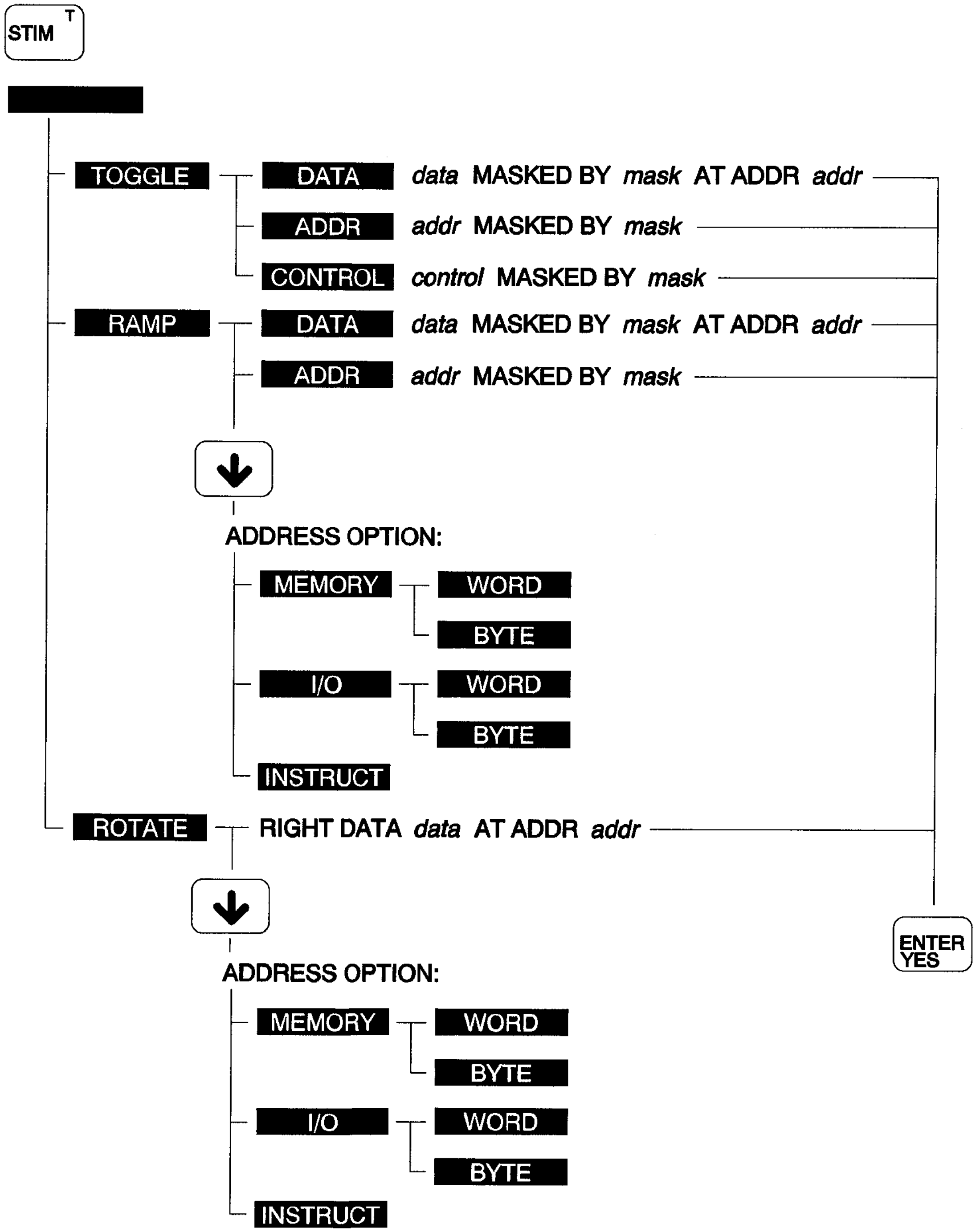
- ✓ RAMP
- ✓ TOGGLE
- ✓ ROTATE

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 T
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
ADDRESS OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ROTATE
RIGHT DATA data AT ADDR addr
ADDRESS 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:

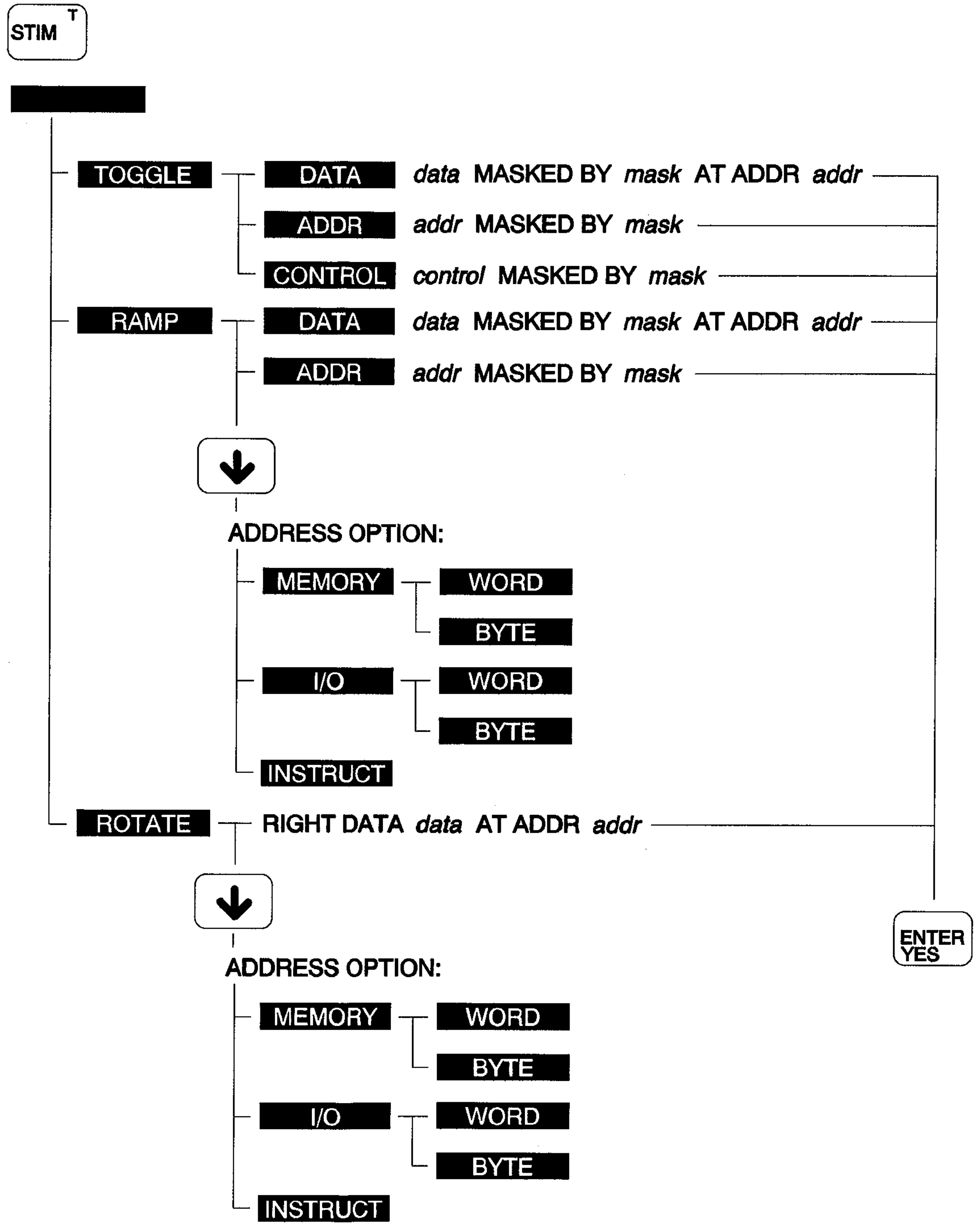
Write 80	1000 0000
Write 82	1000 0010
Write 84	1000 0100
Write 86	1000 0110

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 T
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
ADDRESS OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ROTATE
RIGHT DATA data AT ADDR addr
ADDRESS 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 range. 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
    Address = 1 1111 0000 0000 0000
```

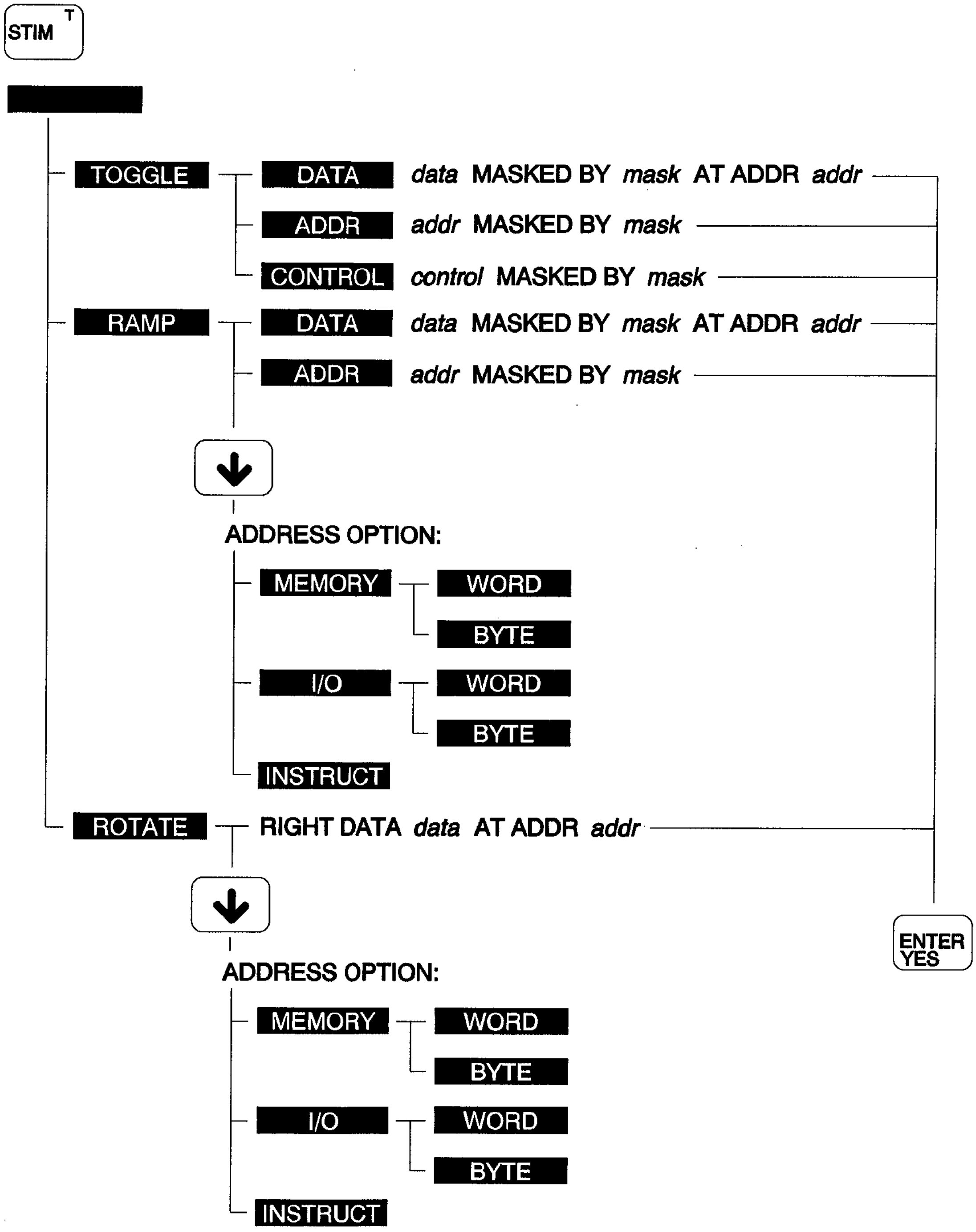
OPERATIONS PERFORMED:

```
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 T
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
↓
ADDRESS OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ROTATE
RIGHT DATA data AT ADDR addr
↓
ADDRESS OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
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)

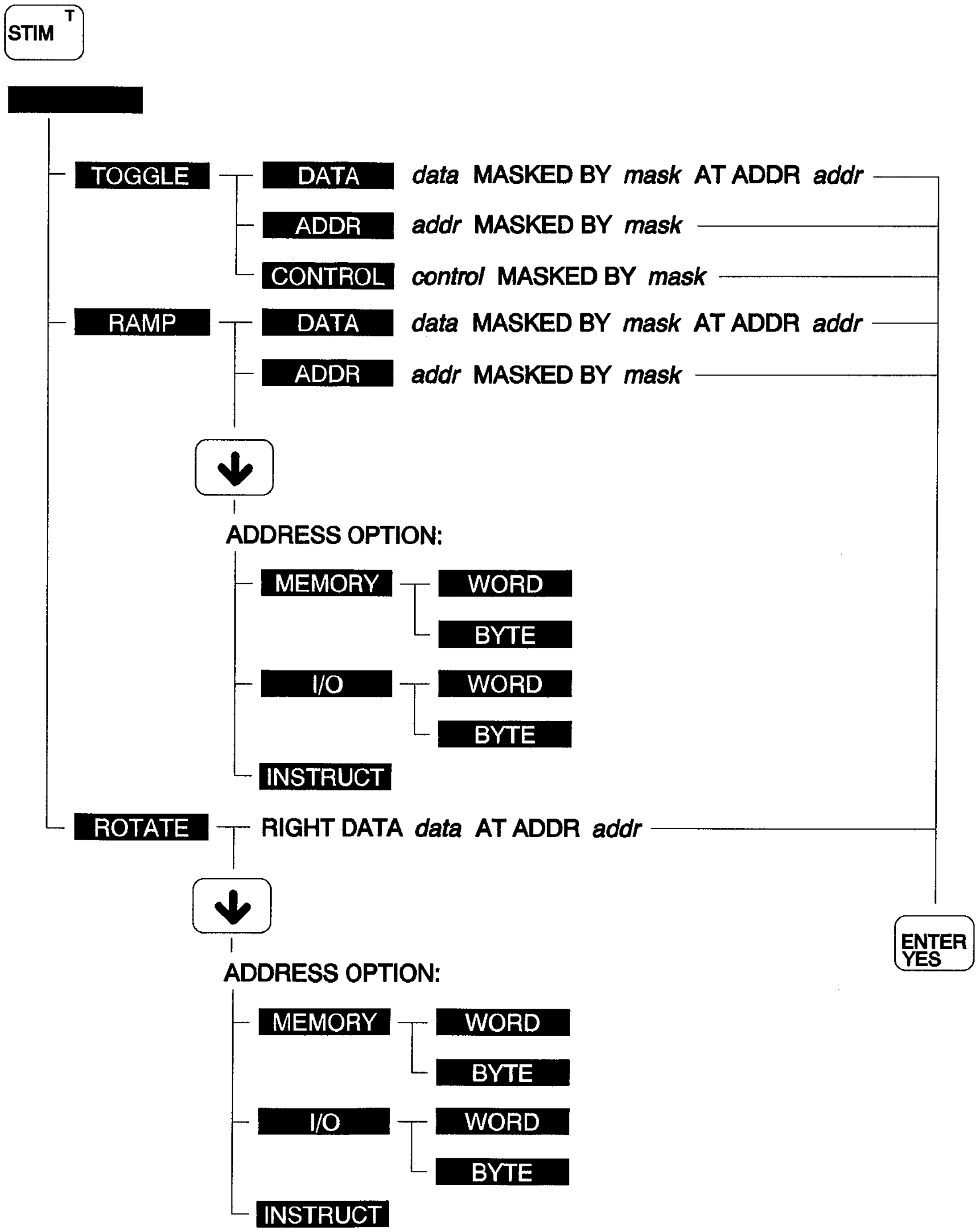
OPERATIONS:

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 T
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*
ADDRESS OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ROTATE
RIGHT DATA *data* AT ADDR *addr*
ADDRESS 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.

