

S-3270
Automated Test System

**S-3270
TEKTEST III
TEST LANGUAGE,
PART TWO**

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

070-3339-00

SEMICONDUCTOR TEST SYSTEMS (STS)
MEASUREMENT SYSTEMS DIVISION

FIRST PRINTING - MAY 1978

SOFTWARE SUPPORT POLICY

Unless otherwise provided, this software is furnished on an "as is" basis. Service, if available for this software, will be provided at the rates in effect at the time service is requested.

SOFTWARE LICENSE

Software supplied by Tektronix, Inc., as a component of a system or as a separate item is furnished under a license for use on a single system and can be copied (with the inclusion of copyright notice) only for use on that single system.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Specification and price change privileges are reserved.

Copyright © 1978 by Tektronix, Inc., Beaverton, Oregon. Printed in the United States of America. All rights reserved. Contents of this publication may not be reproduced in any form without permission of Tektronix, Inc.

PREFACE

This manual describes Part Two of the TEKTEST III Test Language. It contains hardware-oriented statements which are used with the test language statements described in Part One to write test programs for the S-3270 Automated Tests Systems. A test program is written in the Text Editor (EDIT) and translated with the Program Translator (TRAN).

TEKTEST II/III Test Language Part One (062-3375-01) discusses basic concepts such as the general form of statements, specifying numeric constants, writing constants and expressions, and calling subprograms. It also tells how to translate a test program. Part One covers all systems of the S-3200 family; each system type has its own Part Two.

On a S-3270 System, the hardware-oriented statements control the test station, Delta-T Subsystem, DC Subsystem, Programmable Clock Generator, and MC-3 Monitor/Power Supply Option. Refer to *Device Testing Fundamentals* for an overview of functional testing, parametric testing, sector-card operating modes, and the clock generator.



CONTENTS

NOMENCLATURE CONVENTIONS	vii
SELECTING THE TEST STATION ENVIRONMENT	ix
IDENTIFYING THE ACTIVE TEST STATION	x
SECTION 1: DEFINING A PINLIST	
DEFINING A PINLIST	1-1
The PINLIST Statement	1-1
Statement Format Conventions	1-3
IDENTIFYING THE SPECIFIED PIN ELECTRONICS CARD	1-4
The IDENTIFY Function	1-4
SECTION 2: TIMING	
TEST STATION PHASE DISTRIBUTION	2-1
ESTABLISHING THE TIMING REFERENCE	2-4
The CYCLE Statement	2-4
DUT Cycle Limit and Range Restrictions	2-5
Timing Force Data	2-6
PROGRAMMING THE CLOCK PHASES	2-7
The PHASE Statement	2-7
CLOCK PHASE LIMITS	2-9
Pulse Start Time	2-9
Pulse Width	2-9
DUT Cycle Frequency	2-10
Summary of Ranges	2-11
PHASE Statement Modification	2-12
CAUSING A TRIGGER PULSE	2-15
The TRIGGER Statement	2-15
CAUSING A 1 μ s PULSE AT THE REAR PANEL OF THE TSCU	2-16
The TSTRIG Statement	2-16
THE REAL TIME COUNTER	2-17
The CLOCK Variable	2-17
SECTION 3: STIMULUS	
SETTING FORCE VOLTAGES	3-1
The HIDRIVE Statement	3-1
The LODRIVE Statement	3-1
SETTING COMPARE VOLTAGES	3-3
The HICOMPARE Statement	3-3
The LOCOMPARE Statement	3-3

CONNECTING AND DISCONNECTING THE SECTOR-CARD DATA PATHS (1804 TEST STATION ONLY)	3-6
The CONNECT Statement	3-6
The DISCONNECT Statement	3-6
SELECTING THE INITIAL OR PROGRAM CLOCK PHASES	3-15
The CONNECT TO PHASE Statement	3-15
The CONNECT TO DATAPHASE Statement	3-15
ROUTING DATA VIA THE UNDERSOCKET CARD	3-16
The UNDERSOCKET Statement	3-16
INITIALIZING THE TEST STATION	3-17
The INITIALIZE Statement	3-17

SECTION 4: FUNCTIONAL TESTING

FORCING LOGIC LEVELS	4-1
The FORCE Statement	4-1
COMPARING LOGIC LEVELS	4-9
The COMPARE Statement	4-9
INHIBITING THE DRIVER	4-10
The INHIBIT Statement	4-10
DISCONNECTING THE COMPARATOR OUTPUT	4-12
The MASK Statement	4-12
TRANSFERRING DATA	4-14
The LOAD Statement	4-14
MOVING THE PATTERN TO AND FROM THE DUT	4-18
The MOVE Statement	4-18
RETURNING THE CURRENT STATE OF THE TIMEOUT SYSTEM FLAG	4-37
The TIMOUT Function	4-37
DETERMINING THE CLOCK GENERATOR MODE	4-38
The BURST Statement	4-38
READING THE FORCE FLIP-FLOPS	4-39
The DRIVE Function	4-39

SECTION 5: ERROR CONTROL

INDICATING FUNCTIONAL TESTING STATUS	5-1
The ERROR Variable	5-1
POINTING TO THE PATTERN ROWS PRESENTLY AVAILABLE AT THE DUT	5-3
The INDEX Variable	5-3
RENUMBERING THE SHIFT REGISTER ROWS	5-4
The INDEXP Statement	5-4
TESTING FOR AN ERROR CONDITION	5-5
The WHEN ERROR Statement	5-5
DISABLING THE WHEN ERROR STATEMENT	5-7
The CLEAR ERROR Statement	5-7
READING THE ERROR FLIP-FLOPS	5-8
The PINERR Function	5-8
The PINPAS Function	5-8
STOP-ON-ERROR CIRCUITS (ADDITIONAL OPERATING PRINCIPLES)	5-9

SECTION 6: PARAMETRIC TESTING

SETTING AND UNSETTING THE DC AND ΔT SUBSYSTEMS	6-1
The DC and ΔT Subsystem Ranges	6-3
MEASURING VOLTAGE	6-7
The SETUP and UNSET TO MEASURE VOLTAGE Statements	6-7
MEASURING CURRENT	6-11
The SETUP and UNSET TO MEASURE CURRENT . . . FROM DRIVER Statements	6-11
MEASURING CURRENT FROM A POWER SUPPLY	6-15
The SETUP and UNSET TO MEASURE CURRENT . . . FROM VS _n Statements	6-15
MEASURING TIME	6-18
The SETUP and UNSET TO MEASURE TIME Statements	6-18
FORCING VOLTAGE	6-23
The SETUP and UNSET TO FORCE VOLTAGE Statements	6-23
FORCING CURRENT	6-25
The SETUP and UNSET TO FORCE CURRENT Statements	6-25
ASSIGNING MEASUREMENT RESERVED VARIABLES	6-27
The CURRENT Variable	6-27
The VOLTAGE Variable	6-27
The TIME Variable	6-27
SETTING THE 1140A POWER SUPPLIES	6-29
The VS1 through VS4 Statements	6-29
SETTING THE 1140A CURRENT SUPPLIES	6-30
The IS1 and IS2 Statements	6-30

SECTION 7: TSCU OPERATOR INTERACTION

USER-CONTROLLED TESTING	7-1
The ADVANCE Variable	7-1
DISPLAYING TEST RESULTS	7-3
The DISPLAY Statement	7-3
DISPLAYING TEST RESULTS AND INCREMENTING THE INTERNAL COUNTERS	7-5
The SORT Statement	7-5

SECTION 8: DATA LOGGING

LOGICAL UNIT NUMBERS	8-1
Format Conventions	8-1
SUMMARY OF DATA LOGGING STATEMENTS	8-2
READING SAVED ERROR INFORMATION	8-5
The READREG Function	8-5
READING PATTERN DATA	8-6
The SREAD Subroutine	8-6

APPENDIX A: RECIRCULATION AND PARALLEL CHAINING

Recirculation	A-1
Parallel Chaining	A-3
MASK Patterns	A-4
Alternate Data	A-4

APPENDIX B: TRANSLATOR ERROR MESSAGES FOR
TEKTEST II/III HARDWARE-ORIENTED STATEMENTS B-1

APPENDIX C: STATEMENT SUMMARY C-1

NOMENCLATURE CONVENTIONS

This manual uses a standard nomenclature to show the general form of each statement and its elements. The nomenclature conventions are:

Elements shown in upper case letters, special characters, and punctuation marks (including blanks) are *literal* elements. When you use them with the statement, you must type them exactly as shown in the general form.

Elements shown in lower case letters are *variable* elements. When you use them with the statement, you must supply a valid name or value in place of the variable name appearing in the general form. For example, the variable name *pinnum* indicates that you must supply a pin number.

Elements enclosed in square brackets ([]) are *optional* elements. You may supply these elements or not, depending on the way you wish to use the statement. (Since the brackets are a nomenclature convention only, you must not type them when you use the statement.)

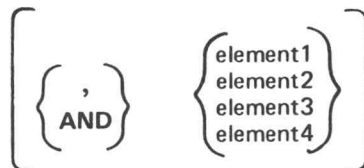
A vertical list of elements enclosed in braces ({}) indicates that you must choose one line from the list when you use the statement. Which element you choose depends on the function you wish the statement to perform. (Since the braces are a nomenclature convention only, you must not type them when you use the statement.)

A vertical list of elements enclosed in square brackets indicates that the element is optional. If you decide to use the element, you must select one line from the vertical list shown. Which element you choose depends on the function you wish the statement to perform.

Elements not enclosed in square brackets or braces are *mandatory* elements – you must supply the element when you use the statement.

When the general form shows the same element twice, separated by an ellipsis (i.e., element, . . . ,element), you may enter the element once or repeat it as many times as desired.

When elements are nested within square brackets and braces, you interpret the brackets and braces by working from the outermost pair of brackets or braces to the innermost pair. For example,



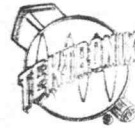
In the above example, the outermost brackets indicate that any elements which are enclosed within the brackets are optional elements. The inner braces indicate that, if you decide to specify the optional elements, you must select one line from each vertical list shown.

This manual assumes that you type a carriage return after each line you type at your terminal. Whenever there is any doubt about the necessity of the carriage return, it is indicated by the symbol ↵. For example,

* ↵

In the above example, the ↵ symbol indicates that the user must type a carriage return after the system prints the asterisk at the terminal.

Throughout this manual, the examples show user-typed information in **boldface**. Information the system prints at your terminal is shown in lightface.



Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

Phone: (503) 644-0161
TWX: 910-467-8708

ADDITIONAL STATION CONTROL DISPLAY ERROR CODES . . . to be incorporated into
Command Language Reference Guide, 062-3315-XX.

The following information should appear under the "A and B Errors - System Errors" heading
in Section 5.

TSCU DISPLAY	MEANING
AA	SHMOO Mode hardware conflict
AB	LIBRARY Mode hardware conflict
B1	LIBRARY file not found
A9	.MTL Mode hardware conflict

Note that the A9 Error Code is a revision of the definition already in the manual.

The following Error Code should be appended to the "A Errors - System Errors" section.

FF	IP3455 Software error - invalid state
----	---------------------------------------

SELECTING THE TEST STATION ENVIRONMENT

The **ENVIRONMENT** statement selects the environment in which you plan to run the test program. This is an optional statement.

The **ENVIRONMENT** statement format is:

ENVIRONMENT IS {
1803
1804
1805
1843
BACKGROUND } [,7PHASE
,14PHASE]

Selecting the environment in which the test runs causes the Translator to flag statements, which are illegal for that environment, with the error message **ILLEGAL ITEM**, and changes the meaning of other statements. If illegal statements exist in the test program and the environment is not specified, the Translator does not flag the statements and the test program halts during test execution.

Omitting the environment statement causes the Translator to select the default environment. Most test programs should use the default environment. The environment statement would be used on dual station systems.

ENVIRONMENT IS 1803 specifies that the device under test (DUT) is being tested on the 1803 Test Station. In this situation, all statements (except the 1840 connection statements), reserved variables, and subprograms described in this manual are legal.

ENVIRONMENT IS 1804 specifies that the DUT is being tested on the 1804 Test Station.

ENVIRONMENT IS 1805 specifies that the DUT is being tested on the 1805 Test Station.

ENVIRONMENT IS 1843 specifies that the DUT is being tested on the 1843 Remote Test Fixture. A special set of **CONNECT** statements (rather than the **CONNECT** statements in Section 3) must be used. *Preparing and Programming the 1843 Remote Test Fixture* (062-3354) discusses these statements. All other statements and reserved variables in this manual are useable.

The **7PHASE** option specifies that the station has a 7-phase clock generator; the Translator will accept 7-phase syntax, and will flag 14-phase syntax as an error. Similarly, the **14PHASE** option specifies that the station has a 14-phase clock generator; the Translator will accept 14-phase syntax, and will flag 7-phase syntax as an error.

ENVIRONMENT IS BACKGROUND specifies that the program is to be run under control of the **REDUCE** program. (See the *Processing Data* manual.) None of the statements or subprograms described in this manual should be programmed. The reserved variable **ADVANCE** may be used. **CTRL/V** simulates pushing the Test Station Control Unit (TSCU) **ADVANCE** button.

IDENTIFYING THE ACTIVE TEST STATION

The need to identify the active test station* occurs when the programmer wishes to write a test program that can run from two or more test stations whose configurations differ. The **STATN** function reads the active test station's number.

The **STATN** function call is:

STATN

The function declaration is:

FUNCTION STATN(0):STATN

For example, when a system has an 1804 and an 1805, the programmer should write the test program so that when the 1804 is active the program skips the 1805 portion. The programmer can also write the test so that it continues only if called from the proper test station.

Example:

```
10.0100 IF(STATN EQ 1) 10.0300,10.0200
10.0200 STOP
10.0300 CONTINUE
```

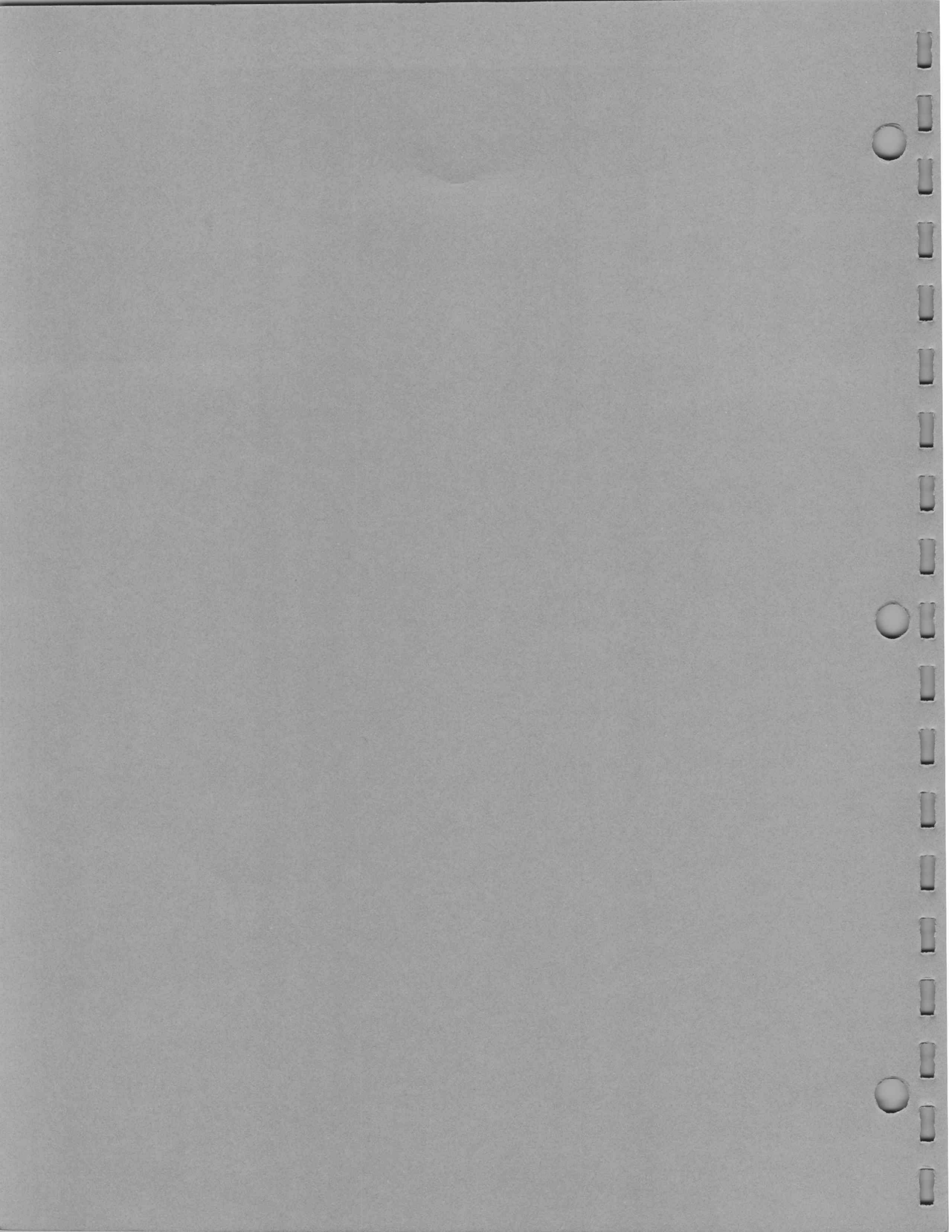
In this example, the program halts if the active test station is other than 1.

*A test station becomes active upon receiving a **START** signal from the Test Station Control Unit.

SECTION 1: DEFINING A PINLIST

This section describes the PINLIST statement and the IDENTIFY function. PINLIST groups under one name individual pins defined in the pin assignment table. See the *Pin Assignment Program* manual for further information.

- **PINLIST** Groups under one name individual pins defined in the pin assignment table
- **IDENTIFY** Reads the identification of the specified pin electronics card

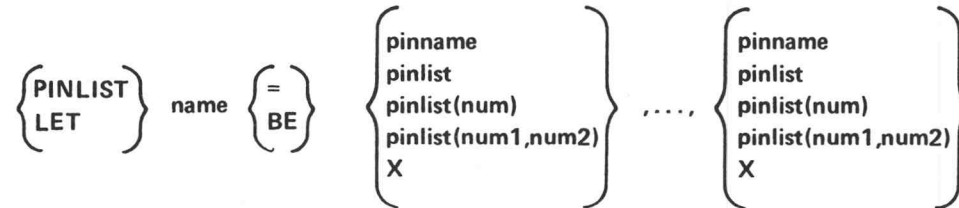


DEFINING A PINLIST

The **PINLIST** statement defines a pinlist. A pinlist groups pins which are associated by some common usage in the test program. A pinlist may be composed of any combination of the following:

- A group of pins defined in the Pin Assignment Program;
- A group of previously defined pinlists; and
- Any portion of a previously defined pinlist.

The **PINLIST** statement format is:



name is the name of the pinlist this statement defines. It consists of one to six alphanumeric characters, excluding special characters, of which the first must be a letter.

pinname is any pin name defined in the Pin Assignment Program that is associated with the test program.

pinlist is any previously defined pinlist name in this test program. **num**, **num1**, and **num2** are positive integer indexes (non-zero). Specifying **pinlist** without any index includes all the pins in **pinlist** in the new pinlist definition.

pinlist(num) selects a pin from **pinlist** to be included in the new pinlist definition. This is a singly indexed pinlist.

pinlist(num1,num2) selects a range of pins from **pinlist** to be included in the new pinlist definition. This is a doubly indexed pinlist.

During a functional test, the first column of the pattern selected by a MOVE or LOAD statement is assigned to the first pin in the pinlist selected by that particular MOVE or LOAD statement, the second column to the second pin, and so forth. The column-skip character, X, indicates that a pattern column is not assigned to a pin.

The PINLIST statement has a line continuation character, /. Use this character to continue a pinlist definition that cannot be contained on a single line.

When you translate a test program, TRAN queries:

PIN ASSIGNMENT TABLE(PIN):

In response, enter the name of a file that contains the pin assignment table related to the test program being translated. The pin assignment table must define all pin names specified in the test program.

Statement Format Conventions

This manual uses the following conventions to simplify statement formats. The mnemonics define what you can specify in the statement.

- pinname** — A pin name
- pinnames** — A group of pin names
- pinlist** — A pinlist name
- pin1** — A pin name or a singly indexed pinlist name
- pins** — A pin name, a group of pin names, a pinlist name, a singly indexed pinlist name, or a doubly indexed pinlist name (see the example below)

These conventions do not indicate what types of pins may be specified (I, O, IO, or any combination).

Also, these conventions do not indicate any difference between two groups of pins. For example, if a statement contains **pins TO pins**, each **pins** refers to a different group of pins.

Example:

If a test program has the pin names PINI1, PINI2, PINI3, PINO1, PINO2, and PINIO defined in its pin assignment table and the pinlist definitions:

```
PINLIST INPUT = PINI1,PINI2,PINI3,PINIO
PINLIST OUTPUT = PINO1,PINO2,PINIO
PINLIST ONE = INPUT(1),OUTPUT(1)
PINLIST ALL = INPUT,OUTPUT(1,2)
```

Then, INPUT(1) is a singly indexed pinlist. OUTPUT(1,2) is a doubly indexed pinlist.

```
PINLIST ONE would contain pins PINI1 and PINO1.
PINLIST ALL would contain pins PINI1, PINI2, PINI3, PINIO, PINO1, and PINO2.
```

IDENTIFYING THE SPECIFIED PIN ELECTRONICS CARD

The **IDENTIFY** function reads the identification of the specified pin electronics card. The VERDICT program uses IDENTIFY to decide on the pin electronics card type.

Value Returned	Pin Electronics Card Type
0	Card for this pin is missing
7	D70

Other values are assigned to other card types or reserved for future use.

The IDENTIFY function call is:

IDENTIFY(pin1)

The function declaration is:

FUNCTION IDENTIFY(T1):SECDAT

SECTION 2: TIMING

The statements discussed in this section pertain to the timing network of the test station. The programmable clock generator controls the timing of the entire system. It performs three functions:

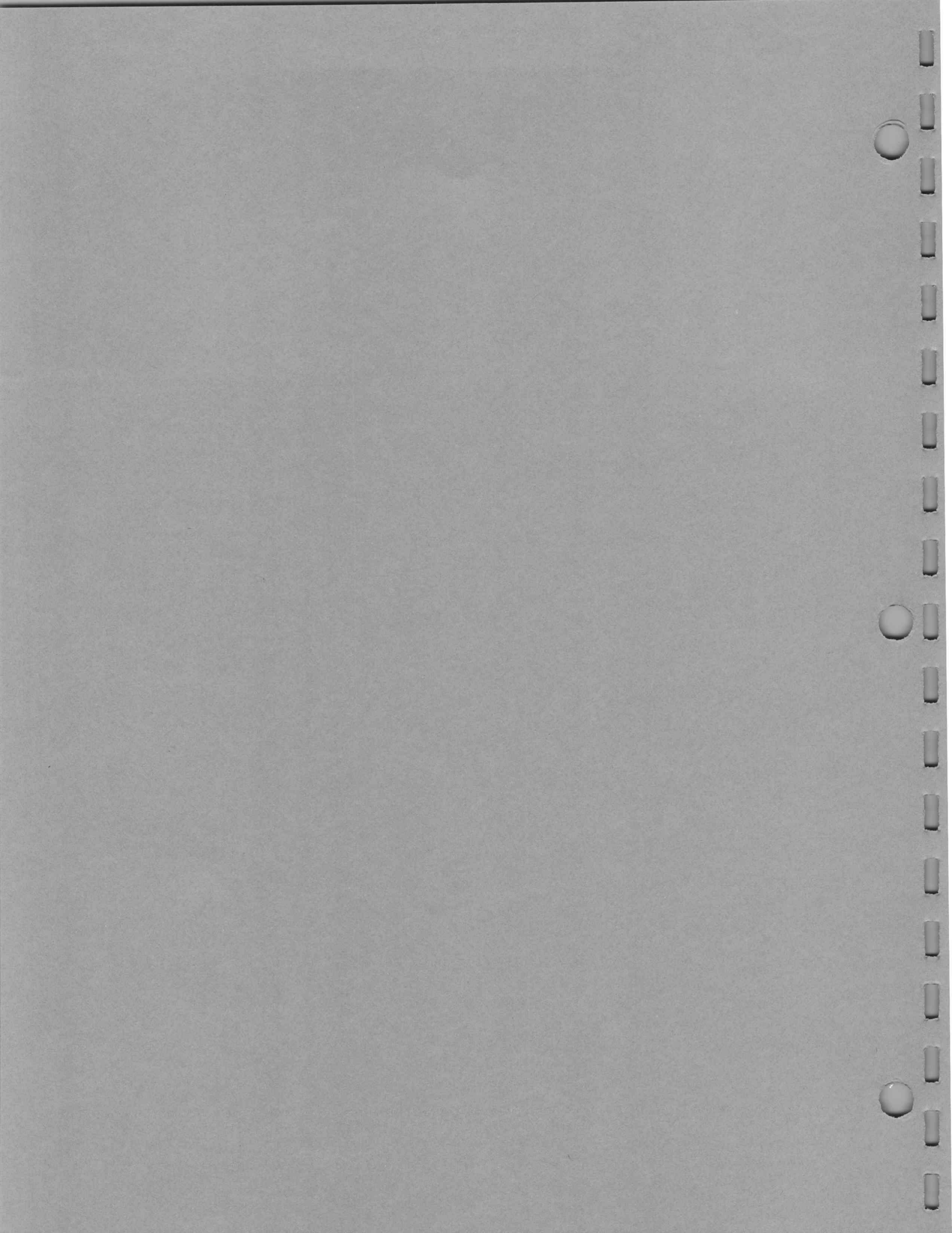
1. Clocks data through the pin electronics card shift registers;
2. Clocks force data; and
3. Clocks compare data.

The clock generator supplies 7 or 14 independently positionable clock phases to the test station. On seven-phase systems, they are called Phase 1 ($\phi 1$), Phase 2 ($\phi 2$), Phase 3 ($\phi 3$), Phase 4 ($\phi 4$), Hicompare ($\phi 5$), Locompare ($\phi 6$), and Dataphase ($\phi 7$). On 14-phase systems, they are called Phase 1 through 14.

The phases are specified relative to a timing reference called t_0 . The CYCLE statement specifies t_0 .

STATEMENTS AND VARIABLES DESCRIBED IN THIS SECTION

- **CYCLE** Establishes the period in which data is sent to and received from the DUT
- **PHASE** Programs the clock phases
- **TRIGGER 1**
• **TRIGGER 2** Cause a trigger pulse at any DUT cycle during a clock sequence
- **TSTRIG** Causes a 1 μ s pulse to occur at the rear panel of the Test Station Control Unit
- **CLOCK** Reads a real-time counter



TEST STATION PHASE DISTRIBUTION

Clock phases are used by the system for two purposes: first, for determining the timing of the data being forced on the DUT, and second, to determine when the output of the DUT will be compared with the expected data.

Each pin electronics card receives four clock phases:

Initial Force (IF),
 Program Force (PF),
 Initial Compare (IC), and
 Program Compare (PC).

It can use one of two "force" phases for forcing data on the DUT. The pin electronics card can use one of two "comparison" phases to determine when the output of the DUT will be compared with the expected output. At the beginning of a test, the driver is connected to the initial compare phase. The sector cards can be programmed to use the program force and program compare phase by means of the CONNECT TO PHASE ON pins statement. The program force phase is then connected to the driver, and the program compare phase is connected to the comparison circuits. The CONNECT TO DATAPHASE ON pins statement will reconnect initial phases.

There are four possible phase distribution configurations. Three use a 2943/2944 Programmable Clock Generator and provide 14 phases; one uses only a 2943 (or a 2941) and provides seven phases. The four configurations are summarized in Table 2-1. Depending on the distribution, a particular phase may be used for either forcing or comparing. Figures 2-1 through 2-4 show how the phases are distributed around the test station.

Table 2-1. Summary of Phase Distribution Configurations

	8 Force 6 Compare	10 Force 4 Compare	12 Force 2 Compare	5 Force 2 Compare
Phase 1	PF 1-16	PF 1-8	PF 1-8	PF 1-16
Phase 2	PF 17-32	PF 9-16	PF 9-16	PF 17-32
Phase 3	PF 33-48	PF 17-24	PF 17-24	PF 33-48
Phase 4	PF 49-64	PF 25-32	PF 25-32	PF 49-64
Phase 5	IC 1-32	PF 33-40	PF 33-40	PC 1-64
Phase 6	IC 33-64	PF 41-48	PF 41-48	IC 1-64
Phase 7	PC 1-16	PF 49-56	PF 49-56	IF 1-64
Phase 8	PC 17-32	PF 57-64	PF 57-64	NA
Phase 9	PC 33-48	IC 1-32	IC 1-64	NA
Phase 10	PC 49-64	IC 33-64	PC 1-64	NA
Phase 11	IF 1-16	PC 1-32	IF 1-16	NA
Phase 12	IF 17-32	PC 33-64	IF 17-32	NA
Phase 13	IF 33-48	IF 1-32	IF 33-48	NA
Phase 14	IF 49-64	IF 33-64	IF 49-64	NA
Last Compare Phase	Phase 6	Phase 10	Phase 9	Phase 6
Phase Distribution	See Figure 2-1	See Figure 2-2	See Figure 2-3	See Figure 2-4

PF = Program Force
 IF = Initial Force
 PC = Program Compare
 IC = Initial Compare

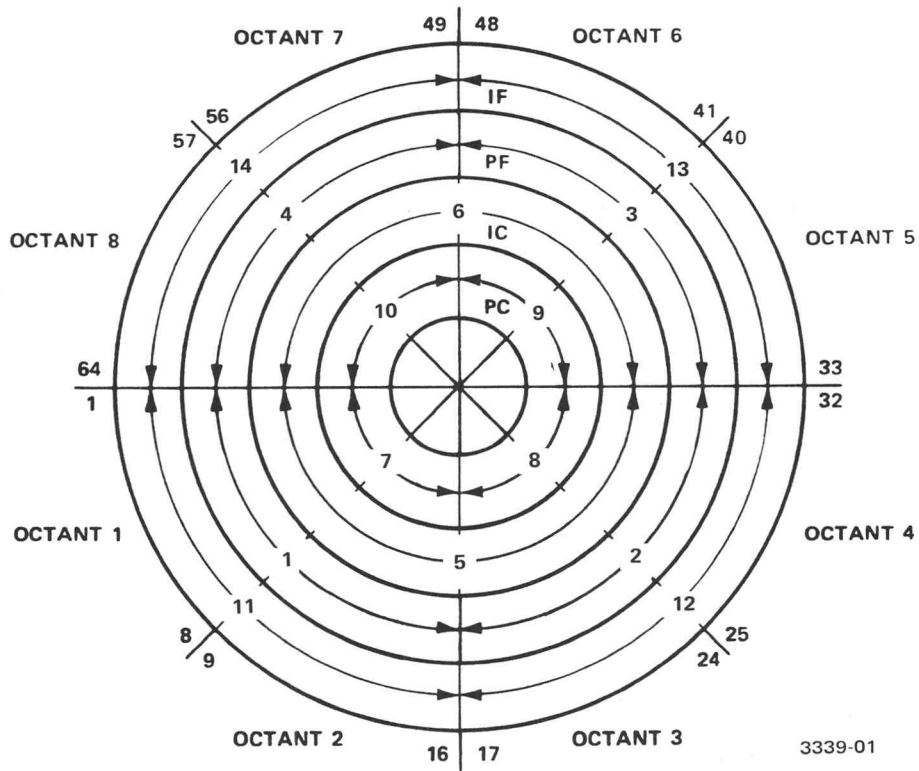


Figure 2-1. 8 Force and 6 Compare Phases

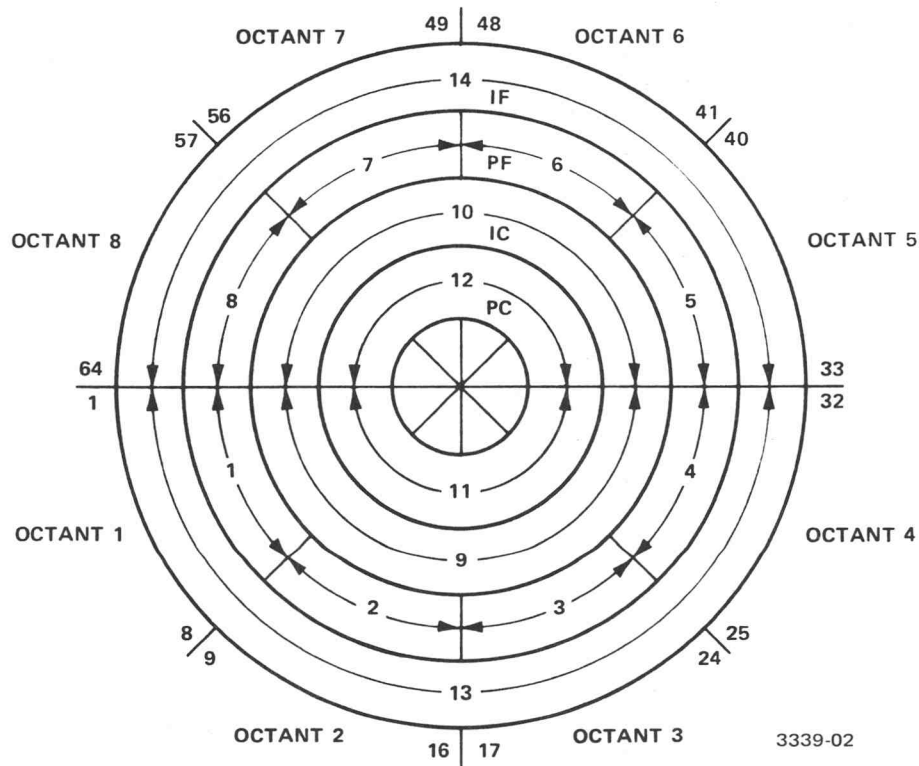


Figure 2-2. 10 Force and 4 Compare Phases

ESTABLISHING THE TIMING REFERENCE

The **CYCLE** statement establishes the period in which data is sent to and received from the DUT. **CYCLE** start is called t_0 . The duration between any two consecutive t_0 s is the DUT cycle.

The **CYCLE** statement format is:

CYCLE = n [,EXTERNAL SYNC]

n specifies the duration of the DUT cycle. It may be any legal expression. Refer to *DUT Cycle Limit and Range Restrictions* for additional information.

EXTERNAL SYNC specifies that the clock generator will be synchronized with an external input. This allows testing of devices that contain their own clocks.

Depending on the sector-card operating mode chosen by the **MOVE** statement (see Section 4), the sector-card shift registers are clocked once, twice or four times during each DUT cycle. The shift register clocks occur every clock cycle. The clock cycle is defined as:

$$\text{clock cycle} = \frac{\text{DUT cycle}}{\text{Clock pulse mode}}$$

Clock Pulse Mode	Sector-Card Mode
1	1 (F,I,C,M)
2	2 and 3 (FC and FI,CM)
4	4 (FICM)

DUT Cycle Limit and Range Restrictions

The S-3270 is capable of test rates up to 20.83 MHz. This frequency corresponds to a minimum clock cycle of 48 ns. Therefore, the minimum DUT cycles are:

Time	Frequency	Clock Mode	Cycle Resolution	
			Fast Range	Slow Range
48 ns	20.8 MHz	1	8 ns	512 ns
96 ns	10.4 MHz	2	16 ns	1024 ns
192 ns	5.2 MHz	4	32 ns	2048 ns

Care should be taken to avoid selecting modes in a test that the programmed DUT cycle time cannot accommodate.