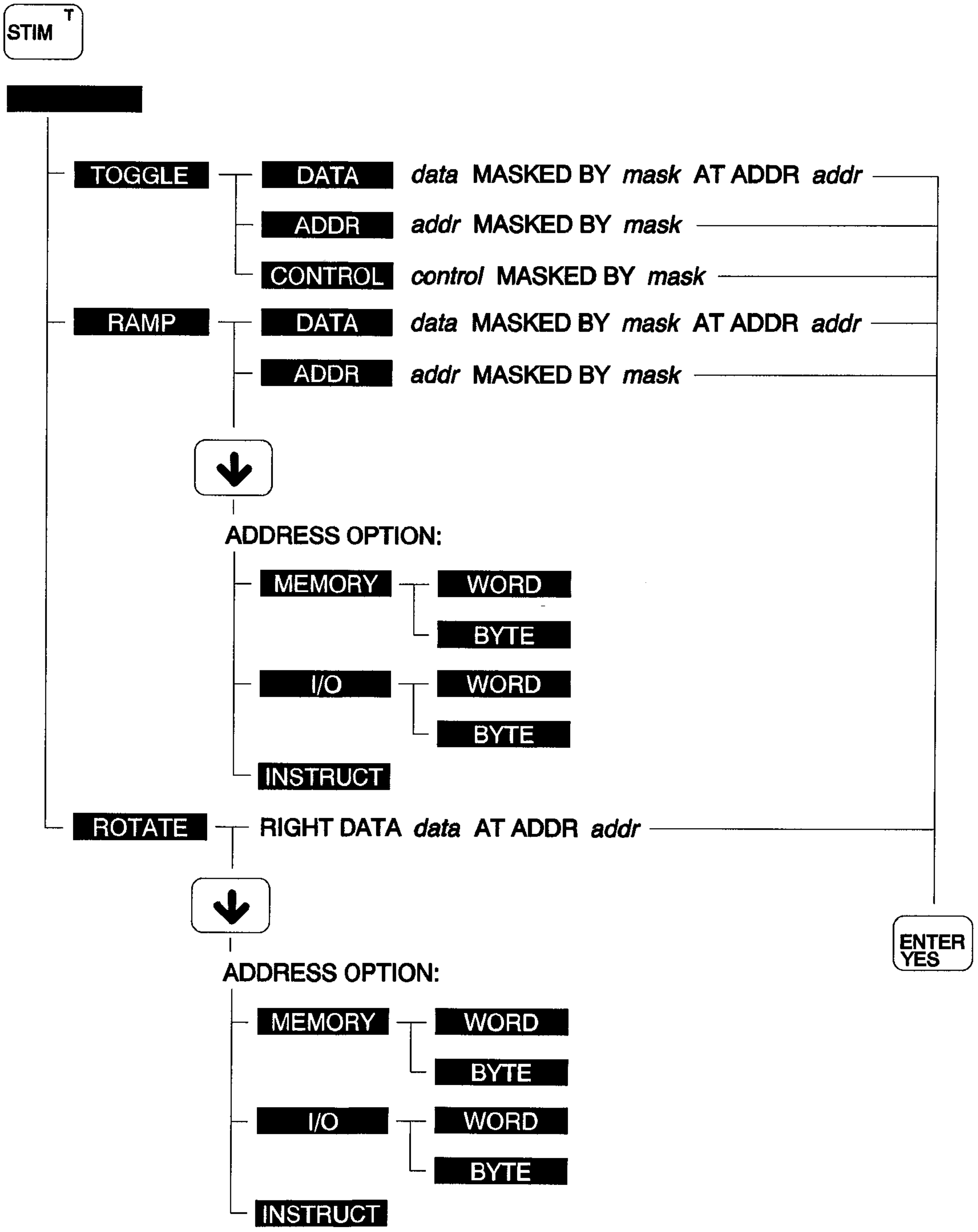
Example:

```
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 T
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
ADDRESS OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ROTATE
RIGHT DATA data AT ADDR addr
ADDRESS OPTION:
MEMORY
WORD
BYTE
I/O
WORD
BYTE
INSTRUCT
ENTER YES

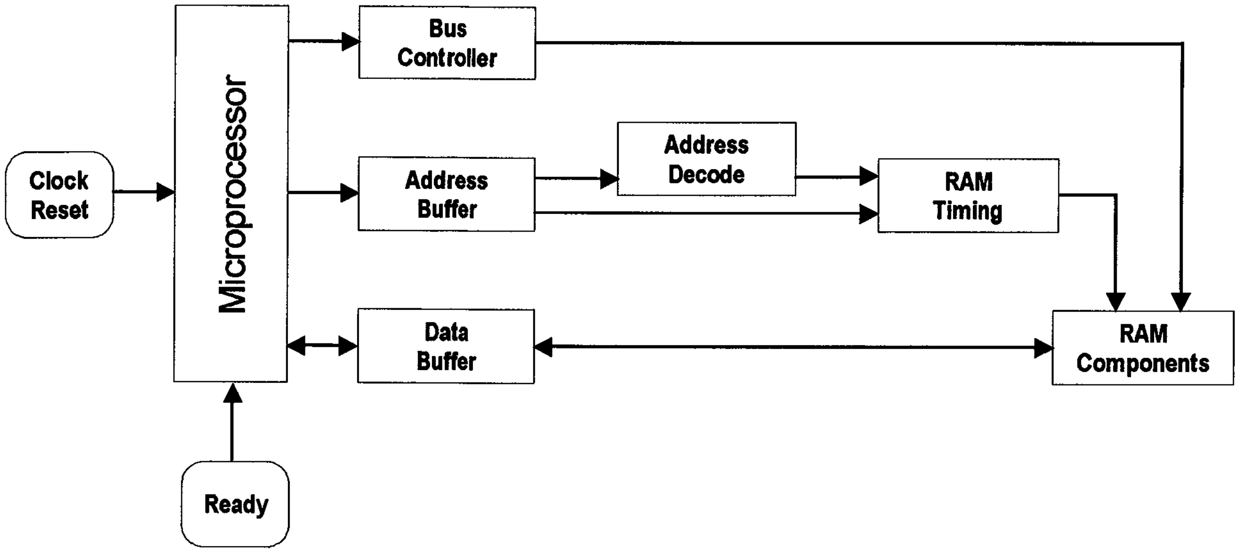
Toggle Control

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

To identify the control lines that are writeable 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.


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

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).

____

____

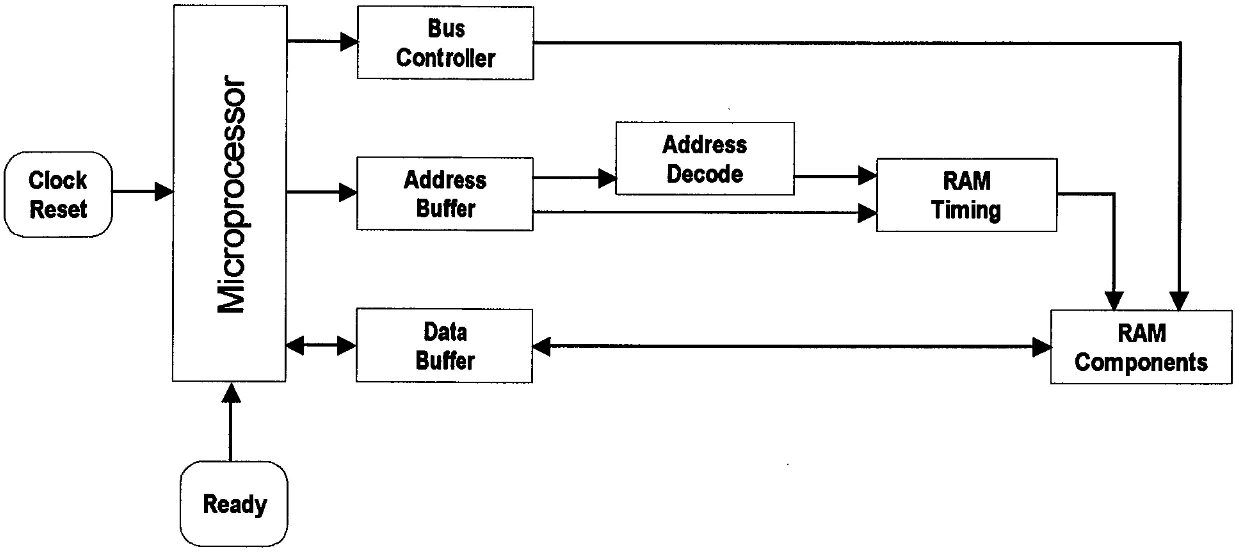
____

⑤ Fault Isolation:

STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.


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

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).

____

____

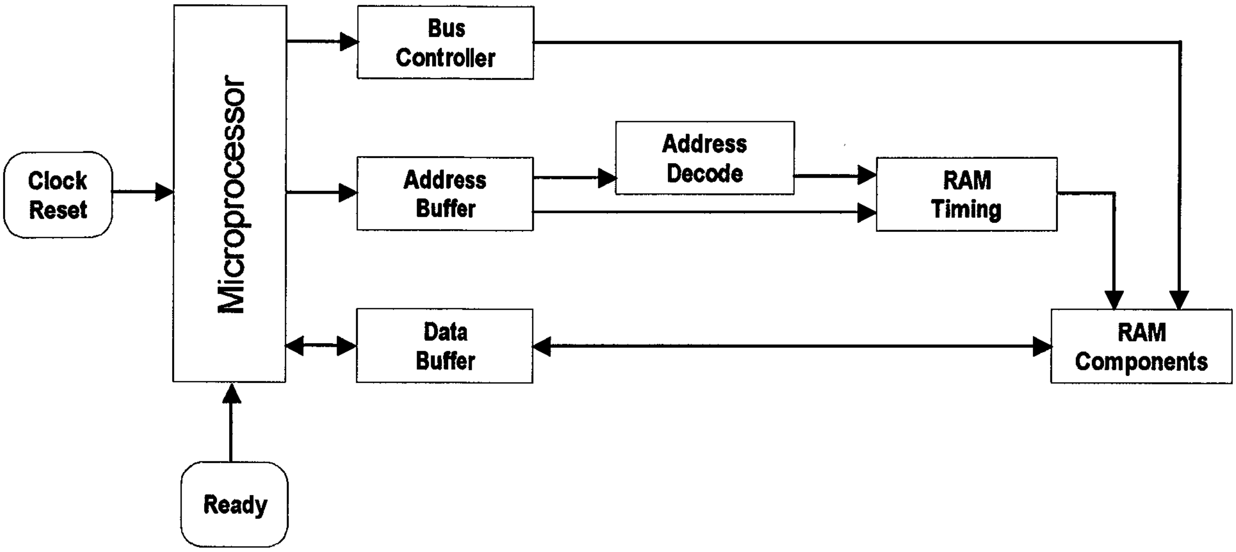
____

⑤ Fault Isolation:

STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.


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

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).

____

____

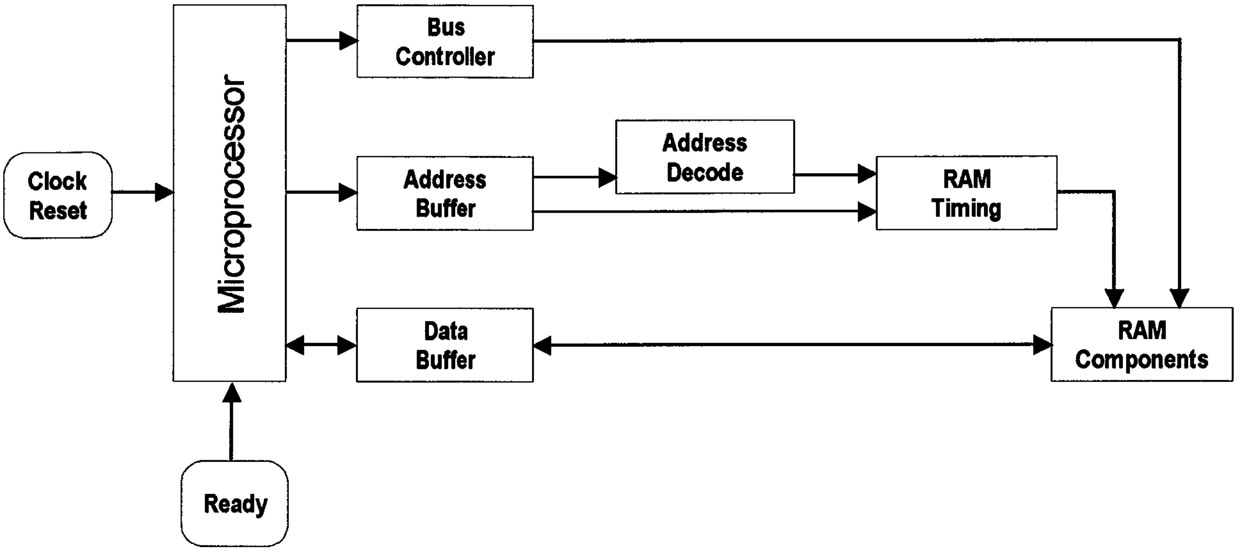
____

⑤ Fault Isolation:

STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.


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

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).

____

____

____

⑤ Fault Isolation:

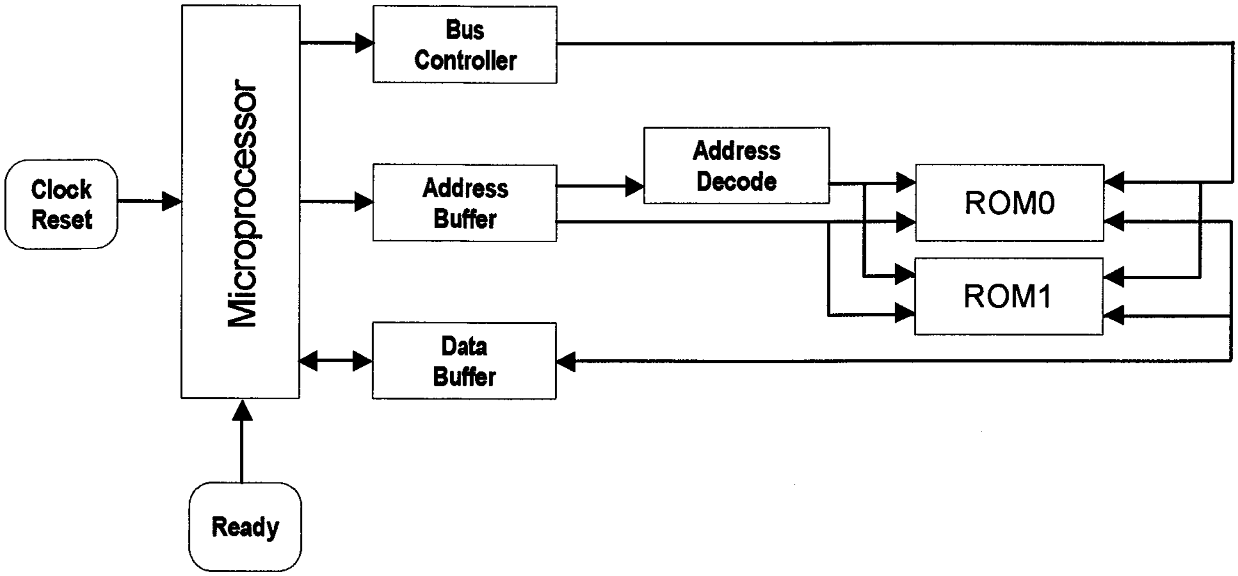
STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.

Section 6

ROM Testing


Bus Controller
Clock Reset
Microprocessor
Address Buffer
Address Decode
ROM0
ROM1
Data Buffer
Ready

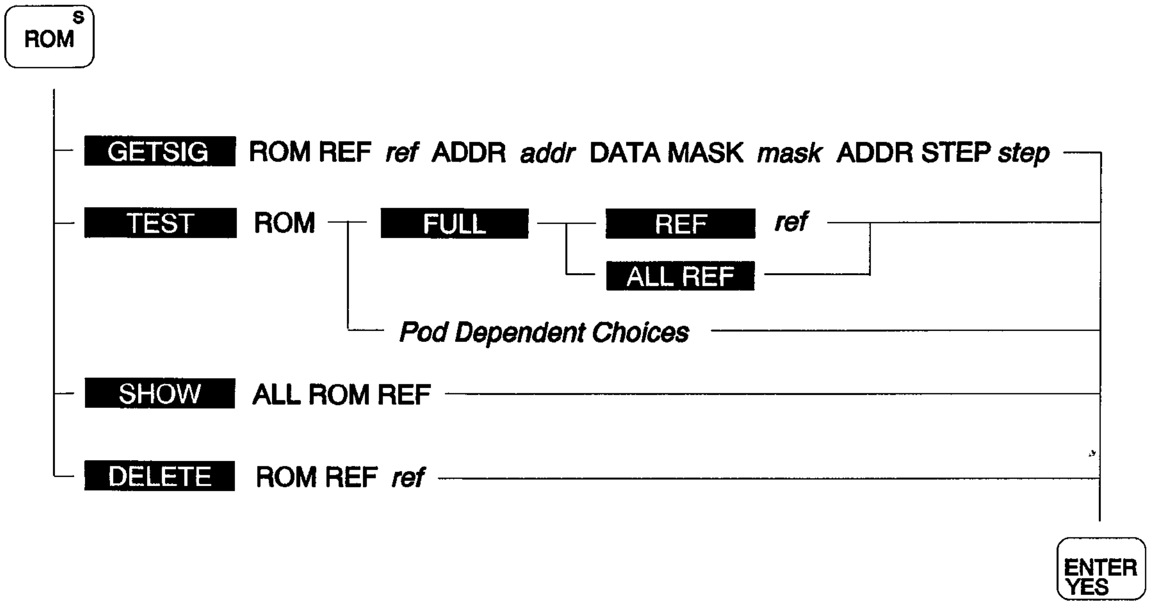
ROM FUNCTIONAL BLOCK

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**
- **Address 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 -EXERCISE 6-1

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

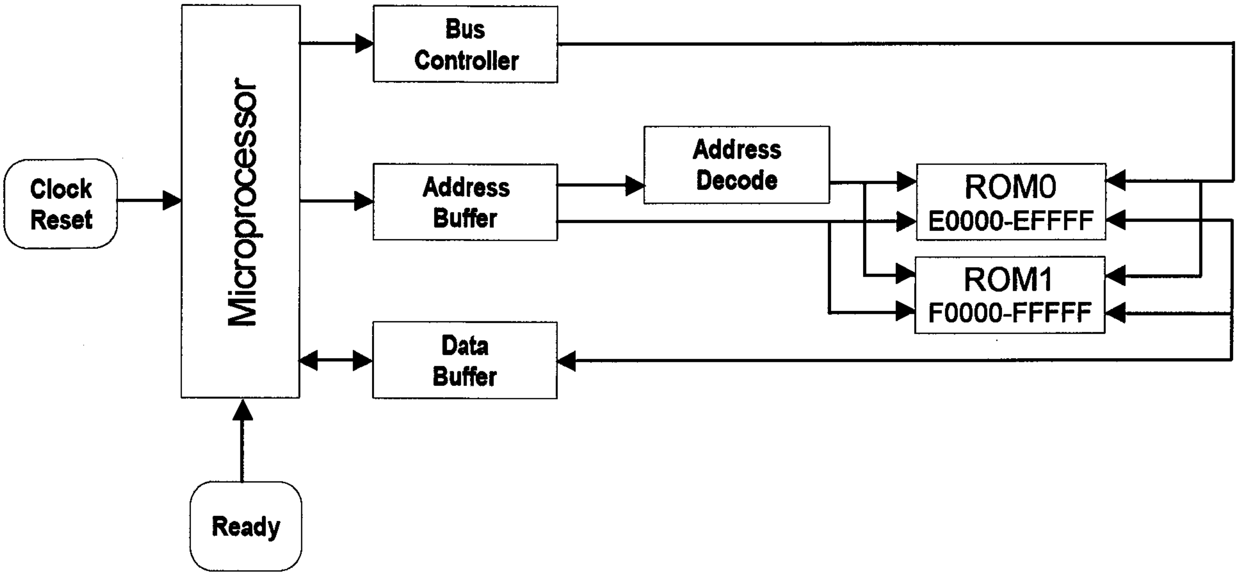
1. *press...* ROM S
2. *press...* F1 GET SIG (get signature)
3. *press...* →
4. *type...* **U27** (REF field)

   Reference designators for devices (enter this field using the alpha characters on the application keypad).

5. *press...* →
6. *type...* **F0000** (First address of U27)
7. *press...* →
8. *type...* **FFFFE** (Last address of U27)
9. *press...* →
10. *type...* **FF** (DATA MASK)

    Mask determines which data bits are used for calculating the signature

11. *press...* →
12. *type...* **2** (Address Step. Read words.)

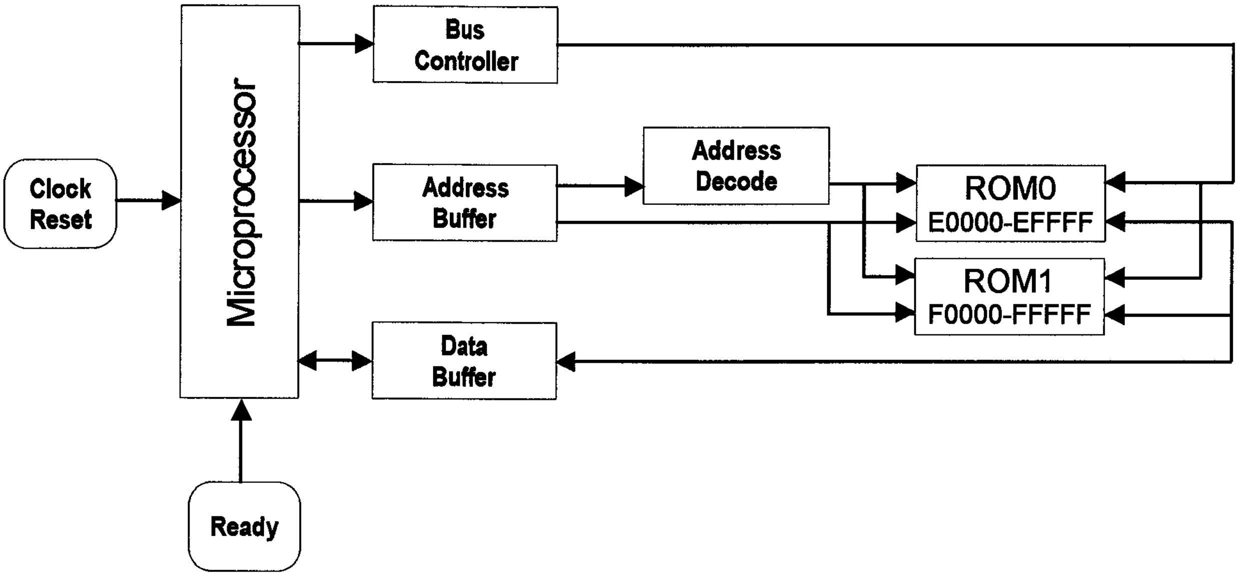

Microprocessor
Clock Reset
Ready
Bus Controller
Address Buffer
Data Buffer
Address Decode
ROM0
E0000-EFFFF
ROM1
F0000-FFFFF

13. *press...* ENTER YES

There will be a brief delay, after which, the following signature will appear on the Mainframe display:

SIGNATURE = F387

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


Bus
Controller
Microprocessor
Clock
Reset
Address
Buffer
Address
Decode
ROM0
E0000-EFFFF
ROM1
F0000-FFFFF
Data
Buffer
Ready

USING THE ROM TEST

The ROM test obtains a signature which is a data compression technique based on the CRC-16 algorithm. This signature is taken from the ROM under test and is compared to a previously stored "known good" signature.

If the ROM signature is found to be incorrect, a diagnostic routine uses all signature information to identify address and data lines that are:

- ✓ open
- ✓ tied to each other
- ✓ stuck high
- ✓ stuck low

Exercise 6-2

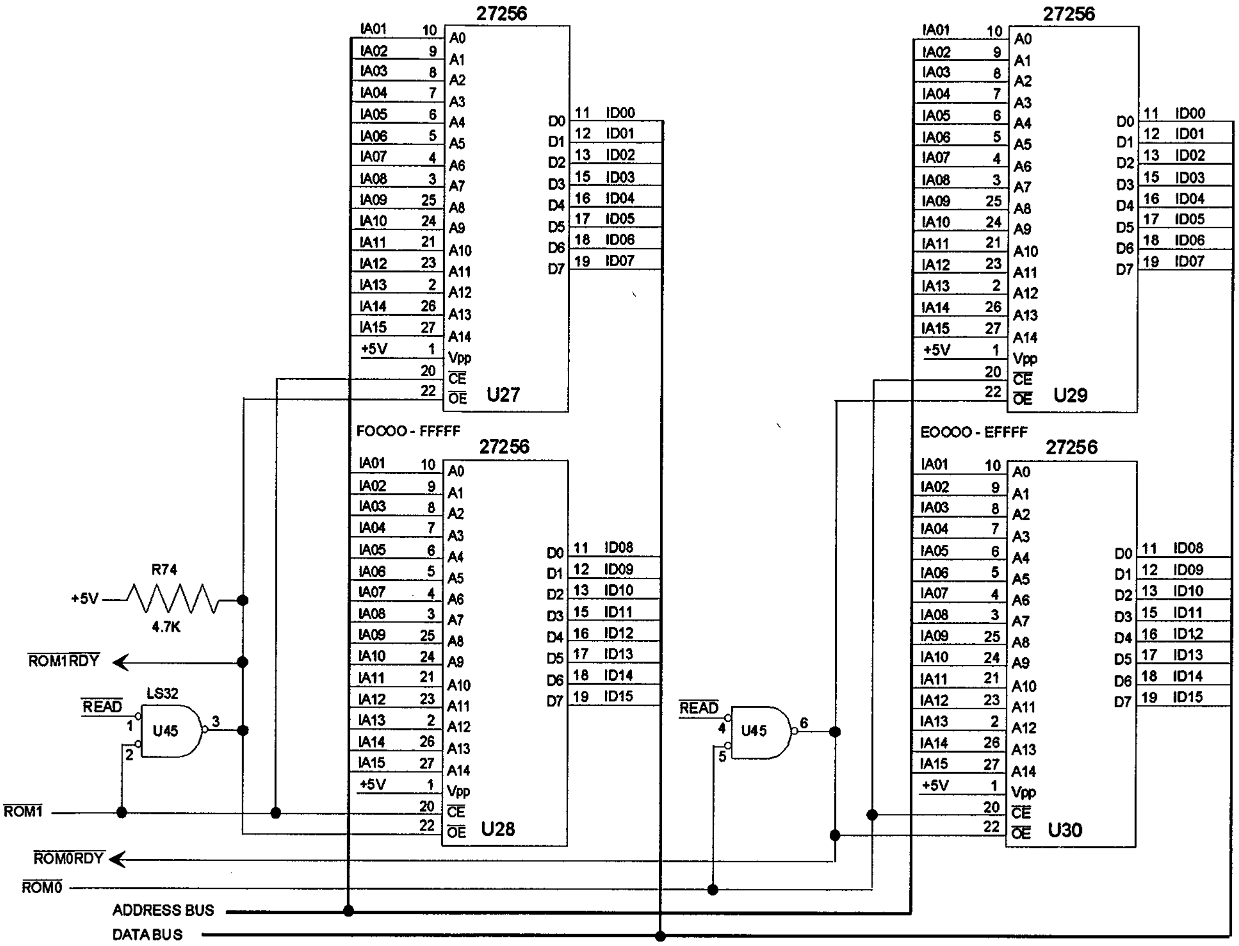
PERFORMING A ROM TEST - EXERCISE 6-2

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

② Write the steps performed in the space provided.

As a class:

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


27256
U27
U28
U29
U30
F0000 - FFFFF
E0000 - EFFFF
IA01 … IA15
A0 … A14
D0 … D7
ID00 … ID07
ID08 … ID15
+5V
Vpp
CE
OE
R74
4.7K
+5V
ROM1RDY
READ
LS32
U45
1
2
3
4
5
6
ROM1
ROM0RDY
ROM0
ADDRESS BUS
DATA BUS

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).

____

____

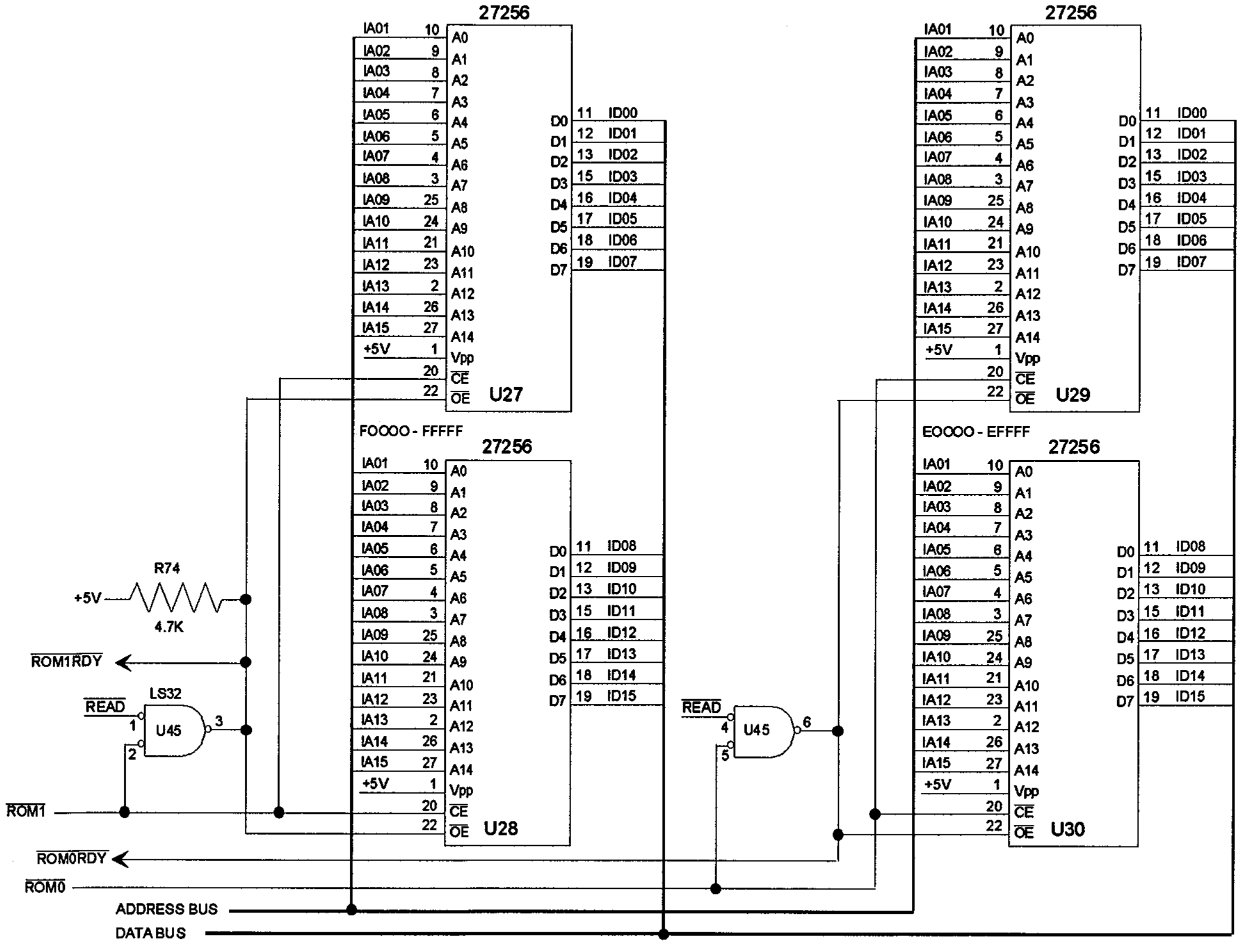
____

⑤ Fault Isolation:

STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.


27256
IA01 10 A0
IA02 9 A1
IA03 8 A2
IA04 7 A3
IA05 6 A4
IA06 5 A5
IA07 4 A6
IA08 3 A7
IA09 25 A8
IA10 24 A9
IA11 21 A10
IA12 23 A11
IA13 2 A12
IA14 26 A13
IA15 27 A14
+5V 1 Vpp
20 CE
22 OE
U27
D0 11 ID00
D1 12 ID01
D2 13 ID02
D3 15 ID03
D4 16 ID04
D5 17 ID05
D6 18 ID06
D7 19 ID07
F0000 - FFFFF
27256
U28
D0 11 ID08
D1 12 ID09
D2 13 ID10
D3 15 ID11
D4 16 ID12
D5 17 ID13
D6 18 ID14
D7 19 ID15
27256
U29
E0000 - EFFFF
27256
U30
R74
+5V
4.7K
ROM1RDY
LS32
READ
1
2
U45
3
ROM1
ROM0RDY
ROM0
ADDRESS BUS
DATA BUS
READ
4
5
U45
6

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).

____

____

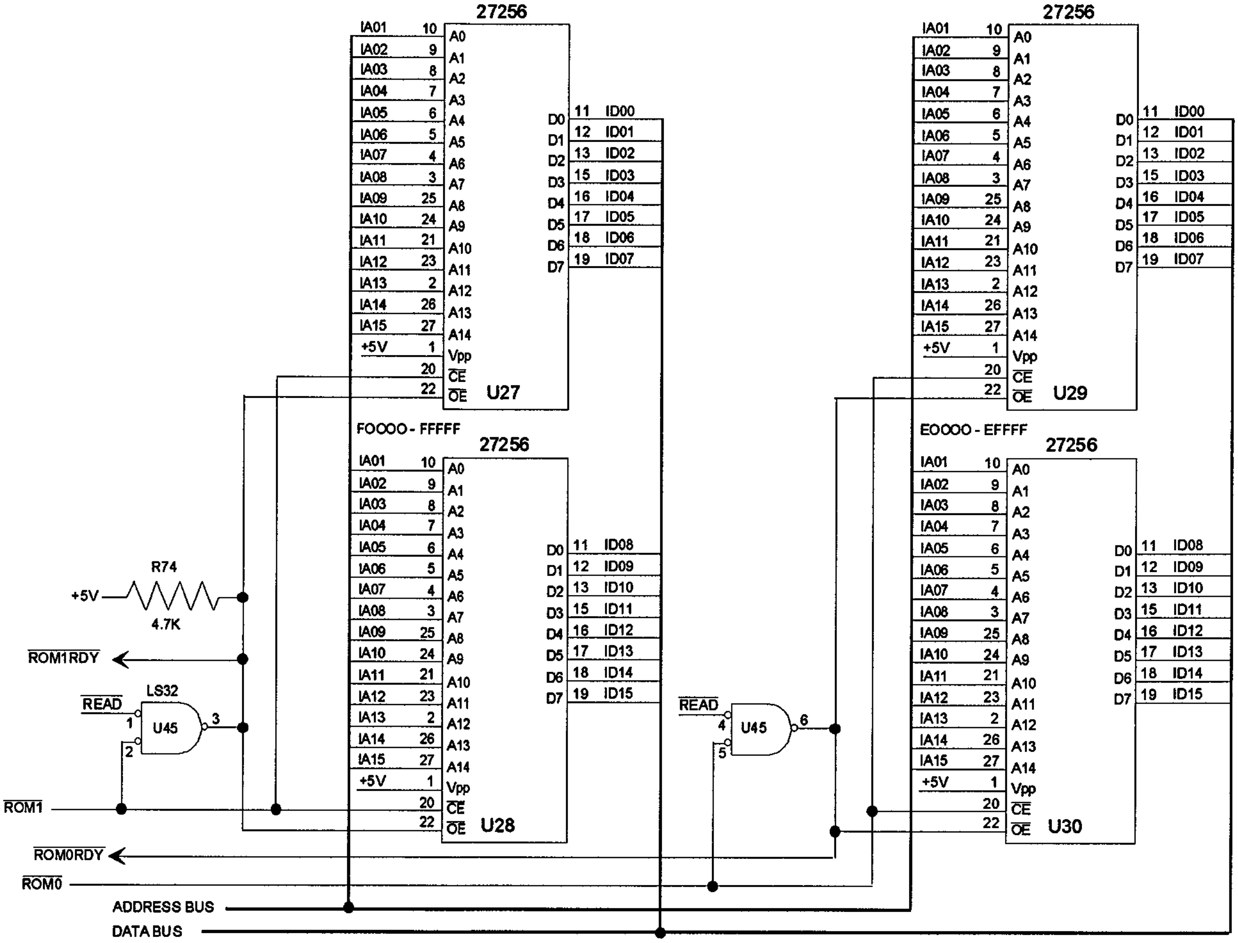
____

⑤ Fault Isolation:

STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate; *Where is the fault?*

⑦ Have the instructor check your results.


27256
U27
U28
U29
U30
F0000 - FFFFF
E0000 - EFFFF
IA01 10 A0
IA02 9 A1
IA03 8 A2
IA04 7 A3
IA05 6 A4
IA06 5 A5
IA07 4 A6
IA08 3 A7
IA09 25 A8
IA10 24 A9
IA11 21 A10
IA12 23 A11
IA13 2 A12
IA14 26 A13
IA15 27 A14
+5V 1 Vpp
20 CE
22 OE
D0 11 ID00
D1 12 ID01
D2 13 ID02
D3 15 ID03
D4 16 ID04
D5 17 ID05
D6 18 ID06
D7 19 ID07
D0 11 ID08
D1 12 ID09
D2 13 ID10
D3 15 ID11
D4 16 ID12
D5 17 ID13
D6 18 ID14
D7 19 ID15
R74
+5V
4.7K
ROM1RDY
READ
LS32
U45
1
2
3
4
5
6
ROM1
ROM0RDY
ROM0
ADDRESS BUS
DATA BUS

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).