The software imposes a limit of 1.048 ms in all modes. Therefore, the DUT cycle may be from 48 ns to 1.048 ms. This range is divided into two parts:

- fast range: 48 ns to 16376 ns, and
- slow range: 16.384 μ s to 1.048 ms.

In the fast range for Clock Mode 1, the DUT cycle is truncated to the smaller multiple of 8 ns. In the slow range for Clock Mode 1, the DUT cycle is rounded to the nearest multiple of 512 ns. See the above chart for the cycle resolution for Clock Mode 2 and Clock Mode 4.

Figure 2-5 below shows the fast and slow ranges.

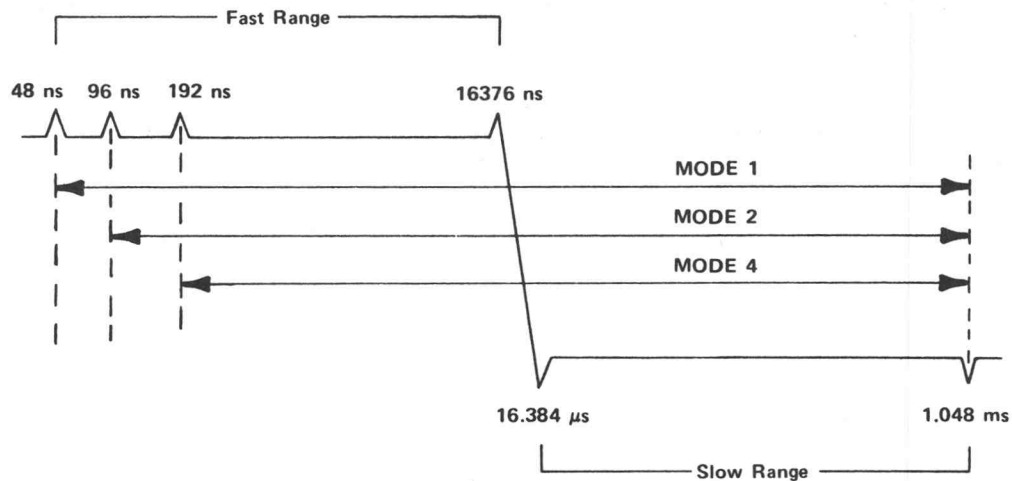


Figure 2-5. DUT Cycle Ranges

3325-02

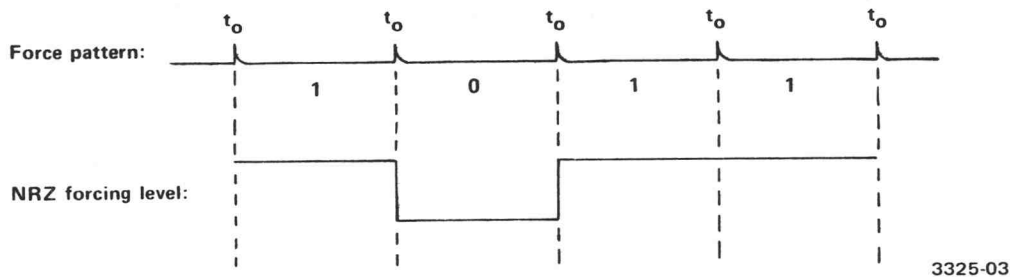
Timing Force Data

When a test runs, the sector-card driver circuits force logic levels on the input pins of the DUT. The FORCE statement (see Section 4) selects the pins forced, the source data, and the timing mode. The force data has three timing modes: Non Return-to-Zero (NRZ), Return-to-Zero (RZ), and Return-to-Complement (RC).

The HIDRIVE and LODRIVE statements (see Section 3) set the voltage levels associated with logic-level one and logic-level zero.

In the NRZ mode, the system forces the specified logic level during the entire DUT cycle. t_0 is the instant at which the DUT is forced with NRZ data. In this case, the CYCLE statement directly controls the timing of the force data.

Example:



In the RZ mode, t_0 does not clock the force data level. The PHASE statement specifies the time during which the system forces the data. Phase 3 is specified relative to t_0 , but is not restricted to occur within the DUT cycle or with the same frequency as t_0 .

If RZ is required, the FORCE statement corresponding to these pins must include the RZ element. Omitting RZ causes the NRZ mode.

The INITIALIZE statement, START button, or CONNECT TO DATAPHASE ON pins statement selects the initial force and compare phases. If a program phase is required, a CONNECT TO PHASE statement must be used.

Examples of the NRZ and RZ modes follow the discussion of the PHASE statement.

PROGRAMMING THE CLOCK PHASES

The **PHASE** statement programs the clock phases. The use of each phase is determined by the choice of phase distribution system.

The **PHASE** statement format is:

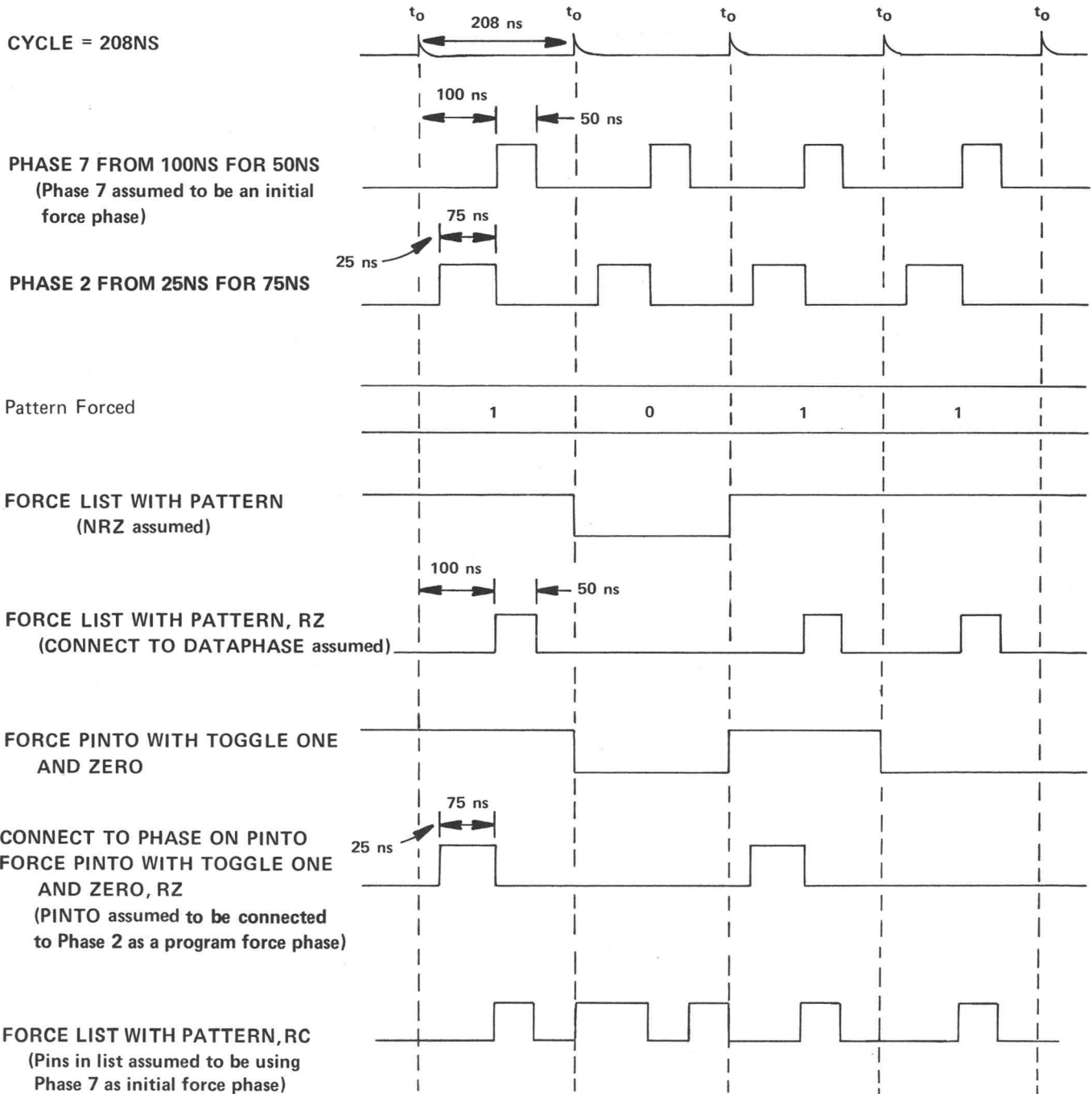
$$\text{PHASEn} \left\{ \begin{array}{l} = \\ \text{FROM} \end{array} \right\} \text{starttime} \left\{ \begin{array}{l} \text{FOR} \\ , \end{array} \right\} \text{width}$$

This statement is correct for those systems with only seven phases; **n** is an integer from 1 to 4. In those systems with 14 clock phases, **n** is an integer from 1 to 14.

starttime specifies when the leading edge of the pulse occurs relative to t_0 . **width** specifies the pulse width.

starttime and **width** are any legal expressions. They are discussed later in this section under the heading **Clock Phase Limits and Scaling**.

Example of NRZ and RZ Timing Modes



3339-05

CLOCK PHASE LIMITS

Certain restrictions must be observed when establishing the seven independent phases relative to the DUT cycle start, t_0 . These restrictions apply equally to the Hicompare, Locompare, Dataphase, and Phase 1 through Phase 14.

In the discussions, **starttime** and **width** refer to the phase statement elements. **starttime** is expressed relative to the DUT cycle start. **width** is expressed relative to **starttime**. The limits for **starttime** and **width** vary depending on the range selected by the CYCLE statement.

Pulse Start-Time

For the fast range, **starttime** may be programmed in 1 ns increments from 0 ns after t_0 to 17 ns before the next t_0 . Programming a start-time that falls within the 17 ns before the next t_0 should be avoided.

For the slow range, **starttime** may be programmed in 512 ns increments from 0 ns after t_0 to 512 ns before the next t_0 . Programming a start-time equal to the CYCLE time should be avoided.

Pulse Width

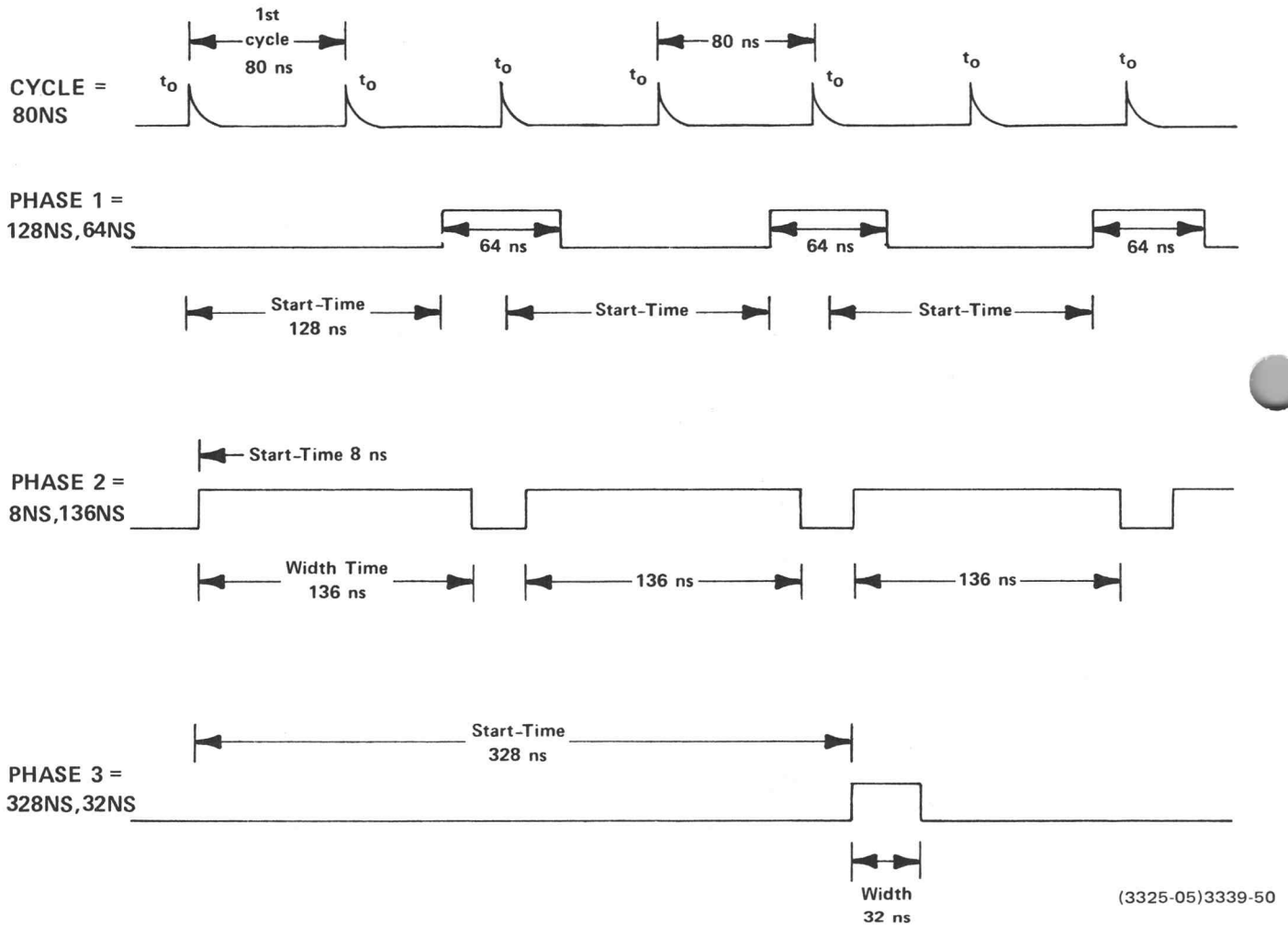
For the fast range, **width** may be programmed from 8 ns after the leading edge (start-time) up to 16 ns of the next leading edge. Programming a width that falls within 16 ns of the next leading edge should be avoided.

For the slow range, **width** may be from 512 ns after the leading edge (start-time) up to 512 ns before the next start-time. Programming a width equal to the CYCLE time should be avoided.

DUT Cycle Frequency

You may program the pulse start-time and width to produce a phase frequency less than the DUT cycle frequency. That is, the programmed pulse does not occur every DUT cycle.

The following example illustrates the effects of specifying a start-time or a width greater than the DUT cycle.



Summary of Ranges

Below are summaries of the limits for **starttime** and **width** for the HICOMPARE, LOCOMPARE, DATA-PHASE and PHASE statements. The CYCLE statement chooses the range.

Fast Range: DUT cycle from 48 ns to 16376 ns.

	starttime	width
FROM	0 ns after t_0	8 ns after starttime
TO	17 ns before the next t_0 -or- 16376 ns after the first t_0 , but not less than 17 ns before the nth t_0	16 ns before the next starttime -or- 16376 ns after starttime , but not less than 16 ns before the nth starttime
starttime and width are positionable in increments of 1 ns.		

Slow Range: DUT cycle from 16.384 μ s to 1.048 ms.

	starttime	width
FROM	0 ns after t_0	512 ns after starttime
TO	512 ns before the next t_0 -or- 1.048 ms after t_0	512 ns before the next starttime -or- 1.048 ms after starttime
starttime and width are positionable in increments of 512 ns. Neither starttime nor width should be equal to CYCLE time.		

PHASE Statement Modification

It is inconsistent to enter values for the phase start-time and width that are in different ranges or are in a different range than selected by the CYCLE statement and other phase statements. If possible, however, the system modifies the phase elements to fit into the range selected by the CYCLE statement. The process is:

- A. The Translator records and formats the phase start-time and width independently of the CYCLE range. If both values fall within the fast range, they round to the nearest multiple of 1 ns. If one of the values falls within the slow range, the Translator assumes both values are in the slow range. The values round to the nearest multiple of 512 ns. If both values fall within the slow range, they round to the nearest multiple of 512 ns.
- B. The system evaluates the CYCLE statement. Since the CYCLE element may be a variable, the decision as to whether CYCLE and the phase ranges are consistent is withheld until the test runs. Once the CYCLE value is established, the system checks it for validity and selects a range.

The system selects the fast range if $48 \text{ ns} \leq \text{CYCLE} \leq 16376 \text{ ns}$. The system selects the slow range if $16377 \text{ ns} \leq \text{CYCLE} \leq 1.048 \text{ ms}$. An error message appears on the Test Station Control Unit (TSCU) and the test halts if CYCLE is less than 48 ns or greater than 1.048 ms.

Then, the system compares the CYCLE range with the ranges of the phase statements, resulting in the following:

1. If CYCLE is in the fast range and any phase is in the slow range, an error message appears and the test halts.
 2. If CYCLE is in the slow range and any phase is in the fast range, the system converts the phase to slow range by rounding both its elements to the nearest multiple of 512 ns.
 3. If both CYCLE and all the programmed phases are in the same range, no inconsistency exists.
- C. When the system encounters a MOVE or LOAD statement, it divides the CYCLE value by the clock mode. If the resultant is less than 48 ns, an error message appears on the TSCU and the test halts.

Examples:

**CYCLE = X
PHASE 1 FROM 200NS FOR 53NS**

If CYCLE is in the fast range, the system implements Phase 1 as programmed. If CYCLE falls within the slow range, the system rounds the PHASE1 elements to the nearest multiple of 512 ns. The pulse start-time and width both become 0 ns. This results in an error condition since the minimum pulse width in the slow range is 512 ns.

**CYCLE = 20US
PHASE 6 FROM 50NS FOR 1000NS**

CYCLE is in the slow range. Therefore, the system adjusts the Hicompare phase into this range. The DUT cycle time becomes 19.968 μ s (a multiple of 512 ns) and Hicompare has a start-time of 0 ns and a width of 1024 ns.

**CYCLE = 500NS
PHASE 3 FROM 10NS FOR 17US**

An error occurs since CYCLE selects the fast range and Phase 3 is in the slow range.

**CYCLE = 50 US (The time specified should be in the range from 21 μ s to 100 μ s.)
PHASE 7 FROM 1US FOR 20US**

Both statements fall within the slow range. The system adjusts the DUT cycle time to 50.176 μ s. The phase seven has a start-time of 1.024 μ s and a width of 19.968 μ s.

DUT at 1804

DUT Cycle Time, τ_c
(in ns of duration)

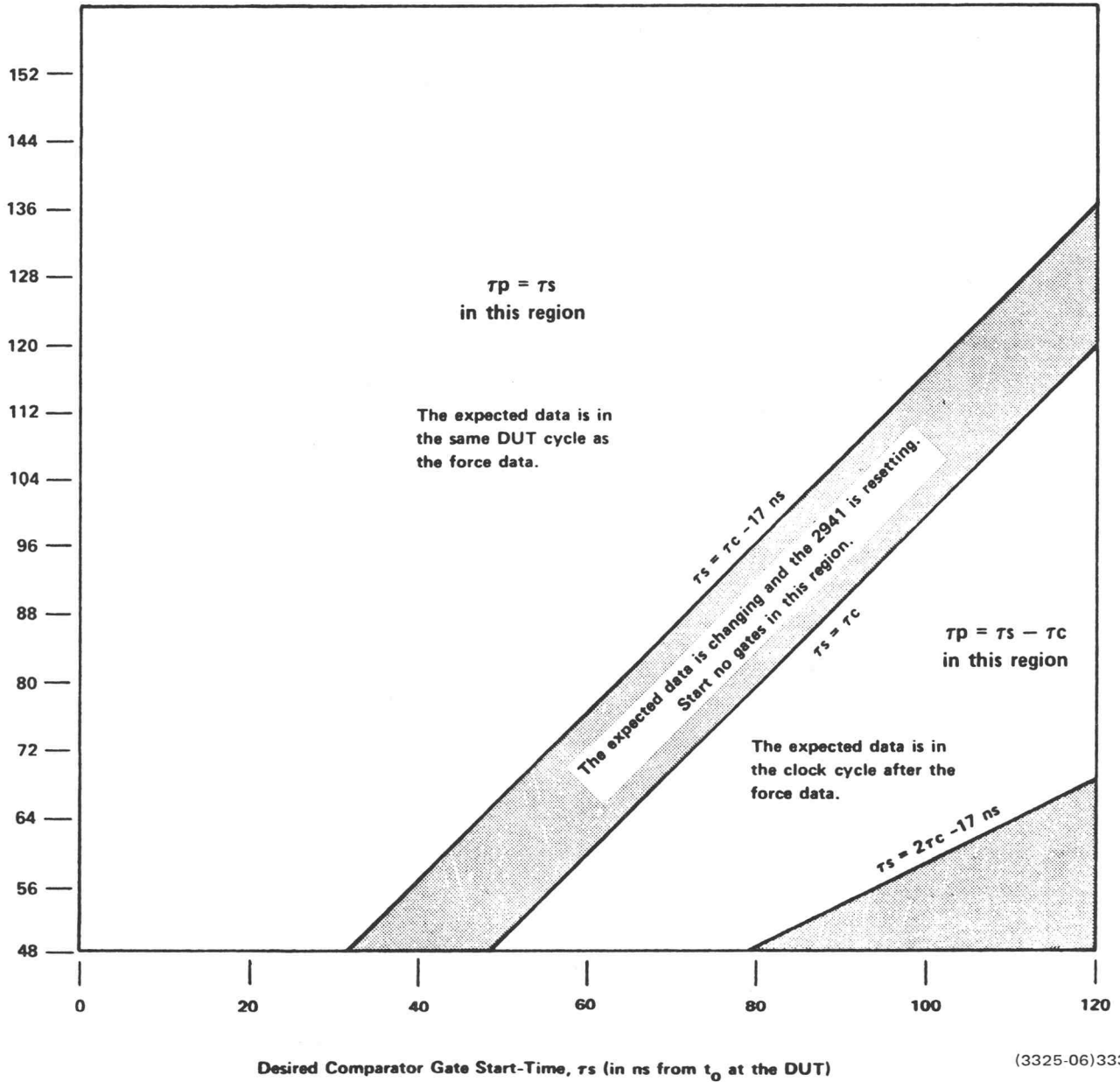


Figure 2-6. CALCULATING THE PROGRAM VALUE, τ_p ,
FOR COMPARATOR-GATE START-TIME

CAUSING A TRIGGER PULSE

The **TRIGGER** and the **TRIGGER2** statements cause a clock trigger pulse at any DUT cycle during a clock sequence. The **TRIGGER** statement must occur before the **MOVE** statement. The pulse occurs approximately 10 ns before the start of the selected DUT cycle. The pulse occurs at the rear panel of the test station at the jack labeled PROGRAMMABLE TRIGGERS.

The **TRIGGER** statement format is:

```
TRIGGER exp[,REPEAT]  
TRIGGER2 exp[,REPEAT]
```

exp indicates the DUT cycle at which the **TRIGGER** pulse occurs. It may be any legal expression that evaluates in the ranges shown below.

Sector Card Operating Mode	Range
1 (F,I,C,M)	$1 \leq \text{exp} \leq 65535$
2 and 3 (FC and FI,CM)	$1 \leq \text{exp} \leq 32767$
4 (FICM)	$1 \leq \text{exp} \leq 16383$

If the **TRIGGER** statement specifies **REPEAT**, the pulse repeats every **exp** DUT cycle.

Example:

```
CYCLE = 100 NS  
TRIGGER2 15  
TRIGGER 10,REPEAT  
MOVE FROM REGISTER (200) TO PINLIST WITH FC
```

A **TRIGGER** pulse occurs every 10th DUT cycle. Since the **MOVE** statement specifies 200 DUT cycles, 20 pulses occur spaced 1000 ns apart.

A **TRIGGER2** pulse occurs on the 15th DUT cycle. Since **REPEAT** was omitted, only one pulse occurs.

CAUSING A 1 μ S PULSE AT THE REAR PANEL OF THE TEST STATION

The **TSTRIG** statement causes a 1 μ s pulse to occur at the rear panel of the test station at the jack labeled PROGRAMMABLE TRIGGERS, TEST. The pulse occurs immediately after the test program encounters the TSTRIG statement.

The TSTRIG statement format is:

TSTRIG

THE REAL-TIME COUNTER

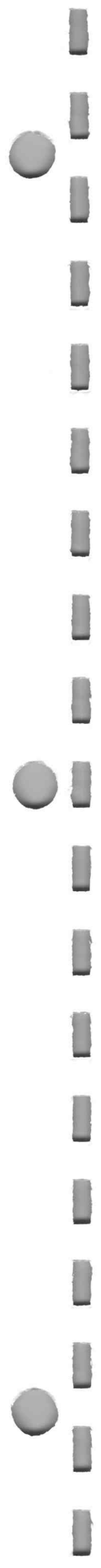
The **CLOCK** variable* reads a real-time counter that increments every 2^{-16} seconds (15.3 μ s). The counter has a maximum count of 4096 seconds (approximately 68.25 minutes).

The real-time counter initializes to zero and begins counting when the TSCU START button is pressed. When the test program encounters **CLOCK** during program execution, the system assigns the value (in seconds) of the real-time counter to **CLOCK**.

The **CLOCK** variable format is:

CLOCK

*See *TEKTEST II/III Test Language, Part One* for information on reserved system variables.



SECTION 3: STIMULUS

The TEKTEST statements described in this section provide stimulus to the DUT, provide expected data information, and make data connection paths through the sector card electronics. These statements are:

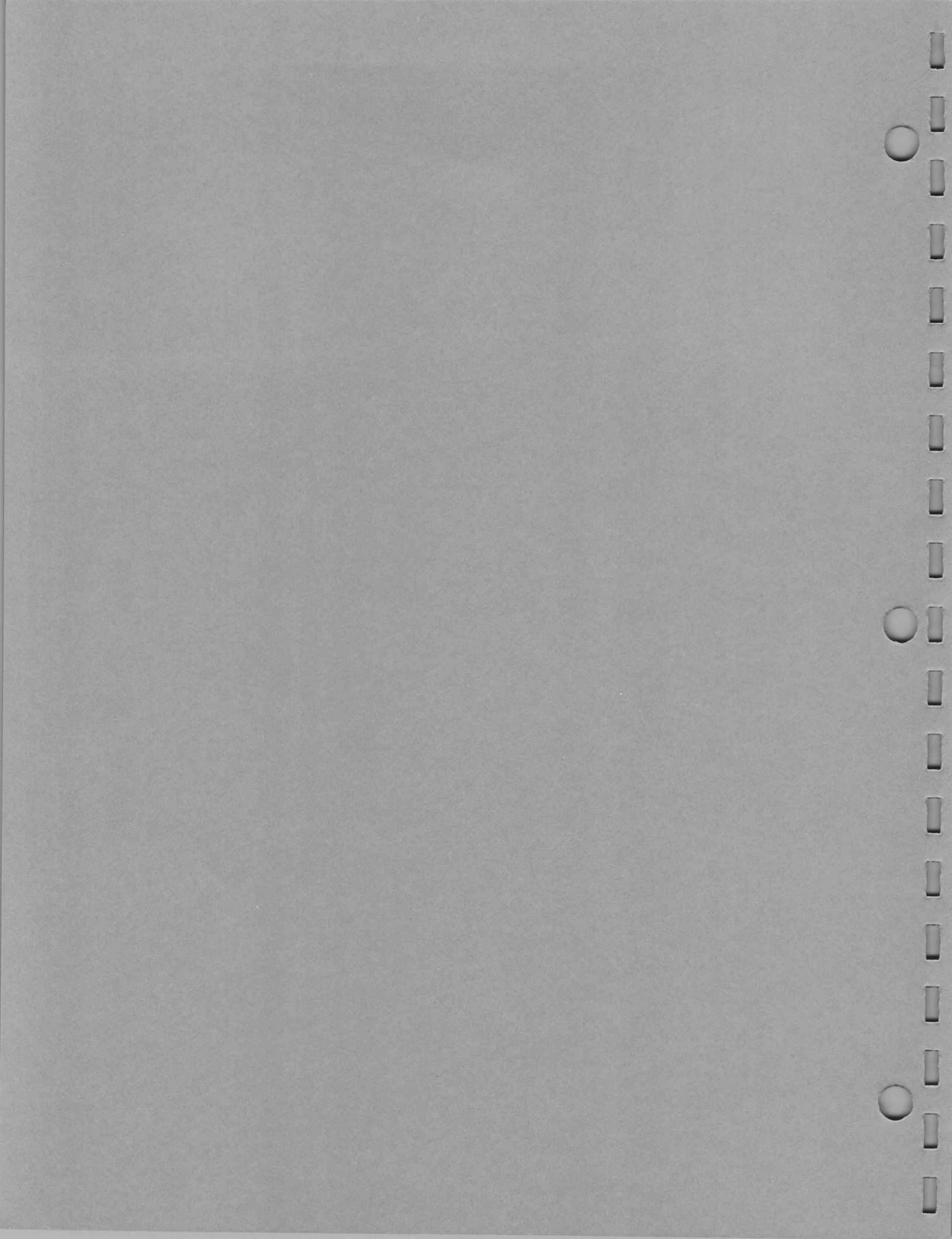
HIDRIVE
LODRIVE
HICOMPARE
LOCOMPARE
CONNECT
DISCONNECT

Mandatory test program statements when making both functional or parametric tests.

CONNECT TO
UNDERSOCKET
INITIALIZE

Optional test program statements.

- **HIDRIVE**
• **LODRIVE**
Enable the Sample and Hold circuitry which sets the sector-card driver voltage levels. The voltage levels are forced on the DUT when the data is forced.
- **HICOMPARE**
• **LOCOMPARE**
Enable the Sample and Hold circuitry which sets the sector-card comparator levels. The comparator monitors the DUT output for the expected data for the selected time frame.
- **CONNECT**
• **DISCONNECT**
Establish the data routes through the sector card electronics. Most connections are made by opening or closing reed switches.
- **CONNECT TO**
Deals with clock phase strobing rather than data routing.
- **UNDERSOCKET**
Allows additional data to be forced on the DUT via the undersocket card.
- **INITIALIZE**
Isolates the DUT from all external connections and clears the 1340 Data Couplers and error flip-flops.



SETTING FORCE VOLTAGES

The **HIDRIVE** and **LODRIVE** statements enable the Sample and Hold (S&H) circuitry which in turn sets the high- and low-voltage levels for the drivers associated with the selected pins. The levels specified in **HIDRIVE** appear at **pins** when logic level 1 is forced; the levels specified in **LODRIVE** appear at **pins** when logic level 0 is forced.

At the beginning of program execution, the system initializes **HIDRIVE** and **LODRIVE** (the S&H circuitry) to 0 V. Observe these rules when using the **HIDRIVE** and **LODRIVE** statements:

1. Both **HIDRIVE** and **LODRIVE** must precede a **FORCE** statement on the same pins.
2. **HIDRIVE** must be set to a value more positive or equal to the **LODRIVE** value ($V_{DH} \geq V_{DL}$).
3. **HIDRIVE** may be programmed from -10 V to +30 V. **LODRIVE** may be programmed from -30 V to +10 V. (The D70 data drivers have a programming resolution of 10 mV. Thus, voltage may be set from 10 mV to 30 V in 10 mV increments. Other type data drivers must be specified within their ranges.)
4. The **HIDRIVE** value minus the **LODRIVE** value must be less than or equal to 30 V: $(V_{DH} - V_{DL}) \leq 30$ V. (Since D70 data drivers have a range of ± 30 V with a maximum swing of 30 V peak to peak, **HIDRIVE** and **LODRIVE** must be set within 30 V of each other.)
5. If both **HIDRIVE** and **LODRIVE** are going to be set to positive voltages, set **HIDRIVE** first. If both **HIDRIVE** and **LODRIVE** are going to be set to negative voltages, set **LODRIVE** first. If **HIDRIVE** is going to be set to a positive voltage and **LODRIVE** to a negative voltage, you may set either first.

The formats of the **HIDRIVE** and **LODRIVE** statements are:

HIDRIVE { $\begin{matrix} = \\ \text{FROM} \end{matrix}$ } voltagevalue ON pins

LODRIVE { $\begin{matrix} = \\ \text{FROM} \end{matrix}$ } voltagevalue ON pins

Elements

voltagevalue is any legal expression.

S&H Discharge Rate

The S&H circuitry is analog in nature. That is, the drive levels may change with time. If a significant delay occurs between the HIDRIVE and LODRIVE statements and the MOVE statement, you may need to re-establish the drive levels. The statements WAIT and IF (ADVANCE) could cause such significant delay.

S&H Feedback Circuitry

HIDRIVE and LODRIVE take precedence over the FORCE statement. Because of feedback circuitry on the S&H board, HIDRIVE programs part of a FORCE WITH ONE operation and LODRIVE programs part of a FORCE WITH ZERO operation. This means that if the sequence of the programming steps are improperly entered, either HIDRIVE or LODRIVE may leave the drivers in a state other than intended. For example, if you enter a FORCE WITH ZERO statement followed by HIDRIVE, HIDRIVE would alter the driver state to 1 since HIDRIVE implies a FORCE WITH ONE operation. On the other hand, if LODRIVE followed a FORCE WITH ONE statement, the driver state would be altered to 0.

SETTING COMPARE VOLTAGES

The **HICOMPARE** and **LOCOMPARE** statements enable the Sample and Hold (S&H) circuitry, which in turn sets the high- and low-voltage levels and timing window for the comparators associated with the selected pins. The comparator expects the voltage for a duration specified in **HICOMPARE** when comparing for 1. It expects the voltage for a duration specified in **LOCOMPARE** when comparing for 0.

Observe these rules when using the **HICOMPARE** and **LOCOMPARE** statements:

1. Both **HICOMPARE** and **LOCOMPARE** compare voltage levels.
2. Both **HICOMPARE** and **LOCOMPARE** must precede a **COMPARE** statement on the same pins.
3. **D70** sector-card comparators have a range of ± 30 V in the 30-volt range and ± 5 V in the 5-volt range. Other sector-card comparator types must be specified within their ranges.
4. The optional **range** element selects the power supply limit and, as mentioned, is either **5** or **30**. The ± 30 -volt range is less sensitive than the ± 5 -volt range but must be selected if the expected voltage is to be greater than 5 V. The ± 5 -volt comparator range provides greater sensitivity. It is the default range.
5. The elements **starttime** and **width** establish the clock generator comparator gates. They specify the timing window in which to expect the data. **starttime** is expressed relative to **CYCLE** t_0 . **width** is expressed relative to **starttime** and specifies duration. The minimum increment is 1 ns*.

The formats of the **HICOMPARE** and **LOCOMPARE** statements are:

Compare Voltages

HICOMPARE=voltagevalue [AT range] ON pins

LOCOMPARE=voltagevalue [AT range] ON pins

*See the **SECTION 2** for a complete account of clock timing procedures.

Phase Times

PHASE {n} { = FROM } starttime { FOR } width

where n indicates any of the comparison phases in your system.

Elements

voltagevalue, **starttime**, and **width** are any legal expressions.

range must be the constant 5 or the constant 30.

S&H Discharge Rate

The S&H circuitry is analog in nature. That is, the compare levels may change with time. If a significant delay occurs between the HICOMPARE and LOCOMPARE statements and the MOVE statement, you may need to re-establish the compare levels. The statements WAIT and IF (ADVANCE) could cause such significant delay.

S&H Feedback Circuitry

HICOMPARE and LOCOMPARE take precedence over the COMPARE statement. Because of feedback circuitry on the S&H board, HICOMPARE programs part of a COMPARE WITH ONE operation and LOCOMPARE programs part of a COMPARE WITH ZERO operation. This means that if the sequence of the programming steps are improperly entered, either HICOMPARE or LOCOMPARE may leave the comparator in a state other than intended. For example, if the user enters a COMPARE WITH ZERO statement followed by HICOMPARE, HICOMPARE would alter the comparator to 1 since HICOMPARE implies a COMPARE WITH ONE operation. On the other hand, if LOCOMPARE followed a COMPARE WITH ONE statement, the comparator would be altered to 0.

Implied CONNECT Statement

Included in the HICOMPARE and LOCOMPARE statements is an implied CONNECT OUTPUT TO COMPARATOR. This eliminates the need to use the CONNECT OUTPUT TO COMPARATOR statement and ensures that the same range is specified in all three statements. It is still necessary, however, to enter the DISCONNECT OUTPUT FROM COMPARATOR statement before making DC measurements if the $2\text{ M}\Omega$ input resistance of the buffer amplifier affects the measurement.

Error Detection

The system does not detect errors under the following conditions (see Figure 3-1):

1. When comparing for 1, the DUT output is greater than the specified HICOMPARE voltagevalue and remains at this level for the specified HICOMPARE width.
2. When comparing for 0, the DUT output is less than the specified LOCOMPARE voltagevalue and remains at this level for the specified LOCOMPARE width.

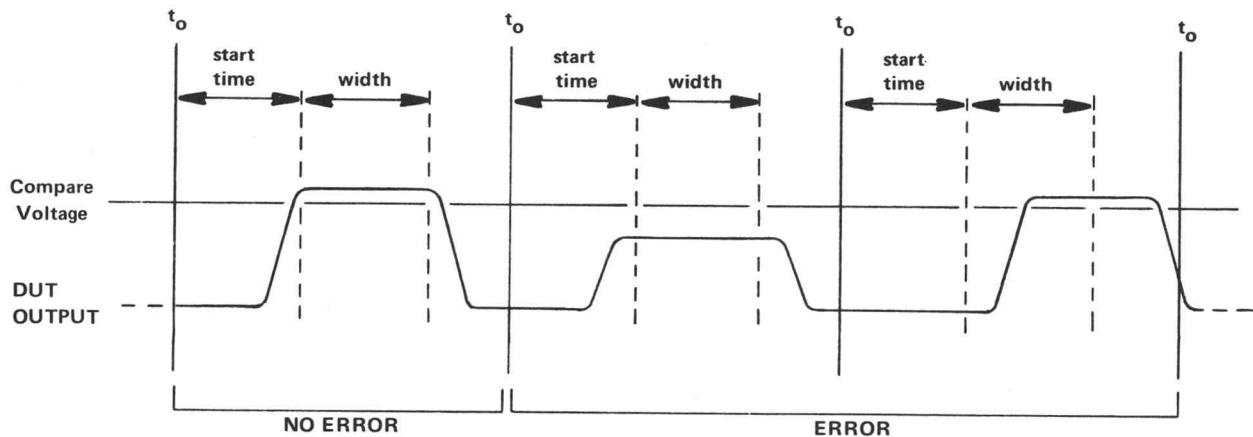


Figure 3-1. Expected data criteria.

3339-06

CONNECTING AND DISCONNECTING THE SECTOR-CARD DATA PATHS (1804 TEST STATION ONLY)

The **CONNECT** and **DISCONNECT** statements respectively close and open one or several reed switches*on the sector cards associated with the selected pins. The **CONNECT** statement allows you to route data to its appropriate destination. An equivalent **DISCONNECT** statement breaks the selected data path. Table 3-1 shows the connections made or broken when you use certain **CONNECT** statements. Note that in most cases **CONNECT** statements which establish a data path will, in turn, also disrupt other paths. Table 3-2 shows how some **CONNECT** statements affect reed switches on the sector cards. Figure 3-2 is an annotated data flow diagram. It points out the paths enabled upon using certain elements of **CONNECT** statements. Equivalent **DISCONNECT** statements break the paths by opening the reed switches.

The formats of the **CONNECT** and **DISCONNECT** statements are:

CONNECT BUFFER TO {
SCOPE1
SCOPE2
AUX1
AUX2
AUX3
AUX4
} ON pin1 [;ND]

CONNECT COMPARATOR {
+
-
} TO {
SCOPE1
SCOPE2
AUX1
AUX2
AUX3
AUX4
} ON pin1 [;ND][,range]

CONNECT COMPUTER TO {
REGISTER
SECTOR
} ON pins

*Or program digital data paths.

CONNECT DATA TO { REGISTER
EBUS } ON pins *

CONNECT DRIVER TO { SCOPE1
SCOPE2
AUX1
AUX2
AUX3
AUX4 } ON pin1 { INPUT ON pins } [;ND]

CONNECT EBUS TO REGISTER ON pins

CONNECT ERRNOT TO { REGISTER
EBUS } ON pins

CONNECT ERROR TO { REGISTER
EBUS
COMPUTER } ON pins

CONNECT EXTERNAL TO SECTOR ON pins*

CONNECT INPUT TO { DRIVER
COMMON
GROUND } ON pins
 { SCOPE1
SCOPE2
AUX1
AUX2
AUX3
AUX4 } ON pin1 } [;ND]

CONNECT { LOAD1
LOAD2 } ON pins [;ND]

CONNECT OUTPUT TO { COMPARATOR,range
LOAD1
LOAD2
COMMON
GROUND } ON pins
 { SCOPE1
SCOPE2
AUX1
AUX2
AUX3
AUX4 } ON pin1 } [;ND]

CONNECT REGISTER TO { REGISTER
SECTOR
COMPUTER
EBUS } ON pins

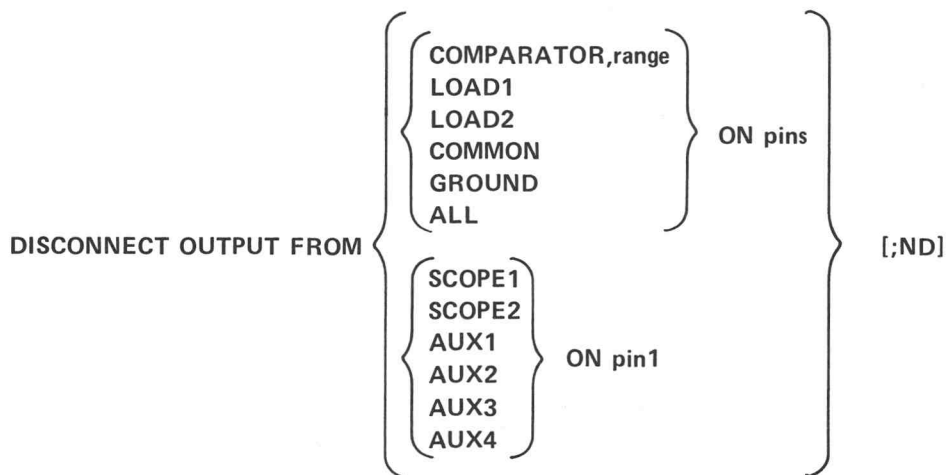
DISCONNECT BUFFER FROM { SCOPE1
SCOPE2
AUX1
AUX2
AUX3
AUX4 } ON pin1 [;ND]

DISCONNECT COMPARATOR { +
- } FROM { SCOPE1
SCOPE2
AUX1
AUX2
AUX3
AUX4 } ON pin1 [;ND]

DISCONNECT DRIVER FROM { { SCOPE1
SCOPE2
AUX1
AUX2
AUX3
AUX4 } ON pin1 } [;ND]
INPUT ON pins

DISCONNECT INPUT FROM { { ALL
DRIVER
COMMON
GROUND } ON pins } [;ND]
{ SCOPE1
SCOPE2
AUX1
AUX2
AUX3
AUX4 } ON pin1

DISCONNECT { LOAD1
LOAD2 } ON pins [;ND]



Elements

BUFFER is the buffer amplifier. If **BUFFER** is the source element, then the output of the buffer amplifier is being connected. If **BUFFER** is the destination element, then the selected source is being connected to the input of the buffer amplifier.

COMPARATOR is the high and low comparator circuitry which feeds information to the error detection circuitry. { \pm } indicates the side of the comparator to be connected. + corresponds to the high-level side (HICOMPARE) and - corresponds to the low-level side (LOCOMPARE). If the range element is specified, it must be either 30 or 5. 30 is the ± 30 -volt range and 5 is the ± 5 -volt range. The 5-volt range is also the default condition. If **COMPARATOR** is the source element, then the output of the comparator is being connected. If **COMPARATOR** is the destination element, then the selected source is being connected to the input of the comparator.

COMPUTER is the central controller.

DRIVER is the high and low driver circuitry which forces data to the DUT. Like **COMPARATOR** and **BUFFER**, the position of the **DRIVER** element in the statement determines if the input or the output of the driver is being connected.

EBUS is the error bus used for serial chaining.

ERROR is the error data from the error detection circuitry. If the data is high, an error was detected during a move operation. The data will remain high during the entire move operation. This error information refers to the error flip-flop and is used with the LOGERROR statement. **ERROR** may be connected to the **COMPUTER** only. The retimed error information from the error detection circuitry is delayed by four cycles and can be connected to the **EBUS** or **REGISTER**.

INPUT is the input path to the DUT.

LOAD1 and **LOAD2** are the special load module circuitry from the load board.

OUTPUT is the output path from the DUT.

ND refers to no delay and allows the system to make faster DC measurements. **ND** instructs the system to continue processing statements while reed switches effected by **CONNECT** and **DISCONNECT** are closed or opened. Without the **ND** element, the system suspends processing of additional statements until the effected reed switch is opened or closed. Thus, the **ND** element can result in more efficient running of the test program. It is good programming practice not to include **ND** in the last **CONNECT** statement in a series. This procedure permits the fastest processing of the **CONNECT** statements to occur, but prevents processing of other parts of the program until all the proper connections have been made.

REGISTER is the 4104-bit shift register.

COMMON is the common bus.

SCOPE1 and **SCOPE2** are the two output jacks at the rear of the 1804 Test Station. These may be used to connect an oscilloscope.

AUX1, **AUX2**, **AUX3**, and **AUX4** are auxiliary channels that may be connected to optional pulse generators, voltmeters, etc., in the system. **AUX1** and **AUX2** are standard with the 64:2 50- Ω matrix. **AUX3** and **AUX4** are only available with the 64:4 SM-1 switching matrix.

SECTOR is the force/compare/inhibit/mask sector-card logic.

GROUND is available at the common bus.

ALL in **DISCONNECT** statements opens all connected paths except those to the 50- Ω matrix.

DATA refers to the compare or expected data.

EXTERNAL refers to external data, for instance, from the 2942 Pattern Generator or the Hardware Sequence Generator.

ERRNOT is error data from the error detection circuitry. **ERRNOT** information differs from **ERROR** information in that it represents cycle-by-cycle error information. Thus, it does not remain high during the entire move operation unless errors are detected during each cycle. **ERRNOT** data runs at approximately 5 MHz and, if stored in the shift register, it is one cycle late.

Prerequisite Statements

These **CONNECT** statements must be preceded by an appropriate **UNSET** statement if a **SETUP** statement has been previously executed.

CONNECT {
 BUFFER
 COMPARATOR { \pm }
 INPUT
 OUTPUT
} TO {
 SCOPE1
 SCOPE2
 AUX1
 AUX2
 AUX3
 AUX4
 COMMON
} ON pin1

Using A 568 Oscilloscope

The use of the optional 568 oscilloscope connected to the AUX path via the OUTPUT path may put a 50 Ω impedance on the DUT output.

Unnecessary DISCONNECT Statements

The following **CONNECT** source elements do not have corresponding **DISCONNECT** source elements:

REGISTER	EBUS
COMPUTER	
ERROR	DATA
ERRNOT	EXTERNAL

Disconnecting a source element in this category occurs when a subsequent **CONNECT** statement is executed employing the element.

Table 3-1

Instructions and Connections

SECTOR CARD (n)			CONNECTIONS MADE (1) OR BROKEN (0)															
DESTINATION →		EBUS				REGISTER				COMPUTER				SECTOR				
SOURCE →		ERRNOT	ERROR	DATA	REGISTER	REGISTER	COMPUTER	EBUS OUT	EBUS IN	ERROR	CARD I. D.	REGISTER	FORCE	EXTERNAL	COMPUTER	PRAM	REGISTER	
INSTRUCTION EXECUTED ↓																		
CONNECT	REGISTER TO	REGISTER				1	0	0	0									
		SECTOR													0	0	0	1
		COMPUTER	0	0	1	0					0	0	1	0				
		EBUS	0	0	0	1					0	0	0	1				
	COMPUTER TO	REGISTER					0	1	0	0								
		SECTOR													0	1	0	0
	ERROR TO	REGISTER	0	1	0	0	0	0	1	0	0	1	0	0				
		EBUS	0	1	0	0					0	1	0	0				
		COMPUTER	1	0	0	0					1	0	0	0				
	EBUS TO	REGISTER					0	0	0	1								
	EXTERNAL TO	SECTOR													1	0	0	0
	DATA TO	REGISTER	0	0	1	0	0	0	1	0	0	0	1	0				
		EBUS	0	0	1	0					0	0	1	0				
	ERRNOT TO	REGISTER	1	0	0	0	0	0	1	0	1	0	0	0				
		EBUS	1	0	0	0					1	0	0	0				
	PRAM TO	SECTOR												0	0	1	0	

SELECTING THE INITIAL OR PROGRAM CLOCK PHASES

The **CONNECT TO PHASE** statement must be included in the test program to select a program phase as the RZ, RO, or RC data gate for the selected pins. This statement allows the sector cards associated with the selected pins to be strobed by a separate clock phase pulse.

The **CONNECT TO DATAPHASE** statement is equivalent to a disconnect phase operation; it connects the initial phases to the selected pins. (See also the **PHASE** and **DATAPHASE** statements.)

The **CONNECT TO** statement format is:

CONNECT TO { **PHASE**
DATAPHASE } **ON pins** [;ND]

Elements

PHASE selects the program force and program compare phases.

DATAPHASE selects the initial force and initial compare phases.

ROUTING DATA VIA THE UNDERSOCKET CARD

The **UNDERSOCKET** statement sends a 16-bit word to the 16 programmable logic lines ($\overline{U0-U15}$) on the undersocket card. The data is sent in parallel and may be used for any purpose.

NPN open-collector transistors control the 16 logic lines. Each has the capacity of sinking 25 mA when programmed low. Each has a maximum open-circuit voltage of +40 V at 10 μ A leakage current.

Also worth noting is the assignment of the logic lines to the B side of the undersocket connector. The least significant bit of the data sent sets $\overline{U0}$ which is connected to B16. The most significant bit sets $\overline{U15}$ on line B15. For example:

```
UNDERSOCKET #101601
```

enables $\overline{U15}$, $\overline{U9}$, $\overline{U8}$, $\overline{U7}$, and $\overline{U0}$.

The **UNDERSOCKET** statement format is:

```
UNDERSOCKET [#]data
```

Elements

data is any legal numeric expression. The system treats a number prefixed with # as an octal number and sends its binary equivalent to the undersocket card. A number not prefixed with # is treated as a decimal number and the system sends its binary equivalent to the undersocket card.

INITIALIZING THE TEST STATION

The **INITIALIZE** statement performs the following functions:

1. Sets all 1140 power supplies to 0.
2. Sets the low driver sample and hold to 0.
3. Sets the high driver sample and hold to 0.
4. Clears the programmable clock generator (distributor clear).
5. Selects initial phase as the phase.
6. Generates a 1340 Data Coupler #1 INITIALIZE signal. The initialized cards are:
 - a. 50 Ω Matrix
 - b. DC Subsystem
 - c. ΔT Subsystem
 - d. Undersocket card
 - e. Scope/Misc.
 - f. System clocks/write, Clock A
 - g. System clocks/read, Clock A – timed interrupt generator bits
 - h. System clocks/write, Clock B – timed interrupt generator bits
 - i. System clocks/read, Clock B – timed interrupt generator bits
 - j. Index counter/read, Index A
 - k. Index counter/write, Index A
 - l. Index counter/read, Index B
 - m. Data to test station/write
 - n. Data to test station/read
7. Generates a 1340 Data Coupler #2 INITIALIZE signal. This list of cards varies with different system configurations, but includes:
 - a. Digitizer A/D/read, A Channel
 - b. Digitizer A/D/read, B Channel
 - c. Digitizer A/D/read, A/B Channel
 - d. Vertical reference
 - e. Horizontal reference
 - f. 3T6 Horizontal Sweep, Delay
 - g. 3S6 Vertical Sweep, B Channel
 - h. 3S6 Vertical Sweep, A Channel

8. Clears all error flip-flops.
9. Isolates the DUT from all external connections.

The INITIALIZE statement format is:

INITIALIZE

Automatic Initialize

The system generates an automatic initialize operation when a STOP statement is encountered in the test program.

The RUN Statement

When you use the RUN statement (see *Part One*) to chain a series of test programs, the STOP statement is not used. Initialization of the 1340 and 1804, therefore, does not occur and all previous connections to the DUT remain intact. The INITIALIZE statement must be specified to initialize the 1340 or 1804.

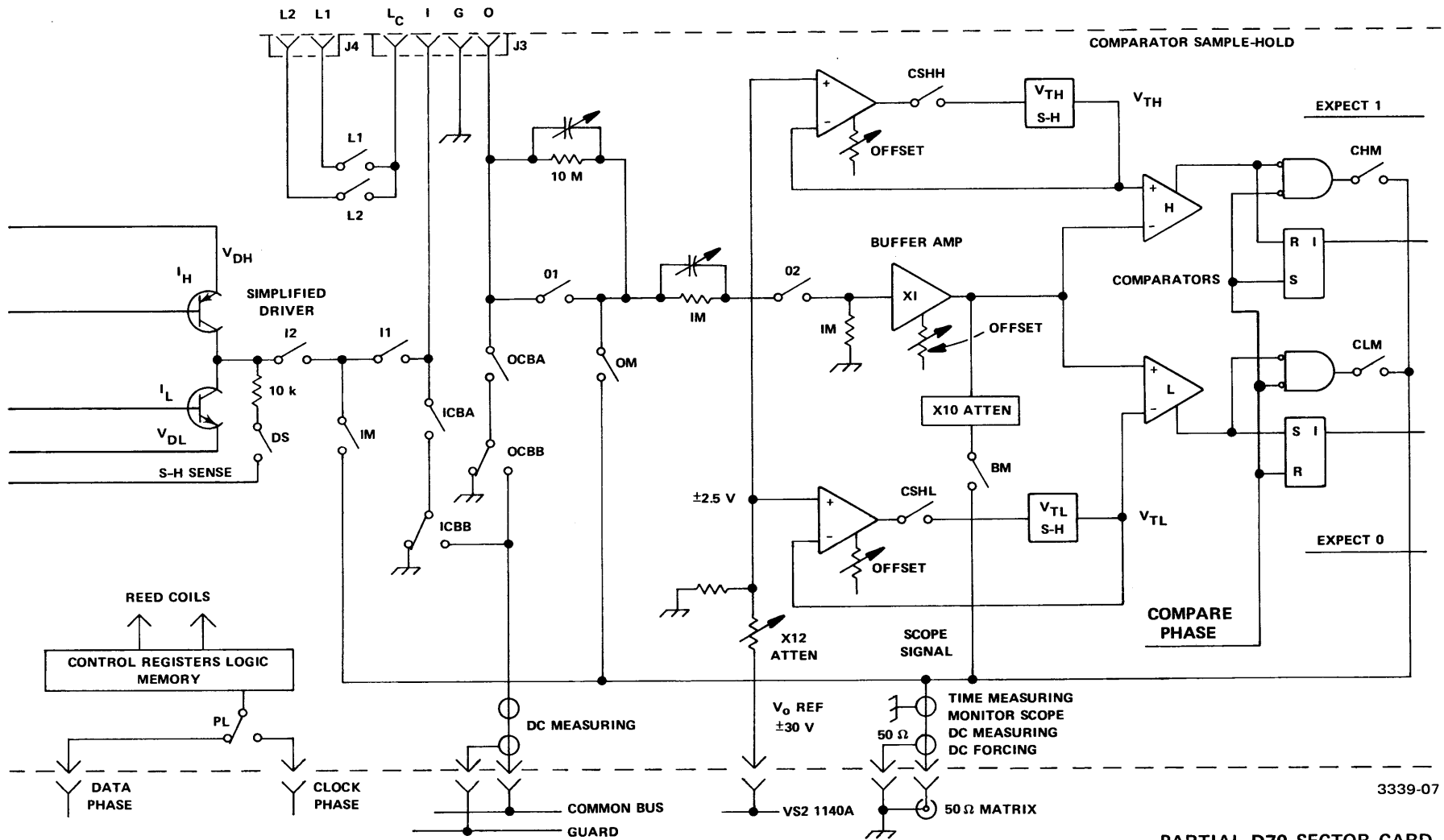


Table 3-2. Effect of Some CONNECT Statements on the Reed Switches

3339-07

PARTIAL D70 SECTOR CARD

Reed Switch Names

CONNECT Statements	DSHH	DSHL	DS	CSHH	CSHL	I1	I2	IM	ICBA	ICBB	OM	BM	CLM	CHM	OCBA	OCBB	O1	O2	L1	L2	
DRIVER TO SCOPE _n ,AUX _n						0	1	1													
COMPARATOR + TO SCOPE _n ,AUX _n											0	0	0	1							
COMPARATOR - TO SCOPE _n ,AUX _n											0	0	1	0							
BUFFER TO SCOPE _n ,AUX _n											0	1	0	0							
INPUT TO COMMON									1	1											
INPUT TO DRIVER						1	1	0													
INPUT TO GROUND									1	0											
INPUT TO SCOPE _n ,AUX _n						1	0	1													
OUTPUT TO COMPARATOR,5V																	1	1			
OUTPUT TO COMPARATOR,30V																	0	1			
OUTPUT TO LOAD1																				1	
LOAD1 ON																				1	1
OUTPUT TO LOAD2																					1
LOAD2 ON																					1
OUTPUT TO <u>COMMON</u> <u>GND</u>															1	1					
															1	0					
OUTPUT TO SCOPE _n ,AUX _n											1	0	0	0			1				
DISCONNECT Statements																					
INPUT FROM ALL						0	0	0	0	0											
OUTPUT FROM ALL											0	0	0	0	0	0	0	0	0	0	0

Reed Switch Functions Legend

DSHH (Driver Sample/Hold High) – Connects memory element to sampling circuit.

DSHL (Driver Sample/Hold Low) – Connects memory element to sampling circuit.

DS (Driver Sense) – Senses output voltage of driver to be applied to I-pin in order to set V_{DH} (voltage, driver-high) or V_{DL} (voltage, driver-low) to the desired value. SH strobe timing controls DSHH and DSHL so that in a sample/hold cycle DS closes first and opens last.

CSHH (Comparator Sample/Hold High) – Connects memory element to sampling circuit.

CSHL (Comparator Sample/Hold Low) – Connects memory element to sampling circuit.

I1 (Input Switch #1)* – Connects I-pin to driver through I2, or to matrix through IM with the driver disconnected.

I2 (Input Switch #2)* – Connects driver to I-pin or to matrix with I-pin disconnected (routed through DC subsystem for functional preconditioning of I-pin before a DC current measurement).

IM (Input (I) Pin to Matrix)* – Connects either driver or I-pin to the matrix with the other disconnected.

ICBA (Input to Common Bus, Switch A) – Input (I) pin to ground or to common bus, depending on ICBB.

ICBB (Input to Common Bus, Switch B) – (1) Grounds input (I) pin, or (2) connects I-pin to common bus, or (3) grounds junction of ICBA and ICBB to stop cross-talk between common bus and I-pin when ICBA is open.

OM (Output to Matrix) – Connects output (O) pin to matrix.

BM (Buffer to Matrix) – Connects output of X1 buffer amplifier through X10 attenuator to matrix and thence to monitor scope. Total attenuation between DUT and monitor is X20 or X120.

CLM (Comparator Low to Matrix) – Output of low comparator via matrix to ΔT subsystem. If START signal, via channel 2; if STOP signal, via channel 1.

CHM (Comparator High to Matrix) – Output of high comparator via matrix to ΔT subsystem, as above.

OCBA (Output to Common Bus, Switch A) – Output (O) pin to ground or to common bus, depending on OCBB.

OCBB (Output to Common Bus, Switch B) – (1) Grounds output (O) pin, or (2) connects O-pin to common bus, or (3) grounds junction of OCBA and OCBB to stop cross-talk between common bus and O-pin when OCBA is open.

O1 (Output Switch #1) – Controls X1-X6 attenuator to buffer and comparator. (1) When closed, comparator range is ± 5 V. (2) When open, comparator range is ± 30 V.

O2 (Output Switch #2) – Disconnects buffer for current and resistance measurements at O-pin.

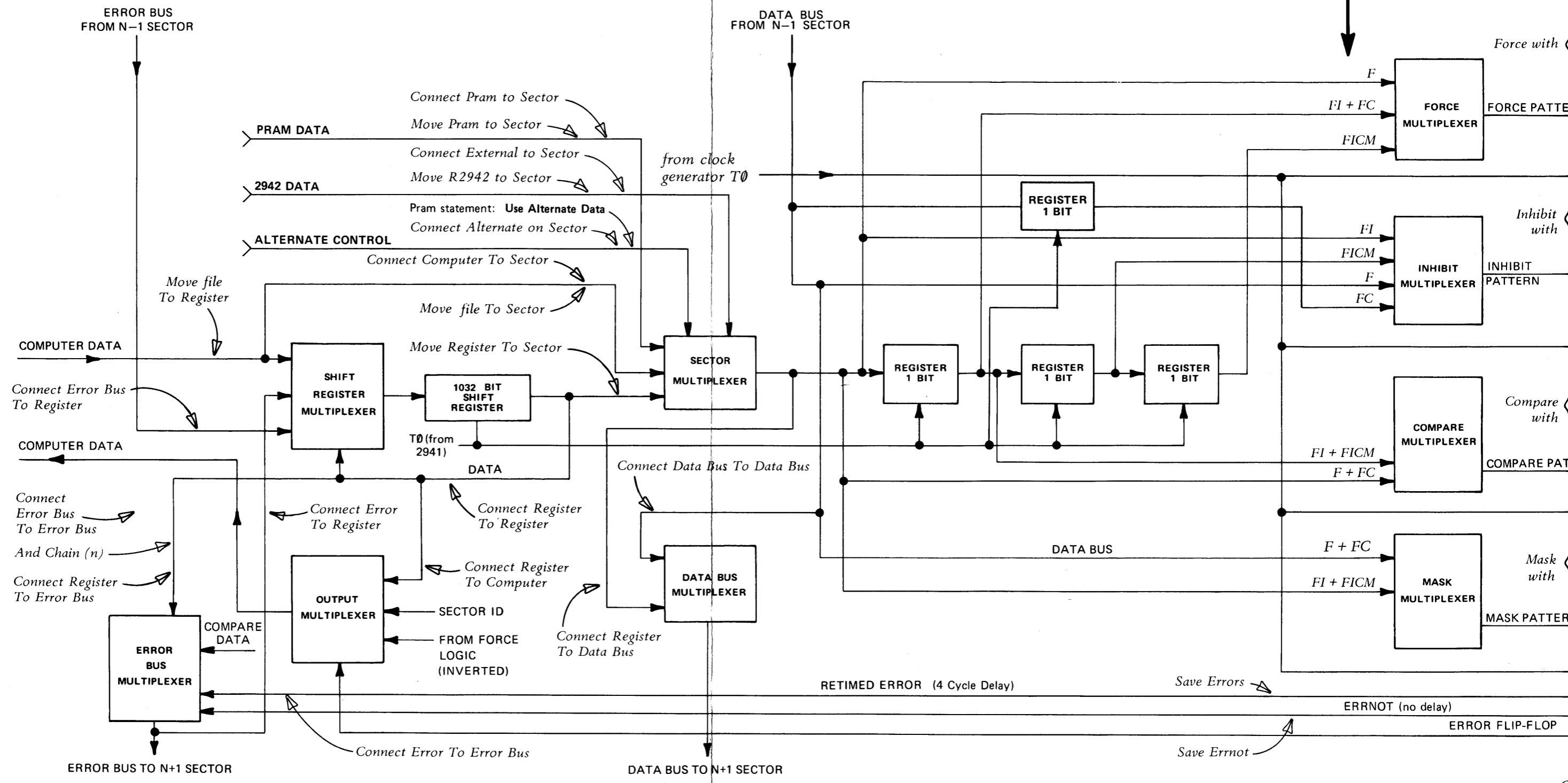
L1 (Load Module #1) – Connects load module #1 to load-common (LC) pin which is then strapped to either the I-pin or O-pin.

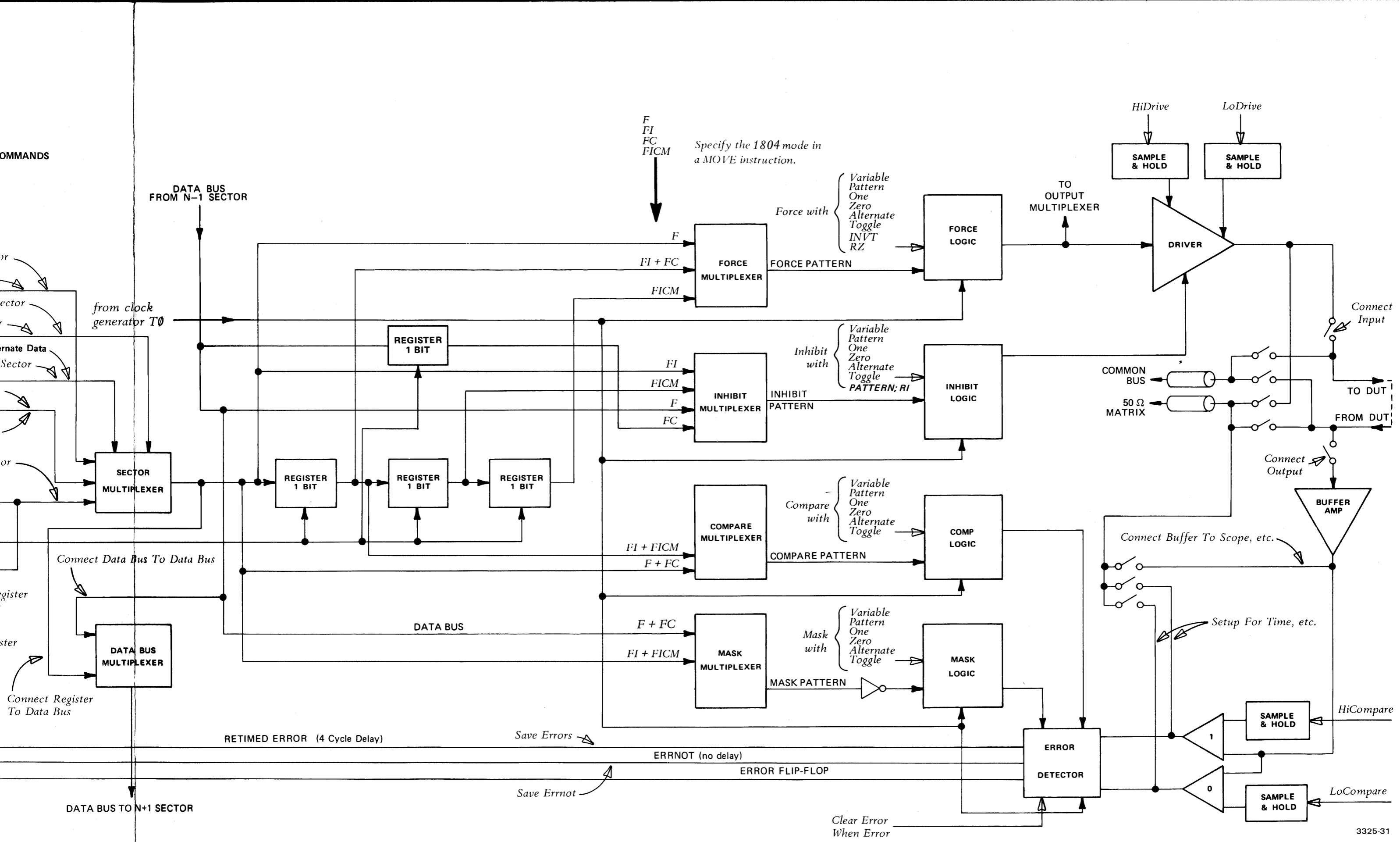
L2 (Load Module #2) – Same as above for Load Module #1.

*Any two or none of these can be closed simultaneously.

NOTE: Titles in *italics> are TEKTEST III TEST LANGUAGE COMMANDS*

F
FI
FC
FICM
Specify the 1804 mode in a MOVE instruction.





DATA FLOW DIAGRAM OF THE D70 SECTOR CARD

SECTION 4: FUNCTIONAL TESTING

The statements and functions described in this section pertain to functional testing. Your system may have up to 64 sector cards, each of which has a shift register that holds as many as 4104 bits of data. One bit from the shift registers is required to define each sector card function. The bits of data form a pattern. Each row of data in the pattern supplies the input and output data to define one state or operating condition of the DUT.

Functional testing consists of applying a pattern to the DUT pins, one row at a time, and checking that the output of the DUT responds as expected. Patterns are moved to the shift registers under control of the test program. The pattern is clocked by the Clock Generator.

Each sector card can perform four clock-rate functions: FORCE, INHIBIT, COMPARE, and MASK. These functions are abbreviated F, I, C, and M, and are all used to perform functional tests.

Functional testing permits serial chaining, in which the shift register from Sector Card (n-1) is connected in series with the shift register of Sector Card (n). This allows shift register patterns greater than 4104.

Statements and Functions Described in this Section

- **FORCE** Selects the source data forced between specific logic levels on DUT input pins
- **COMPARE** Checks the level of DUT output pins against expected levels
- **INHIBIT** Disconnects the driver associated with DUT input pins
- **MASK** Disconnects the comparator output associated with DUT output pins
- **LOAD** Transfers blocks of data from a source file to the pin electronics card shift registers
- **MOVE** Moves the pattern to and from the DUT
- **TIMOUT** Returns the current state of the TIMEOUT system flag
- **BURST** Determines the programmable clock generator mode
- **DRIVE** Reads the pin electronics card force flip-flops

FORCING LOGIC LEVELS

When a test runs, the sector-card driver circuits force logic levels on the input pins of the DUT. The **FORCE** statement selects the pins forced and the source data.