____

____

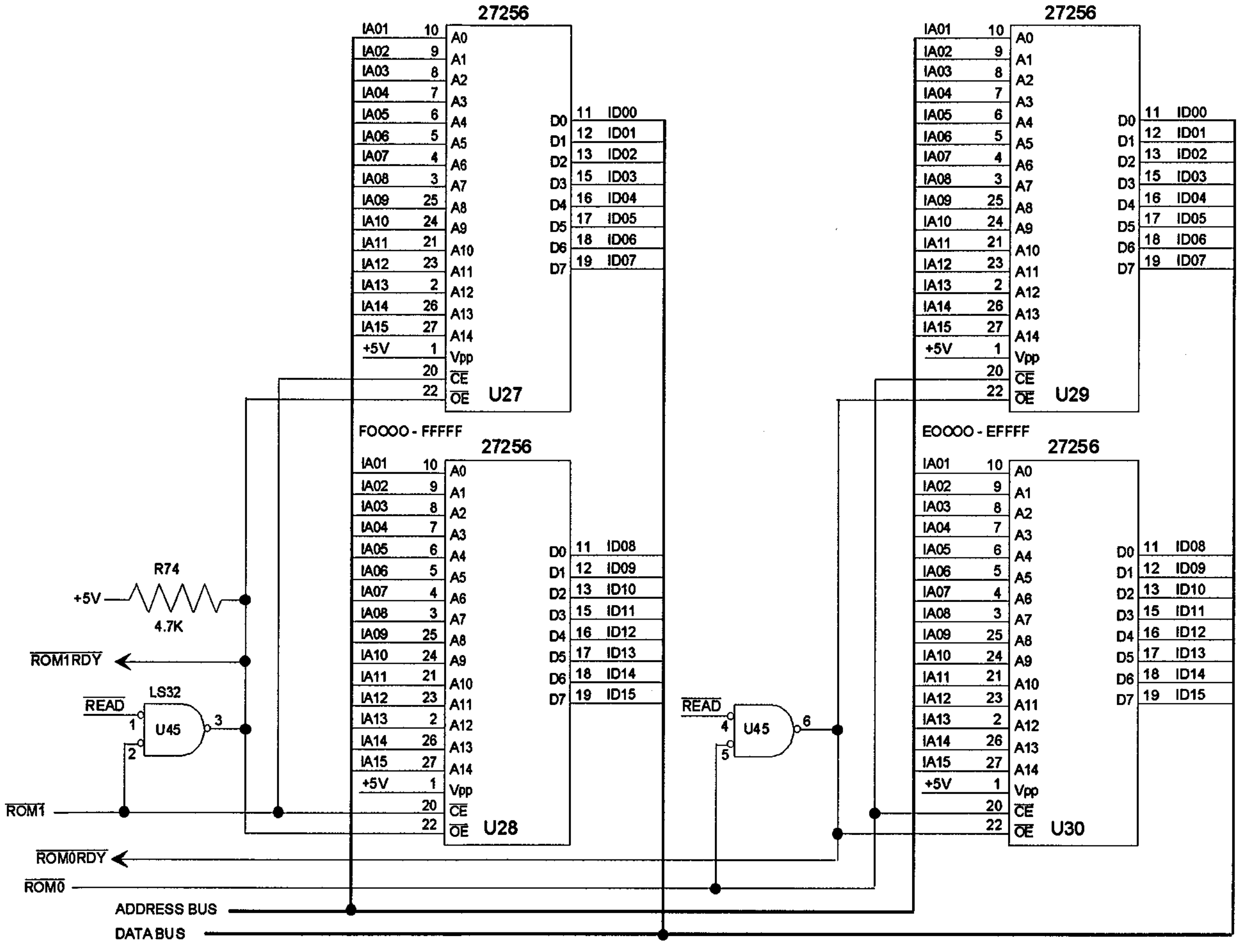
____

⑤ Fault Isolation:

STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.


27256
IA01 10 A0
IA02 9 A1
IA03 8 A2
IA04 7 A3
IA05 6 A4
IA06 5 A5
IA07 4 A6
IA08 3 A7
IA09 25 A8
IA10 24 A9
IA11 21 A10
IA12 23 A11
IA13 2 A12
IA14 26 A13
IA15 27 A14
+5V 1 Vpp
20 CE
22 OE
U27
D0 11 ID00
D1 12 ID01
D2 13 ID02
D3 15 ID03
D4 16 ID04
D5 17 ID05
D6 18 ID06
D7 19 ID07
FOOOO - FFFFF
27256
U28
D0 11 ID08
D1 12 ID09
D2 13 ID10
D3 15 ID11
D4 16 ID12
D5 17 ID13
D6 18 ID14
D7 19 ID15
27256
U29
EOOOO - EFFFF
27256
U30
R74
+5V
4.7K
ROM1RDY
READ
LS32
U45
1
2
3
ROM1
READ
U45
4
5
6
ROM0RDY
ROM0
ADDRESS BUS
DATA BUS

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).

____

____

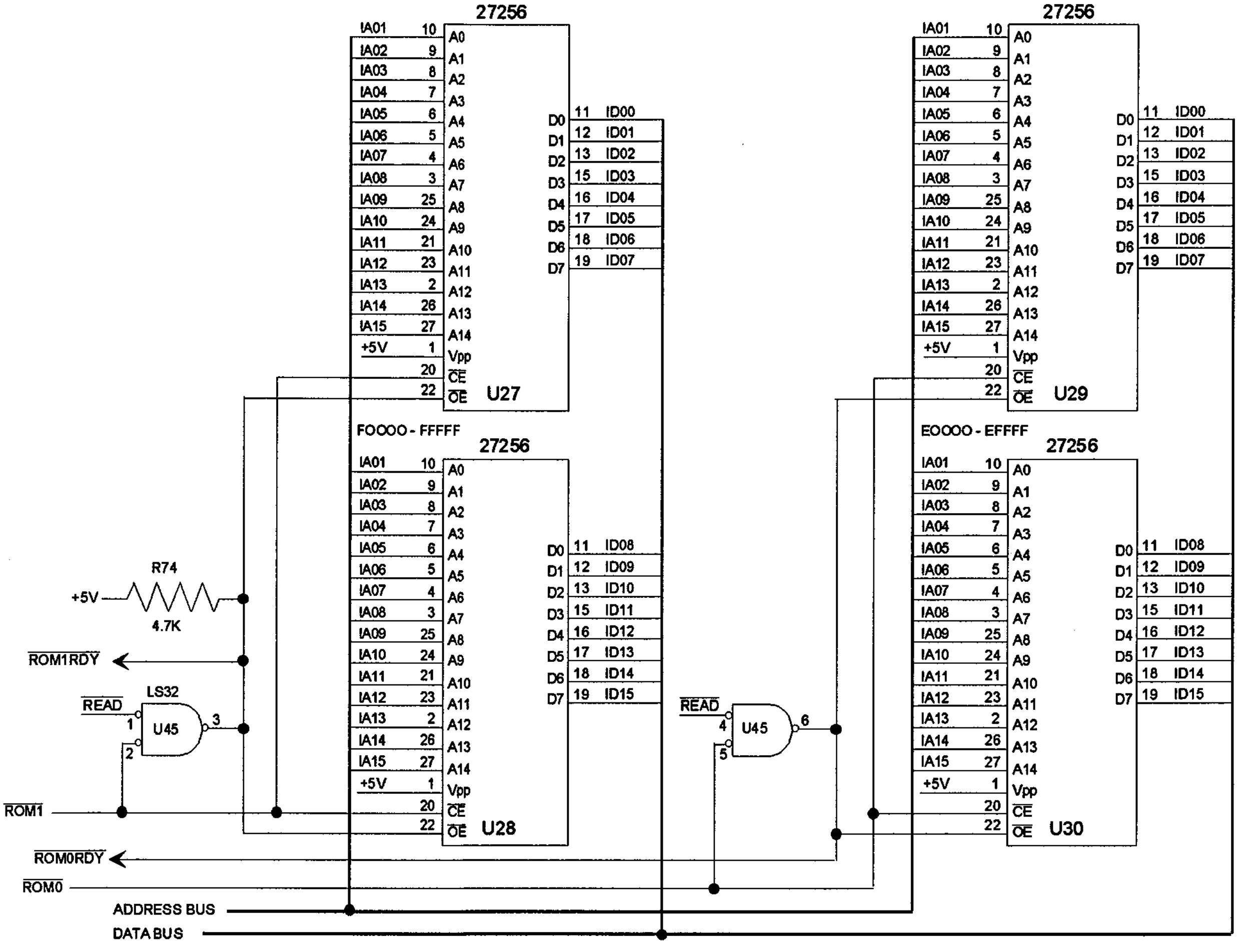
____

⑤ Fault Isolation:

STIMULUS COMMANDS	NODES	RESPONSE

⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.


27256
U27
U28
U29
U30
F0000 - FFFFF
E0000 - EFFFF
IA01
IA02
IA03
IA04
IA05
IA06
IA07
IA08
IA09
IA10
IA11
IA12
IA13
IA14
IA15
+5V
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
Vpp
CE
OE
D0
D1
D2
D3
D4
D5
D6
D7
ID00
ID01
ID02
ID03
ID04
ID05
ID06
ID07
ID08
ID09
ID10
ID11
ID12
ID13
ID14
ID15
R74
4.7K
+5V
ROM1RDY
READ
LS32
U45
ROM1
ROM0RDY
ROM0
ADDRESS BUS
DATA BUS

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	NODES	RESPONSE

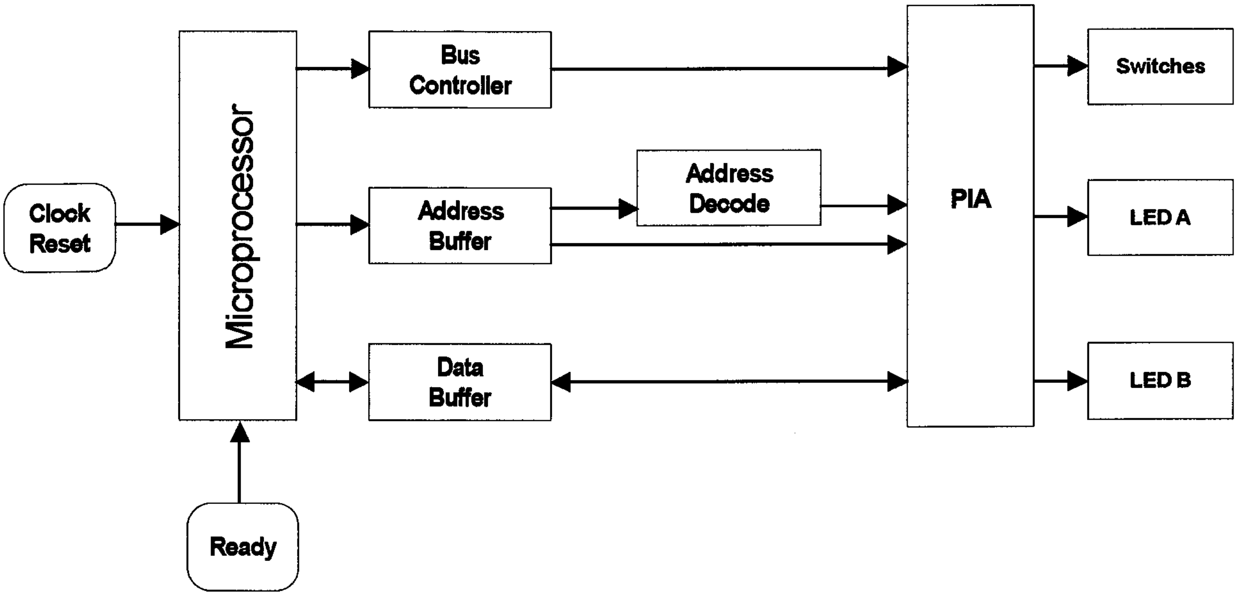
⑥ Evaluate: *Where is the fault?*

⑦ Have the instructor check your results.

Section 7

Parallel Interface Adapter

The Parallel Interface functional block will be used as an example for developing a test in an area not covered by a 9100FT *built-in test.*


Clock
Reset
Microprocessor
Ready
Bus
Controller
Address
Buffer
Data
Buffer
Address
Decode
PIA
Switches
LED A
LED B

UNDERSTANDING THE PARALLEL INTERFACE ADAPTER BLOCK

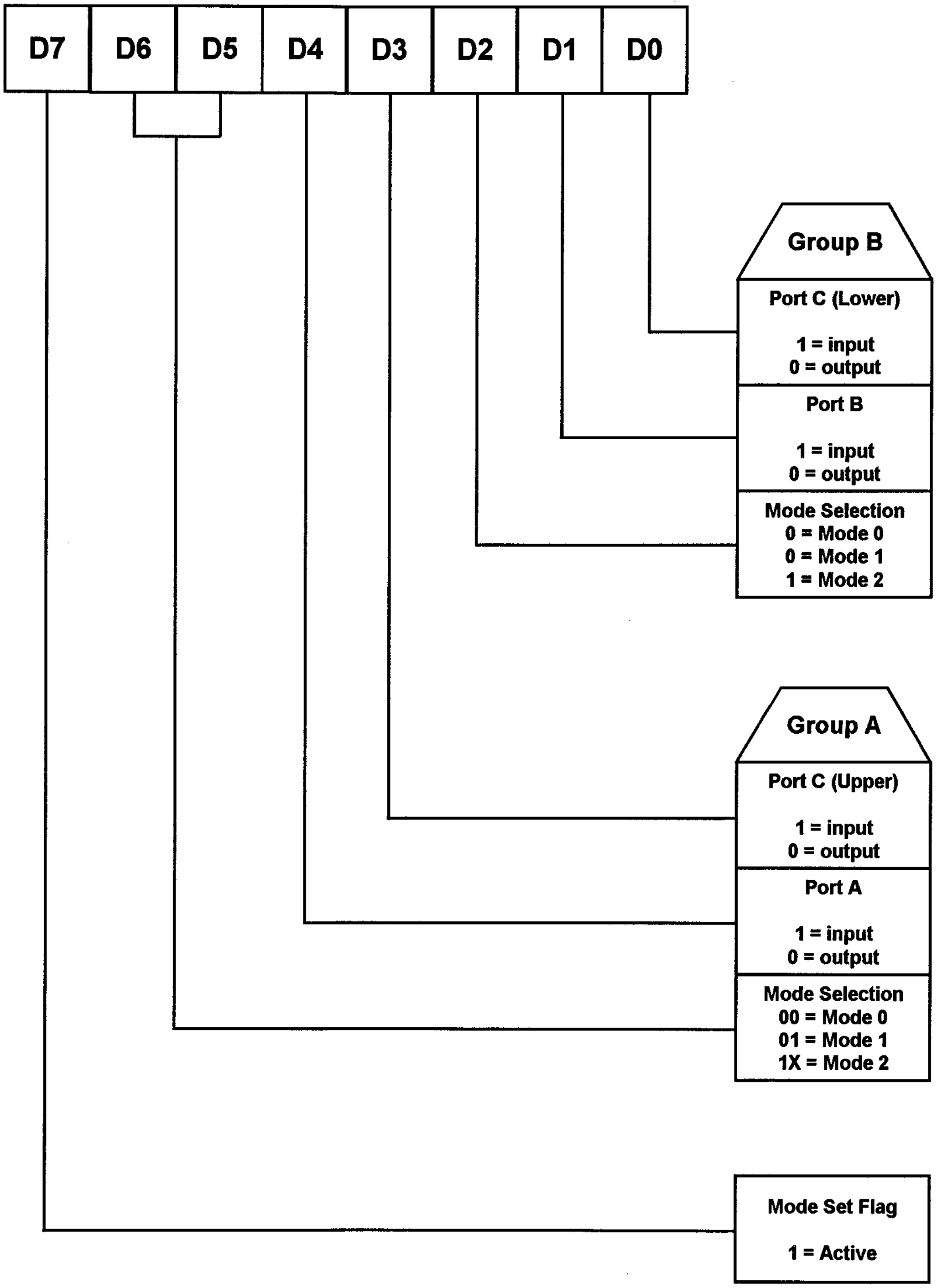
As a class:

Describe the basic operation of the functional block named "PIA" (Parallel Interface Adapter*). Refer to illustration on previous page (7-2) to help 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 (PPI), or a Programmable Communications Adapter (PCA).


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 - EXERCISE 7-1

① Determine the control word to activate:

PORT A = output
PORT B = output
PORT C = input
Control Word = ________________

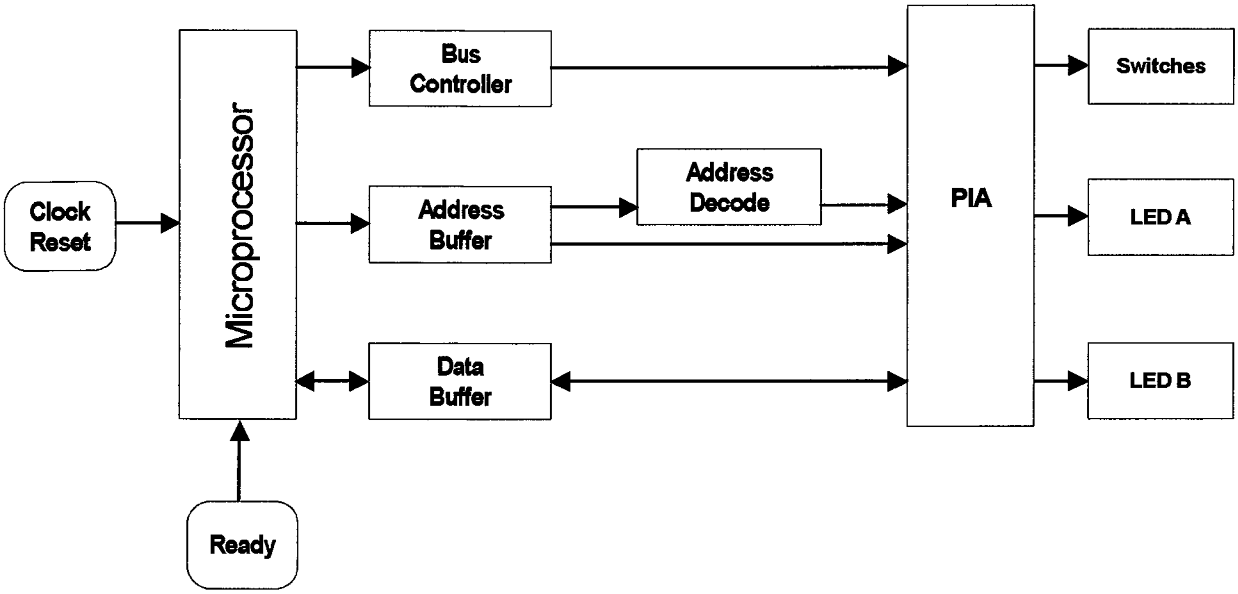
② Determine the PIA Addresses:

PORT A = ________________
PORT B = ________________
PORT C = ________________
Control Word = ________________

③ Configure the PIA.

④ Verify configuration by displaying a 7 on LED A.

⑤ Display a .F on LED B.


Clock
Reset
Microprocessor
Ready
Bus Controller
Address Buffer
Data Buffer
Address Decode
PIA
Switches
LED A
LED B

Exercise 7-2

LED FUNCTIONAL TEST FOR THE PIA - EXERCISE 7-2

① Configure the PIA for the proper input/output setting of each port.

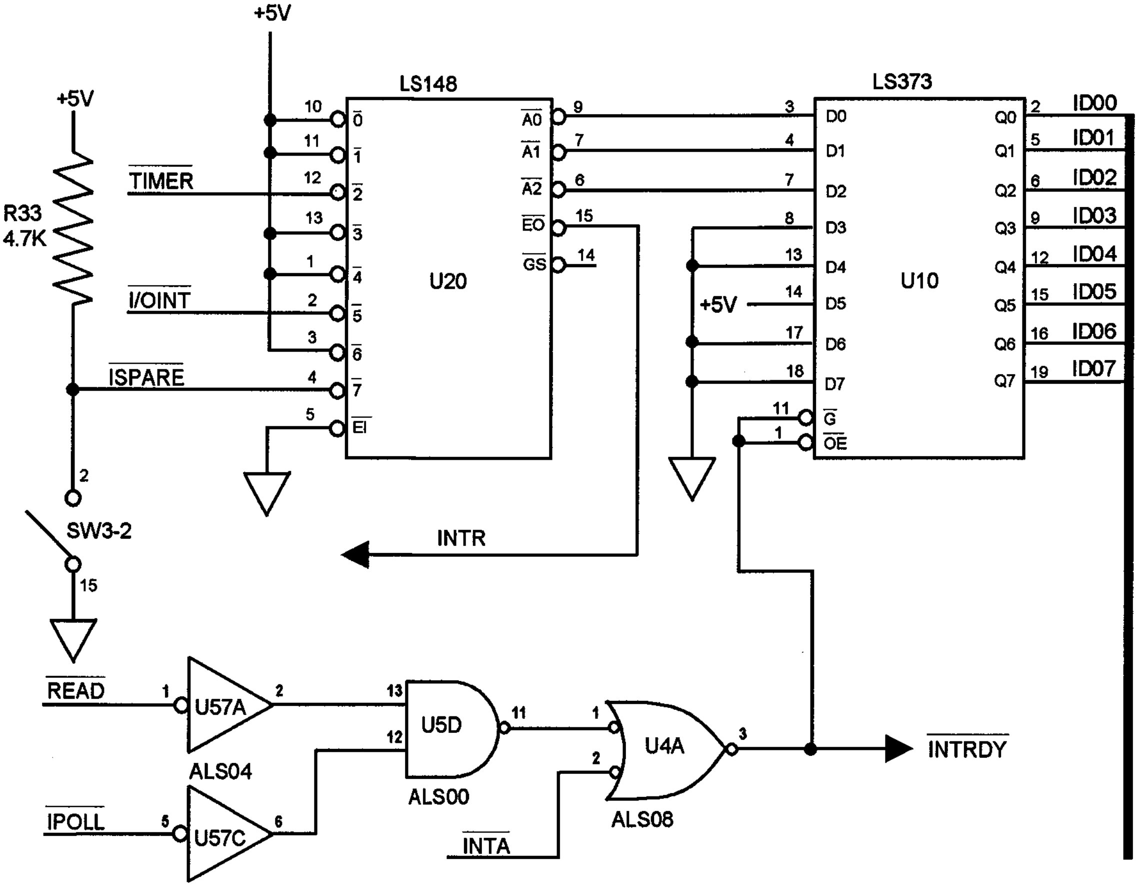
② Use the I/O module (pin mode) with the 40 pin clip to functionally test the PIA chip.

As a class:

Discuss the immediate mode capabilities of the I/O Module before beginning the test.

Section 8

Fault Isolation


+5V
+5V
R33
4.7K
TIMER
I/OINT
ISPARE
2
SW3-2
15
LS148
U20
10
11
12
13
1
2
3
4
5
0
1
2
3
4
5
6
7
EI
A0
A1
A2
EO
GS
9
7
6
15
14
INTR
LS373
U10
3
4
7
8
13
14
17
18
11
1
+5V
D0
D1
D2
D3
D4
D5
D6
D7
G
OE
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
2
5
6
9
12
15
16
19
ID00
ID01
ID02
ID03
ID04
ID05
ID06
ID07
READ
1
U57A
2
ALS04
IPOLL
5
U57C
6
13
12
U5D
11
ALS00
INTA
1
2
U4A
3
ALS08
INTRDY

INTERRUPT CIRCUITRY STIMULUS

Exercise 8-1

INTERRUPT CIRCUITRY STIMULUS - EXERCISE 8-1

Develop a procedure to verify the Interrupt Circuitry is operating correctly.

As a class:

Review test strategy.

As a class:

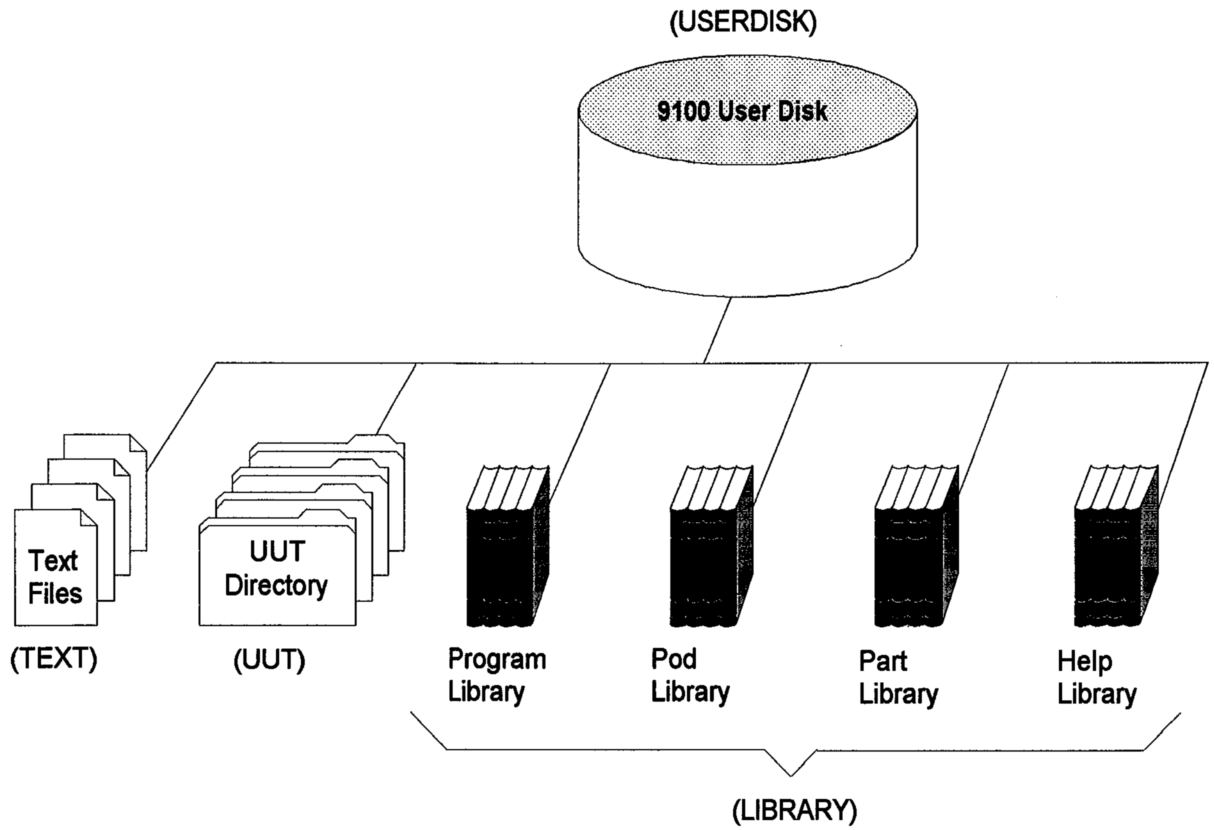
Review exercise when complete.

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 incorporated a userdisk that contains data and programs for one or more units under test. To provide additional userdisks, you add more floppy disks.


(USERDISK)
9100 User Disk
Text Files
(TEXT)
UUT Directory
(UUT)
Program Library
Pod Library
Part Library
Help Library
(LIBRARY)

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:

✓ **Test Files** - Operator's instructions and program documentation are stored in text documents.

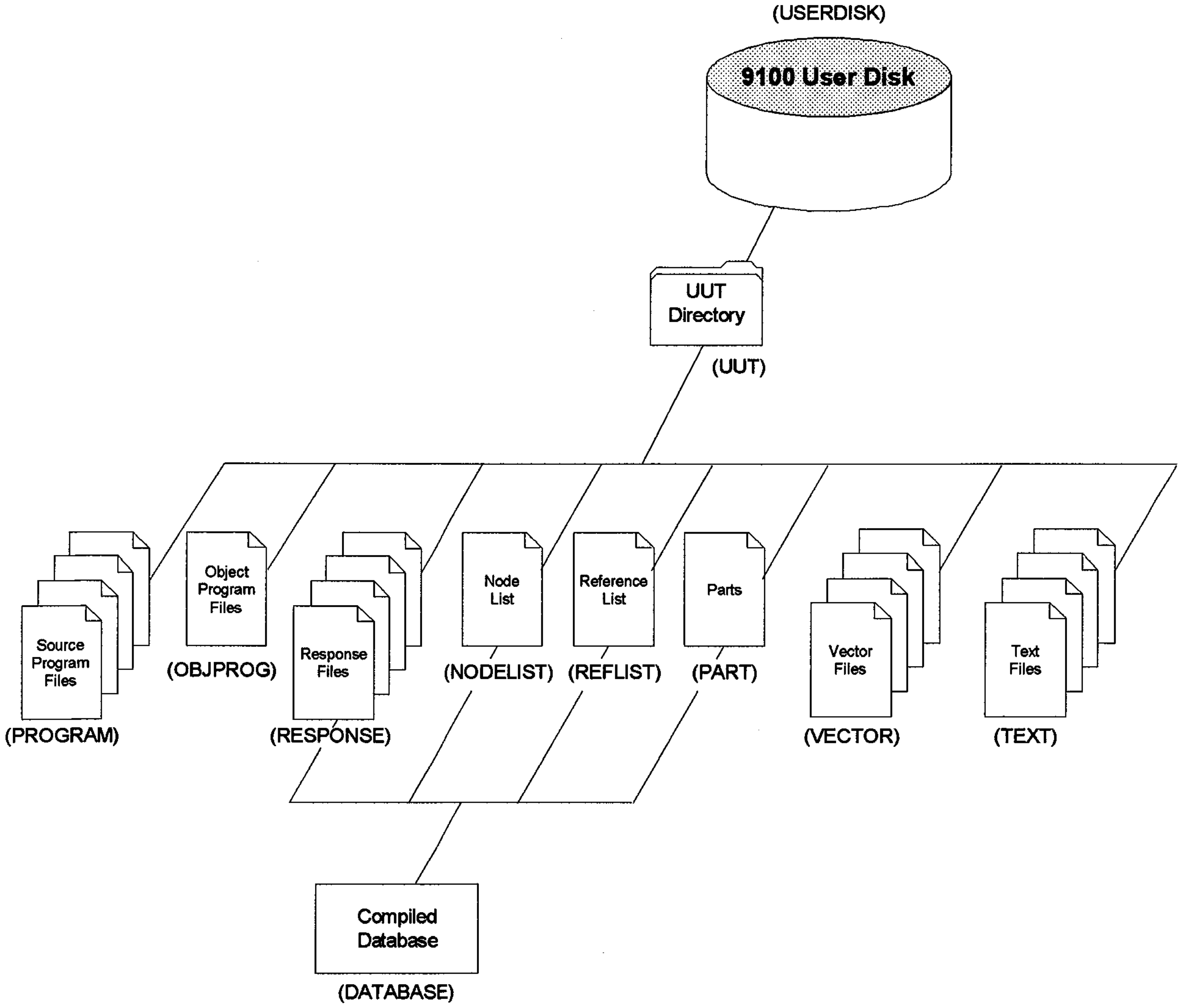
✓ **Part Library** - The Part Library consists solely of part description. The Part Library is shared by all the UUT descriptions on the same userdisk, so that the part description does not have to be duplicated for each UUT that uses the part.

✓ **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.

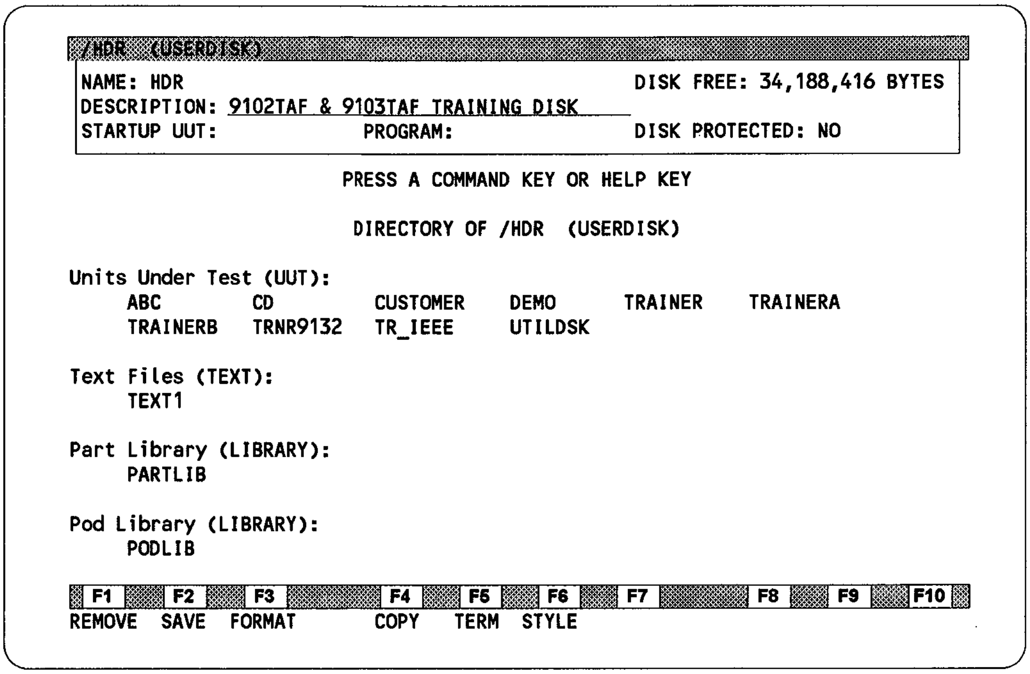
✓ **UUT Directories** - Each userdisk may contain one or more UUT directory.


(USERDISK)
9100 User Disk
UUT Directory
(UUT)
Source Program Files
(PROGRAM)
Object Program Files
(OBJPROG)
Response Files
(RESPONSE)
Node List
(NODELIST)
Reference List
(REFLIST)
Parts
(PART)
Vector Files
(VECTOR)
Text Files
(TEXT)
Compiled Database
(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 Database** - A compiled database contains responses, reference designators, parts descriptions, 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: 34,188,416 BYTES
DESCRIPTION: 9102TAF & 9103TAF TRAINING DISK
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      TRAINER    TRAINERA
     TRAINERB   TRNR9132   TR_IEEE     UTILDSK

Text Files (TEXT):
     TEXT1

Part Library (LIBRARY):
     PARTLIB

Pod Library (LIBRARY):
     PODLIB
```