You must pre-establish the force paths with a **CONNECT** statement. Set the drive levels with the **HIDRIVE** and **LODRIVE** statements, which must precede the **FORCE** statement.

FORCE programs inhibit to zero; thus no inhibiting occurs. **FORCE** must therefore precede an **INHIBIT** statement on the same pins.

You may force data in three timing modes: Return-to-Zero (RZ), Non Return-to-Zero (NRZ), and Return-to-Complement (RC).

The **FORCE** statement format is:

FORCE pins WITH {
PATTERN
ALTERNATE PATTERN AND ONE
ALTERNATE PATTERN AND ZERO
ONE
ZERO
TOGGLE ONE AND ZERO
TOGGLE ZERO AND ONE
}

{ ;RZ, INVERT
;RZ
;RC }

-or-

FORCE pinlist WITH {
expression
ALTERNATE PATTERN AND expression
TOGGLE expression
}

{ ;RZ, INVERT
;RZ
;RC }

Elements

- **PATTERN** is pattern source data, which is established by **MOVE** and **CONNECT** statements.
- **ONE** is a logic one (1), which is fixed data.
- **ZERO** is a logic zero (0), which is fixed data.
- **expression** is any legal expression. The value of **expression** is interpreted as a 16-bit binary integer. Each of the 16 bits is sent to a different pin in the pinlist: the least significant bit (the rightmost bit) to the first pin in the pinlist, the next bit to the second pin in the pinlist, and so on.

If less than 16 pins exist in the pinlist, the bits without corresponding pins are ignored. Conversely, if more than 16 pins exist in the pinlist, pin 17, pin 18, etc., are not sent pattern bits.

- **ALTERNATE** permits switching between **PATTERN** and the second source data (**ONE**, **ZERO**, or **expression**), starting with **PATTERN**. **ALTERNATE** must be in a **MOVE** statement.
- **ALTERNATE PATTERN AND ONE** alternates between **PATTERN** and **ONE**, starting with **PATTERN**.
- **ALTERNATE PATTERN AND ZERO** alternates between **PATTERN** and **ZERO**, starting with **PATTERN**.
- **ALTERNATE PATTERN AND expression** alternates between **PATTERN** and the value of **expression**, starting with **PATTERN**.
- **TOGGLE** alternates between the first source data (**ONE**, **ZERO**, or **expression**) and its complement.
- **TOGGLE ONE AND ZERO** alternate between **ONE** and **ZERO**. The logic zero level becomes effective immediately. The logic one level becomes effective at the beginning or the first clock cycle of the **MOVE** statement.
- **TOGGLE ZERO AND ONE** alternates between **ZERO** and **ONE**. The logic one level becomes effective immediately. The logic zero level becomes effective on the first clock cycle of the **MOVE** statement.
- **TOGGLE expression** alternates between the value of **expression** and its complement.

Optional Elements

- **RZ** indicates the Return-to-Zero timing mode. In this mode, the **CYCLE** statement does not clock the forcing level. Instead, the **DATAPHASE** and **PHASE** statements specify the time during which the system forces the data. When you specify **RZ**, the data level remains at a logic zero until a logic one is forced.

In the Non Return-to-Zero timing mode (**NRZ**), the **CYCLE** statement directly controls the timing of the forcing level. The system assumes the **NRZ** mode if you omit **RZ**.

- **INVERT** inverts the forcing level.

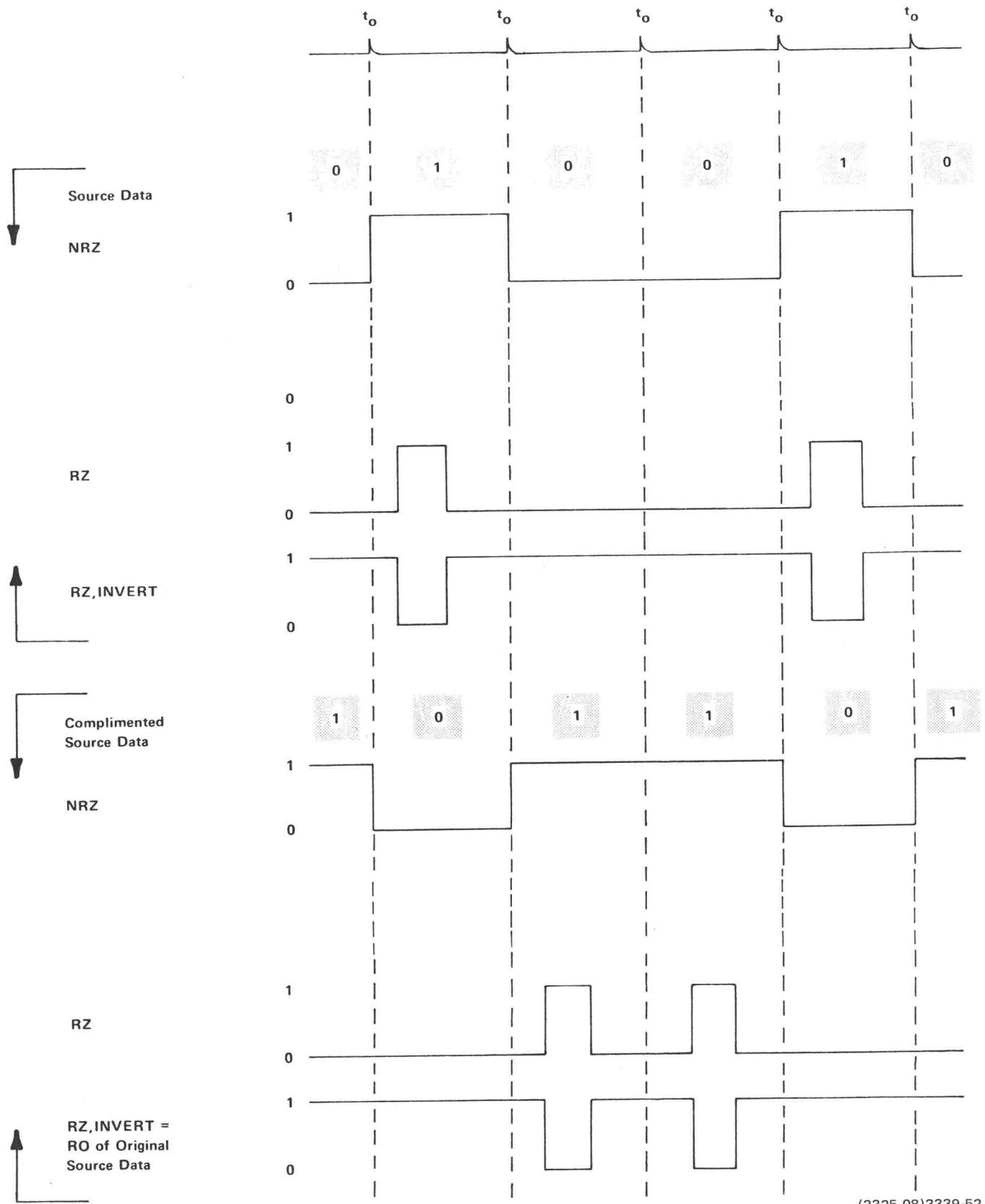
Return-to-One Timing Mode

The complemented source data plus **RZ,INVERT** indicate the Return-to-One (**R0**) timing mode. In this mode the data level remains at a logic one until a logic zero is forced.

Figure 4-1A illustrates the Return-to-One timing.

RC indicates Return-to-Complement mode. In this mode, the data being forced on the DUT is inverted when the phase is false and is not inverted when the phase is true.

Figure 4-1B illustrates Return-to-Complement timing.



(3325-08)3339-52

Figure 4-1A. Return-to-One (RO) Timing

Driver Logic Preset

The statement sequence

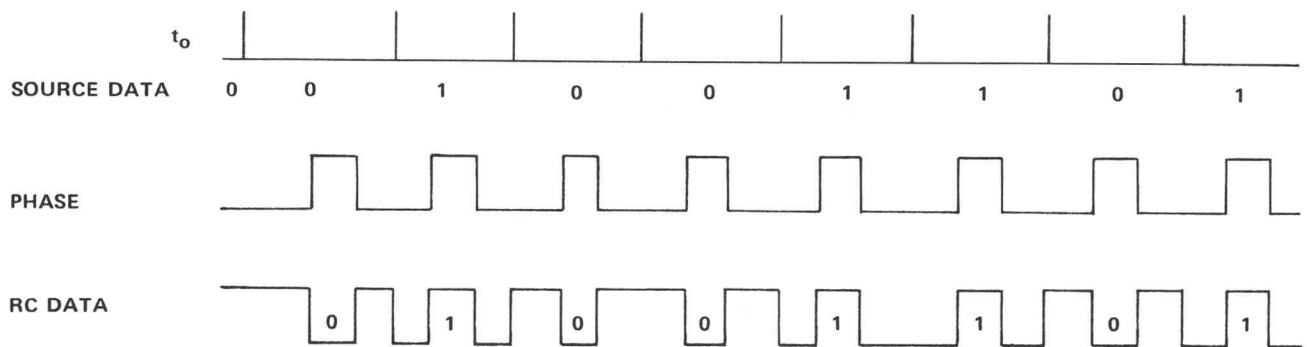
```
FORCE pins WITH ONE;RZ  
FORCE pins WITH ZERO
```

causes the driver to go to a **ONE** state for a few microseconds before it goes to the specified **ZERO** state. This happens because the one statement **FORCE pins WITH ZERO** must change two control bits when switching the data source from **ONE;RZ** to **ZERO**.

To avoid this problem, use the statement sequence shown below to perform the same operation. In this case, the separate statements **FORCE pins WITH ZERO;RZ** and **FORCE pins WITH ZERO** each change one of the control bits.

```
FORCE pins WITH ONE;RZ  
FORCE pins WITH ZERO;RZ  
FORCE pins WITH ZERO
```

Figure 4-2 shows the driver logic preset (before the first clock cycle).



(3325-09)3339-53

Figure 4-1B. Return-to-Complement (RC) Data



FORCE
FORCE,COMPARE

Source	Executed Before First Clock Cycle				During First Clock Cycle				During Second Clock Cycle			
	Elements				Elements				Elements			
	None	RC	RZ	RZ,INVERT	None	RC	RZ	RZ,INVERT	None	RC	RZ	RZ,INVERT
PATTERN	*	*	0	1	PAT	$\overline{\text{PAT}} \oplus \text{PHASE}$	$\text{PAT} \cdot \text{PHASE}$	$\overline{\text{PAT}} + \overline{\text{PHASE}}$	PAT	$\overline{\text{PAT}} \oplus \text{PHASE}$	$\text{PAT} \cdot \text{PHASE}$	$\overline{\text{PAT}} + \overline{\text{PHASE}}$
ALTERNATE PATTERN AND ONE	*	*	0	1	PAT	$\overline{\text{PAT}} \oplus \text{PHASE}$	$\text{PAT} \cdot \text{PHASE}$	$\overline{\text{PAT}} + \overline{\text{PHASE}}$	1	$\overline{\text{PHASE}}$	PHASE	$\overline{\text{PHASE}}$
ALTERNATE PATTERN AND ZERO	*	*	0	1	PAT	$\overline{\text{PAT}} \oplus \text{PHASE}$	$\text{PAT} \cdot \text{PHASE}$	$\overline{\text{PAT}} + \overline{\text{PHASE}}$	0	PHASE	0	1
ONE	1	0	0	1	1	$\overline{\text{PHASE}}$	PHASE	$\overline{\text{PHASE}}$	1	$\overline{\text{PHASE}}$	PHASE	$\overline{\text{PHASE}}$
ZERO	0	1	0	1	0	PHASE	0	1	0	PHASE	0	1
TOGGLE ONE AND ZERO	0	1	0	1	1	$\overline{\text{PHASE}}$	PHASE	$\overline{\text{PHASE}}$	0	PHASE	0	1
TOGGLE ZERO AND ONE	1	0	0	1	0	PHASE	0	1	1	$\overline{\text{PHASE}}$	PHASE	$\overline{\text{PHASE}}$

- * = No change
- PAT = the source data
- $\overline{\text{PAT}}$ = the complemented source data
- PHASE = any of the five clock phases (Phase 1 through Phase 4, or Dataphase)
- $\overline{\text{PHASE}}$ = the complemented phase
- $\text{PAT} \cdot \text{PHASE}$ = logically ANDs the source data and the phase
- $\overline{\text{PAT}} + \overline{\text{PHASE}}$ = logically ORs the complemented source data and the complemented phase
- $\overline{\text{PAT}} \oplus \text{PHASE}$ = logically exclusive ORs the complemented data with the phase

Figure 4-2. Driver Logic Preset (Before First Clock Cycle)

COMPARING LOGIC LEVELS

The **COMPARE** statement checks the level of DUT output pins against the expected levels specified by source data. Set the reference levels with the **HICOMPARE** and **LOCOMPARE** statements, which must precede the **COMPARE** statement.

COMPARE programs mask to zero; thus no masking occurs. **COMPARE** must therefore precede a **MASK** statement on the same pins.

The **COMPARE** statement format is:

COMPARE pins WITH {
PATTERN
ALTERNATE PATTERN AND ONE
ALTERNATE PATTERN AND ZERO
ONE
ZERO
TOGGLE ONE AND ZERO
TOGGLE ZERO AND ONE
}

-or-

COMPARE pinlist WITH {
expression
ALTERNATE PATTERN AND expression
TOGGLE expression
}

Refer to the **Elements** discussion in the **FORCE** statement for an explanation of these elements.

INHIBITING THE DRIVER

The **INHIBIT** statement disconnects the driver associated with DUT input pins at clock rate. When inhibited, the driver switches to a high impedance state. The driver is inhibited in the manner specified by the source data: a logic one corresponds to the inhibit state; a logic zero corresponds to the no inhibit state.

Since the **FORCE** statement programs inhibit to zero, an **INHIBIT** statement must follow a **FORCE** statement on the same pins.

The **INHIBIT** statement format is:

INHIBIT pins WITH {
 PATTERN
 ALTERNATE PATTERN AND ONE
 ALTERNATE PATTERN AND ZERO
 ONE
 ZERO
 TOGGLE ONE AND ZERO
 TOGGLE ZERO AND ONE
}

-or-

INHIBIT pinlist with {
 expression
 ALTERNATE PATTERN AND expression
 TOGGLE expression
}

-or-

INHIBIT pinlist with PATTERN;RI

Refer to the **Elements** discussion in the **FORCE** statement for an explanation of these elements.

Notes

The **INHIBIT WITH ONE** statement inhibits the driver.

The **INHIBIT WITH ZERO** statement enables the driver. This allows the driver to continue forcing at the programmed state.

Applications

Some applications of the INHIBIT statement are to:

- Inhibit the driver when the sector is connected to an input pin.
- Inhibit the driver when the sector is connected to an I/O bus pin on the DUT, and the sector is comparing.

RI return to INHIBIT. This modifier causes the driver to be inhibited when the force phase is false, and causes the driver to be not inhibited when the phase is true.

DISCONNECTING THE COMPARATOR OUTPUT

The **MASK** statement disconnects the comparator output associated with DUT output pins at clock rate. This disregards any errors that occur at the DUT. The comparator is masked in the manner specified by the source data: a logic one corresponds to the mask state; a logic zero corresponds to the no mask state.

Since the **COMPARE** statement programs mask to zero, a **MASK** statement must follow a **COMPARE** statement on the same pins.

The **MASK** statement format is:

MASK pins WITH {
PATTERN
ALTERNATE PATTERN AND ONE
ALTERNATE PATTERN AND ZERO
ONE
ZERO
TOGGLE ONE AND ZERO
TOGGLE ZERO AND ONE
}

-or-

MASK pinlist WITH {
expression
ALTERNATE PATTERN AND expression
TOGGLE expression
}

Refer to the **Elements** discussion in the **FORCE** statement for an explanation of these elements.

Notes

The **MASK WITH ONE** statement masks the comparator. The comparator continues to observe the DUT output, but any detected errors are blocked.

The **MASK WITH ZERO** statement unmask the comparator. This allows the comparator to continue at the programmed state. Errors are then detected.

Applications

Some applications of the MASK statement are to:

- Mask the comparator when the DUT output is undefined.
- Mask the comparator when DUT output data is of no interest.
- MASK and COMPARE a logic one (when automode switching is enabled) to switch from COMPARE-MASK to FORCE-INHIBIT.

TRANSFERRING DATA

The **LOAD** statement transfers blocks of data from a source file located on a disk or in core, to the sector-card shift registers indicated by a pinlist.

LOAD formats and loads data into the shift registers before a **MOVE** statement. **MOVE** then draws this data from the registers to perform the sector functions defined by the **FORCE**, **INHIBIT**, **COMPARE**, and **MASK** statements. Although similar to the **MOVE** statement, **LOAD** does not clock the DUT while the transfer occurs, and the destination of the data can only be the shift registers.

Unless you include the **APPEND** option in the **LOAD** statement, the first bit to enter the shift registers always loads into the start of the registers, i.e., **INDEX = 1**.

Whenever the system encounters a **LOAD** statement, it clears the error flip-flops before executing the **LOAD**.

The **LOAD** statement format is:

LOAD [**FROM**] { **CORE**
DISK } **filnam** [[**expression1**,] **expression2**] **TO pinlist**

[**WITH mode**]

[**AND** { **APPEND (location)**
CHAIN (number)
NO CONNECTS } ... **AND** { **APPEND (location)**
CHAIN (number)
NO CONNECTS }]

Elements

- **CORE** indicates that the data transfers from the specified core file.
- **DISK** indicates that the data transfers from the specified pattern file on the current disk in use.
- **filnam** is the name of the type PAT source file from which the data transfers.
- **expression1** and **expression2** are any legal expressions. The system rounds the values of the expressions to integers. **expression1** indicates the first source file pattern row to be loaded into the shift registers. The default value of **expression1** is one. **expression2** indicates the last pattern row to be loaded.
- **CORE filnam** transfers all the data in the specified core file.
- **CORE filnam (expression2)** transfers rows 1 through **expression2** of the specified core file.
- **CORE filnam (expression1,expression2)** transfers rows **expression1** through **expression2** of the specified core file.
- **DISK filnam** transfers all the data in the specified pattern file.
- **DISK filnam (expression2)** transfers rows 1 through **expression2** of the specified pattern file.
- **DISK filnam (expression1,expression2)** transfers rows **expression1** through **expression2** of the specified pattern file.

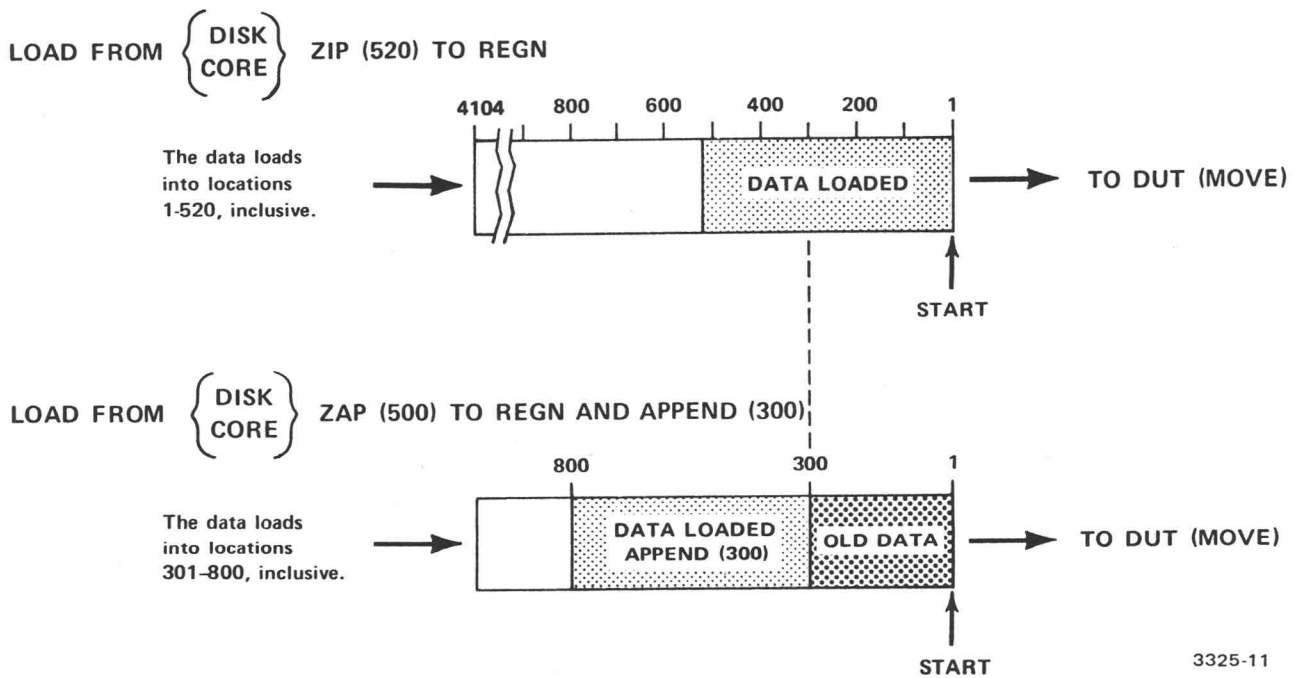
Optional Elements

- **mode** specifies the sector card operating mode. The modes are:
 - F,I,C,M (or simply F). If you omit **mode**, the system assumes F,I,C,M.
 - FC
 - FI,CM (or simply FI)
 - FICM

The following options may be used only once in a LOAD statement:

- **APPEND (location)** defines where in the shift registers the data should be placed. **location** is any legal expression. The system rounds the value of **location** to an integer. The value of (**location** + 1) specifies the first shift register row that is appended. The default value of **location** is zero. If you omit **APPEND**, the first pattern row is always appended to zero, and loads into the start of the registers, i.e., INDEX = 1. Ensure that the pattern length plus the append is less than 4104 bits.

Example:



3325-11

- **CHAIN (number)** causes the pattern to be loaded into **number** consecutive shift registers for each pin in the pinlist. **number** is any integer between 1 and 32. **CHAIN** allows patterns too long to be contained in a single shift register to be loaded into **number**-shift registers, connected in a series. This is serial chaining. The data path for serial chaining is the E BUS. Refer to the **Connections** discussion in the **MOVE** statement for additional information on the connections.
- **NO CONNECTS** suppresses all data path connections which are normally made by the **LOAD** statement. The programmer must make his own connections.

Notes

- When programming two consecutive **LOAD** statements, follow the first **LOAD** with a **CONNECT REGISTER TO REGISTER** statement. This prevents loss of data loaded with the first **LOAD** statement.

```
LOAD sourcefile TO PIN1
CONNECT REGISTER TO REGISTER ON PIN1
LOAD sourcefile TO PIN2
```

- The table in the **Notes** discussion of the **MOVE** statement indicates how the **LOAD** and **MOVE** statements recirculate the shift registers and connect the mode control bits.

MOVING THE PATTERN TO AND FROM THE DUT

The **MOVE** statement is the system ignition for functional testing. It is used to move the pattern to and from the DUT in the manner specified. The **MOVE** statement:

- accesses a pattern on a pattern source (in the shift registers, in core, or on a disk);
- formats the pattern according to the specified mode;
- programs and enables the clock generator; and
- moves the pattern data to the DUT to perform the sector card functions.

MOVE contains an implicit **CONNECT . . . TO SECTOR** on the specified pinlist.

All shift registers are clocked by **MOVE**, regardless of whether or not they are included in the **MOVE** pinlist. Depending upon the source data specified in **FORCE,COMPARE**, **FORCE**, **COMPARE**, **INHIBIT**, or **MASK** statements, errors may be generated by sector cards not included in the **MOVE** statement. Drivers may change states even though the **MOVE** pinlist does not include them. Pins included in the **MOVE** pinlist are automatically connected to a pattern source. Use **CONNECT** and **LOAD** statements to connect the pins and load the pattern.

Whenever the system encounters a **MOVE** statement, it clears the error flip-flops before executing the **MOVE**.

The **MOVE** statement stops by one of these methods:

- The end of the clock generator pulses specified by the **MOVE** statement.
- Error or pass conditions occur, and **WHEN ERROR** or **WHEN PASS** statements are programmed.
- Pressing the **STOP** button on the Test Station Control Unit.
- If the execution time for the **MOVE** exceeds the value specified by the **TIMEOUT** option.

The MOVE statement format is:

MOVE [FROM] REGISTER ([expression1,] expression2) TO pinlist [WITH mode]

AND { ALTERNATE CHAIN (number)
SAVE { ERRORS
ERRNOT
DATA }
NO CONNECTS
TIMEOUT (time) } ... AND { ALTERNATE CHAIN (number)
SAVE { ERRORS
ERRNOT
DATA }
NO CONNECTS
TIMEOUT (time) }

-or-

MOVE [FROM] { CORE
DISK } filnam [[expression1,] expression2] TO pinlist [WITH mode]

AND { ALTERNATE CHAIN (number)
SAVE { ERRORS
ERRNOT
DATA }
NO CONNECTS
APPEND (location) } ... AND { ALTERNATE CHAIN (number)
SAVE { ERRORS
ERRNOT
DATA }
NO CONNECTS
APPEND (location) }

-or-

MOVE [FROM] { R2942
PRAM } ([expression1,] expression2) TO pinlist [WITH mode]

AND { ALTERNATE CHAIN (number)
SAVE { ERRORS
ERRNOT
DATA }
NO CONNECTS
APPEND (location) } ... AND { ALTERNATE CHAIN (number)
SAVE { ERRORS
ERRNOT
DATA }
NO CONNECTS
APPEND (location) }

Elements

- **R2942** indicates that the pattern source is the R2942 hardware pattern generator.
- **PRAM** indicates that the pattern source is the PRAM (pattern random access memory).
- **REGISTER** indicates that the pattern source is the shift registers.
- **CORE** indicates that the pattern source is in core.
- **DISK** indicates that the pattern source is on the current disk in use.
- **filnam** is the name of the pattern file in core or on the disk.
- **expression1** and **expression2** are any legal expressions. They specify how many DUT cycles the MOVE statement generates. The number of DUT cycles generated is (**expression2** - **expression1** + 1). The system rounds the values of the expressions to integers.

expression1 indicates the first pattern row of the test. The default value of **expression1** is one.
expression2 indicates the last pattern row of the test. **expression1** is useful only when you are not serial chaining.

Examples:

MOVE REGISTER (10) . . . AND SAVE ERRORS

Ten DUT cycles occur, with rows 1 through 10 of the pattern moved to the DUT.

MOVE REGISTER (11,20) . . . AND SAVE ERRORS

Again, ten DUT cycles occur, but rows 11 through 20 of the pattern are moved to the DUT.

You may program the clock generator to free run during a MOVE REGISTER statement by specifying **expression2** equal to zero.

Example:

MOVE REGISTER (10,0) . . .

The clock generator begins free running with bit 10 in the shift registers. Bit 10 is the first bit moved to the DUT.

The clock generator continues free running until:

- An error occurs and WHEN ERROR is programmed.
 - A pass condition occurs and WHEN PASS is programmed.
 - The STOP button on the Test Station Control Unit is pressed.
 - A timeout occurs as a result of the **TIMEOUT** option.
-
- **REGISTER (expression2)** indicates that the pattern source is rows 1 through **expression2** of the register pattern data.
 - **REGISTER (expression1,expression2)** indicates that the pattern source is rows **expression1** through **expression2** of the register pattern data.
 - **CORE filnam** indicates that the pattern source is all the data in the specified core file.
 - **CORE filnam (expression2)** indicates that the pattern source is rows 1 through **expression2** of the specified core file.
 - **CORE filnam (expression1,expression2)** indicates that the pattern source is rows **expression1** through **expression2** of the specified core file.
 - **DISK filnam** indicates that the pattern source is all the data in the specified pattern file.
 - **DISK filnam (expression2)** indicates that the pattern source is rows 1 through **expression2** of the specified pattern file.
 - **DISK filnam (expression1,expression2)** indicates that the pattern source is rows **expression1** through **expression2** of the specified pattern file.

Optional Elements

- **mode** specifies the sector card operating mode. The modes are:
 - F,I,C,M (or simply F). If you omit **mode**, the system assumes F,I,C,M.
 - FC
 - FI,CM (or simply FI)
 - FICM

The following options may be used only once in a MOVE statement:

- **ALTERNATE** must be included in the MOVE statement if FORCE,COMPARE, FORCE, COMPARE, INHIBIT, or MASK statements specified **ALTERNATE PATTERN** as the pattern source, and you wish the source data to change at the beginning of each clock cycle. With **ALTERNATE** specified, the number of DUT cycles is twice the number of words specified by **expression1** and **expression2**. If you **ALTERNATE** and specify a **SAVE** option, only errors detected by the pattern are saved.

Example:

MOVE REGISTER (10) . . . AND ALTERNATE

Twenty cycles are sent to the DUT. If you specified a **SAVE** option, only ten bits of data (from the error circuits) would be saved.

- **CHAIN (number)**. In a MOVE REGISTER statement, **CHAIN** is legal with or without a **SAVE** option. In a MOVE CORE or MOVE DISK statement, **CHAIN** is legal only with a **SAVE** option.

CHAIN (number) causes **number** registers on the clockwise side of each pin in **pinlist** to be serially chained. **number** is an integer between 1 and 32. The data path for serial chaining is the EBUS. Refer to the **Connections** discussion for additional information on the connections.

CHAIN (number) AND SAVE causes **number** registers on the counterclockwise side of each pin in **pinlist** to be serially chained to contain error output from the sector comparators corresponding to **pinlist**.

In a MOVE FROM REGISTER statement, **CHAIN AND SAVE** re-establishes the connections between registers chained on the clockwise side of each pin by a LOAD statement. Therefore, **number** registers are chained on both sides of the sectors in **pinlist**.

- **SAVE ERRORS** causes error information from the error detectors on each sector card to be saved in the shift registers. The last four rows are not passed into the shift registers, and the first four pattern rows saved are always zero. To save all the errors, four extra clock cycles must be specified by the MOVE statement. When examining the contents of the shift register, simply ignore the first four rows and assume that the first pattern word is in the fifth row.

Example:

MOVE REGISTER (8) . . . AND SAVE ERRORS

This statement moves four rows of data and saves errors.

- **SAVE ERRNOT*** causes error information from the error detectors on each sector card to be saved in the shift registers. A one is stored when an error occurs. A zero is stored when no error occurs. The error information is delayed one cycle before entering the shift registers. This differs from **SAVE ERRORS**, whose information appears in the shift registers four words late.

In order for **ERRNOT** to perform correctly, at least 70 ns must elapse between the time the error occurs and the next t_0 . This enables the result to be stable at the shift register clock. The system detects errors up to the trailing edge of the latest comparator gate.

- **SAVE DATA** causes the external input pattern for **pinlist** to be saved in the shift registers. Since the pattern stored is from the comparison circuit, program the following statements to make the force bits available for storage:

**COMPARE forcepins WITH PATTERN
MASK forcepins WITH ONE**

The pattern words are delayed one cycle before entering the shift registers. Therefore, word (n) of the pattern is stored as word (n+1) in the shift registers. **SAVE DATA** operates correctly up to a 20 MHz clock rate.

- **NO CONNECTS** suppresses all data path connections which are normally made by the MOVE statement. The programmer must make his own connections.

*Note that the logic polarity of **SAVE ERRNOT** in D70 sectors is opposite that of **SAVE ERRNOT** in D1B sectors.

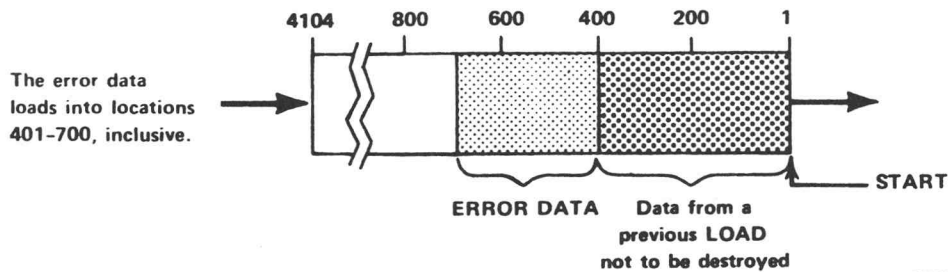
- **TIMEOUT (time)** is legal in a MOVE REGISTER, MOVE PRAM, or MOVE R2942 statement; it is not legal in a MOVE CORE or MOVE DISK statement. **TIMEOUT** specifies the maximum time in seconds allowed for the completion of the statement. **time** is a constant between .00 and .99. If the execution time for the MOVE exceeds the value specified, the system aborts MOVE. Use **TIMEOUT** to guard against endless looping in the event of a mistake in the pattern program.

If a WHEN ERROR or WHEN PASS statement is enabled, and an error or pass condition occurs, control passes to the next sequential statement in the program. The occurrence of a timeout abort does not affect the contents of the ERROR variable.

- **APPEND (location)** is legal only in a MOVE CORE or MOVE DISK statement, and only with a **SAVE** option. You may not use the **CHAIN** and **APPEND** elements in the same MOVE statement. **APPEND (location)** defines where in the shift registers the error data should be saved. **location** is any legal expression. The system rounds the value of **location** to an integer. The value of **(location + 1)** specifies the first shift register row that is appended. The default value of **location** is zero. The data is placed directly after **location**. If you omit **APPEND**, the first data row is always appended to zero, and loads into the start of the registers, i.e., **INDEX = 1**.

Example:

MOVE FROM CORE VIP (1,300) TO PINX AND SAVE ERRORS AND APPEND (400)



3325-12

Connections

The following charts show the connections made and broken with the LOAD and MOVE statements. Sector Card (n) refers to the sector card specified in the pinlist. Sector Card (n+1), Sector Card (n-1), etc., refer to sector cards which do not explicitly appear in the pinlist but are programmed by serial chaining through the CHAIN element option.