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.

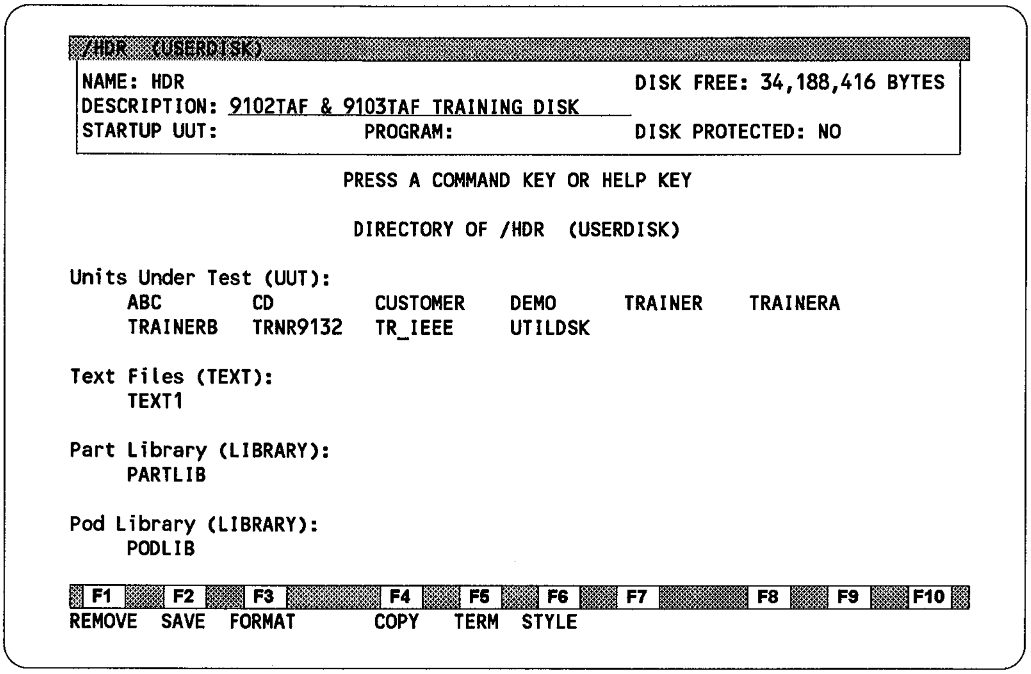
The information window and edit window for the HDR userdisk appears in the display. From this point on, you enter commands from the programmer's keyboard.

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

✓ **DESCRIPTION** - Contains pertinent data about the edited disk: the owner's name, the date the disk was created and for what application. This window does not require any information and can be skipped.

✓ **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.) *Startup UUT* does not require any information and can be skipped.

✓ **STARTUP PROGRAM** - Specifies the first program executed once Mainframe is booted up. This window does not require any information and can be skipped.



```
/HDR  (USERDISK)
NAME: HDR                                       DISK FREE: 34,188,416 BYTES
DESCRIPTION: 9102TAF & 9103TAF TRAINING DISK
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      TRAINER    TRAINERA
     TRAINERB   TRNR9132   TR_IEEE     UTILDSK

Text Files (TEXT):
     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
```

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 micro floppy disk drive.

✓ F4 **(COPY)**

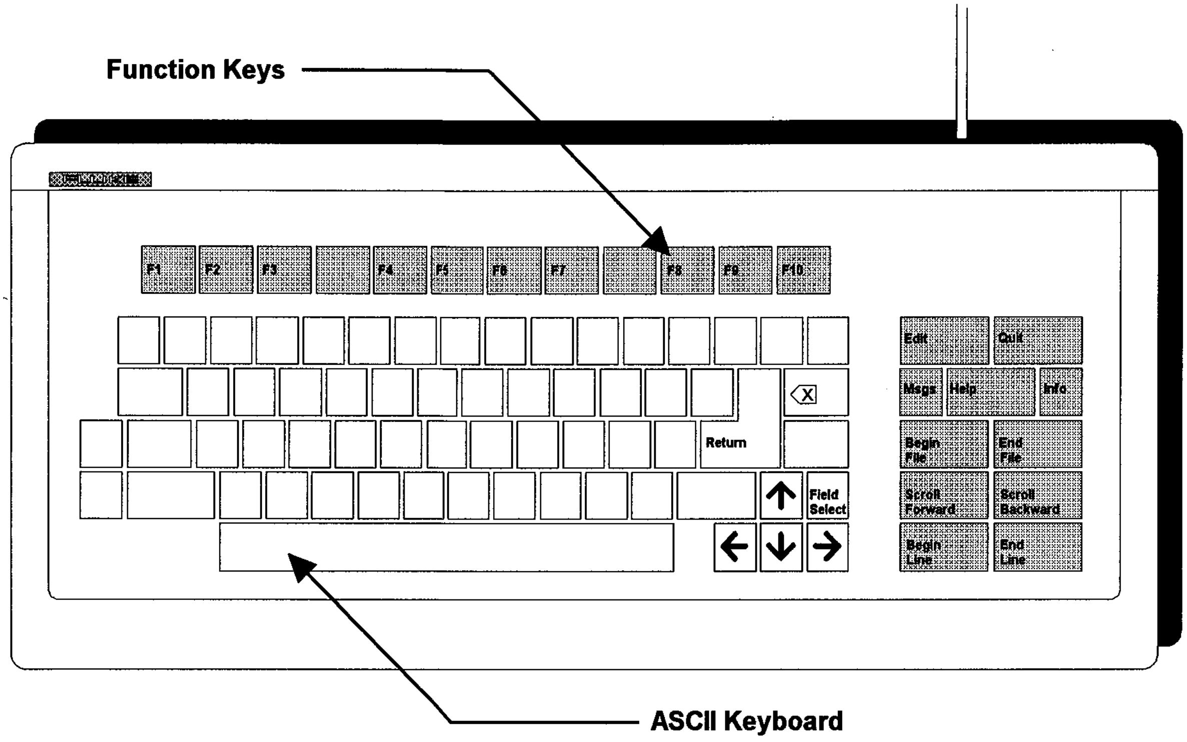
Copies an existing file to a *different disk using the same name* or to the *same disk using a different name.*

✓ F5 **(TERM)**

Enters the terminal emulator.

✓ F6 **(STYLE)**

Allows selection of brief or long directory list.


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

The editor is redirected to either DR1 (userdisk) or any item displayed in the HDR userdisk window. A prompt appears to let you enter the name and type of what you want to edit.

type.... /hdr/trainer or trainer followed by a return. (not case sensitive)

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 type... as **UUT** using the FIELD SELECT Key and RETURN. You are now at the UUT window.

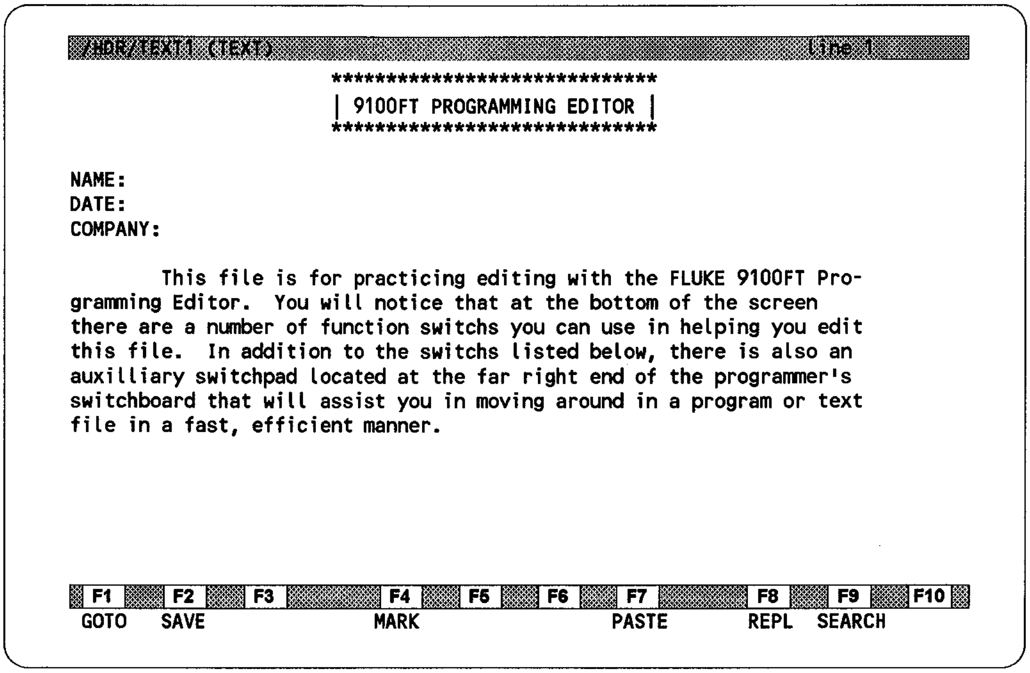
press... the RETURN key

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... the QUIT key

The USERDISK Directory is now displayed.



```
                         ******************************
                         | 9100FT PROGRAMMING EDITOR |
                         ******************************

NAME:
DATE:
COMPANY:

        This file is for practicing editing with the FLUKE 9100FT Pro-
gramming Editor.  You will notice that at the bottom of the screen
there are a number of function switchs you can use in helping you edit
this file.  In addition to the switchs listed below, there is also an
auxilliary 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.
```

TEXT ENTRY

TEXT1 is used to practice the editor functions. This file concentrates on editor functions rather than TL/1 syntax.