SECTOR CARD (n)				CONNECTIONS MADE (1) OR BROKEN (0)																			
DESTINATION →				EBUS				REGISTER				COMPUTER				SECTOR							
SOURCE →				ERRNOT	ERROR	DATA	REGISTER	REGISTER	COMPUTER	EBUS OUT	EBUS IN	ERROR	CARD I. D.	REGISTER	FORCE	EXTERNAL	COMPUTER	PRAM	REGISTER				
INSTRUCTION EXECUTED																							
LOAD { CORE DISK } (executed when clock is off)			1	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0				
	CHAIN (3)		2	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0				
LOAD { CORE DISK } (executed when clock is on)			3					0	1	0	0												
	CHAIN (3)		4					0	0	0	1												
MOVE { CORE DISK }			5	1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0				
		SAVE	6	*See lines 12-14				0	0	1	0	*See lines 12-14				0	1	0	0				
	CHAIN (3)	SAVE	7					0	0	0	1					0	1	0	0	0	1	0	0
MOVE REGISTER	CHAIN (3)	SAVE	8					0	0	0	1					0	0	0	1	0	0	0	1
		SAVE	9	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1				
	CHAIN (3)		10	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1				
			11	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1				
	SAVE	ERRORS	12	0	1	0	0	*See lines 8 (CHAIN) or 9 (NO CHAIN)				0	1	0	0	0	0	0	1				
		ERRNOT	13	1	0	0	0					1	0	0	0	0	0	0	0	0	0	1	
		DATA	14	0	0	1	0					0	0	1	0	0	0	1	0	0	0	0	1
MOVE PRAM	CHAIN (3)	SAVE	15	*See lines 12-14				0	0	0	1	*See lines 12-14				0	0	1	0				
		SAVE	16					0	0	1	0					0	0	1	0	0	1	0	0
			17	1	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0				
		SAVE	ERRORS	18	0	1	0	0	*See lines 8-9				0	1	0	0	0	0	1	0			
			ERRNOT	19	1	0	0	0					1	0	0	0	0	0	0	0	1	0	
	DATA		20	0	0	1	0	0					0	1	0	0	0	1	0	0	1	0	
MOVE R2942	CHAIN (3)	SAVE	21	*See lines 12-14							1	*See lines 12-14				1	0	0	0				
		SAVE	22							1						1	0	0	0				
			23	1	0	0	0	1				1				1	0	0	0				
		SAVE	ERRORS	24	0	1	0	0	*See lines 8-9				0	1	0	0	1	0	0	0			
			ERRNOT	25	1	0	0	0					1	0	0	0	1	0	0	0	0	0	0
			DATA	26	0	0	1	0					0	0	1	0	1	0	0	0	0	0	0

*Because of the many options available with some instructions, this chart has been abbreviated to save space; therefore, some lines do not completely specify the instruction to be executed. For example, consider the instruction MOVE REGISTER AND SAVE DATA. Entering the table above first on line 9, you are referred to lines 12-14 for the SAVE options. Since you wish to save DATA, you would select line 14 to combine with line 9 for the complete instruction.

SECTOR CARD (n + 1) (CHAINS n + 1)				CONNECTIONS MADE (1) OR BROKEN (0)																
DESTINATION →		SOURCE →		EBUS			REGISTER				COMPUTER			SECTOR						
INSTRUCTION EXECUTED ↓				ERRNOT	ERROR	DATA	REGISTER	REGISTER	COMPUTER	EBUS OUT	EBUS IN	ERROR	CARD I. D.	REGISTER	FORCE	EXTERNAL	COMPUTER	PRAM	REGISTER	
LOAD { CORE DISK }			1																	
	CHAIN (3)		2																	
MOVE { CORE DISK }			3																	
		SAVE	4																	
	CHAIN (3)	SAVE	5	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
MOVE REGISTER	CHAIN (3)	SAVE	6	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	7																	
	CHAIN (3)		8																	
			9																	
	SAVE	ERRORS		10																
		ERRNOT		11																
		DATA		12																
MOVE PRAM	CHAIN (3)	SAVE	13	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	14																	
			15																	
	SAVE	ERRORS		16																
		ERRNOT		17																
		DATA		18																
MOVE R2942	CHAIN (3)	SAVE	19	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	20																	
			21																	
	SAVE	ERRORS		22																
		ERRNOT		23																
		DATA		24																

SECTOR CARD (n + 2) (CHAINS n + 2)				CONNECTIONS MADE (1) OR BROKEN (0)																
DESTINATION →		SOURCE →		EBUS			REGISTER			COMPUTER			SECTOR							
INSTRUCTION EXECUTED				ERRNOT	ERROR	DATA	REGISTER	REGISTER	COMPUTER	EBUS OUT	EBUS IN	ERROR	CARD I. D.	REGISTER	FORCE	EXTERNAL	COMPUTER	PRAM	REGISTER	
LOAD { CORE DISK }			1																	
	CHAIN (3)		2																	
MOVE { CORE DISK }			3																	
		SAVE	4																	
MOVE REGISTER	CHAIN (3)	SAVE	5	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
	CHAIN (3)	SAVE	6	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	7																	
	CHAIN (3)		8																	
				9																
	SAVE	ERRORS		10																
ERRNOT			11																	
DATA			12																	
MOVE PRAM	CHAIN (3)	SAVE	13	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	14																	
			15																	
	SAVE	ERRORS		16																
		ERRNOT		17																
		DATA		18																
MOVE R2942	CHAIN (3)	SAVE	19	0	0	0	0	1	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	20																	
			21																	
	SAVE	ERRORS		22																
		ERRNOT		23																
		DATA		24																

SECTOR CARD (n + 3) (CHAINS n + 3)				CONNECTIONS MADE (1) OR BROKEN (0)																
DESTINATION →			EBUS				REGISTER				COMPUTER				SECTOR					
SOURCE →			ERRNOT	ERROR	DATA	REGISTER	REGISTER	COMPUTER	EBUS OUT	EBUS IN	ERROR	CARD I. D.	REGISTER	FORCE	EXTERNAL	COMPUTER	PRAM	REGISTER		
INSTRUCTION EXECUTED ↓																				
LOAD { CORE DISK }			1																	
	CHAIN (3)		2																	
MOVE { CORE DISK }			3																	
		SAVE	4																	
	CHAIN (3)	SAVE	5	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
MOVE REGISTER	CHAIN (3)	SAVE	6	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	7																	
	CHAIN (3)		8																	
			9																	
	SAVE	ERRORS	10																	
		ERRNOT	11																	
		DATA	12																	
MOVE PRAM	CHAIN (3)	SAVE	13	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	14																	
			15																	
	SAVE	ERRORS	16																	
		ERRNOT	17																	
		DATA	18																	
MOVE R2942	CHAIN (3)	SAVE	19	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	20																	
			21																	
	SAVE	ERRORS	22																	
		ERRNOT	23																	
		DATA	24																	

SECTOR CARD (n - 1) (CHAINS n - 1)				CONNECTIONS MADE (1) OR BROKEN (0)																
DESTINATION →			1	EBUS			REGISTER			COMPUTER			SECTOR							
SOURCE →				ERRNOT	ERROR	DATA	REGISTER	REGISTER	COMPUTER	EBUS OUT	EBUS IN	ERROR	CARD I. D.	REGISTER	FORCE	EXTERNAL	COMPUTER	PRAM	REGISTER	
INSTRUCTION EXECUTED																				
LOAD { CORE DISK }			1																	
	CHAIN (3)		2	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
MOVE { CORE DISK }			3																	
		SAVE	4																	
	CHAIN (3)	SAVE	5																	
MOVE REGISTER	CHAIN (3)	SAVE	6	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	7																	
	CHAIN (3)		8	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
			9																	
	SAVE	ERRORS		10																
		ERRNOT		11																
		DATA		12																
MOVE PRAM	CHAIN (3)	SAVE	13	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	14																	
			15																	
	SAVE	ERRORS		16																
		ERRNOT		17																
		DATA		18																
MOVE R2942	CHAIN (3)	SAVE	19	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	
		SAVE	20																	
			21																	
	SAVE	ERRORS		22																
		ERRNOT		23																
		DATA		24																

SECTOR CARD (n - 2) (CHAINS n - 2)				CONNECTIONS MADE (1) OR BROKEN (0)															
DESTINATION			EBUS		REGISTER				COMPUTER				SECTOR						
SOURCE			ERRNOT	ERROR	DATA	REGISTER	REGISTER	COMPUTER	EBUS OUT	EBUS IN	ERROR	CARD I. D.	REGISTER	FORCE	EXTERNAL	COMPUTER	PRAM	REGISTER	
INSTRUCTION EXECUTED																			
LOAD { CORE DISK }			1																
	CHAIN (3)		2	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0
MOVE { CORE DISK }			3																
		SAVE	4																
	CHAIN (3)	SAVE	5																
MOVE REGISTER	CHAIN (3)	SAVE	6	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0
		SAVE	7																
	CHAIN (3)		8	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0
			9																
	SAVE	ERRORS	10																
		ERRNOT	11																
DATA		12																	
MOVE PRAM	CHAIN (3)	SAVE	13	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0
		SAVE	14																
			15																
	SAVE	ERRORS	16																
		ERRNOT	17																
		DATA	18																
MOVE R2942	CHAIN (3)	SAVE	19	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0
		SAVE	20																
			21																
	SAVE	ERRORS	22																
		ERRNOT	23																
		DATA	24																

Notes

- Figure 4-3 flowcharts the MOVE REGISTER statement.
- The **MOVE** and **LOAD** statements recirculate the shift registers (see Appendix A) and connect the mode control bits (SRI, EOB, and SIS), as indicated in the table below. Note that the MOVE CORE and MOVE DISK statements always recirculate the shift registers.

Statement	Recirculates		Connects	
	Yes	No	All	None
LOAD { CORE DISK } ...	X		X	
LOAD { CORE DISK } ... AND CHAIN		X		X
LOAD { CORE DISK } ... AND NO CONNECTS	X			X
LOAD { CORE DISK } ... AND CHAIN AND NO CONNECTS		X		X
MOVE REGISTER ...	X		X	
MOVE REGISTER ... AND CHAIN		X	X	
MOVE REGISTER ... AND NO CONNECTS	X			X
MOVE REGISTER ... AND CHAIN AND NO CONNECTS		X		X
MOVE { CORE DISK } ...	X		X	
MOVE { CORE DISK } ... AND CHAIN	X		X	
MOVE { CORE DISK } ... AND NO CONNECTS	X			X
MOVE { CORE DISK } ... AND CHAIN AND NO CONNECTS	X			X



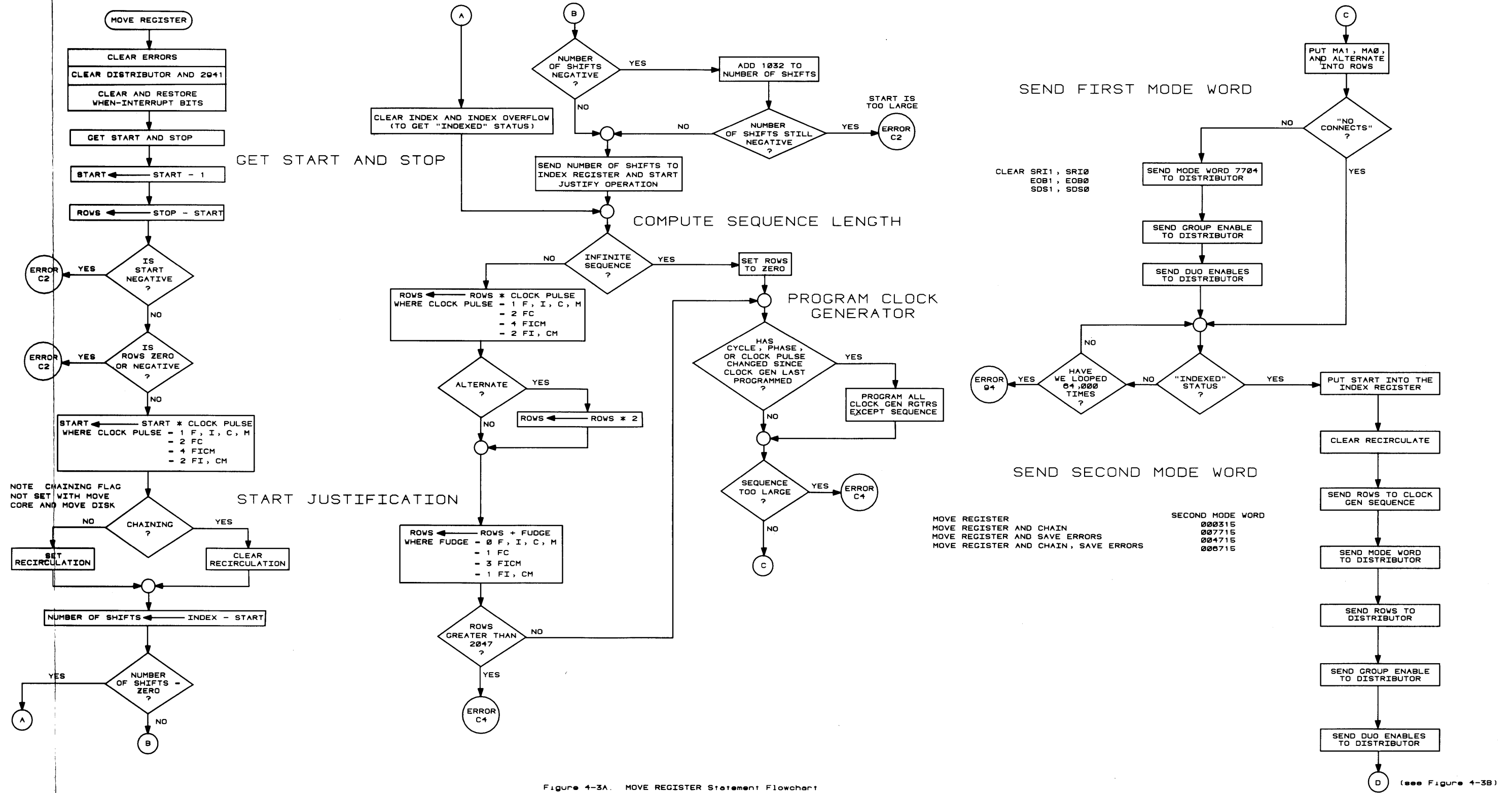
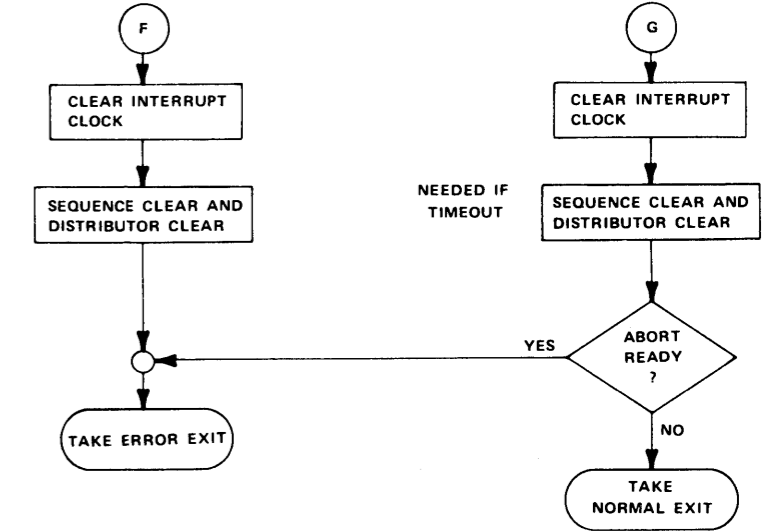
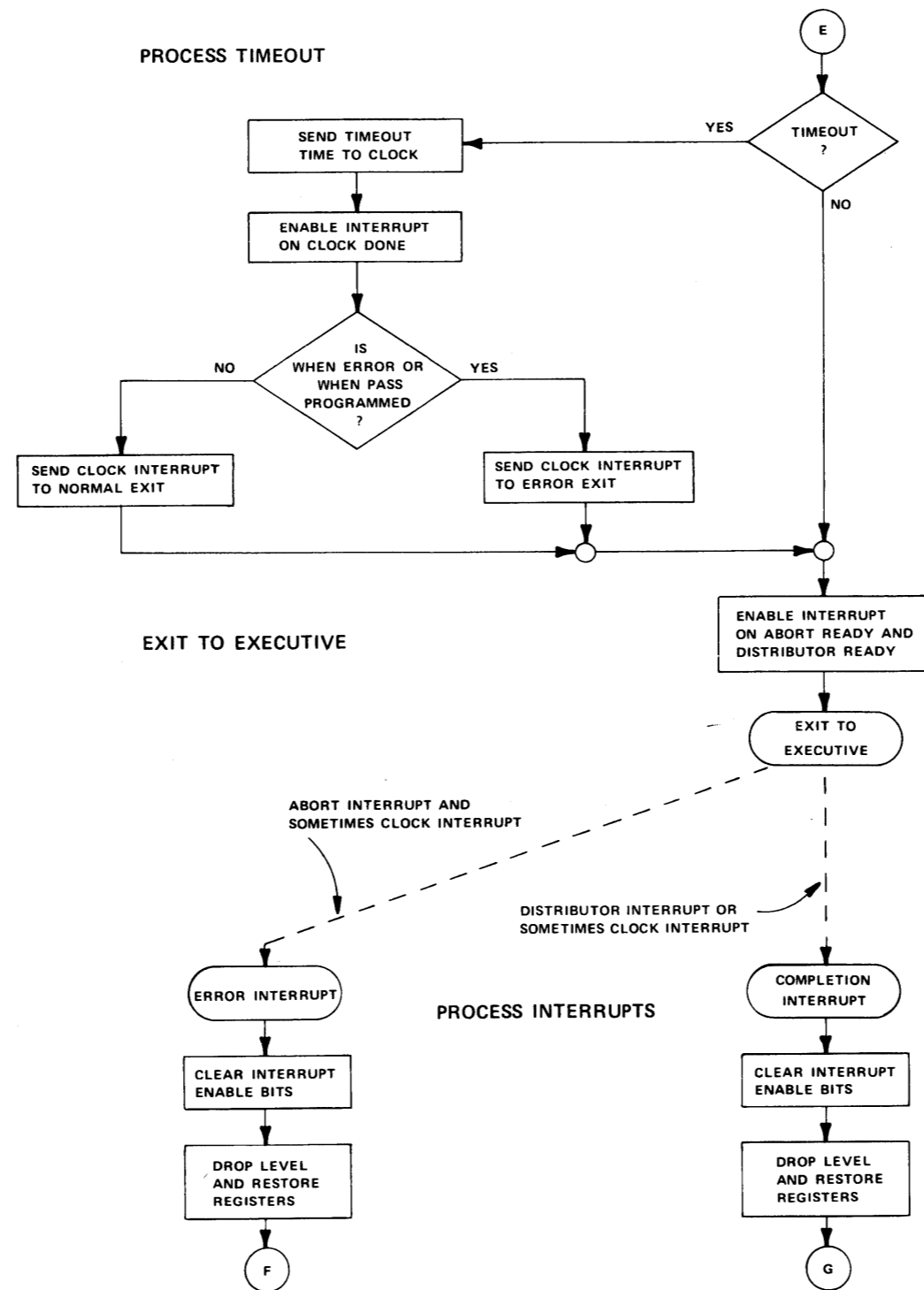
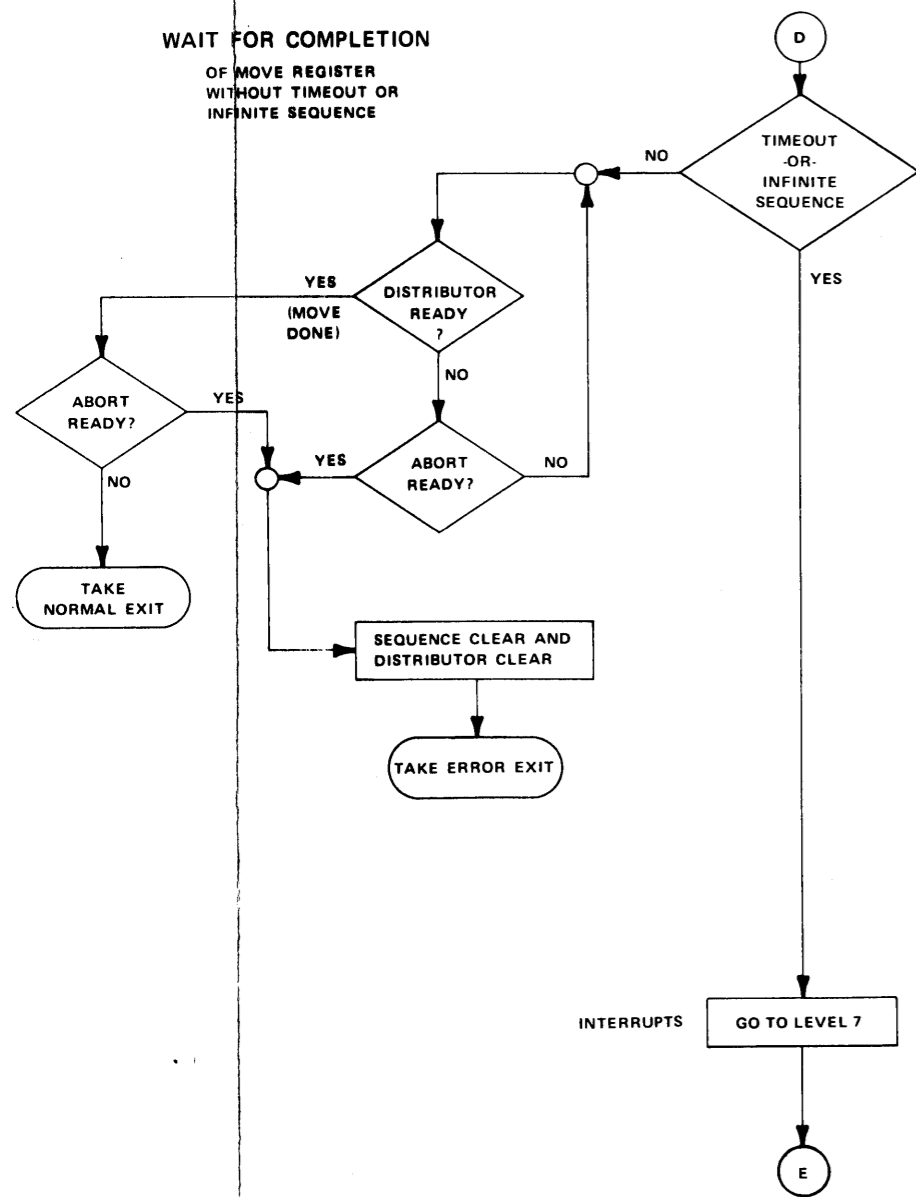


Figure 4-3A. MOVE REGISTER Statement Flowchart

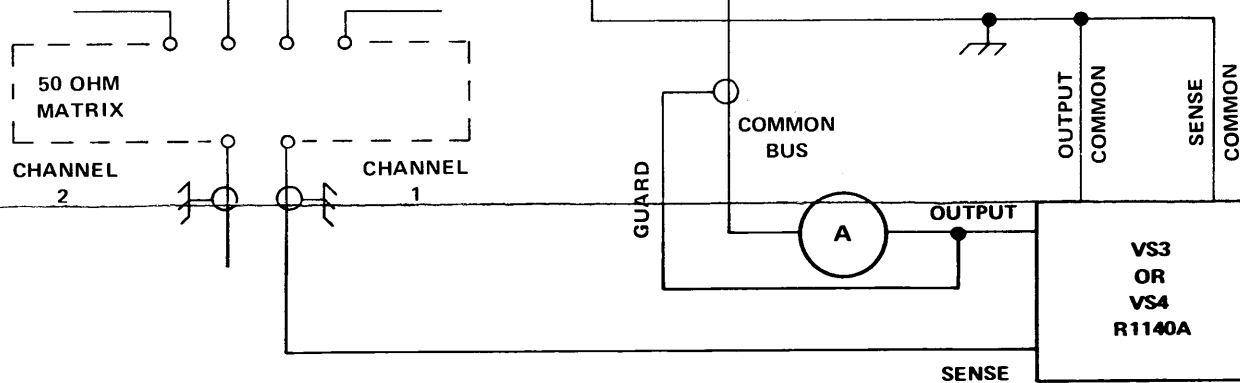
REV A SEP 1980



3325-32

Figure 4-3B. MOVE REGISTER Statement Flowchart (continued)

CONNECTOR BOARD



3325-33

Reed Switch Names

SETUP Statements	DSHH	DSHL	DS	CSHH	CSHL	I1	I2	IM	ICBA	ICBB	OM	BM	CLM	CHM	OCBA	OCBB	O1	O2	L1	L2	Matrix 1 2
FOR I STIMULATE ON O or IO											1						1				1
FOR V STIMULATE ON O or IO											1						1				1
FOR CURRENT FROM (VS3) ON I (1 nA - 10 mA)									1	1											1
FOR CURRENT FROM (VS3) ON I (10 mA - 450 mA)						1		1	1	1											1
FOR CURRENT FROM (VS3) ON O (10 mA - 450 mA)											1				1	1	1				1
FOR CURRENT FROM (VS3) ON IO (1 nA - 10 mA)									1	1											1
FOR CURRENT FROM (VS3) ON IO (10 mA - 450 mA)									1	1	1						1				1
FOR CURRENT FROM DRIVER ON I							1	1	1	1											1
FOR CURRENT FROM DRIVER ON O							1	1							1	1					1
FOR CURRENT FROM DRIVER ON IO							1	1	1	1											1
FOR VOLTAGE ON I TO I						1		1	1	1											1
FOR VOLTAGE ON I TO O									1	1	1						1				1
FOR VOLTAGE ON I TO IO						1		1	1	1											1
FOR VOLTAGE ON O TO I						1		1							1	1					1
FOR VOLTAGE ON O TO O											1				1	1	1				1
FOR VOLTAGE ON O TO IO						1		1							1	1					1
FOR VOLTAGE ON IO TO I						1		1	1	1											1
FOR VOLTAGE ON IO TO O									1	1	1					1					1
FOR VOLTAGE ON IO TO IO						1		1	1	1											1
TO FORCE CURRENT ON I						1		1													1
TO FORCE VOLTAGE ON I						1		1													1
TO FORCE CURRENT ON O											1						1				1
TO FORCE VOLTAGE ON O																					1
TO FORCE CURRENT ON IO											1							1			1
TO FORCE VOLTAGE ON IO											1							1			1
TO MEASURE CURRENT FROM (VS3) ON I (1 nA - 10 mA) (10 mA - 450 mA)						1		1	1	1											1 1
TO MEASURE CURRENT FROM (VS3) ON O (10 mA - 450 mA)											1				1	1	1				1
TO MEASURE CURRENT FROM (VS3) ON IO (1 nA - 10 mA) (10 mA - 450 mA)									1	1								1			1 1
TO MEASURE CURRENT FROM DRIVER ON I							1	1	1	1											1
TO MEASURE CURRENT FROM DRIVER ON O							1	1													1
TO MEASURE CURRENT FROM DRIVER TO IO							1	1	1	1											1
TO MEASURE VOLTAGE ON I TO I						1		1	1	1											1
TO MEASURE VOLTAGE ON I TO O									1	1	1						1				1
TO MEASURE VOLTAGE ON I TO IO						1		1	1	1											1
TO MEASURE VOLTAGE ON O TO I						1		1	1	1											1
TO MEASURE VOLTAGE ON O TO O											1				1	1	1				1
TO MEASURE VOLTAGE ON O TO IO						1		1							1	1					1
TO MEASURE VOLTAGE ON IO TO I						1		1	1	1											1
TO MEASURE VOLTAGE ON IO TO O									1	1					1	1					1
TO MEASURE VOLTAGE ON IO TO IO						1		1	1	1											1
TO MEASURE VOLTAGE ON I									1	1											
TO MEASURE VOLTAGE ON O															1	1					
TO MEASURE VOLTAGE ON IO									1	1											
TO MEASURE CURRENT FROM (VS3) ON O (1 nA - 10 mA)															1	1					1

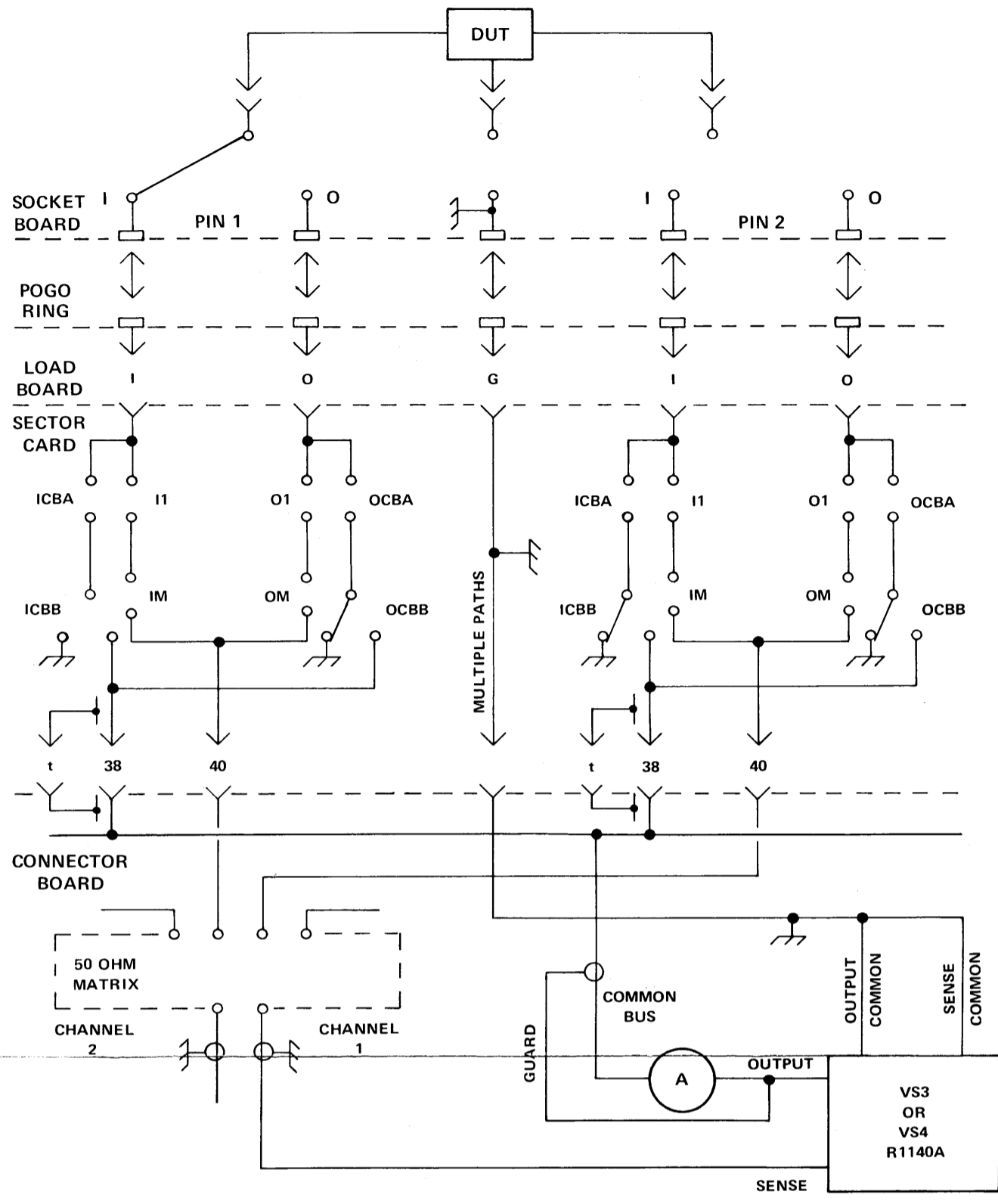


Table 6-1. Effect of SETUP Statements on the Reed Switches

3325-33

Reed Switch Names

SETUP Statements	DSHH	DSHL	DS	CSHH	CSHL	I1	I2	IM	ICBA	ICBB	OM	BM	CLM	CHM	OCBA	OCBB	O1	O2	L1	L2	Matrix 1 2
FOR I STIMULATE ON O or IO											1						1				1
FOR V STIMULATE ON O or IO											1						1				1
FOR CURRENT FROM (VS3 VS4) ON I (1 nA - 10 mA)									1	1											1
FOR CURRENT FROM (VS3 VS4) ON I (10 mA - 450 mA)						1		1	1	1											1
FOR CURRENT FROM (VS3 VS4) ON O (10 mA - 450 mA)											1				1	1	1				1
FOR CURRENT FROM (VS3 VS4) ON IO (1 nA - 10 mA)									1	1											1
FOR CURRENT FROM (VS3 VS4) ON IO (10 mA - 450 mA)									1	1	1						1				1
FOR CURRENT FROM DRIVER ON I							1	1	1	1											1
FOR CURRENT FROM DRIVER ON O							1	1							1	1					1
FOR CURRENT FROM DRIVER ON IO							1	1	1	1											1
FOR VOLTAGE ON I TO I						1		1	1	1											1
FOR VOLTAGE ON I TO O									1	1	1						1				1
FOR VOLTAGE ON I TO IO						1		1	1	1											1
FOR VOLTAGE ON O TO I						1		1							1	1					1
FOR VOLTAGE ON O TO O											1				1	1	1				1
FOR VOLTAGE ON O TO IO						1		1							1	1					1
FOR VOLTAGE ON IO TO I						1		1	1	1											1
FOR VOLTAGE ON IO TO O									1	1	1						1				1
FOR VOLTAGE ON IO TO IO						1		1	1	1											1
TO FORCE CURRENT ON I						1		1													1
TO FORCE VOLTAGE ON I						1		1													1
TO FORCE CURRENT ON O											1						1				1

MEASURING VOLTAGE

The **SETUP TO MEASURE VOLTAGE** statement performs the following operations:

1. Disables Scope 1.
2. Connects the first pin to the common bus.
3. Connects the second pin to Channel 1 of the 50 Ω matrix. Disconnects the driver on the second pin if it is an I or IO pin.
4. Connects the matrix to the second pin's sector-card electronics.
5. Cause a 1 ms delay.

The **UNSET** statement complements the above steps. See Figure 6-1.

The formats of the **SETUP** and **UNSET TO MEASURE VOLTAGE** statements are:

$$\text{SETUP } \left\{ \begin{array}{l} \text{TO MEASURE VOLTAGE} \\ \text{FOR VOLTAGE} \end{array} \right\} \text{ ON pins [TO pin1] } \left[\begin{array}{l} \left\{ \begin{array}{l} \text{AT} \\ , \end{array} \right\}^* \\ \text{range} \end{array} \right]$$
$$\text{UNSET } \left\{ \begin{array}{l} \text{TO MEASURE VOLTAGE} \\ \text{FOR VOLTAGE} \end{array} \right\} \text{ ON pins [TO pin1]}$$

Elements

If **pins** is an IO pin, it is connected via the I side of the IO sector card path. It is also connected to the common bus. If **pins** is a group of pin names, all pins are connected to the common bus and are shorted together.

If **pin1** is an IO pin, it too is connected via the I side of the IO sector card path. It is also connected to the 50 Ω matrix. If you omit the second pin, the system makes the voltage measurement relative to the DC Subsystem ground.

range is any legal numeric expression. Omitting **range** causes the system to use the 10-volt range as a default.

*A comma is not legal for single-ended voltage measurements.

Disconnections Required

- Disconnect the scope or auxiliary ports from the 50 Ω matrix before trying to connect the DUT to it.
- Disconnect the driver from the pin to which you are connecting the DC Subsystem.

Effects on the Common Bus

When you are using the common bus for SETUP TO MEASURE VOLTAGE, you should avoid using the INITIALIZE and STOP statements. In addition, you should avoid pressing the STOP button on the Test Station Control Unit. Failure to observe these precautions may cause the DUT pins to be grounded for a short time and result in DUT failure or reed switch damage on the sector card. You should always use the UNSET statement before performing any of the above operations.

Settling Time

On the 100 mV range, the settling time for measuring voltage is 4 ms.