Exercise 9-1

TEXT ENTRY USING TEXT1 - EXERCISE 9-1

1.

press... the EDIT key on programmer's keyboard.

edit... file text1 on /hdr.

press... the FIELD SELECT key to select type TEXT.

2. Position cursor with the arrow keys and record in designated spaces: your name, date, and company

3. Save your changes using softkey F2 (SAVE).

4. F1 (GOTO)

The GOTO function allows you to "jump" to a specific line in a long program without scrolling.

press... the F1 key - GOTO Line 14.

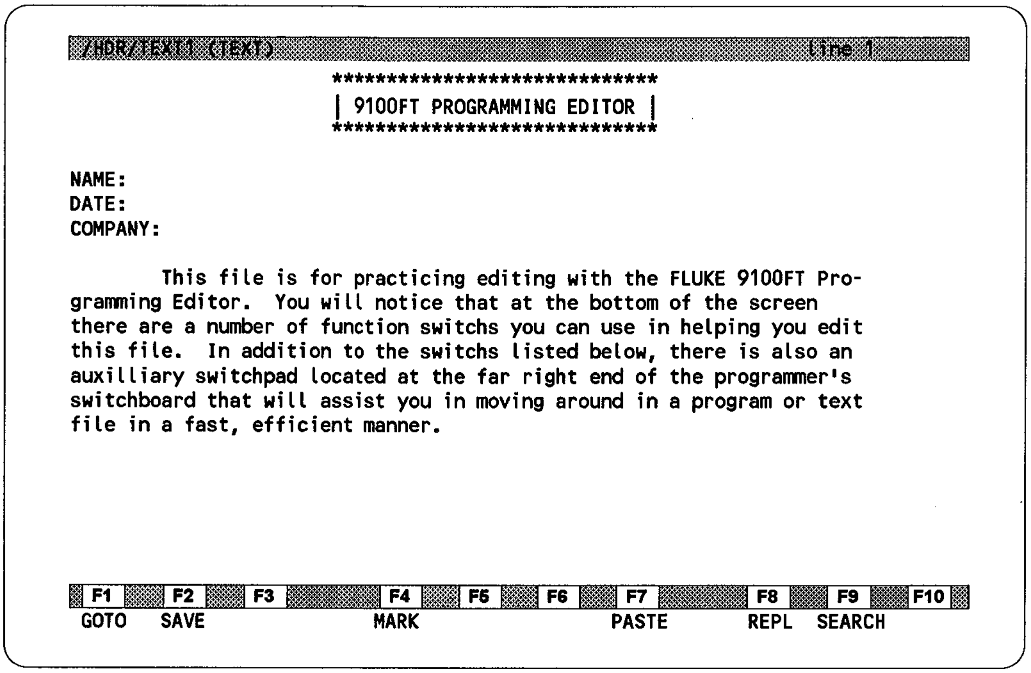
5. F4 (MARK Function)

press... the F4 key - The Mainframe will now use the cursor position as a reference point.

position cursor... at the first character of the line.

press... the ↓ key 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.



```
/HDR/TEXT1 (TEXT)                                                   Line 1
                           *******************************
                           | 9100FT PROGRAMMING EDITOR |
                           *******************************

NAME:
DATE:
COMPANY:

        This file is for practicing editing with the FLUKE 9100FT Pro-
gramming Editor.  You will notice that at the bottom of the screen
there are a number of function switchs you can use in helping you edit
this file.  In addition to the switchs listed below, there is also an
auxilliary 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.
```

6. F5 (CUT Function)

   press... the F5 key - the lines previously marked with MARK function are cut. The cut text is stored in a buffer for later retrieval. See step (7)

7. F7 (PASTE Function)

   press... the F7 key - text previously stored in cut buffer appears on screen.

   The PASTE function is useful in programs where a line of text or various commands are used repeatedly.

8. GOTO the line on which your name appears.

9. MARK your name, date, and company.

   press... the F6 key (YANK)

   What is the difference between YANK and CUT?

10. F9 (SEARCH Function)

    This function allows you to search for a string of text.

    press... the F9 key - the editor asks you what you want to search for.

    type... the word "switch". Notice that the cursor advances to the first instance of the word "switch".

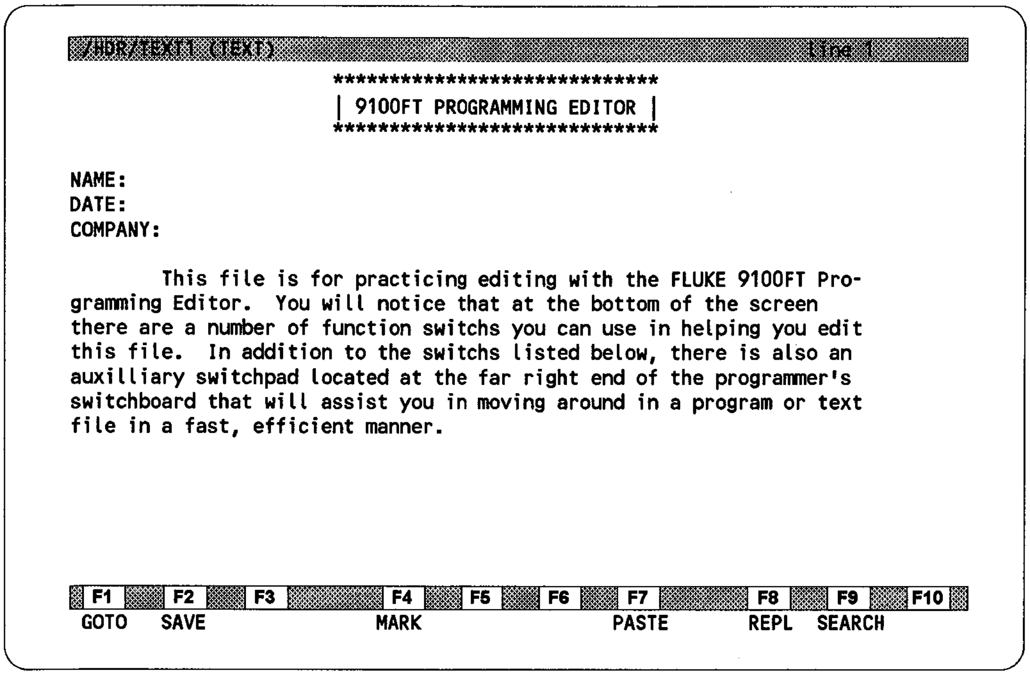
11. F8 (REPLACE Function)

    Replaces a specified string of characters with another string of characters.

    press... the F8 key - the editor asks for the characters to be replaced.

    type... the word "switch"

    press... the RETURN key - The editor asks for the characters you want in the place of the word "switch".



```
/HDR/TEXT1 (TEXT)                                              Line 1
                    *******************************
                    | 9100FT PROGRAMMING EDITOR |
                    *******************************

NAME:
DATE:
COMPANY:

        This file is for practicing editing with the FLUKE 9100FT Pro-
gramming Editor.  You will notice that at the bottom of the screen
there are a number of function switchs you can use in helping you edit
this file.  In addition to the switchs listed below, there is also an
auxilliary 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
```

type... the word "key"

press... RETURN Notice 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)

press... the F2 key - The file is saved.

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 key again.

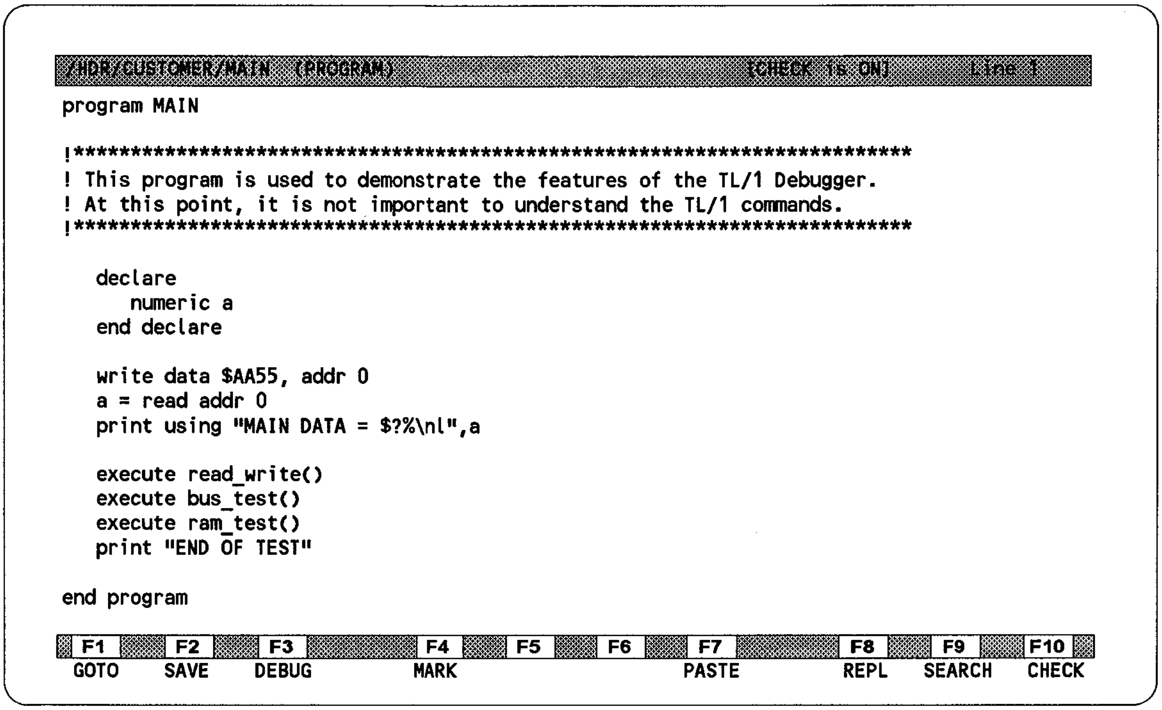
15. Exiting the editor

press... the QUIT key on auxiliary keypad. Each time QUIT is pressed you move up one level 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/MAIN  (PROGRAM)                               [CHECK is ON]      Line 1
program MAIN

!*************************************************************************
! This program is used to demonstrate the features of the TL/1 Debugger.
! At this point, it is not important to understand the TL/1 commands.
!*************************************************************************

   declare
      numeric a
   end declare

   write data $AA55, addr 0
   a = read addr 0
   print using "MAIN DATA = $?%\nl",a

   execute read_write()
   execute bus_test()
   execute ram_test()
   print "END OF TEST"

end program
```

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.

Exercise 9-2

USING THE DEBUGGER - EXERCISE 9-2

1. Edit program MAIN

   press... EDIT
   type... /hdr/customer/main
   press... RETURN, then FIELD SELECT PROGRAM, RETURN

   You should now be viewing the program **MAIN**. At this point, it is not necessary to concentrate on the TL/1 statement.

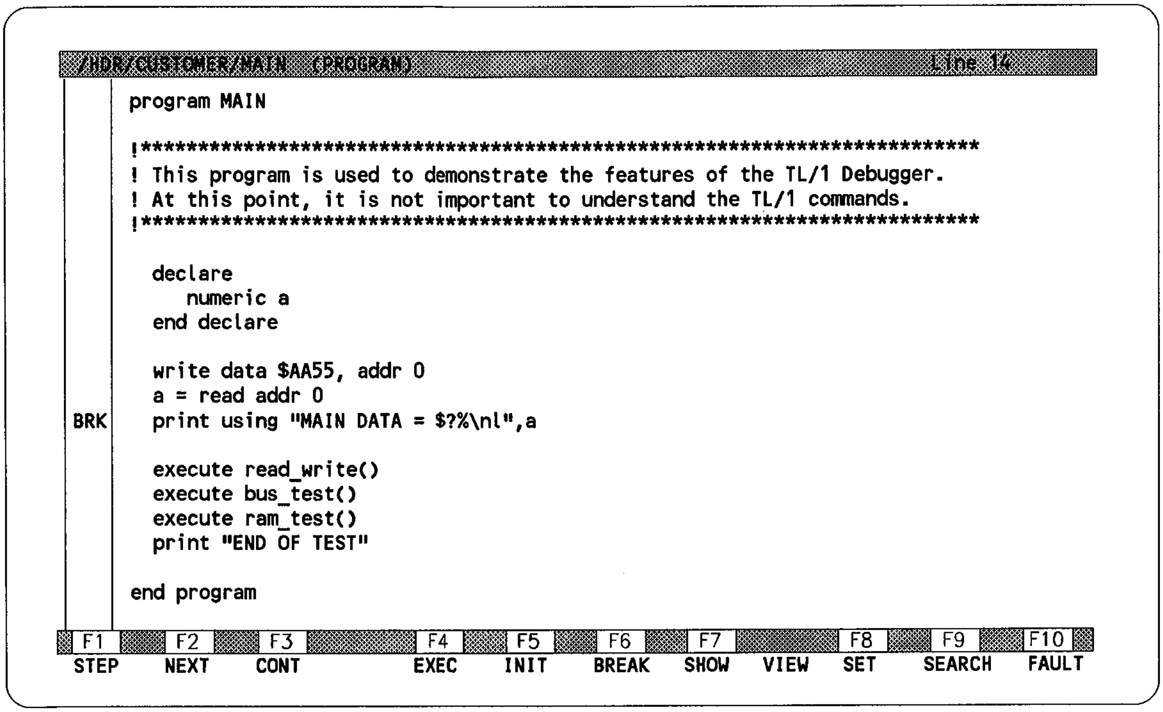
2. F3 (DEBUG)

   press... the F3 key to load the debugger. The bottom of the screen will display LOADING DEBUGGER...

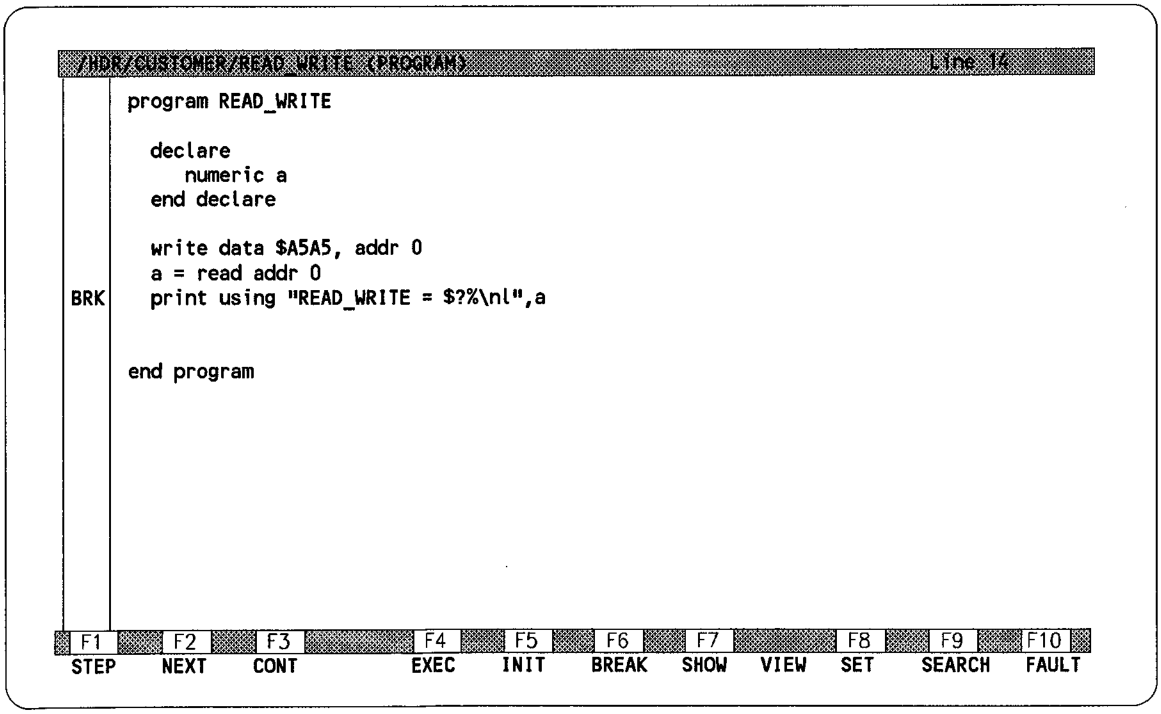
   A status column appears on left of screen when debugger has been loaded. New function key labels now become visible.

3. F9 (SEARCH)

   press... the F9 key - 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.



```
/HDR/CUSTOMER/MAIN  (PROGRAM)                                     Line 14

     program MAIN

     !*******************************************************************************
     ! This program is used to demonstrate the features of the TL/1 Debugger.
     ! At this point, it is not important to understand the TL/1 commands.
     !*******************************************************************************

       declare
          numeric a
       end declare

       write data $AA55, addr 0
       a = read addr 0
BRK    print using "MAIN DATA = $?%\nl",a

       execute read_write()
       execute bus_test()
       execute ram_test()
       print "END OF TEST"

     end program
```



```
/HDR/CUSTOMER/READ_WRITE (PROGRAM)                                Line 14

     program READ_WRITE

       declare
          numeric a
       end declare

       write data $A5A5, addr 0
       a = read addr 0
BRK    print using "READ_WRITE = $?%\nl",a


     end program
```

4. F6 (BREAK)

press... the F6 key 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 commands(s) on that line.

5. □ (VIEW)

press... the VIEW function key located between the F7 and F8 function keys.

The view softkey prompts for a program name and then displays the program.

At The View Program Prompt:

type... read_write

press... RETURN

You should be looking at the program read_write.

press... F9 (SEARCH)

Search for print and set a breakpoint

press... □ (VIEW)

type... main at the prompt

press... RETURN

You are now back to the MAIN program.

6. F4 (EXECUTE)

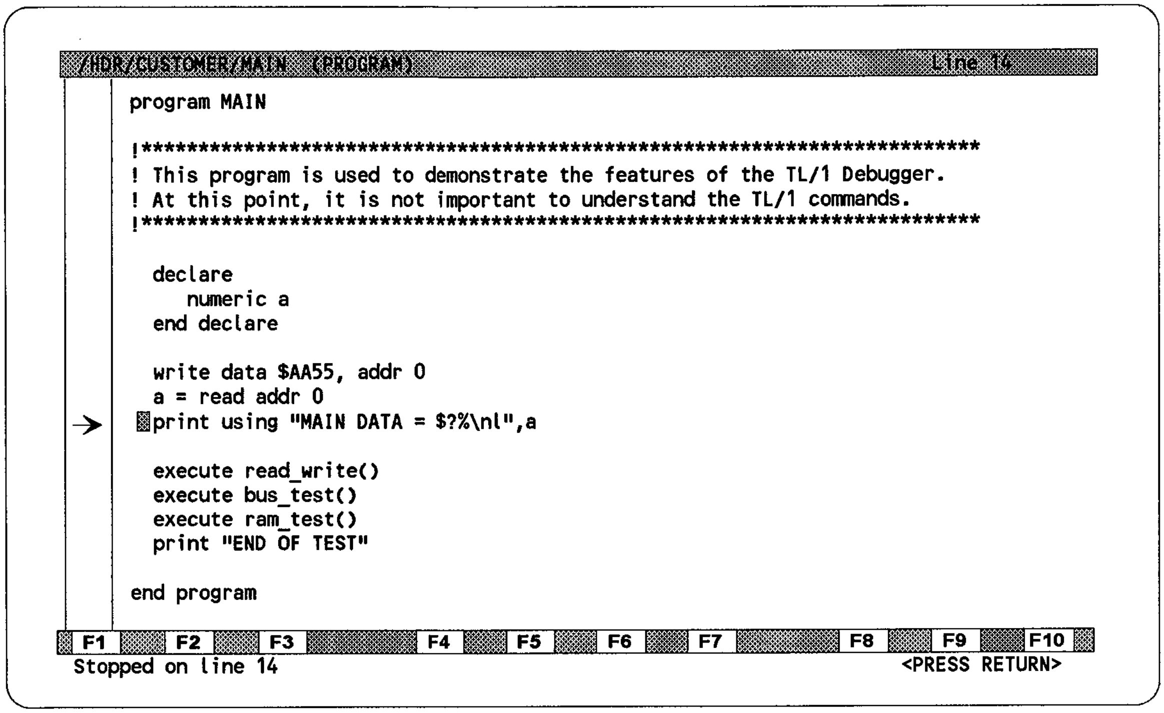
To begin program execution:

press... F4 (EXECUTE)

press... RETURN for the default filename MAIN

At line 14, a breakpoint is encountered and program execution is halted.

press... RETURN to view the function key labels.