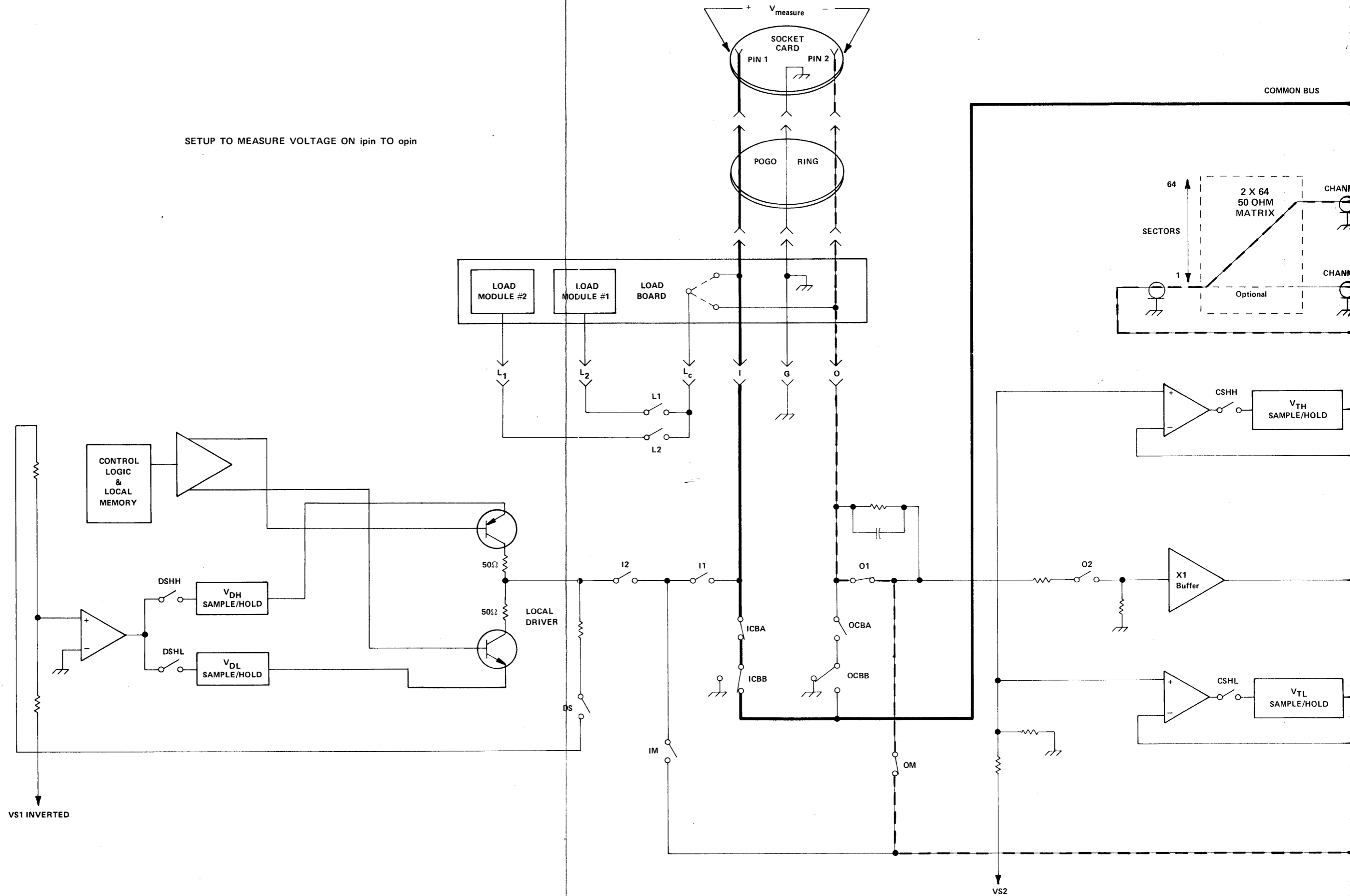
Effects on the DC Subsystem

This statement does not disable the DC Subsystem and does not open a connection made previously with SETUP FOR ISTIMULATE or VSTIMULATE. However, a SETUP FOR CURRENT or VOLTAGE statement should not be in operation.

Effects on the Comparator

The comparator, if connected, is forced to the 5-volt range if the second pin in a SETUP TO MEASURE VOLTAGE statement is an O pin. The UNSET TO MEASURE VOLTAGE statement forces the comparator, if connected, to the 30-volt range.

SETUP TO MEASURE VOLTAGE ON ipin TO opin



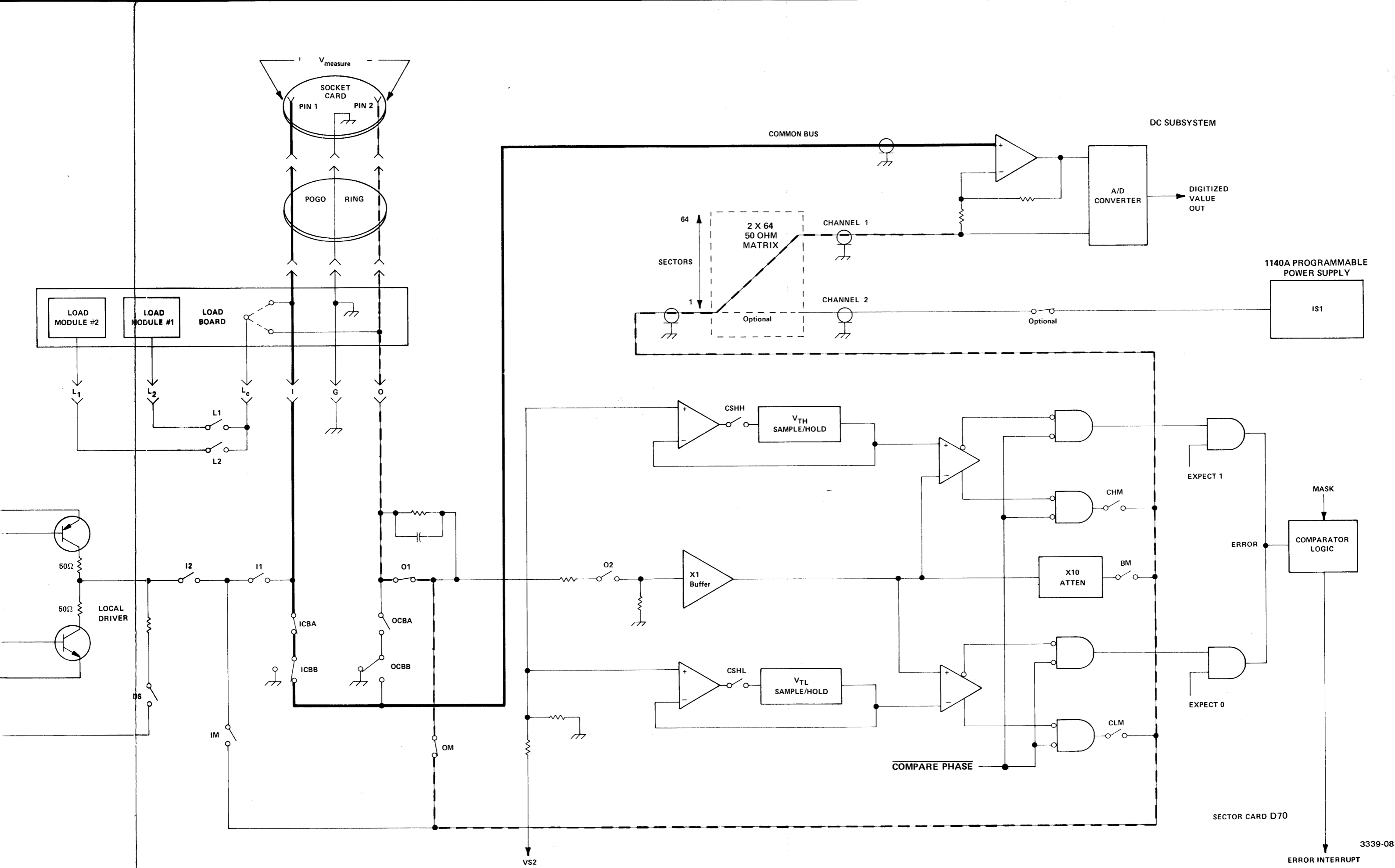


Figure 6-1. INTERCONNECTIONS FOR MEASURING VOLTAGE

MEASURING CURRENT

The **SETUP TO MEASURE CURRENT . . . FROM DRIVER** statement performs the following operations:

1. Disables Scope 1.
2. Causes a 3 ms delay if the DC Subsystem is programmed to the 100 mA range. Otherwise, it causes a 1 ms delay.
3. Connects the driver to the 50 Ω matrix.
4. Connects channel 1 of the matrix to the pin's sector-card electronics.
5. Causes a 1 ms delay.
6. Connects the pin to the common bus.
7. Causes a 1 ms delay.

The **UNSET** statement complements the above steps. See Figure 6-2.

The formats of the **SETUP** and **UNSET TO MEASURE CURRENT . . . FROM DRIVER** statements are:

SETUP { TO MEASURE CURRENT
FOR CURRENT } ON pin1 FROM DRIVER { AT } range

UNSET { TO MEASURE CURRENT
FOR CURRENT } ON pin1 FROM DRIVER

Elements

If **pin1** is an IO pin, it is connected via the I side of the IO sector card path. If **pin1** is an O pin, the driver is connected to the output pin. (This is the only way in which a driver may be connected to an O pin without external wiring.)

DRIVER is the high and low driver sample and hold circuitry which forces data to the DUT.

range is any legal numeric expression.

Measuring Current on More Than One Pin

The `SETUP TO MEASURE CURRENT . . . FROM DRIVER` statement uses the common bus. You can measure current on more than one pin by connecting additional pins to the common bus.

Effects on the Driver

The `UNSET TO MEASURE CURRENT . . . FROM DRIVER` statement leaves the driver disconnected.

Effects on the Common Bus

When you are using the common bus for `SETUP TO MEASURE CURRENT`, you should avoid using the `INITIALIZE` and `STOP` statements. In addition, you should avoid pressing the `STOP` button on the Test Station Control Unit. Failure to observe these precautions may cause the DUT pins to be grounded for a short time and result in DUT failure or reed switch damage on the sector card. You should always use the `UNSET` statement before performing any of the above operations.

Disconnections Required

When you are performing a current measurement, you must make sure that the driver is disconnected from the pin to which you are connecting the DC Subsystem.

Maximum Range for Measuring Current from the Driver

The maximum range for measuring current from the driver is 100 mA.

Kelvin Sensing

When the DC Subsystem is in the 10 mA range or above, remote Kelvin sensing of the forcing voltage can occur at the DUT. The sense path is via the O port of the DUT if the pin is an IO pin. Kelvin sensing can only occur on the sector card when the DUT pin is an I or O pin. Kelvin sensing at the DUT produces a more accurate measurement.

RETURNING THE CURRENT STATE OF THE TIMEOUT SYSTEM FLAG

Whenever a timeout occurs as the result of the **TIMEOUT** element in a **MOVE REGISTER** statement, a system flag is set to one. The **TIMOUT** function returns the current state of this system flag. The system clears (sets to zero) the flag at the beginning of a **MOVE REGISTER** or **LOAD** statement. (No other way of setting the flag to zero exists.)

The **TIMOUT** function call is:

TIMOUT

The function declaration is:

FUNCTION TIMOUT(0):TIMOUT

DETERMINING THE CLOCK GENERATOR MODE

The **BURST** statement determines whether the programmable clock generator is in a free-running or programmed mode. In free-running mode, the number of clock cycles is infinite. In programmed mode, the number of clock cycles is finite. The BURST free-running mode is different from the MOVE REGISTER free-running mode.

The BURST statement format is:

BURST { ON
OFF }

- **BURST ON** turns off the programmed mode and puts the clock generator in a free-running mode. All seven clock phases run continuously at their programmed rates, start times, and width; only the number of clock cycles is unprogrammed.
- **BURST OFF** turns off the free-running mode and puts the clock generator in the programmed mode.

The following statements are illegal during a burst:

MOVE
PHASE
DATAPHASE
CYCLE
HICOMPARE . . . FROM X FOR Y
LOCOMPARE . . . FROM X FOR Y
WHEN ERROR
WHEN PASS
WHEN OVERFLOW
WHEN ERROR OR OVERFLOW
CLEAR ERROR

The following statements may cause undesirable results if used prior to a BURST ON or during a burst operation:

{ FORCE
COMPARE
FORCE,COMPARE
INHIBIT
MASK } WITH PATTERN

READING THE FORCE FLIP-FLOPS

The **DRIVE** function reads the force flip-flop on the specified sector card. The value returned by **DRIVE** is:

- one if the force flip-flop is set and
- zero if the force flip-flop is clear.

The **DRIVE** function call is:

DRIVE(pin1)

The function declaration is:

FUNCTION DRIVE(T1):SECDAT

NOTE

The result of DRIVE does not reflect the elements INVERT or RZ in the FORCE statement.



SECTION 5: ERROR CONTROL

The statements, variables, and functions described in this section pertain to functional-testing error control. This section also discusses the stop-on-error circuits.

Error Detection

Errors are detected only during comparator gates. Gates are programmable over a wide range of times for both gate start-time and width. Errors can therefore be detected over a time period that approaches two clock cycles. For example, the earliest that the system can detect an error is for a gate whose start-time was programmed at t_0 . The latest the system can detect an error is near the trailing edge of a gate whose start-time was as late as possible in the cycle and whose width was as large as possible. This places the trailing edge late in the next cycle.

Statements, Variables, and Functions Described in this Section

- **ERROR** Indicates the status of functional testing

- **INDEX** Points to the shift register pattern rows presently available at the DUT

- **INDEXP** Renumbers the rows in the shift registers

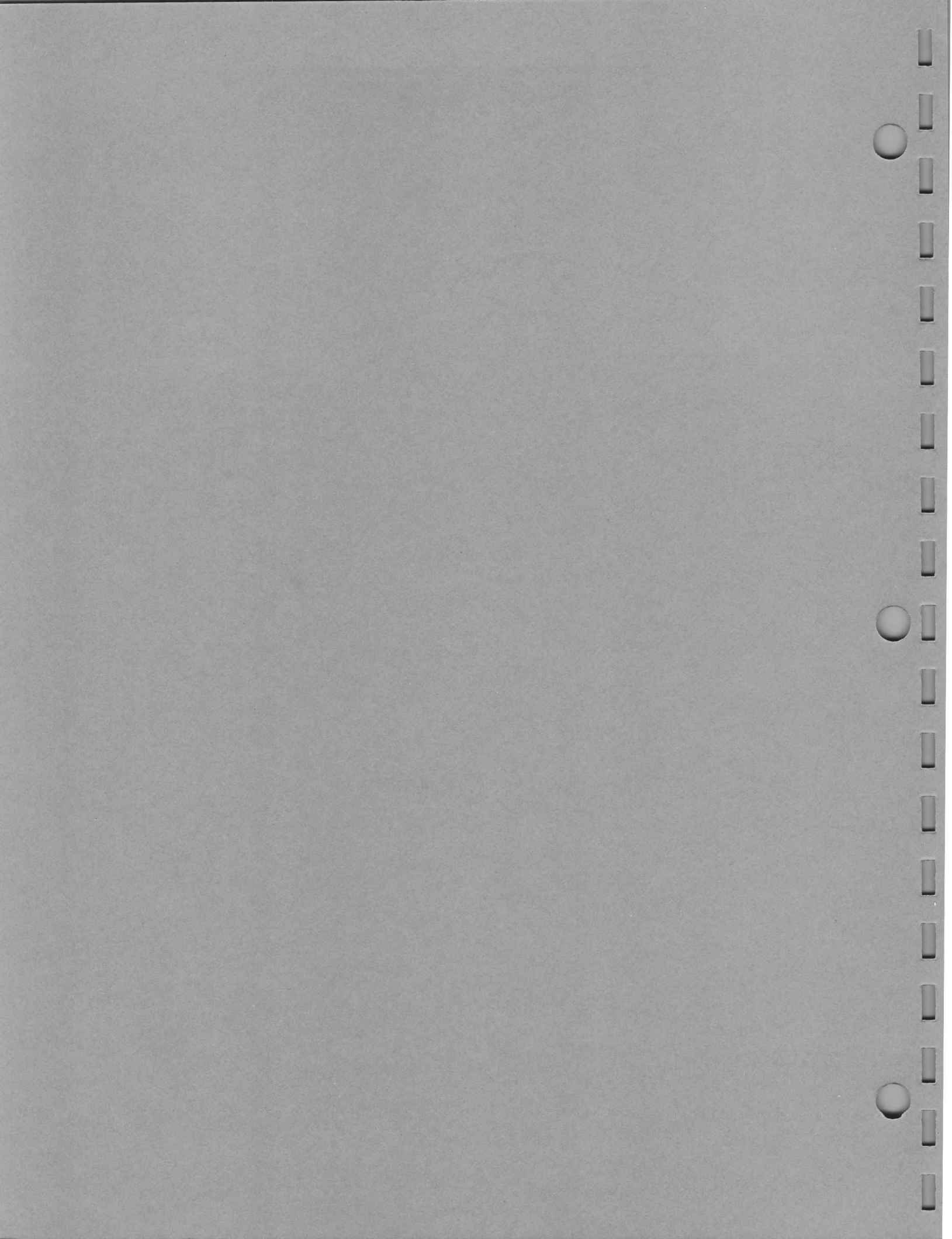
- **WHEN ERROR** Tests for an error condition and clears the error flip-flops

- **WHEN PASS** Tests for a pass condition and clears the error flip-flops

- **CLEAR ERROR** Disables the WHEN ERROR and WHEN PASS statements and clears the error flip-flop

- **PINERR** Reads the pin electronics card error flip-flops

- **PINPAS** Reads the pin electronics card error flip-flops



INDICATING FUNCTIONAL TESTING STATUS

The **ERROR** variable* indicates the status of functional testing: that is, whether a functional error has occurred or has not occurred. Any discrepancy between the expected output and the actual output of the DUT constitutes an error.

Once an error occurs, the system sets the value of **ERROR** and the error flip-flops. **ERROR** remains set until a **STOP**, **INITIALIZE**, **CLEAR ERROR**, **MOVE**, **LOGREG**, **READREG**, or **SREAD** statement is encountered.

At any given time in the program, **ERROR** has a value of one or zero. If an error occurs on any of the 64 sector comparators, the value of **ERROR** is one (or true). If an error does not occur, the value of **ERROR** is zero (or false).

ERROR may be used in any legal expression, or as the element in a logical **IF** statement.* The **ERROR** variable format is:

ERROR

If you wish to store the value of **ERROR**, use **ERROR** in an expression:

variable = ERROR

If you wish to transfer program control when an error occurs, use **ERROR** as the element in a logical **IF** statement:

IF (ERROR) linenumber [,linenumber]

ERROR indicates functional disparities only. The **COMPARE** and **FORCE,COMPARE** statements determine whether an error has or has not occurred. The **HICOMPARE** and **LOCOMPARE** statements specify the time window in which to expect the data.

*See the *Test Language, Part One* for information on reserved system variables and the **IF** statement and its elements.

Example:

>3.3 X = ERROR

If an error occurs, X = 1. If an error does not occur, X = 0.

>10.4 IF(ERROR) 12.0,10.5

If an error occurs (ERROR is true), program control transfers to line 12.0000. If an error does not occur (ERROR is false), program control transfers to line 10.5000.

POINTING TO THE PATTERN ROWS PRESENTLY AVAILABLE AT THE DUT

The **INDEX** variable* counts each bit of data as the pattern is clocked out of the sector card shift registers. **INDEX** (the index counter) points to the shift register pattern rows presently available at the DUT.

INDEX may be used in any legal expression. The **INDEX** variable format is:

INDEX

If you wish to store the value of **INDEX**, use **INDEX** in an expression:

variable = INDEX

If an error occurs during functional testing, use **INDEX** to determine on which pattern bit the error occurred.

The maximum index count is 262656.

Example:

```
.  
. .  
. .  
>2.3 WHEN ERROR 5.0  
>2.4 MOVE REGISTER (4) TO ALL WITH F  
. .  
. .  
>5.0 X = INDEX  
. .  
. .  
>5.4 RETURN
```

When an error occurs, **WHEN ERROR** transfers program control to line 5.0000. **INDEX** determines on which pattern bit the error occurred.

*See the *Test Language, Part One* for information on reserved system variables.

RENUMBERING THE SHIFT REGISTER ROWS

The **INDEXP** statement renumbers the rows in the shift registers. The renumbered data becomes available to the DUT at the next clock cycle.

The **INDEXP** statement format is:

INDEXP expression

expression may be any legal expression which returns a positive or negative integer value. The value of **expression** resets the position of **INDEX** to ahead or behind its previous position. New pattern row 1 is equal to present pattern row $[1 - (\mathbf{expression})]$.

Example:

>5.5 **INDEXP -49**

Renumbers all of the pattern rows in the shift registers such that the present row 50 becomes row 1 of the pattern. $[1 - (-49)] = 50$.

TESTING FOR AN ERROR CONDITION

The **WHEN ERROR** statement clears the error flip-flops when it is executed and tests for an error condition during functional testing. If an error occurs, **WHEN ERROR** discontinues testing and transfers program control to the line number specified. An error consists of a discrepancy between the expected DUT output and the actual output.

A **WHEN ERROR** statement must precede a **MOVE** statement. The **CLEAR ERROR** statement disables the **WHEN ERROR** statement.

The **WHEN ERROR** statement format is:

WHEN ERROR *linenumber*

linenumber is the line number of the statement to which program control transfers if an error occurs. **linenumber** usually starts a routine which displays the **FAIL** indicator and contains a **CLEAR ERROR** statement.

WHEN ERROR operates in a manner similar to a subroutine call. Thus, you must include a **RETURN** statement in the error routine. When program control encounters the **RETURN** statement, it transfers to the line immediately following the **MOVE** statement where the error occurred.

An error can be detected only during a high- or low-comparator gate. (See the **HICOMPARE** and **LOCOMPARE** statements in Section 3.)

If **WHEN ERROR** is programmed and the Pram is off:

- At 100 ns after the first error is detected, the data to the sector drivers will not change.
- At 100 ns after the first error is detected, the sector shift registers are no longer clocked. The position of the data can be determined with the **INDEX** statement.

Example:

```
.  
. .  
. .  
>11.5 WHEN ERROR 21.0  
>11.6 MOVE . . .  
>11.7 PRINT "START NEXT FUNCTIONAL TEST",CR  
. .  
. .  
>21.0 PRINT "FAILED ON ROW",INDEX:IO,CR  
>21.1 DISPLAY,FAIL  
>21.2 CLEAR ERROR  
>21.3 RETURN  
. .  
. .
```

When an error occurs, WHEN ERROR discontinues testing and transfers program control to line 21.0000. When program control encounters the RETURN statement, it transfers to line 11.7000.

DISABLING THE WHEN ERROR STATEMENT

The **CLEAR ERROR** statement disables the **WHEN ERROR** statement and clears the error flip-flops.

The **CLEAR ERROR** statement format is:

CLEAR ERROR

Example:

<pre> . . . >9.0 WHEN ERROR 100.0 >9.1 MOVE ... >9.2 DISPLAY,PASS >9.3 CLEAR ERROR >9.4 CALL 200.0 </pre>	<p>CLEAR ERROR disables the WHEN ERROR statement and clears the error flip-flops set by the MOVE statement.</p>
---	--

READING THE ERROR FLIP-FLOPS

The **PINERR** function reads the contents of the error flip-flops on the specified sector cards. The value returned by PINERR is:

- one if any of the error flip-flops are set and
- zero if none of the error flip-flops are set.

The PINERR function call is:

PINERR(pins)

The function declaration is:

FUNCTION PINERR(T):SECDAT

The **PINPAS** function reads the contents of the error flip-flops on the specified sector cards. The value returned by PINPAS is:

- one if any of the error flip-flops are clear and
- zero if none of the error flip-flops are clear.

The PINPAS function call is:

PINPAS(pins)

The function declaration is:

FUNCTION PINPAS(T):SECDAT

STOP-ON-ERROR CIRCUITS (ADDITIONAL OPERATING PRINCIPLES)

The stop-on-error circuits are designed to assist you in identifying malfunctioning DUT pins, and in determining the stimulus conditions which caused the pins to produce errors. Ideally, the stop-on-error circuits would be able to perform these functions:

1. "Freeze" the stimulus conditions at the instant the DUT produced an error.
2. Indicate which word of the pattern was being processed when the error occurred.
3. Indicate which pin of the DUT produced the error.

Because of delays in the error circuitry, not all of the above ideal functions are always directly available. However, using the programming techniques presented below will enable you to gather most of the desired information.

At the beginning of a test, the system disables the stop-on-error circuits. The system enables the stop-on-error circuits when it encounters a WHEN ERROR statement in your program. During program execution, the system disables the stop-on-error circuits when it encounters a CLEAR ERROR statement, or when an error interrupt occurs. In the case of an error interrupt, the system does not disable the circuits until the program branches to the line number specified by the WHEN ERROR statement. After disabling the stop-on-error circuits, the system does not enable them until it again encounters the WHEN ERROR statement.

At 110 ns after the system detects an error, it "freezes" the error circuits. If the system detects the first error at the leading edge of the comparator gate, it only records the errors which occur during that comparator gate pulse. Programming a comparator gate width greater than 48 ns causes the system to truncate the gate width to 48 ns when it detects an error. If no error occurs at the leading edge of the comparator gate, but instead a DUT output pin goes to an error level during the gate time, a portion of the next gate will occur (if the next gate would normally occur in 48 ns or less) before the system can "freeze" the error circuits. This means that errors occurring during two different cycles may be recorded. (See Figure 5-1 through Figure 5-4.)

Since the stop-on-error circuits perform as described above, it is possible to formulate these two rules:

1. If a DUT output pin fails to reach the expected level during the minimum propagation delay time, the system stores only those errors which occur within the same cycle.

2. If the DUT output pins all reach the expected levels during the minimum propagation delay time, but one or more of the pins leaves its expected level while the comparator gate is still true, the system stores the errors occurring during the current gate. In addition, the system stores any errors which occur during the first portion of the next gate (if the next gate begins in less than 110 ns from the error detected during the current gate).

The system stops force data, Dataphase, and the index counter between 85 ns and 100 ns from the time it detects the first error. At that time, the system "freezes" the driver output logic level. The system truncates all phases in progress and does not generate any additional pulses. If the cycle time was 100 ns or greater, the index counter accurately reflects the cycle during which the error occurred. You can then use the INDEX variable to determine the position in the shift register of the data used to stimulate the pin in error.

If the cycle time was less than 100 ns, the test overruns the error — that is, the index counter does not reflect the actual clock cycle during which the error occurred. The index counter may overcount by as much as two counts. You can force the overcount to be constant by programming the comparator gates so that their leading edges do not occur between 85 ns and 100 ns before a t_0 . If the overcount is constant, you then can subtract it from the reserved variable INDEX to determine the position in the shift register of the data used to stimulate the pin in error.

To program a constant overcount for the index counter, use these rules:

1. For a constant index overcount of 2, program the start-time to be (a) equal to or less than the cycle time minus 17 ns, and (b) equal to or greater than two times the cycle time minus 85 ns. That is,

$$\text{START-TIME} \begin{array}{l} \leq (\text{CYCLE TIME} - 17 \text{ ns}) \\ \geq (2 * \text{CYCLE TIME} - 85 \text{ ns}) \end{array}$$

2. For a constant index overcount of 1, program the start-time to be (a) equal to or less than two times the cycle time minus 100 ns, and (b) equal to or less than the cycle time minus 17 ns, and (c) equal to or greater than the cycle time minus 85 ns, and (d) equal to or greater than zero. That is,

$$\text{START-TIME} \begin{array}{l} \leq (2 * \text{CYCLE TIME} - 100 \text{ ns}) \\ \leq (\text{CYCLE TIME} - 17 \text{ ns}) \\ \geq (\text{CYCLE TIME} - 85 \text{ ns}) \\ \geq 0 \text{ (zero)} \end{array}$$

3. For no constant index overcount, program the start-time to be (a) equal to or less than the cycle time minus 100 ns, and (b) equal to or greater than zero. That is,

$$\text{START-TIME} \begin{array}{l} \leq (\text{CYCLE TIME} - 100 \text{ ns}) \\ \geq 0 \text{ (zero)} \end{array}$$

All phases are stopped within 110 ns of detecting an error.

The beginning of a clock cycle, t_0 , is the time at which the NRZ data changes at the driver output, hence at the DUT input (the time at which the DUT "sees" the forced data). You program the DUT clock (which is one of the programmable clock phases) to begin after the maximum data setup time required by the DUT, and the comparator gates to follow the DUT clock by the specified propagation delay time of the DUT outputs. Thus, the comparator gate start-time is the sum of the DUT setup time and the DUT propagation delay time.

Cycle time is usually chosen to match the maximum frequency for the DUT.

Figure 5-6 illustrates how you can adjust either the cycle time or the gate start-time to force the index counter overrun to be constant. If you cannot test the DUT satisfactorily by using the setting required for the index counter, perhaps you can provide the input data in RZ mode at a later time than t_0 in the cycle. You then can move both the clock phase and the comparator gate to a later time in the clock cycle.

If you wish to observe the DUT statically at the time of the error, you can use the index counter and the constant index overrun to program a second pass which will stop on the error word. For example, assume that a user programmed a constant index overrun of two. The user could then use the program steps shown below to cause the test to execute a second pass and stop at an error.

Example:

```
>9.1 WHEN ERROR 10.1
>9.2 MOVE REGISTER (1,END) TO PINS
>9.3 CONTINUE
.
.
.
>10.1 MOVE REGISTER (1,(INDEX-2)) TO PINS
>10.2 CONTINUE
.
.
.
```

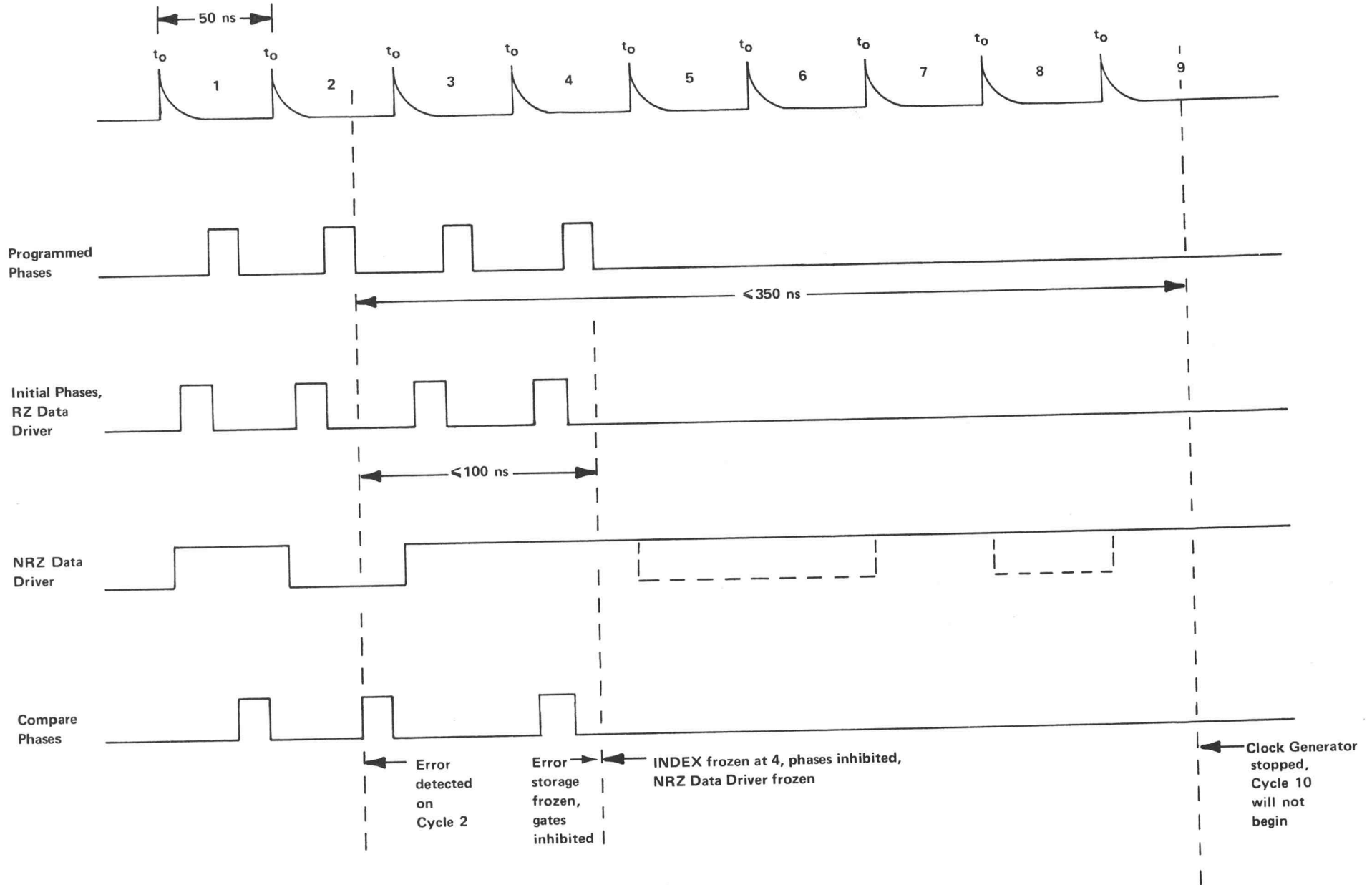
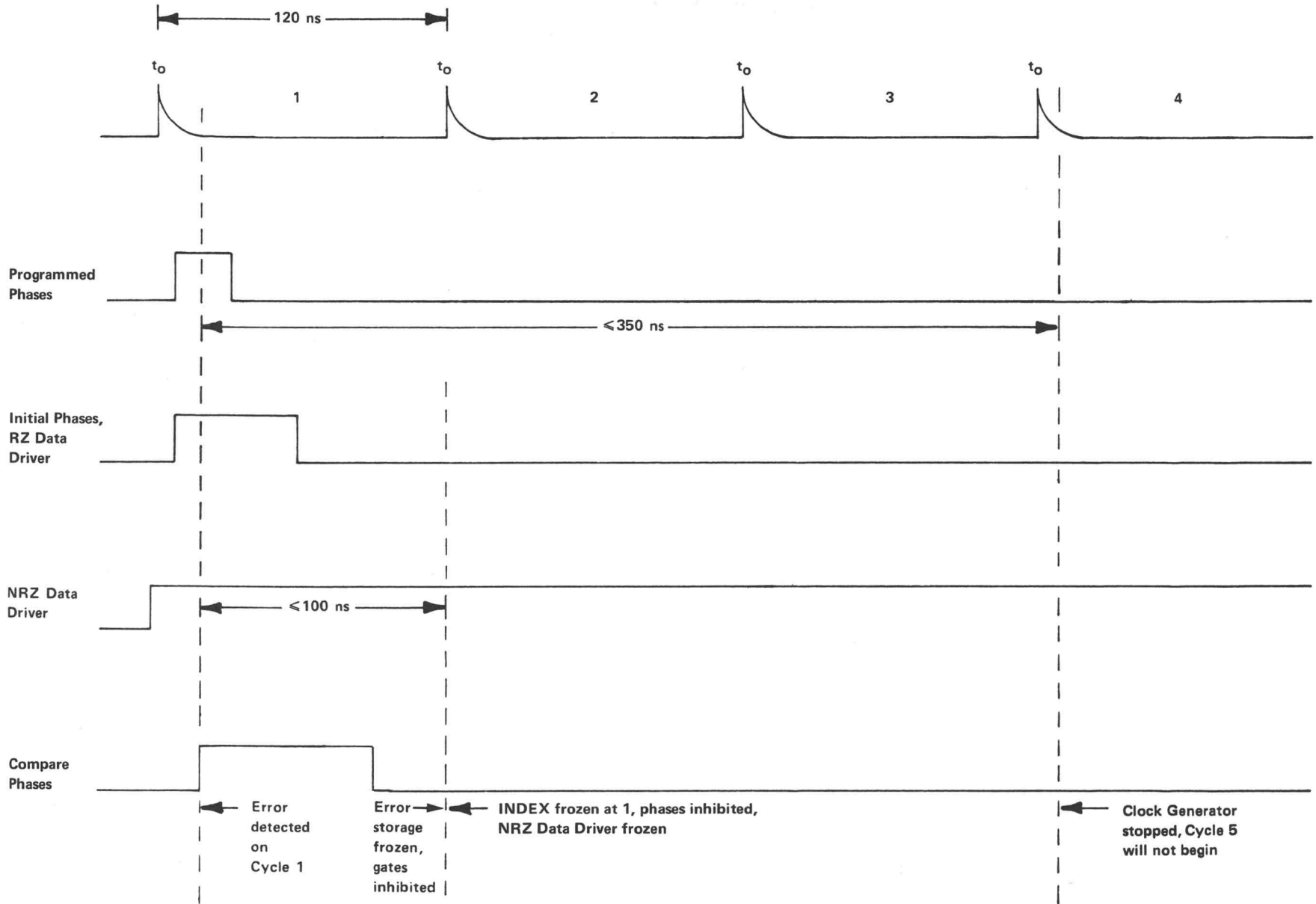
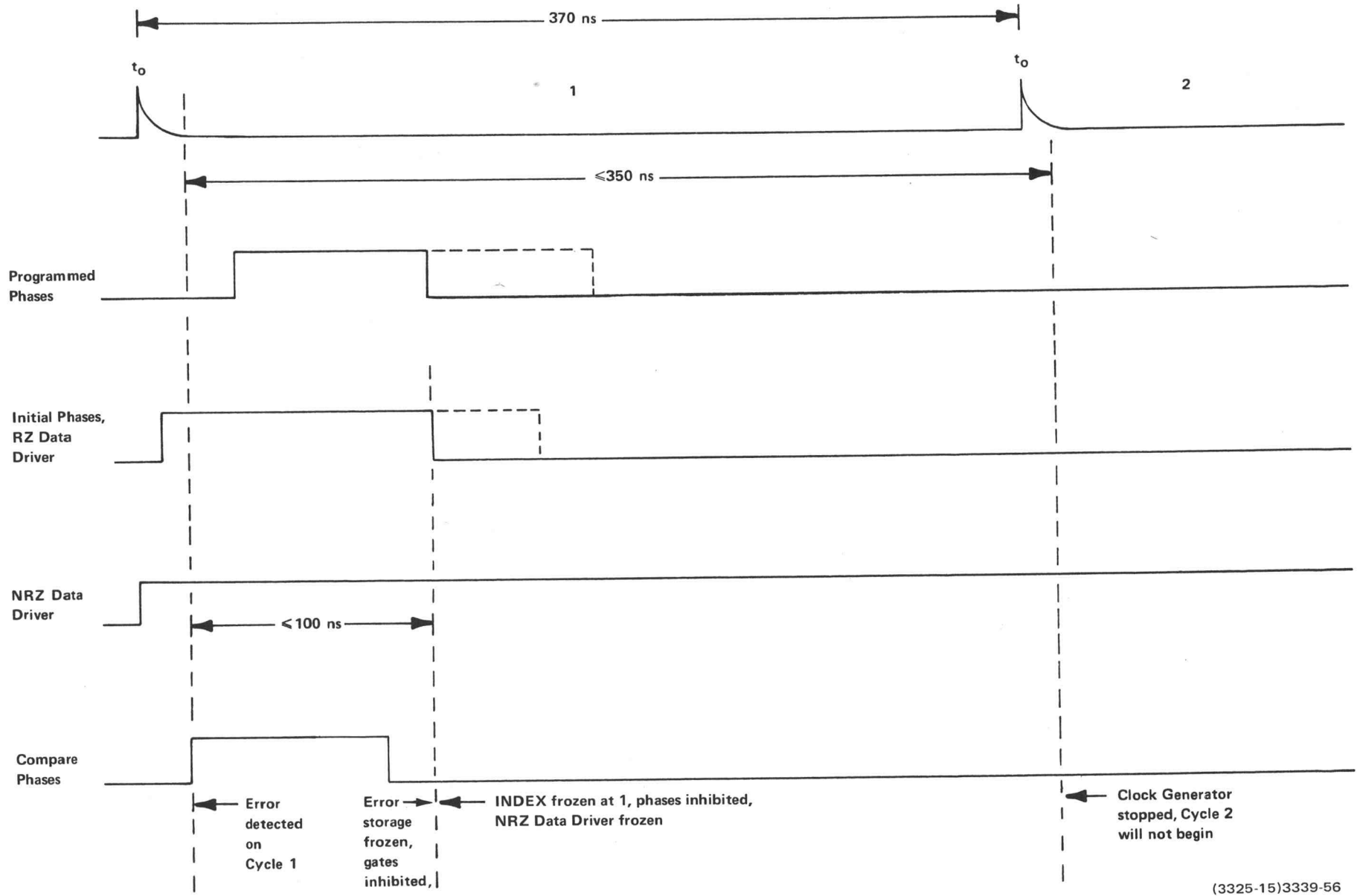