```
/HDR/CUSTOMER/MAIN  (PROGRAM)                                      Line 14

      program MAIN

      !*************************************************************************
      ! This program is used to demonstrate the features of the TL/1 Debugger.
      ! At this point, it is not important to understand the TL/1 commands.
      !*************************************************************************

        declare
           numeric a
        end declare

        write data $AA55, addr 0
        a = read addr 0
 →     ▒print using "MAIN DATA = $?%\nl",a

        execute read_write()
        execute bus_test()
        execute ram_test()
        print "END OF TEST"

      end program
```

```
Stopped on line 14                                              <PRESS RETURN>
```

7. F7 (SHOW)

press... F7 (SHOW) a variable prompt appears.

type... **a**

press... RETURN a=43605 (decimal) will appear.

To see a value in hex, press the SHIFT and F7 simultaneously. The variable **a** should still be there.

press... RETURN

You should now see a=AA55 `(hex)`.

8. F8 (SET)

press... F8 (SET) allows you to assign a value to a variable.

type... **a** then press RETURN then type $55AA.
Note: ($) *indicates the number is a hex value.*

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

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

press... ↓ set a breakpoint a breakpoint at line 17. Repeat ↓ to set a breakpoint at line 18.

Note: Breakpoints may be set or cleared while program execution is halted.

press... F3 The Applications Display will read:
MAIN DATA = $55AA

The programmer's screen will display the word LOADING.... until execution halts at line 9 of the program read_write.

press... F3 again. This time the Application Display prints read_write data=$A5A5 and program execution halts at line 17 of the program main.

press... RETURN

10. F1 (STEP)

STEP causes the current TL/1 command to be executed, steps to the next command, then halts prior to it's execution.

Note: 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 Execution will halt on the first command line of the program **bus_test** (line 3).

press... F3 (CONT). Execution continues and halts at line 18 of program **main**.

11. F2 (NEXT)

NEXT causes the current TL/1 command line to be executed, steps to the next command line, then halts prior to it's execution.

Note: 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) Unlike the STEP function, the NEXT function did not allow the program to halt inside the called program.

press... F3 (CONT) to end the program.

12. F5 (INIT) clears all breakpoints and reset all variables.

press... F5 (INIT) Observe, all breakpoints are cleared and you can no longer SHOW the variable "a".

press... □ (VIEW) to view the program read_write and to verify the breakpoint set has also been cleared.

press... □ (VIEW) to 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)

    Toggles the fault window on and off.

    press... F10 (FAULT)

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

16. To continue program execution...

    press... F3 (CONT)

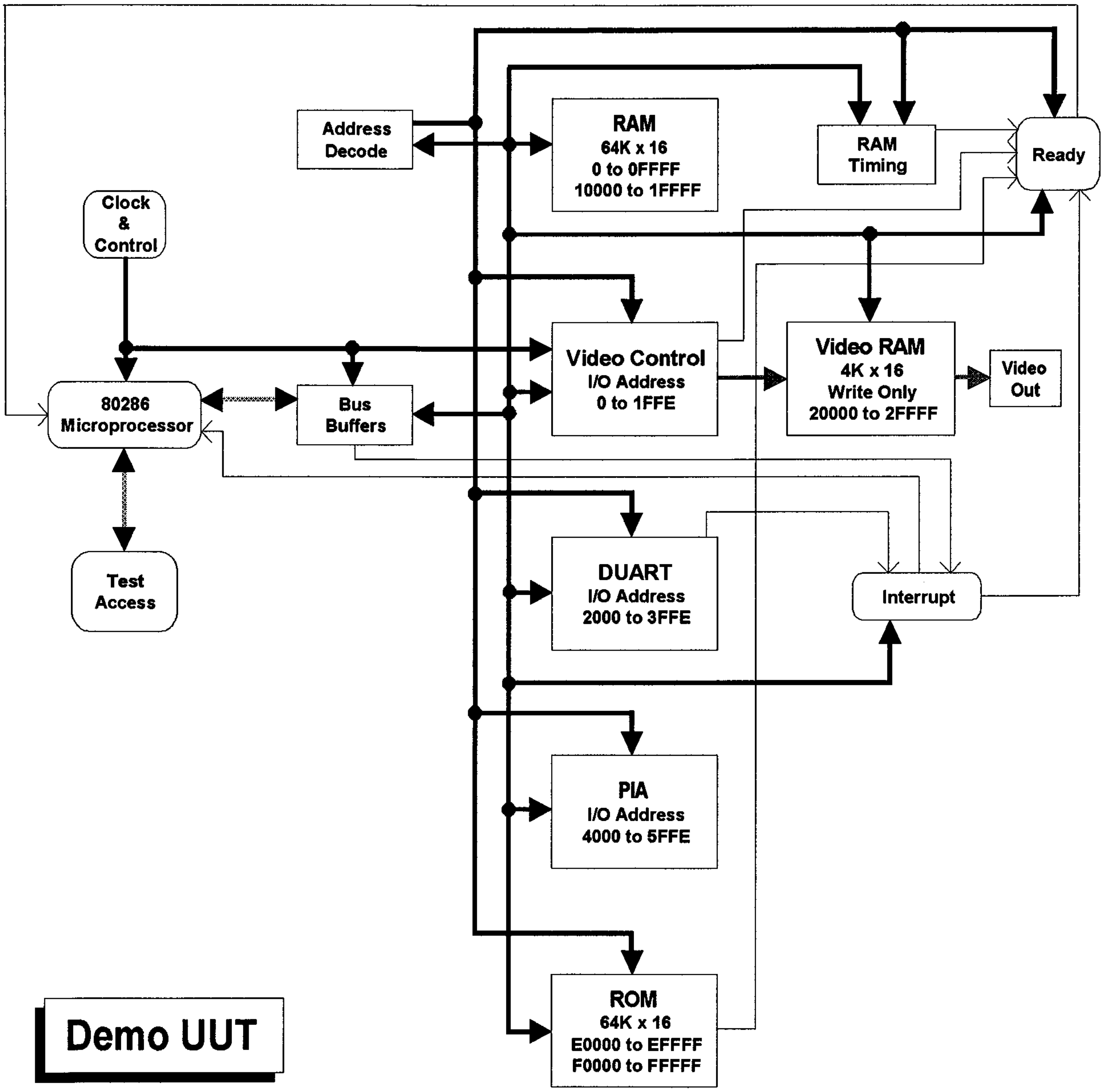
17. Leave the debugger and return to the Applications keypad.

Section 10

Functional Test

This section is a series of exercises designed to introduce the programming environment of the 9100FT. *The exercises will familiarize you with the operation of the:*

- ❍ Program debugger
- ❍ Built-in functions
- ❍ Measurement devices
- ❍ TL/1 language.


Address Decode
RAM
64K x 16
0 to 0FFFF
10000 to 1FFFF
RAM Timing
Ready
Clock & Control
Video Control
I/O Address
0 to 1FFE
Video RAM
4K x 16
Write Only
20000 to 2FFFF
Video Out
80286 Microprocessor
Bus Buffers
Test Access
DUART
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

BUS TEST

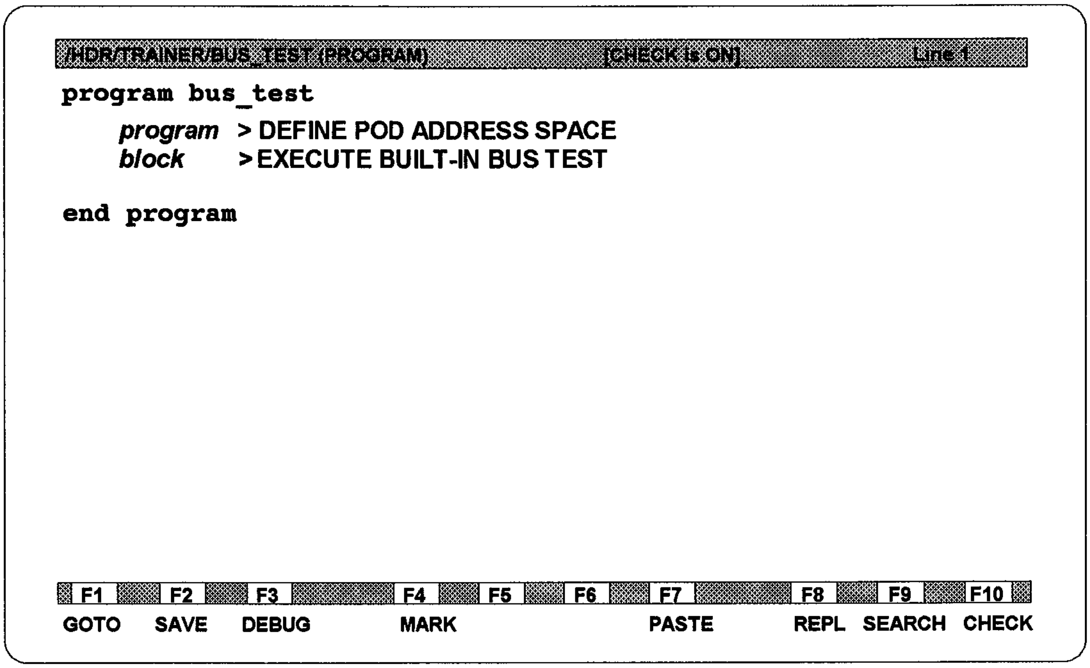
The built-in Bus Test is an easy, fast method of testing the UUT microprocessor bus and provides excellent diagnostic information to the user. Bus Test detects faults that can cause most all other functional tests to fail.

Bus Test checks.....

- ✓ Address
- ✓ Data
- ✓ Control Lines

......to be sure each can be driven high and low individually. 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/BUS_TEST (PROGRAM)            [CHECK is ON]            Line 1
program bus_test
    program  > DEFINE POD ADDRESS SPACE
    block    > EXECUTE BUILT-IN BUS TEST

end program
```

Bus Test Structure

The basic structure of the program used to test the bus functional area of the UUT is illustrated on page (10-4) to the left.

Exercise 10-1

DESIGN THE "BUS_TEST" PROGRAM - EXERCISE 10-1

① Edit the program "bus_test" for the UUT trainer.

Define the correct pod addressing space before entering the Bus Test Command. The address space option is pod dependent. This information can be found in the Pod Manual or the Supplemental Pod Manual.

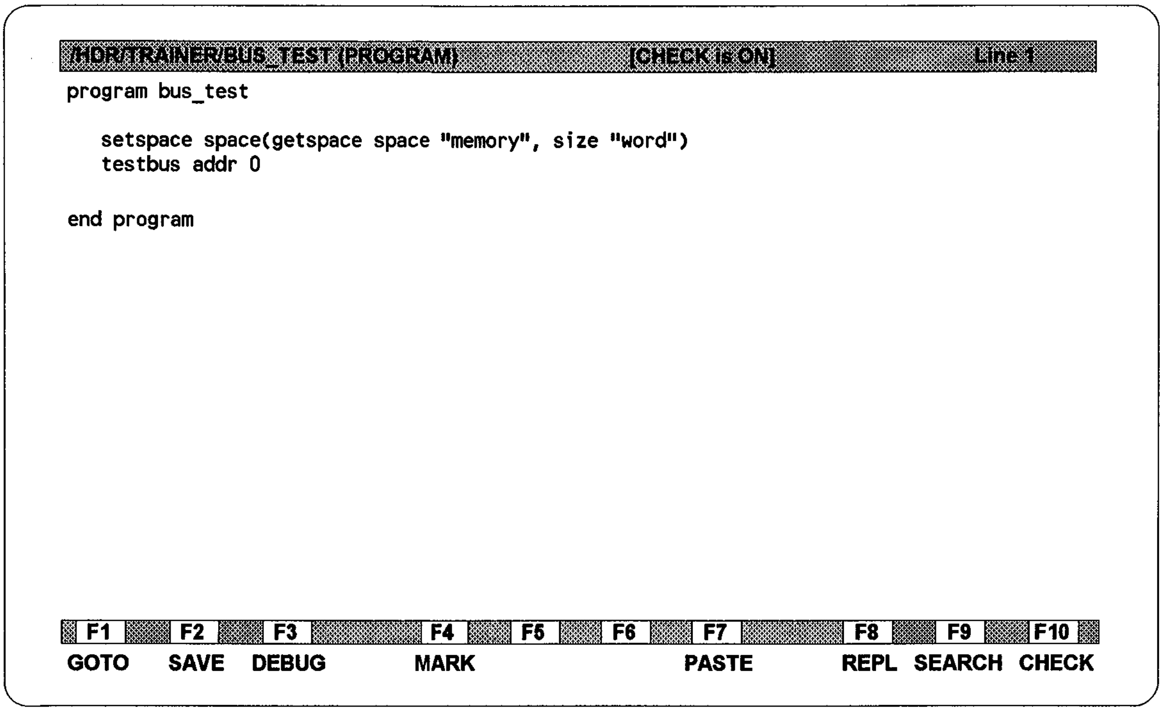
For exercises in this section, we will consider only the following 80286 pod address space option:

- ✓ 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 is the proper form of the getspace *command for initiating **"memory word access"** in the pod?*

The 9100FT retrieves the proper value from the Pod Library. The operator directs this number to the pod using the setspace command.



/HDR/TRAINER/BUS_TEST (PROGRAM) [CHECK is ON] Line 1

```
program bus_test

   setspace space(getspace space "memory", size "word")
   testbus addr 0

end program
```

2. **Set the pod to perform memory word accesses.** Use the getspace and setspace commands in the command line.

   Your program should look similar to the figure illustrated on page (10-6) to the left.

3. **Modify the setspace command line** using Positional Notation .

4. **Perform the actual Bus Test.**

   Enter the command that will perform the built-in Bus Test on the trainer UUT. The command asks for an address at which the test can perform the reads and writes. The address specified in the bus test should be a memory location that is read and writeable. Other locations should only be specified if they physically exist and are not written to by other devices.

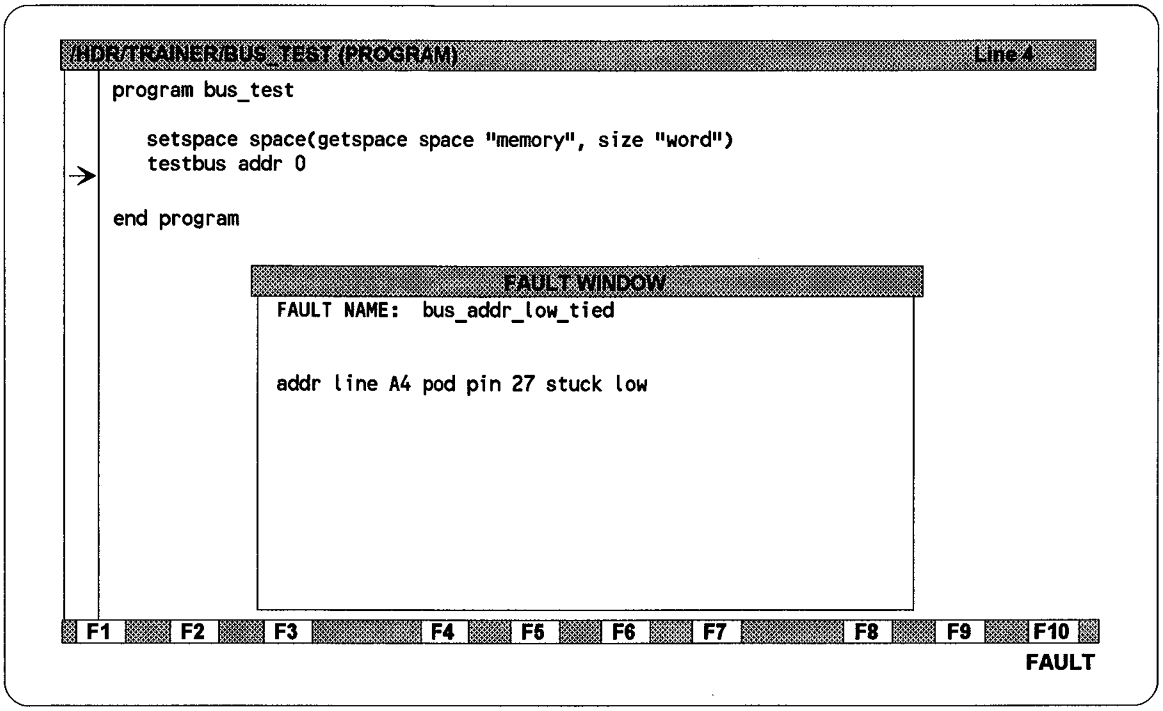
   Use address 0 in your testbus command line. Address 0 is the first address of RAM space on the UUT.

5. **Check your program for errors** using the F10 (CHECK) function.

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

   Execution should have completed with:

   ```
   status = PASS.
   ```



/HDR/TRAINER/BUS_TEST (PROGRAM) Line 4

```
program bus_test

   setspace space(getspace space "memory", size "word")
   testbus addr 0

end program
```

FAULT WINDOW

```
FAULT NAME:  bus_addr_low_tied


addr line A4 pod pin 27 stuck low
```

Bus Test Faults

Exercise 10-2

STUCK ADDRESS - EXERCISE 10-2

① Select fault switch SW2-5 (Set to ON).

② Execute the "bus_test" program and record the name and description of the fault.

Exercise 10-3

STUCK DATA - EXERCISE 10-3

① Select fault switch SW2-7 (Set to ON).

② Execute the "bus_test" program and record the name and description of the fault.

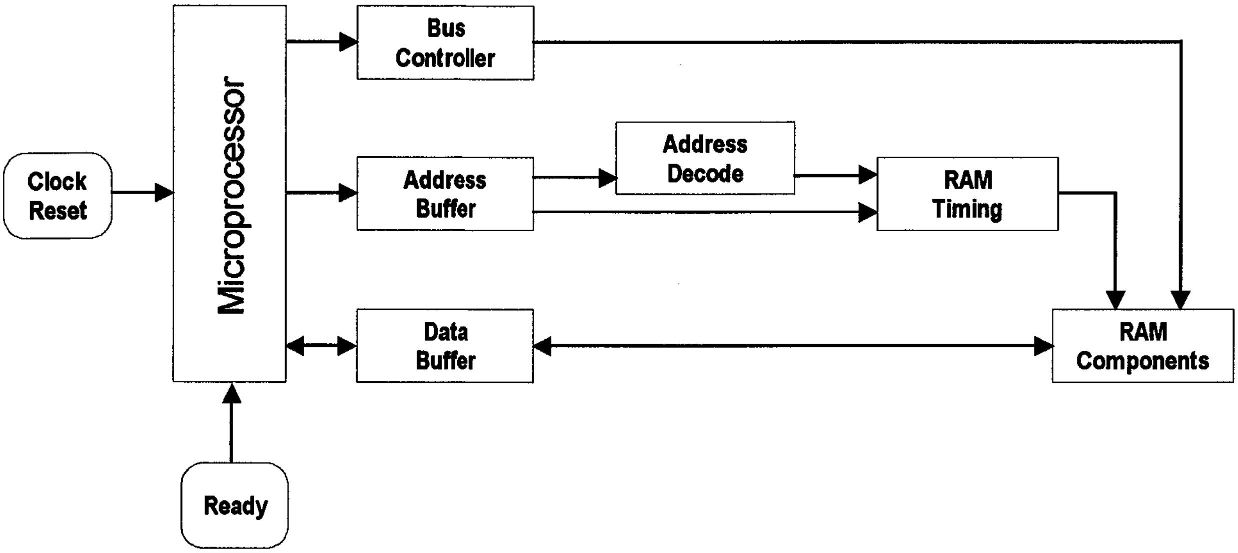
Exercise 10-4

FAULT - EXERCISE 10-4

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

① Press and ***hold down*** the [RESET] button on the Trainer UUT.

② Execute the "bus_test" program and record the name and description of the fault.


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

RAM Test

Exercise 10-5

DESIGN THE "RAM_TEST" PROGRAM - EXERCISE 10-5

Design a program "ram_test" under directory UUT/TRAINER, to test UUT RAM over the address range of 0-1FFFE using the testramfast command.

When performing the test:

- ✓ check all 16 data bits
- ✓ use the default delay
- ✓ set seed to 1

(Don't forget the Address Option)

Exercise 10-4

RAM DATA FAULT - EXERCISE 10-4

① Select Fault SW1-5 (Set to OFF)

② Execute "ram_test" and record the name and description of the fault.

Exercise 10-7

RAM ADDRESS FAULT - EXERCISE 10-7

① Select Fault SW4-8 (Set to ON)

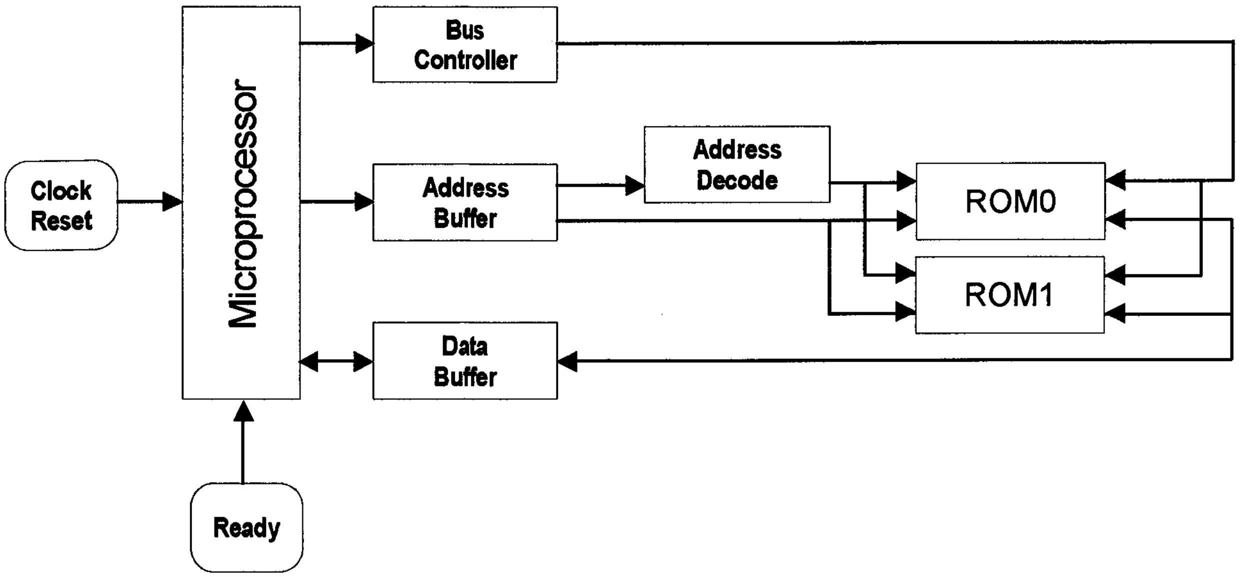
② Execute "ram_test" and record the name and description of the fault.

Exercise 10-8

RAM CANNOT MODIFY FAULT - EXERCISE 10-8

① Select Fault SW1-4 (Set to OFF)

② Execute "ram_test" and record the name and description of the fault.


Microprocessor
Bus Controller
Address Buffer
Data Buffer
Address Decode
ROM0
ROM1
Clock Reset
Ready

ROM Test

Exercise 10-9

DESIGN THE "ROM_TEST" PROGRAM - EXERCISE 10-9

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

Exercise 10-10

DATA STUCK - EXERCISE 10-10

① Select SW3-1 (Set to ON)

② Execute the "rom_test" program and record the name an description of the fault.

Exercise 10-11

ALL DATA STUCK - EXERCISE 10-11

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

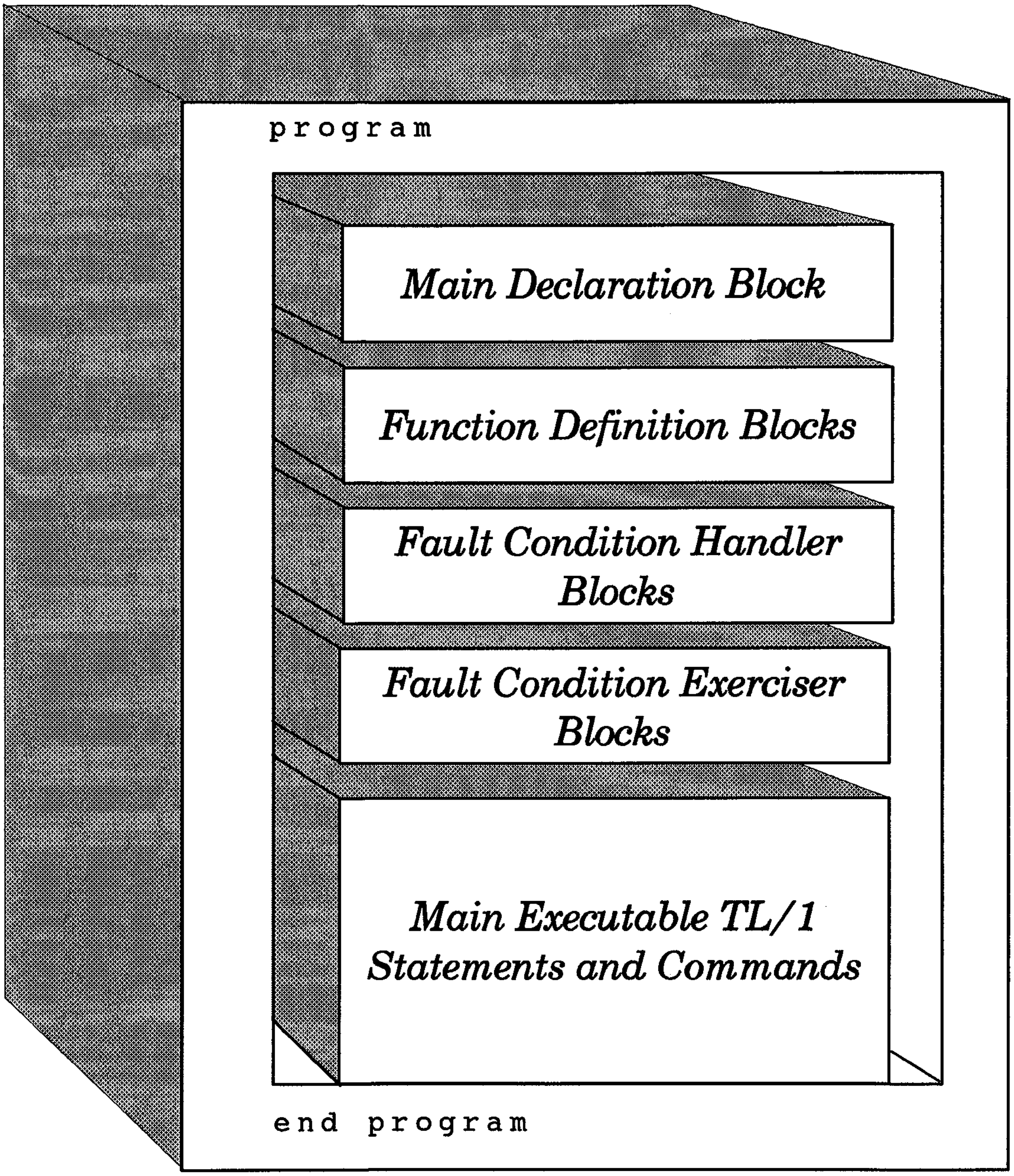
① Begin by selecting SW3-7 (Set to ON)

② Execute the "rom_test" 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 which simplify the process of writing test and troubleshooting programs are available.*


program
Main Declaration Block
Function Definition Blocks
Fault Condition Handler Blocks
Fault Condition Exerciser Blocks
Main Executable TL/1 Statements and Commands
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
```

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)

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 hexadecimal notation. Hexadecimal 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"
```

```
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
        !numeric x
     end declare

          !Run this program in debug with both variables
          !commented out then run it with only one
          !commented out and then run it again 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 (i.e. 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 that 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 status between 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
not	Logical negation

Logical Operators

& and	Logical AND
| or	Logical OR
^ xor	Logical exclusive OR
~ cpl	One's complement

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 representation 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 significant 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
return
abort
goto / label
wait

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 (e.g. operator display/keypad, 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 (i.e. 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 returns a numeric value that represents the "channel" to be input from. In this example the channel points to /term2 (the programmers keyboard). The numeric variable in_chan must be used within each input command statement. 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 inputting 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 executing instructions in the `runuut` mode.

runuutvirtual — causes the UUT to begin executing instructions from it's own memory, asynchronously to the system.

Prompting the Operator

connect
clip
assign
probe
assoc

Termination Status

fault
passes/fails
refault
handle / end handle

GFI FUNCTIONS

gfi hint	adds a node to the end of the GFI hint list.
gfi suggest	returns the next node in the GFI hint list.
gfi clear	Erases GFI summary and GFI suggestion list.
gfi control	determines whether a program is being executed under GFI control.
gfi device	returns the name of GFI measurement device.
gfi test	executes all stimulus routines associated with a pin or device.
gfi status	returns the status of a node.
gfi accuse	returns string containing GFI accusation 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	returns the name of the node being tested.
dbquery	returns information 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

defines a reference designator. It associates a reference designator with a window location and a part shape previously defined by a define part command. To display the reference designator, use a draw command.

define text

defines 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

draws previously defined UUT components in a window on 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

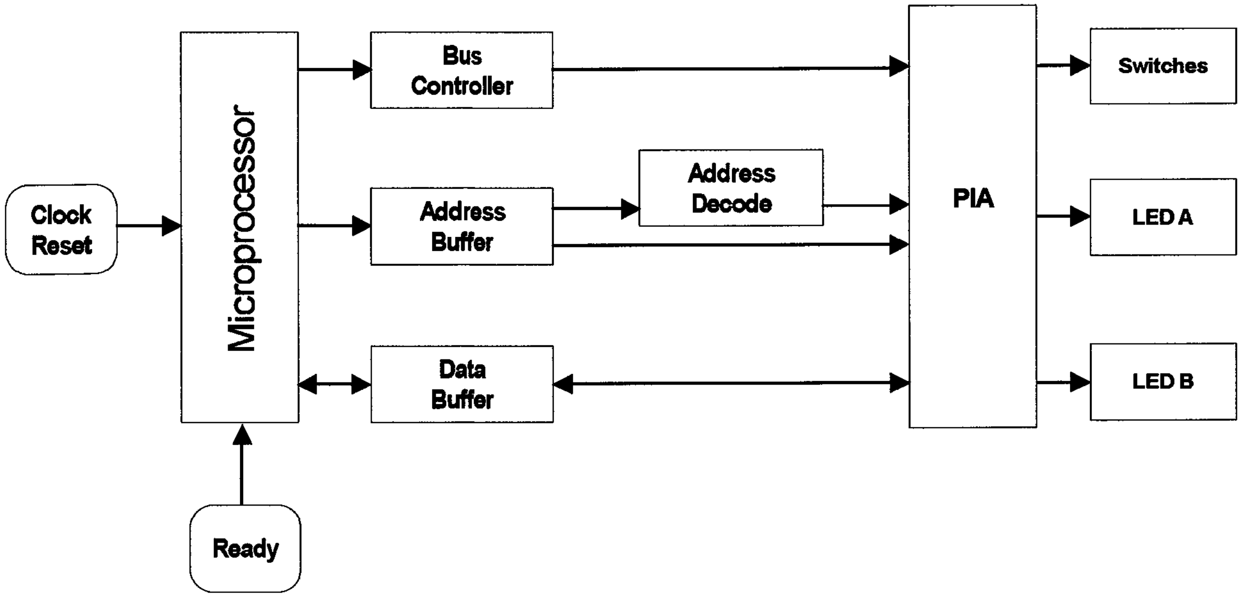
displays and reads from a menu defined by the define menu command.

remove

removes definitions made by a define command.

Section 12

PIA Functional Test


Clock
Reset
Microprocessor
Ready
Bus
Controller
Address
Buffer
Data
Buffer
Address
Decode
PIA
Switches
LED A
LED B

PIA FUNCTIONAL TEST

Exercise 12-1

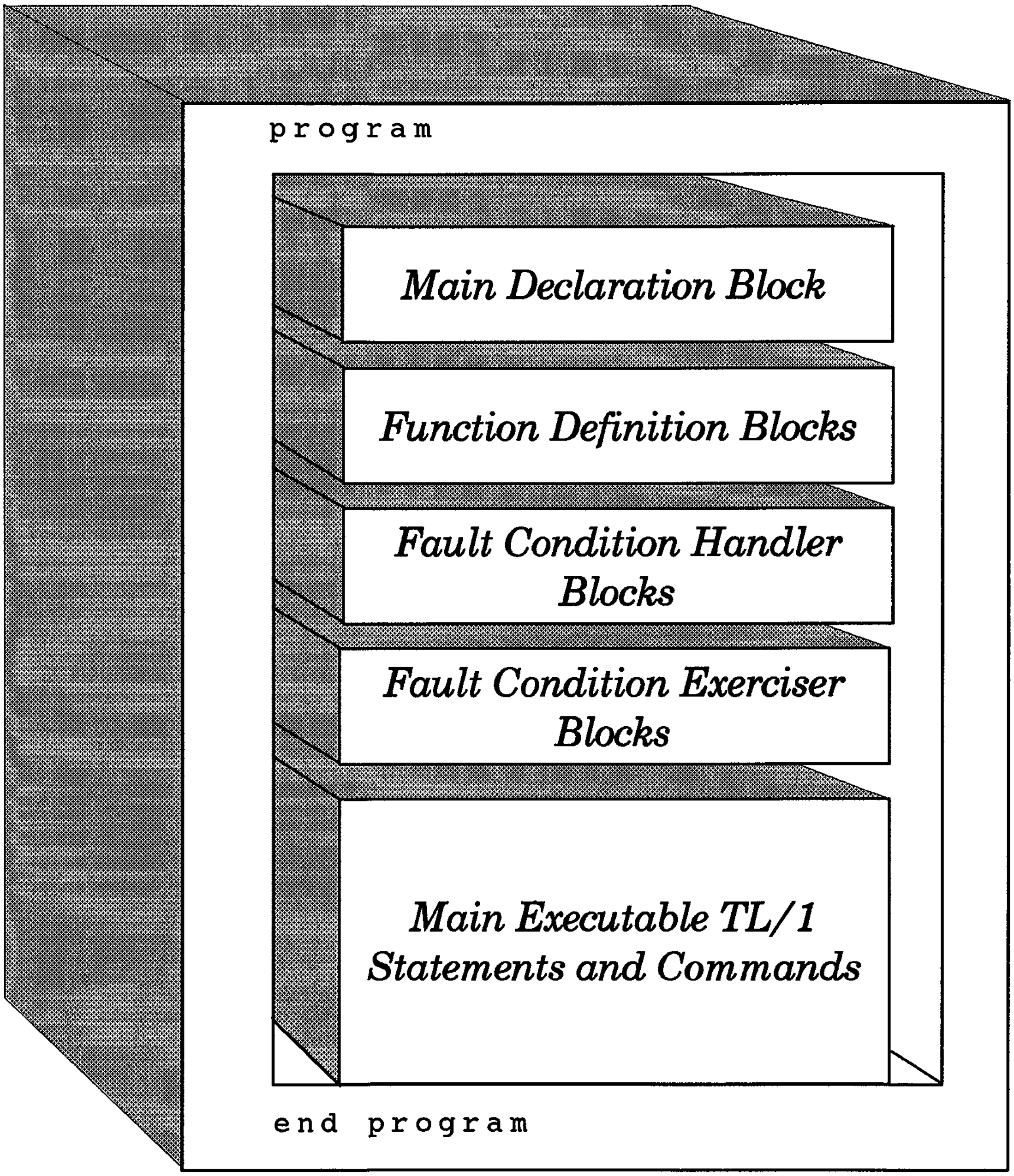
PIA FUNCTIONAL TEST - EXERCISE 12-1

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
Main Declaration Block
Function Definition Blocks
Fault Condition Handler Blocks
Fault Condition Exerciser Blocks
Main Executable TL/1 Statements and Commands
end program

HANDLERS EXERCISE

Exercise 13-1

HANDLERS - 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 9100FT 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 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 (9100FT)

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, test 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 Star, 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 of 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 9100FT 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 9100FT 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 considerable 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, re-programmable 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 9100FT 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 sequences 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 9100FT to perform tests on a UUT.

TRI-STATE

Being in a high impedance state.

UUT (UNIT UNDER TEST)

The circuit board or system being tested by the (9100FT).

USERDISK

In the 9100FT, 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 changes 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 9100FT 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.

Appendix B

Technical Data Sheets

TITLE	NUMBER
9100A Series Digital Testers	A0266C
9132A Memory Interface Pod	A0321A
Microprocessor Pod Adapters	A0355B
Vector Output I/O Module	A0357A
9100A Synchronizing I/O Module and Probe with "Int" Mode	A0367A
9132A-68030 Processor Support Package	A0372A
9132A-64180 Processor Support Package	A0373A
9132A-68020 Processor Support Package	A0374A
9132A-80386 Processor Support Package	A0379A
9132A-80386SX Processor Support Package	A0380A
9132A-68000 Processor Support Package	A0384A
9132A-80486 Processor Support Package	A0385A
9132A-7810 Processor Support Package	A0386A
9132A-80286 Processor Support Package	A0387A
9100A-030 9010 to 9100FT Programming Translator	A0392A
IEEE 488 / Plus Option 9105A-015 / 9100A-015 UGK	A0393B
9132A-68HC11 Processor Support Package	A0406A
Using Multiple I/O Modules For Driving Patterns and Gathering Signatures	L0045A
Putting a Software Switch into your GFI Database	L0046A

Appendix c

Application Notes

TITLE	NUMBER
Determining the suitability of Digital Designs for testing with the 9132A Memory Interface Pod	B0186A
9100FT Testing of a CGA Card with the Vector I/O Module	B0192A
Developing Test Vectors for a Video Controller Chip	B0200A
Testing DMA Circuitry on the IBMPC-AT	A0350A
Adding IEEE-488 Instruments to the 9100FT Digital Test System	A0349A
9100FT Production Testing	L0030A