Figure 5-1. Stop-on-Error Circuits, Example 1.



(3325-14)3339-55

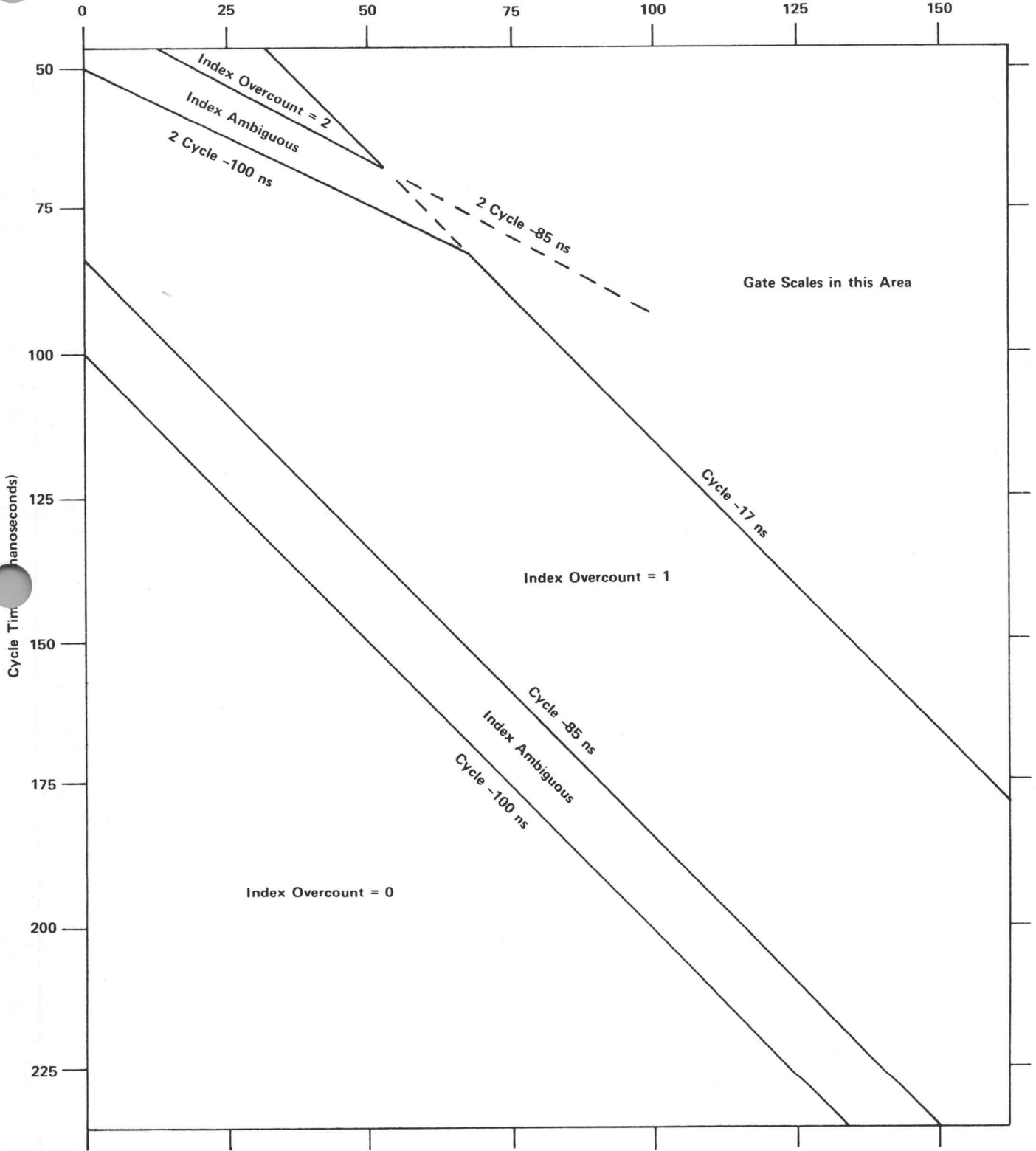
Figure 5-2. Stop-on-Error Circuits, Example 2



(3325-15)3339-56

Figure 5-3. Stop-on-Error Circuits, Example 3.

Gate Start-Time (in nanoseconds)



3325-18

Figure 5-4. Index Overcounts.



SECTION 6: PARAMETRIC TESTING

The TEKTEST statements described in this section specify the type of parametric measurement to be made, read the measurement result, and set the 1140A current and voltage supplies. These statements are:

SETUP
UNSET Mandatory test program statements when making parametric tests.

CURRENT
VOLTAGE (or AUTOV)
TIME Mandatory measurement reserved variables.

VS1-VS4
IS1, IS2 Optional test program statements.

- **SETUP**
UNSET Respectively set and unset the DC and ΔT Subsystems in the 1804 Test Station. The measurement modes available are:

Force voltage	DC Subsystem Measurement Modes
Force current	
Measure voltage	
Measure current while forcing voltage	
Measure time	ΔT Subsystem Measurement Mode

- **CURRENT**
VOLTAGE
(or AUTOV) Initiate their respective measurements to be made in addition to storing the results.

- **TIME** Stores the result of a time measurement but does not initiate the measurement to be made. This is accomplished during a MOVE statement as defined by the **TRIGGER** element in a SETUP TO MEASURE TIME statement.

- **VS1 through VS4** Set the 1140A voltage levels and current limits and allow you to easily redefine the values established with the **SETUP** statement without requiring you to employ the **UNSET** statement first.

- **IS1**
IS2 Set the 1140A current supply levels and voltage limits and also allow you to easily redefine the values established with a **SETUP** statement without requiring you to employ the **UNSET** statement first.

SETTING AND UNSETTING THE DC AND ΔT SUBSYSTEMS
(1804 TEST STATION ONLY)

SETUP { TO MEASURE VOLTAGE
FOR VOLTAGE } ON pins [TO pin1] { AT^{*}
, } range

SETUP { TO MEASURE CURRENT
FOR CURRENT } ON pin1 FROM { DRIVER
VS3=voltage
VS4=voltage } { AT
, } range

SETUP { TO FORCE VOLTAGE
FOR VSTIMULATE } ON pin1 FROM { VS3=voltage
VS4=voltage } { AT
, } currentlimit

SETUP { TO FORCE CURRENT
FOR ISTIMULATE } ON pin1 FROM IS1=current { AT
, } voltage limit

SETUP { TO MEASURE TIME
FOR TIME } { FROM REFERENCE
ON pin1 { +
- } } TO pin1 { +
- } { AT
, } range, TRIGGER exp

SETUP { FOR
TO MEASURE } EXTERNAL { RESISTANCE
VOLTAGE } [MEASUREMENT] ON pin1 TO pin1**

UNSET { TO MEASURE VOLTAGE
FOR VOLTAGE } ON pins [TO pin1]

* A comma is not legal for single-ended voltage measurements.

** Refer to your system digital voltmeter manual for a statement description.

UNSET { TO MEASURE CURRENT
FOR CURRENT } ON pin1 FROM { DRIVER
VS3
VS4 }

UNSET { TO FORCE VOLTAGE
FOR VSTIMULATE } ON pin1

UNSET { TO FORCE CURRENT
FOR ISTIMULATE } ON pin1

UNSET { TO MEASURE TIME
FOR TIME } { FROM REFERENCE
ON pin1 { +
- } } TO pin1 { +
- }

UNSET { FOR
TO MEASURE } EXTERNAL { RESISTANCE
VOLTAGE } [MEASUREMENT] ON pin1 TO pin1*

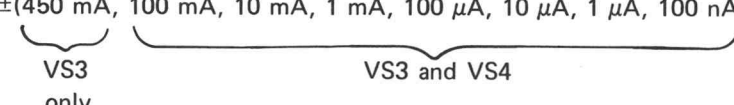
*Refer to your system digital voltmeter manual for a statement description.

The DC and ΔT Subsystems Ranges

DC Subsystem

For voltage: $\pm(100 \text{ V}, 10 \text{ V}, 1 \text{ V}, 0.1 \text{ V})$

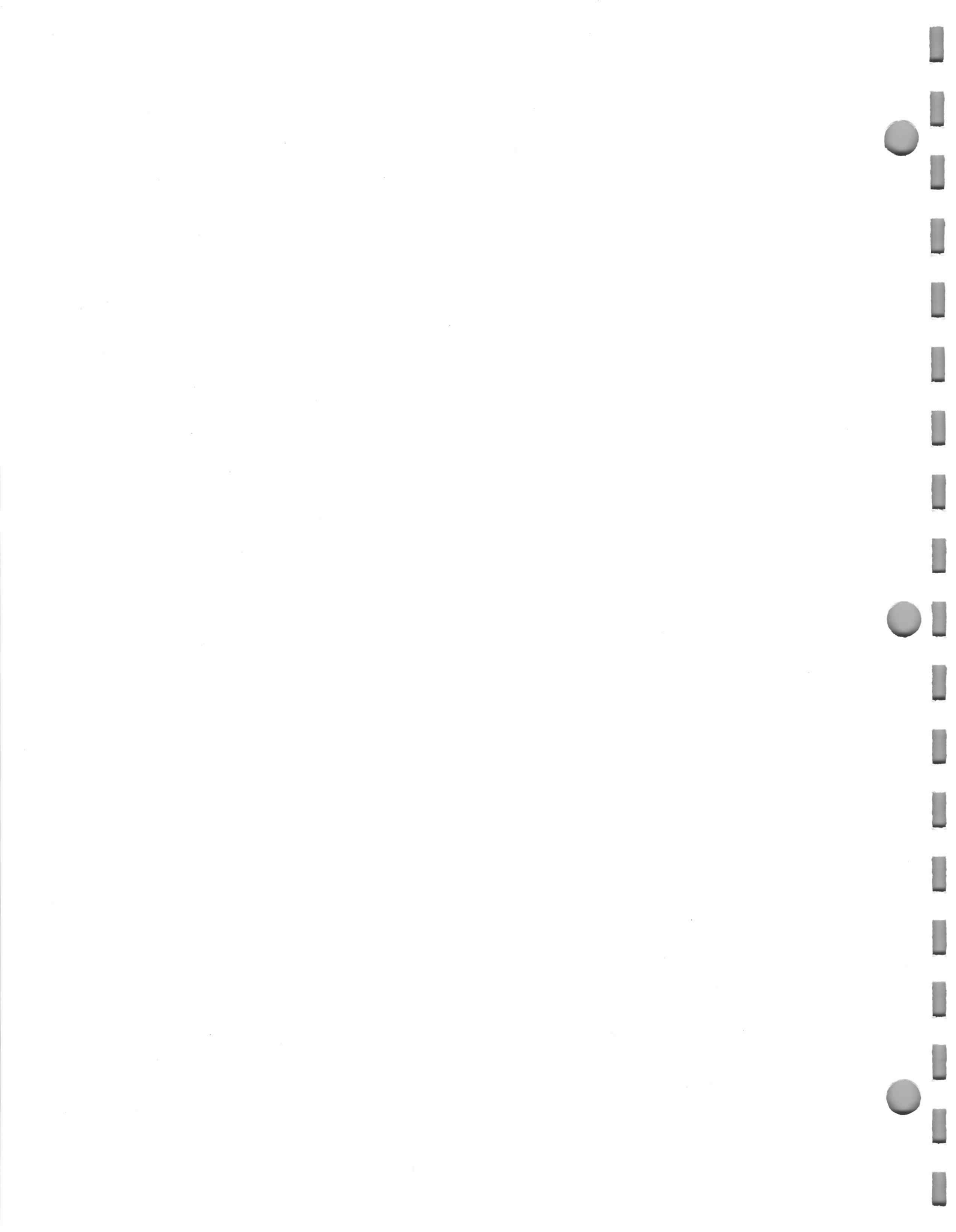
For current: $\pm(450 \text{ mA}, 100 \text{ mA}, 10 \text{ mA}, 1 \text{ mA}, 100 \mu\text{A}, 10 \mu\text{A}, 1 \mu\text{A}, 100 \text{ nA})$



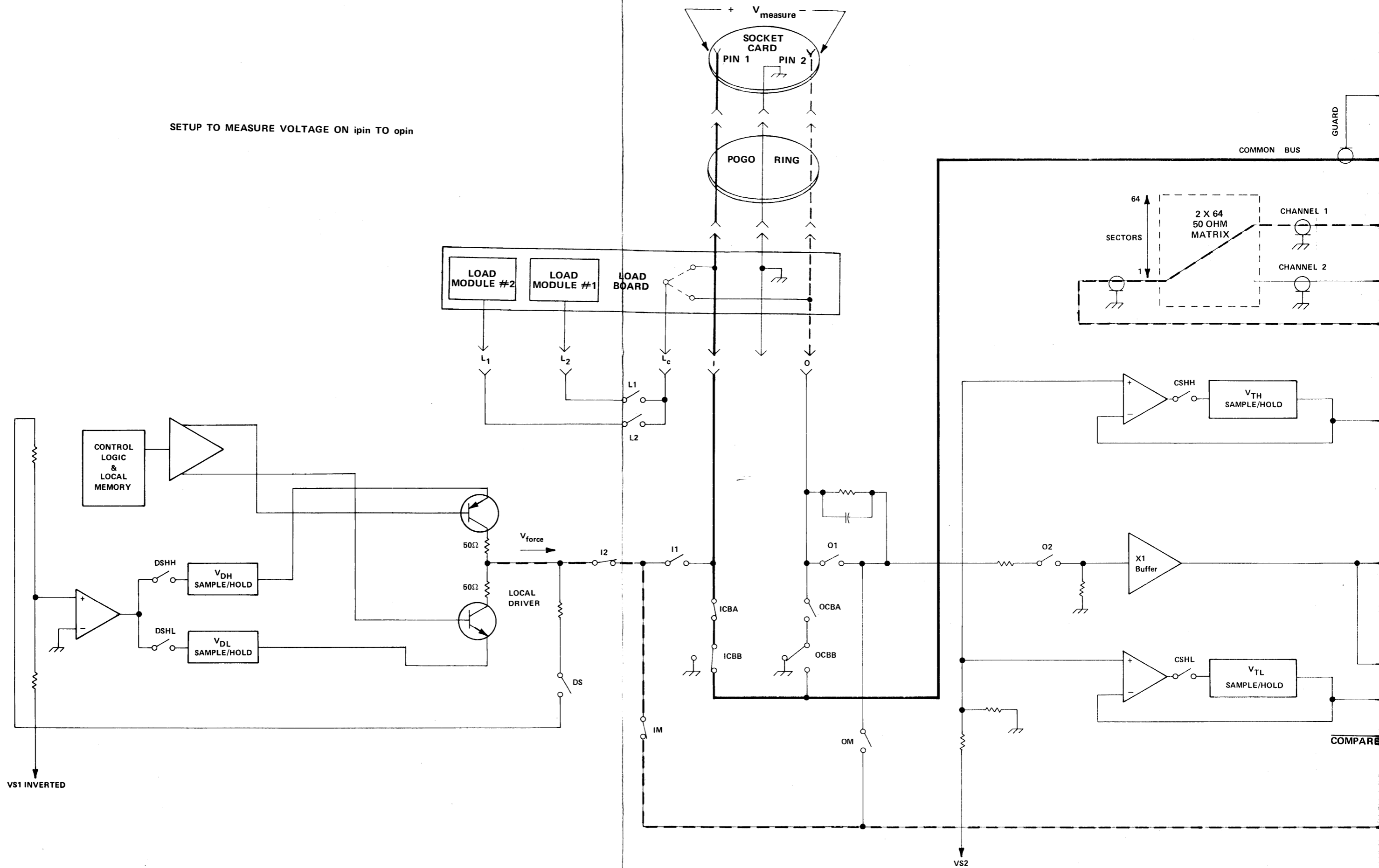
ΔT Subsystem

$\pm(1 \text{ ms}, 100 \mu\text{s}, 10 \mu\text{s}, 1 \mu\text{s}, 100 \text{ ns})$

See also the following interconnecting diagrams and Table 6-1, which illustrates how each SETUP statement affects (closes) the reed switches on the sector cards. In general, the UNSET statement complements the appropriate SETUP statement as a guard against unintentional connections. Thus, UNSET statements open the reed switches affected by SETUP.



SETUP TO MEASURE VOLTAGE ON ipin TO opin



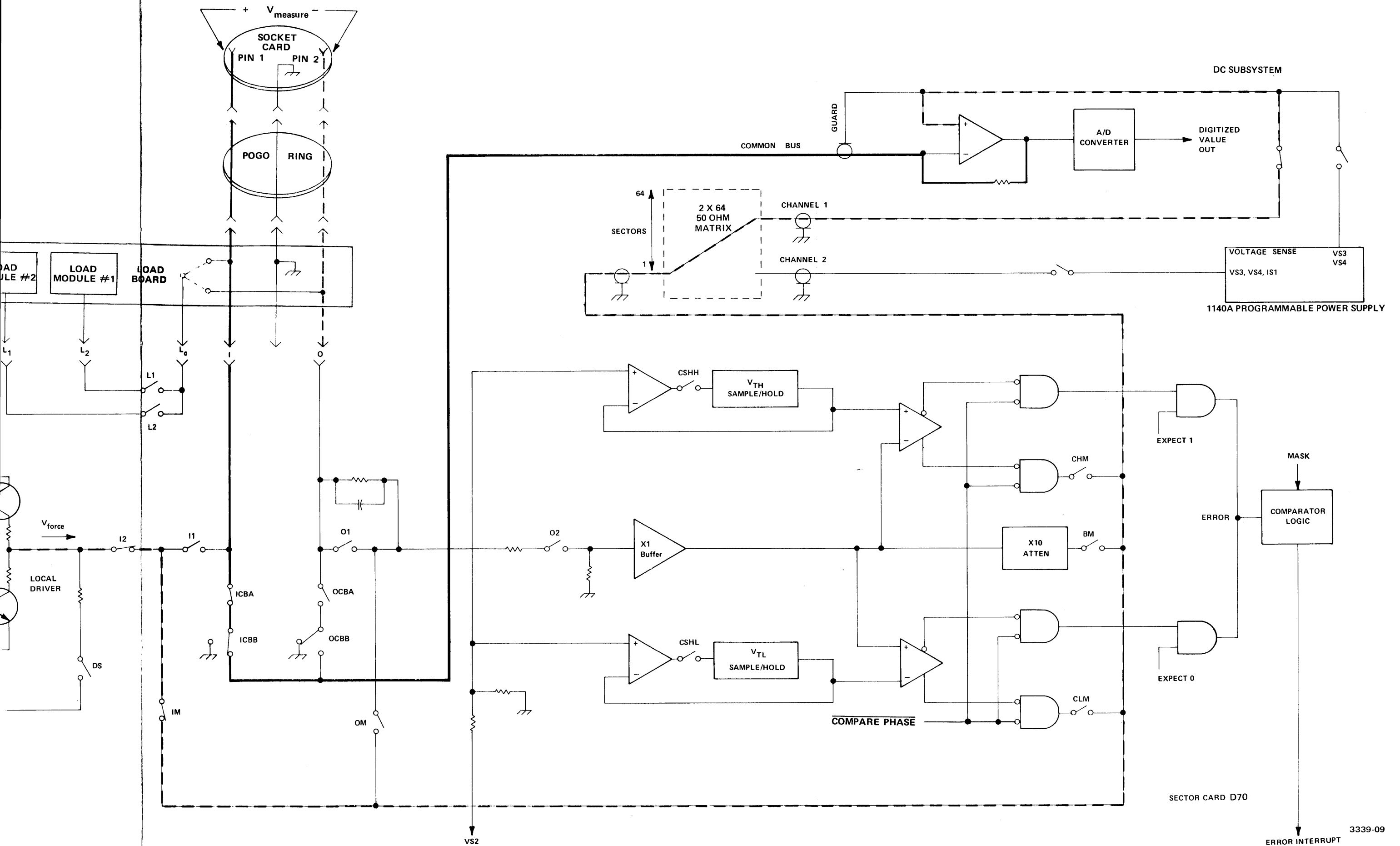


Figure 6-2. INTERCONNECTIONS FOR MEASURING CURRENT (DRIVER PRECONDITIONING)

MEASURING CURRENT FROM A POWER SUPPLY

The **SETUP TO MEASURE CURRENT . . . FROM VS_n** statement performs the following operations:

1. Disables Scope 1.
2. Programs the VS3 or VS4 power supply.
3. Causes a 3 ms delay if the DC Subsystem is programmed to the 100 mA range. Otherwise, it causes a 1 ms delay.
4. Connects the pin to the common bus.*
5. If the DC Subsystem range is 10 mA to 450 mA, it connects the pin to the matrix.
6. Connects channel 1 of the 50 Ω matrix to the pin's sector-card electronics.
7. Causes a 1 ms delay.

The **UNSET** statement complements the above steps. See Figure 6-3.

The formats of the **SETUP** and **UNSET TO MEASURE CURRENT . . . FROM VS_n** statements are:

$$\text{SETUP } \left\{ \begin{array}{l} \text{TO MEASURE CURRENT} \\ \text{FOR CURRENT} \end{array} \right\} \text{ ON pin1 FROM } \left\{ \begin{array}{l} \text{VS3=voltage} \\ \text{VS4=voltage} \end{array} \right\} \left\{ \begin{array}{l} \text{AT} \\ \text{' } \end{array} \right\} \text{ range}$$
$$\text{UNSET } \left\{ \begin{array}{l} \text{TO MEASURE CURRENT} \\ \text{FOR CURRENT} \end{array} \right\} \text{ ON pin1 FROM } \left\{ \begin{array}{l} \text{VS3} \\ \text{VS4} \end{array} \right\}$$

Elements

If **pin1** is an IO pin, the common bus is connected via the I side of the IO sector card path and the matrix (sense line) is connected via the O side of the IO path, placing the sense line as close to the DUT as possible.

VS3=voltage and **VS4=voltage** are any legal numeric expressions. See the ranges for the DC Subsystem for acceptable values.

range is any legal numeric expression.

*The DUT is connected to the common bus after VS3 or VS4 is connected to the common bus to avoid having the DUT charge the common bus.

Measuring Current on More Than One Pin

Although this statement causes the pin to be connected to the common bus, a group of pins may not be used since channel 1 of the matrix may be connected. However, measuring current on more than one pin may be achieved by using the `CONNECT TO COMMON` statement.

Effects on the DC Subsystem

This statement does not disable the DC Subsystem and does not open a connection previously made with a `SETUP TO FORCE VOLTAGE` or `SETUP TO FORCE CURRENT` statement. However, a `SETUP TO FORCE VOLTAGE` or `CURRENT` should not be in operation when executing this statement.

Effects on the Common Bus

When you are using the common bus for `SETUP TO MEASURE CURRENT`, you should avoid using the `INITIALIZE` and `STOP` statements. In addition, you should avoid pressing the `STOP` button on the Test Station Control Unit. Failure to observe these precautions may cause the DUT pins to be grounded for a short time and result in DUT failure or reed switch damage on the sector card. You should always use the `UNSET` statement before performing any of the above operations.

Effects on the Driver

The `SETUP TO MEASURE CURRENT . . . FROM VSn` statement disconnects the driver on the 10 mA to 450 mA ranges. The driver must be disconnected to get to the sense line connected.

Effects on the Comparator

The comparator, if connected, is forced to the 5-volt range if you are using the 10 mA to 450 mA ranges with a `SETUP TO MEASURE CURRENT . . . FROM VSn` statement employing an `O` or `IO` pin. The `UNSET TO MEASURE CURRENT` statement forces the comparator, if connected, to the 30-volt range.

Disconnections Required

When you are performing a current measurement, you must make sure that the driver is disconnected from the pin to which you are connecting the DC Subsystem.

Kelvin Sensing

When the DC Subsystem is in the 10 mA range or above, remote Kelvin sensing of the forcing voltage can occur at the DUT. The sense path is via the O port of the DUT if the pin is an IO pin. Kelvin sensing can only occur on the sector card when the DUT pin is an I or O pin. Kelvin sensing at the DUT produces a more accurate measurement.

MEASURING TIME

The **SETUP TO MEASURE TIME** statement performs the following operations:

1. Connects the first pin to the 50 Ω matrix.*
2. Connects channel 2 of the matrix to the first pin's sector-card electronics.*
3. Connects the second pin to the matrix.
4. Connects channel 1 of the matrix to the second pin's sector-card electronics.
5. Causes a 1 ms delay.

The **UNSET** statement complements the above steps.

The formats of the **SETUP** and **UNSET TO MEASURE TIME** statements are:

$$\text{SETUP } \left\{ \begin{array}{l} \text{TO MEASURE TIME} \\ \text{FOR TIME} \end{array} \right\} \left\{ \begin{array}{l} \text{FROM REFERENCE} \\ \text{ON pin1 } \left\{ \begin{array}{l} + \\ - \end{array} \right\} \end{array} \right\} \text{ TO pin1 } \left\{ \begin{array}{l} + \\ - \end{array} \right\} \left\{ \begin{array}{l} \text{AT} \\ , \end{array} \right\} \text{ range, TRIGGER exp}$$
$$\text{UNSET } \left\{ \begin{array}{l} \text{TO MEASURE TIME} \\ \text{FOR TIME} \end{array} \right\} \left\{ \begin{array}{l} \text{FROM REFERENCE} \\ \text{ON pin1 } \left\{ \begin{array}{l} + \\ - \end{array} \right\} \end{array} \right\} \text{ TO pin1 } \left\{ \begin{array}{l} + \\ - \end{array} \right\}$$

Elements

REFERENCE is the timing signal from the programmable clock generator to the ΔT Subsystem (see also the following notes).

pin1 $\left\{ \begin{array}{l} + \\ - \end{array} \right\}$ refers to the two levels of the comparator. + indicates the high-level side and - indicates the low-level side.

range is any legal numeric expression. The system rounds the expression to the nearest range. The ranges are: 1 ms, 100 μs , 10 μs , 1 μs , and 100 ns.

TRIGGER exp ensures that the ΔT Subsystem is properly armed at the beginning of a clock cycle. The **exp** element is an expression which specifies the t_0 on which the ΔT Subsystem is to be armed.

*These steps are omitted if the element **REFERENCE** is chosen.

When Time Measurements Are Made

Time measurements are made during a MOVE statement, typically on the last programmed cycle. The **TRIGGER** element in the SETUP TO MEASURE TIME statement defines when the time measurement is enabled (when the ΔT Subsystem is armed). For example, if the **TRIGGER exp** element is 5, the time measurement is made during the fifth clock cycle of the move operation.

Effects on the TRIGGER Statement

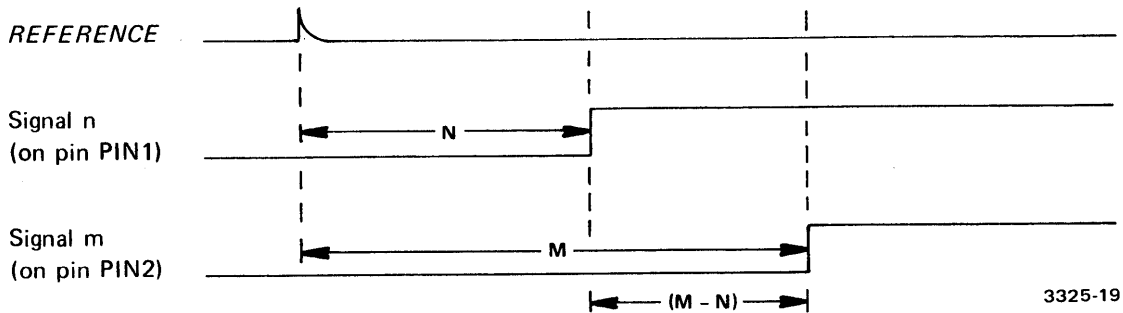
Use of the TRIGGER element in the SETUP TO MEASURE TIME statement does not preclude the use of the clock generator trigger output as a scope trigger when not measuring time.

Using REFERENCE to Make a Time Measurement

Two SETUP statements are required to make a time measurement if the **REFERENCE** element is specified. This is because the *REFERENCE* signal is generated at an unspecified, but constant, time during the test.

Therefore, make a time measurement on one pin, then a second measurement using a different pin. The difference in times is the result (the final time measurement).

For example:



Program the following statements to use the *REFERENCE* signal.

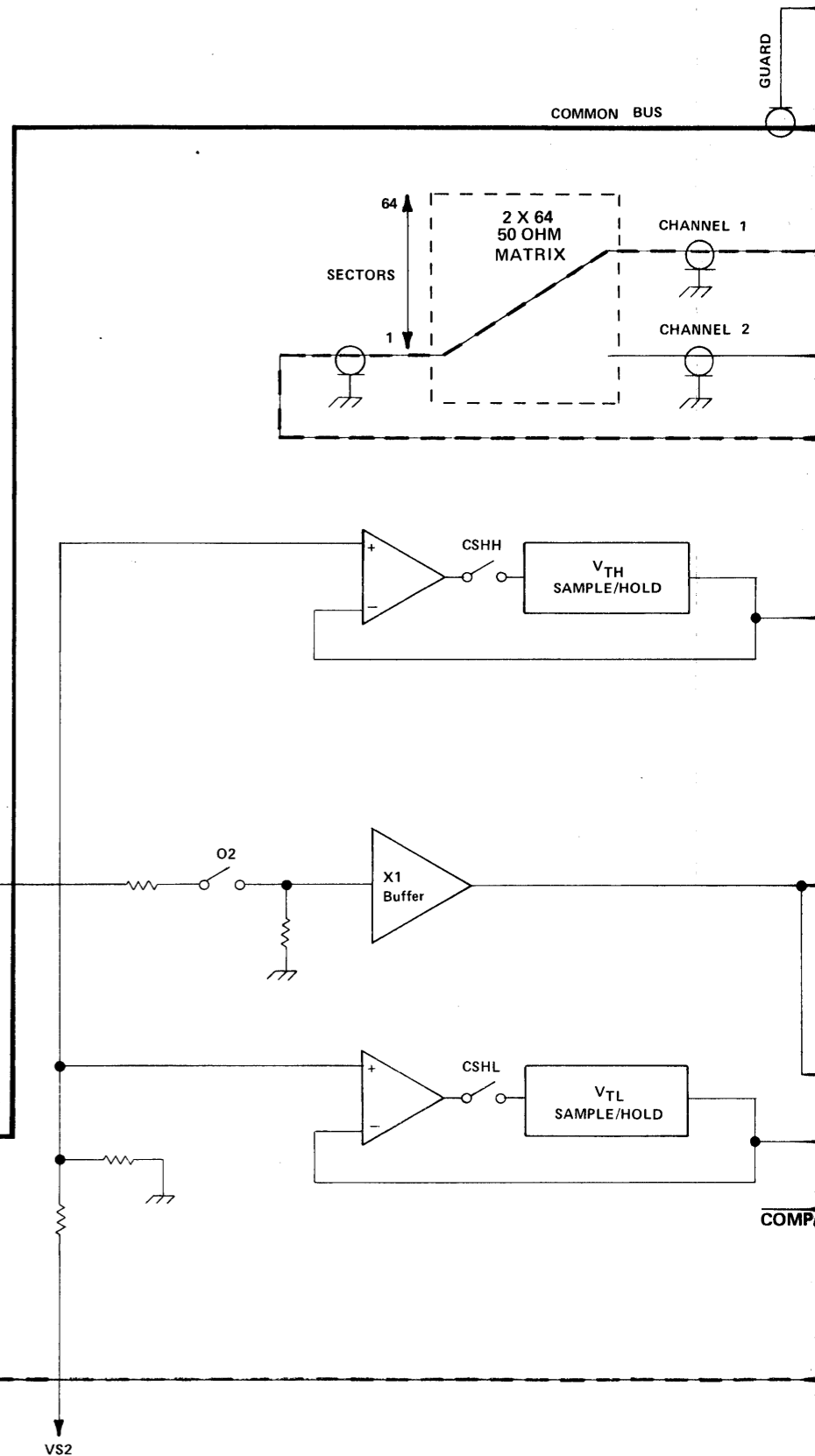
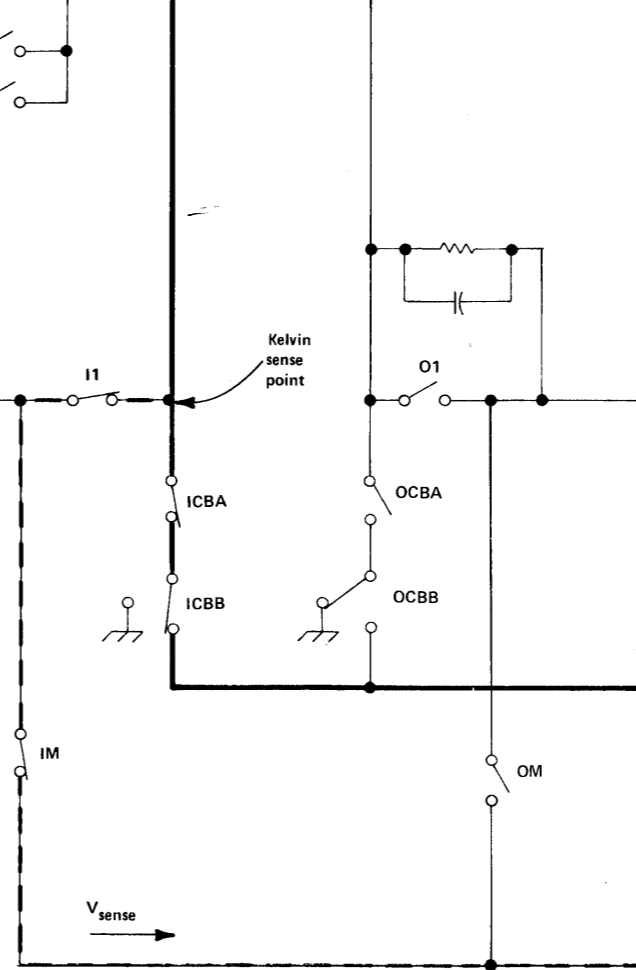
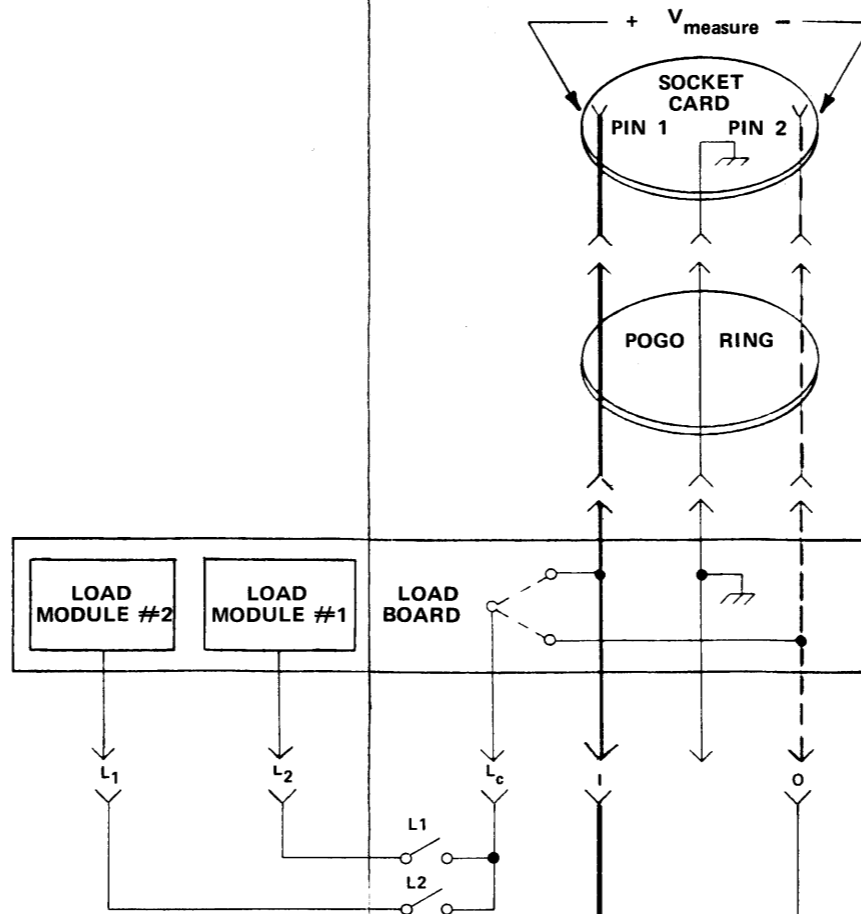
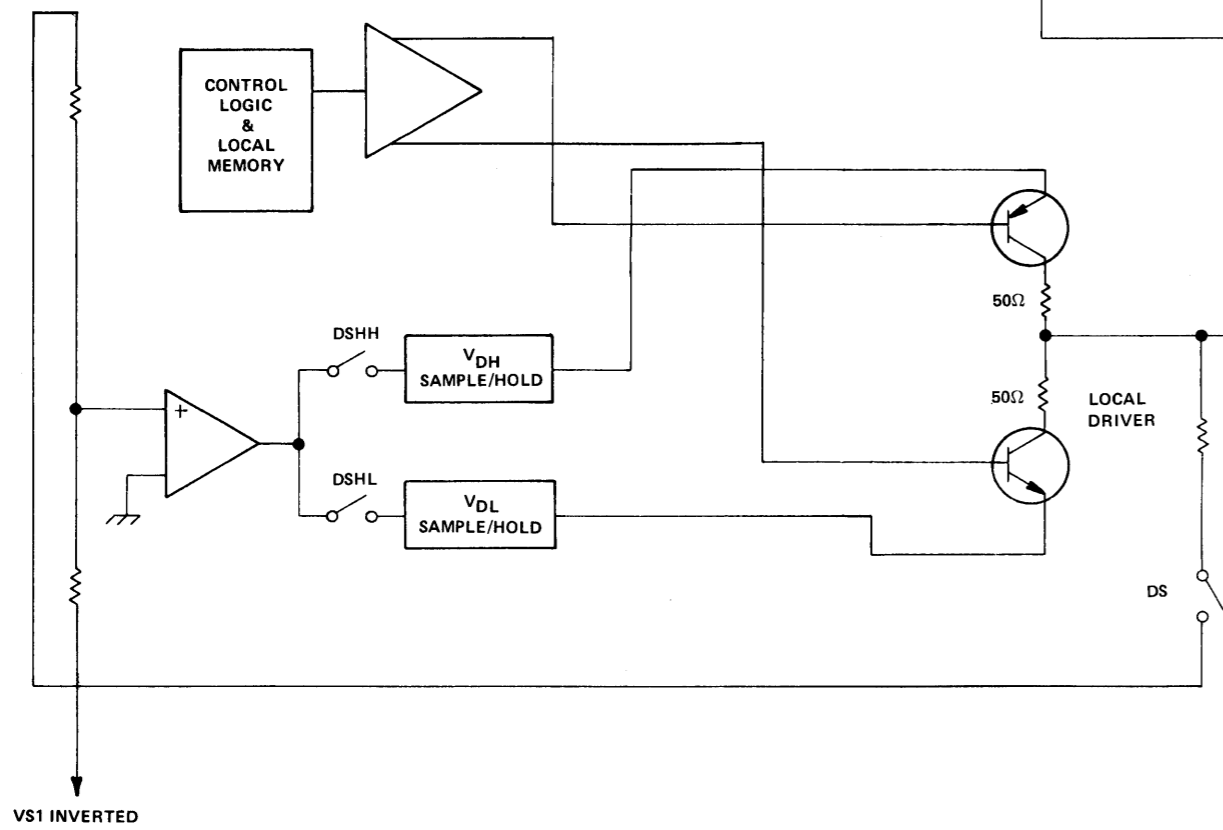
```
SETUP TO MEASURE TIME FROM REFERENCE TO PIN1 . . .  
MOVE . . .  
N = TIME  
SETUP TO MEASURE TIME FROM REFERENCE TO PIN2 . . .  
MOVE . . .  
M = TIME  
DIFFER = M - N
```

The same measurement could have been made by using the statements:

```
SETUP TO MEASURE TIME FROM PIN1 TO PIN2 . . .  
MOVE . . .  
DIFFER = TIME
```



SETUP TO MEASURE VOLTAGE ON ipin TO opin



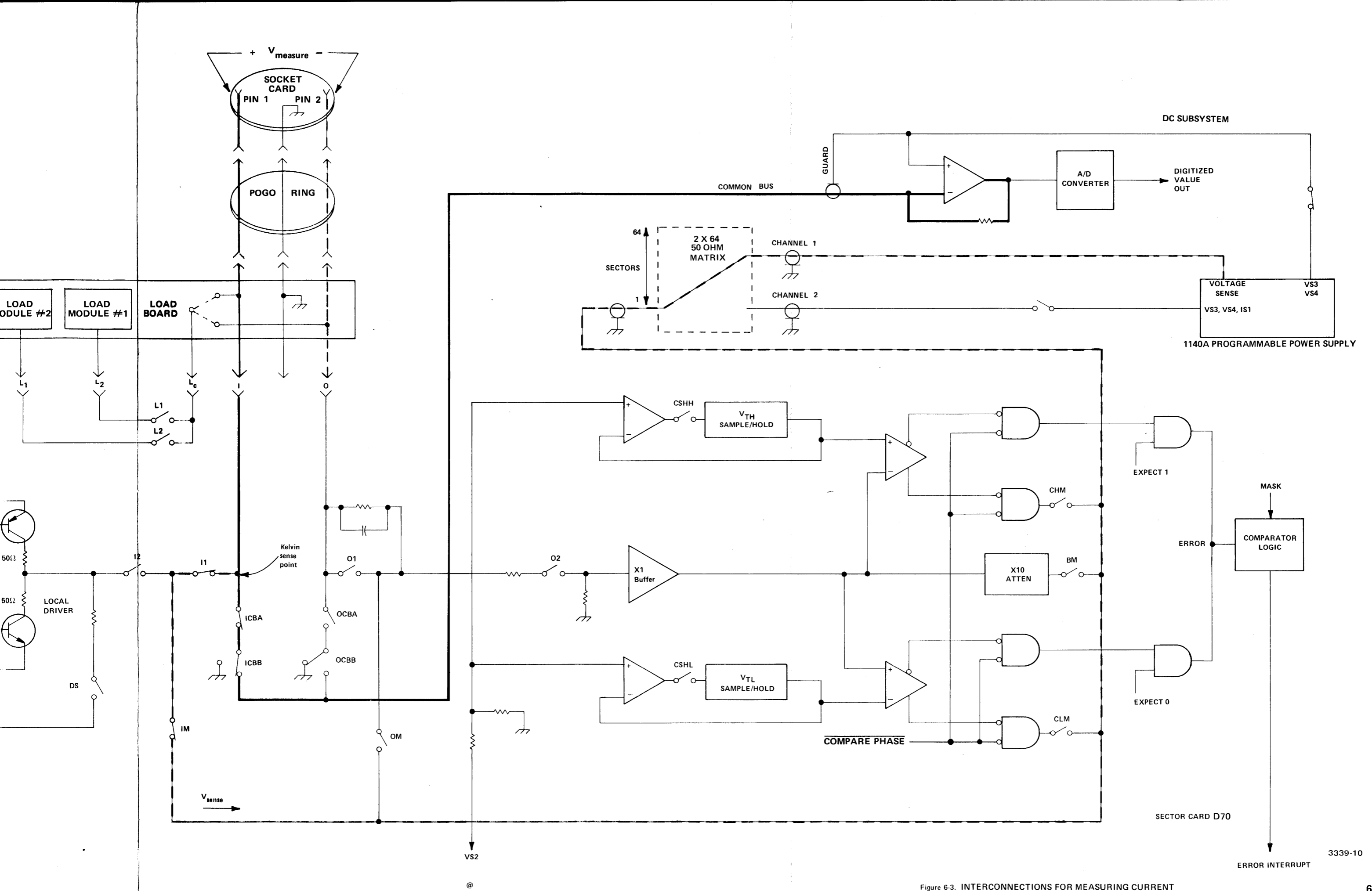


Figure 6-3. INTERCONNECTIONS FOR MEASURING CURRENT

FORCING VOLTAGE

The **SETUP TO FORCE VOLTAGE** statement performs the following operations:

1. Disables Scope 2.
2. Programs the VS3 or VS4 supply.
3. Causes a 1 ms delay.
4. Connects the pin to the 50 Ω matrix. Disconnects the driver if the pin is an I pin.
5. Connects channel 2 of the matrix to the sector card associated with the pin.
6. Causes a 1 ms delay.

The **UNSET** statement complements the above steps.

The formats of the **SETUP** and **UNSET TO FORCE VOLTAGE** statements are:

$$\text{SETUP } \left\{ \begin{array}{l} \text{TO FORCE VOLTAGE} \\ \text{FOR VSTIMULATE} \end{array} \right\} \text{ ON pin1 FROM } \left\{ \begin{array}{l} \text{VS3=voltage} \\ \text{VS4=voltage} \end{array} \right\} \left\{ \begin{array}{l} \text{AT} \\ ' \end{array} \right\} \text{ currentlimit}$$
$$\text{UNSET } \left\{ \begin{array}{l} \text{TO FORCE VOLTAGE} \\ \text{FOR VSTIMULATE} \end{array} \right\} \text{ ON pin1}$$

Elements

If **pin1** is an IO pin, the 50 Ω matrix is connected via the O side of the IO sector card path.

voltage and **currentlimit** are any legal numeric expressions.

Effects on the DC Subsystem

This statement does not disable the DC Subsystem and does not open a connection made previously with a SETUP TO MEASURE CURRENT or VOLTAGE. However, a SETUP TO FORCE VOLTAGE or CURRENT should not be in operation when executing this statement.

If a SETUP TO MEASURE CURRENT statement is in effect when executing this statement, both statements must not use the same voltage source.

Effects on the Comparator

The comparator, if connected, is forced to the 5-volt range if the pin in a SETUP TO FORCE VOLTAGE statement is an O or IO pin. The UNSET TO FORCE VOLTAGE statement forces the comparator, if connected, to the 30-volt range.

FORCING CURRENT

The **SETUP TO FORCE CURRENT** statement performs the following operations:

1. Disables Scope 2.
2. Causes a 1 ms delay to allow the 50 Ω matrix to discharge.
3. Connects the pin to channel 2 of the matrix. Disconnects the driver if the pin is an I pin.
4. Connects channel 2 of the matrix to the pin's sector-card electronics.
5. Sets the IS1 current supply.

The **UNSET** statement complements the above steps.

The formats of the **SETUP** and **UNSET TO FORCE CURRENT** statements are:

SETUP { **TO FORCE CURRENT**
 FOR ISTIMULATE } **ON pin1 FROM IS1=current** { **AT** } **voltagelimit**

UNSET { **TO FORCE CURRENT**
 FOR ISTIMULATE } **ON pin1**

Elements

If **pin1** is an IO pin, the 50 Ω matrix is connected via the O side of the IO sector card path. (When measuring a voltage and forcing a current on an IO pin, the voltage is measured as close to the DUT as possible.)

IS1=current is any legal numeric expression. See the current supply statement (IS1) for acceptable range values.

voltagelimit is any legal numeric expression. See the ranges for the DC Subsystem for acceptable values.

Effects on the Comparator

The comparator, if connected, is forced to the 5-volt range if the pin is an O or IO pin in a SETUP TO FORCE CURRENT statement. The UNSET TO FORCE CURRENT statement forces the comparator, if connected, to the 30-volt range.

The IS1 Current Supply

IS1 should not be enabled when executing this statement. It is disabled by an UNSET TO FORCE CURRENT statement.

Effects on the DC Subsystem

This statement does not disable the DC Subsystem and does not open a connection made previously with a SETUP TO MEASURE CURRENT or a SETUP TO MEASURE VOLTAGE. However, a SETUP TO FORCE VOLTAGE or TO FORCE CURRENT statement should not be in operation when executing this statement.

Channel 2 Matrix and Ground Relationship

During execution of a SETUP TO FORCE CURRENT statement, current moves between channel 2 of the matrix and ground. Should the connection between channel 2 and ground be opened, for example with a DISCONNECT statement, IS1 will go to its voltage limit and charge channel 2. If the connection between channel 2 and ground is then made, channel 2 will discharge and may produce a high transient current.

ASSIGNING MEASUREMENT RESERVED VARIABLES

CURRENT, **VOLTAGE**, and **TIME** are reserved variables. In the test program, they can be assigned to on-going numeric results of parametric measurements by using them as expressions and in statements which allow expressions (for example, IF statements). The three reserved variables listed above correspond respectively to the measurements defined and established by:

SETUP TO MEASURE CURRENT
SETUP TO MEASURE VOLTAGE
SETUP TO MEASURE TIME

Current and voltage measurements are actually made each time the reserved variables **CURRENT** or **VOLTAGE** follow their respective **SETUP** statements. Each time these variables appear in the test program, the measurements are made. If you wish to store the result of a measurement, the appropriate statement:

variable= {
CURRENT
VOLTAGE
}

should be employed. This statement makes the measurement and stores the result in **variable**. Time measurements, however, are made during a **MOVE** statement. The **TRIGGER** element in the **SETUP TO MEASURE TIME** statement defines when the time measurement is enabled. The time measurement result is saved as an analog voltage and can be read by using the reserved variable **TIME**. For example,

variable=TIME (This reads the A/D converter.)

The reserved variable **AUTOV** can be used in place of the reserved variable **VOLTAGE**. **AUTOV** allows you to measure voltage and store or print the result. **AUTOV** employs the user-specified range element in the **SETUP TO MEASURE VOLTAGE** statement to make the initial voltage measurement. However, if the actual voltage is outside the specified range, **AUTOV** automatically scales up or down as necessary and repeats the measurement.

CURRENT, VOLTAGE, and AUTOV perform a new measurement each time one is encountered during test program execution. The statement PRINT (VOLTAGE+VOLTAGE)/2 will print the average of two measurements. Before the S-3200 System Instruction Processor initiates a measurement, it waits to allow the voltages in the DC Subsystem to settle. The setting times are as follows:

Settling Times			
Range	Measure VOLTAGE	Measure CURRENT	
		NOT From DRIVER	From DRIVER
100 mV	4 ms		
1 V	.6 ms		
10 V	.6 ms		
100 V	1.2 ms		
100 nA		25 ms	25 ms
1 μ A		8 ms	9 ms
10 μ A		1.5 ms	6 ms
100 μ A		0.3 ms	0.45 ms
1 mA		0	0
10 mA		0.2 ms	0
100 mA		1.5 ms	0
450 mA		10.0 ms	

AUTOV may perform up to four successive measurements. It must wait for the voltage in the DC Subsystem to settle before each measurement. For example, to measure 0 V on the 100 V range will involve repeated measurement on each range until the 100 mV range is reached. The sum of all settling times would be 6.4 ms.

SETTING THE 1140A POWER SUPPLIES

The **VS1** through **VS4** statements set the voltage levels and current limits for the voltage sources 1 through 4, respectively. Since VS1 and VS2 are used for the HIDRIVE and LODRIVE power supplies, you can consider them as system dedicated. Thus, you most likely will not employ these statements. VS3 and VS4, however, are used to make parametric measurements with the DC Subsystem and the ΔT Subsystem. VS3 and VS4 allow you to redefine the levels established with the SETUP TO FORCE VOLTAGE statement while leaving the data paths unaltered.

The formats of the VS1 through VS4 statements are:

$$\left. \begin{array}{l} \text{VS1} \\ \text{VS2} \\ \text{VS3} \\ \text{VS4} \end{array} \right\} = \text{voltagevalue} \left\{ \begin{array}{l} \\ \text{AT} \end{array} \right\} \text{currentlimit}^*$$

Elements

voltagevalue and **currentlimit** are any legal numeric expressions. VS1 and VS2 are used internally by the system for the HIDRIVE and LODRIVE power supplies. (See their descriptions in Section 3 for their range limitations.) The **voltagevalue** and **currentlimit** for VS3 and VS4 are as follows:

VS3 has a range of ± 39.99 V programmable in 10 mV steps with a maximum current of ± 450 mA.

VS4 has a range of ± 99.99 V programmable in 10 mV steps with a maximum current of ± 120 mA.

*VS5-VS8 are documented in the *MC-3 Monitor/Power Supply Option* manual. VS5-VS8 do not have equivalent SETUP statements.

SETTING THE 1140A CURRENT SUPPLIES

The **IS1** and **IS2** statements set the current values for current supply 1 and the optional current supply 2, respectively. These statements allow you to redefine the current value established with the **SETUP TO FORCE CURRENT** statement while leaving the data paths unaltered. **IS1** and **IS2** can only be used when you are forcing current and measuring voltage.

The formats of the **IS1** and **IS2** statements are:

$$\text{IS1}=\text{currentvalue} \left. \begin{array}{c} ' \\ \text{AT} \end{array} \right\} \text{voltage\textit{limit}}$$
$$\text{IS2}=\text{currentvalue} \left. \begin{array}{c} ' \\ \text{AT} \end{array} \right\} \text{voltage\textit{limit}}$$

Elements

currentvalue and **voltage\textit{limit}** are any legal numeric expressions. **voltage\textit{limit}** is determined by the nature of the DUT. **currentvalue** may be:

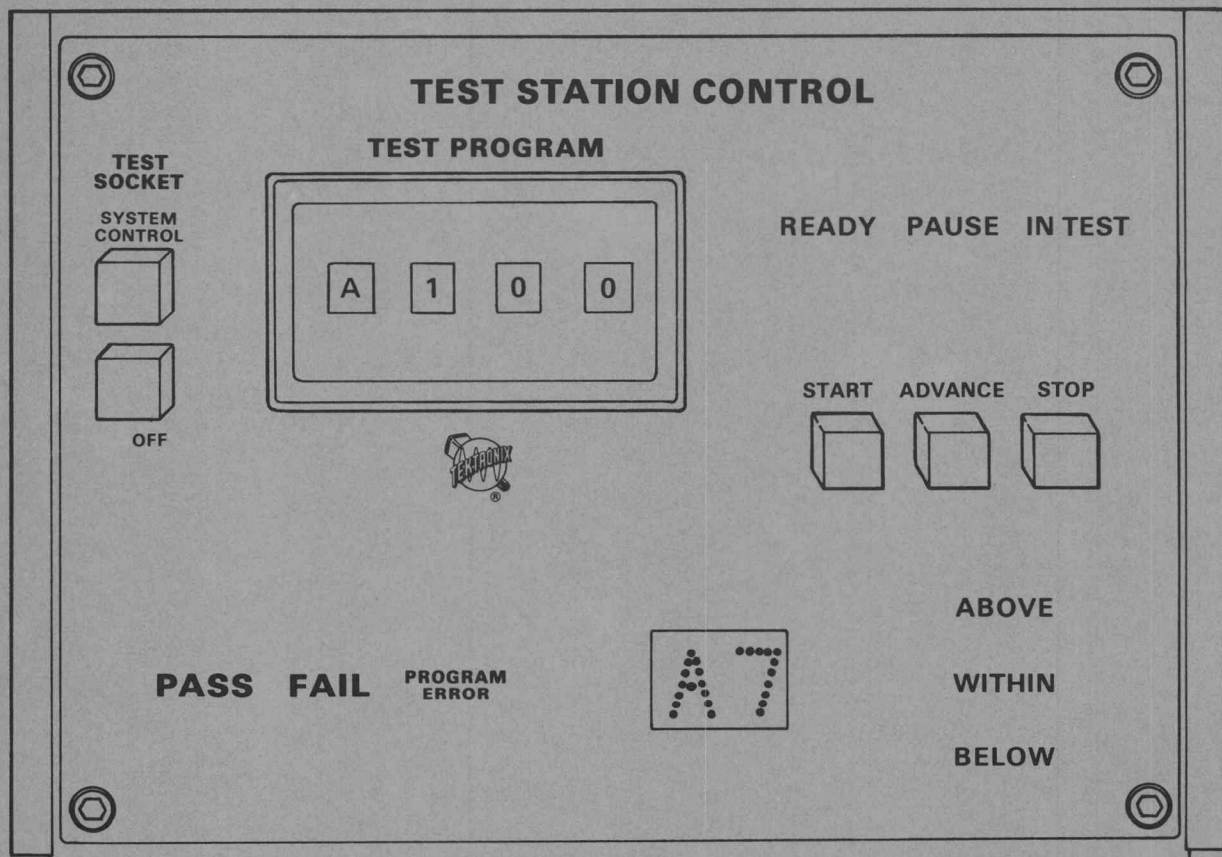
- ±200 mA in 100 μ A steps
- ±20 mA in 10 μ A steps
- ±2 mA in 1 μ A steps

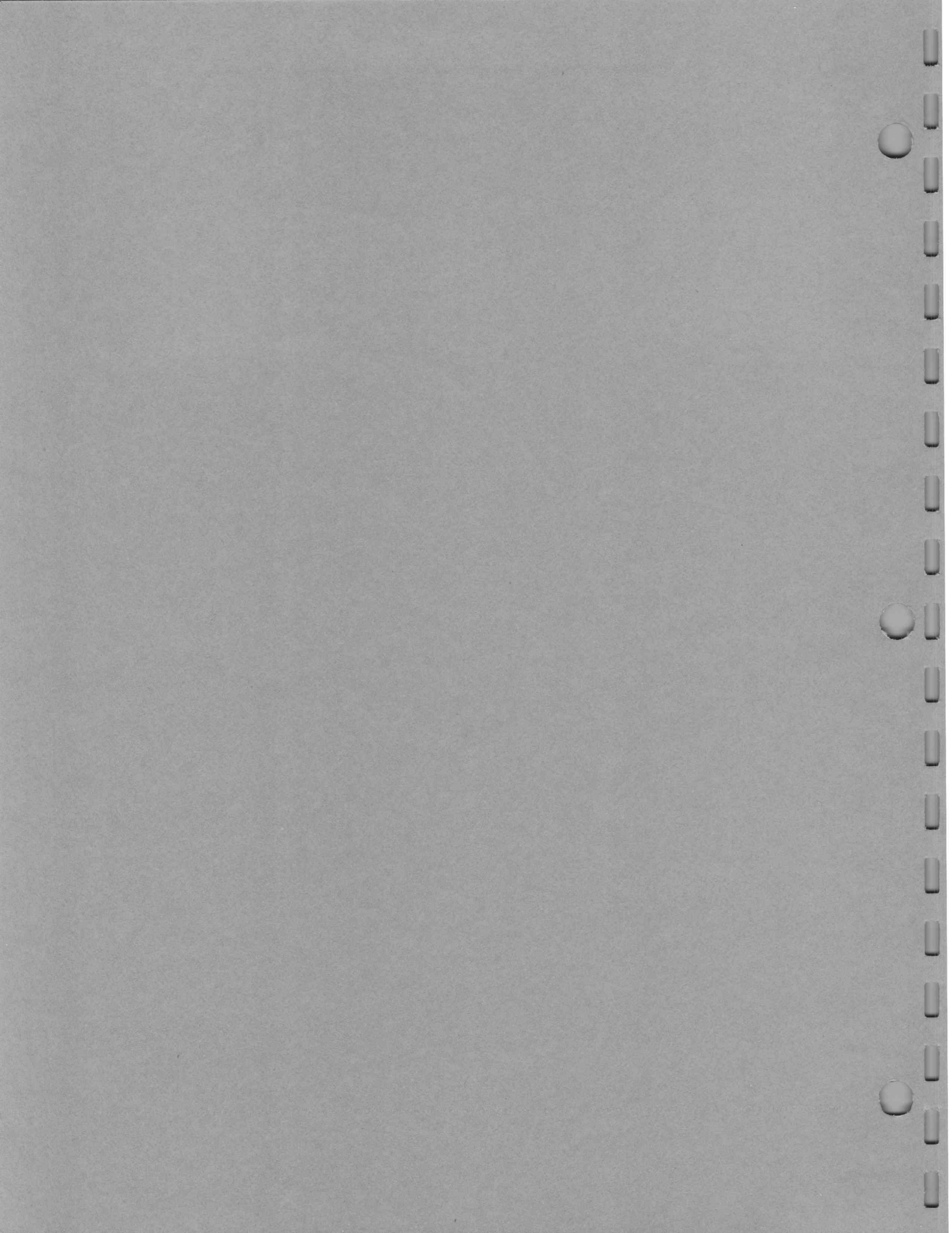
SECTION 7: TSCU OPERATOR INTERACTION

The statements and variables described in this section pertain to operator interaction at the Test Station Control Unit (TSCU). You initiate testing and interpret results through the TSCU. For further information on these procedures, see Section 5 of the *Command Language Reference Guide*.

Statements and Variables in this Section

- **ADVANCE** Provides user control of testing
- **DISPLAY** Displays test results on the TSCU
- **SORT** Performs the same function as **DISPLAY** and stores the number of times the **SORT** elements are accessed.





USER-CONTROLLED TESTING

The **ADVANCE** variable* indicates if the TSCU ADVANCE button has been pushed. ADVANCE has a value of one if the button is pressed and a value of zero if it is not pressed.

ADVANCE may be used as the element in a logical IF statement,* or in any legal expression. The ADVANCE variable format is:

ADVANCE

Using ADVANCE as the element in a logical IF statement provides user control of testing. Programming ADVANCE in this manner causes the test to stay in a loop until you press the ADVANCE button on the TSCU. This allows you to make necessary adjustments. The TSCU PAUSE indicator, which indicates program looping, lights as a result of ADVANCE being programmed in an IF statement. When you press ADVANCE, the program proceeds to run and the PAUSE indicator turns off.

The statement for this use is:

IF (ADVANCE) linenummer [,linenummer]

If the ADVANCE button is pressed, the program branches true; if not pressed, the program branches not true.

ADVANCE has a value of either one or zero, depending on the condition of the ADVANCE button. You may use this value in an expression.

Example:

<pre>>6.5 VAR = 4 >6.6 X = VAR + ADVANCE >6.7 DISPLAY (X)</pre>	Depending on whether or not the ADVANCE button is pressed, either four or five will be displayed.
---	---

*See the *Test Language, Part One* for information on reserved system variables and the IF statement and its elements.

Example:

```
>1.0 IF (ADVANCE) 17.0,4.0
>2.0 * STATEMENTS 4.0 THROUGH 9.0 CONSTITUTE A
>3.0 * TEST TO DETERMINE THE VALUE OF VOLTAGE
.
.
.
>9.0
>10.0 IF (LOWER LIMIT<VOLTAGE<UPPER LIMIT) 11.0,13.0,15.0
>11.0 DISPLAY BELOW
>12.0 GOTO 1.0
>13.0 DISPLAY WITHIN
>14.0 GOTO 1.0
>15.0 DISPLAY ABOVE
>16.0 GOTO 1.0
>17.0 * TEST PROGRAM PROCEEDS
```

If the TSCU ADVANCE button is not pressed (a value of zero), the program branches not true to line 4.000. This causes the PAUSE indicator to light and the program stays in a loop until the ADVANCE button is pressed. When the ADVANCE button is pressed (a value of one), the program branches true to line 17.0000 and falls out of the loop.

```
>101.0 * TEST TO DETERMINE THE VALUE OF VOLTAGE
.
.
.
>105.0 IF (LOWER LIMIT<VOLTAGE<UPPER LIMIT) 106.0,107.0,108.0
>106.0 DISPLAY BELOW
>106.5 GOTO 101.0
>107.0 DISPLAY WITHIN
>107.5 GOTO 109.0
>108.0 DISPLAY ABOVE
>109.0 IF (ADVANCE) 110.0,101.0
>110.0 * TEST PROGRAM PROCEEDS
```

In this example, the PAUSE indicator does not light unless VOLTAGE is WITHIN limits.

DISPLAYING TEST RESULTS

The **DISPLAY** statement displays test results on the TSCU test result indicators and PROGRAM ERROR readout display.

The test result indicators operate only when you program them with a **DISPLAY** or **SORT** statement. You may program any or all of these lights to indicate the pass/fail results of the test and whether these results are above, within, or below the programmed limits.

In addition, you may program **DISPLAY** to display on the readout a user-defined, two-digit number. If these display numbers are the same as the Instruction Processor error codes, the **PROGRAM ERROR** indicator will not light.

The **DISPLAY** statement format is:

```
DISPLAY expression  [ ,PASS  
                    [ ,FAIL  
                    [ ,ABOVE . . . ,PASS  
                    [ ,WITHIN      ,FAIL  
                    [ ,BELOW      ,ABOVE  
                               [ ,WITHIN  
                               [ ,BELOW
```

expression may be any legal expression that returns a value between 00 and 99 inclusive. Numbers greater than 99 cause the error code D0 to be displayed on the **PROGRAM ERROR** readout display.

PASS, FAIL, ABOVE, WITHIN, and BELOW are the test result indicators.

Example:

<p>>6.5 VAR = 4 >6.6 X = VAR + ADVANCE >6.7 DISPLAY (X)</p>	<p>Displays the value of the expression (X) in the readout display.</p>
<p>>8.0 DISPLAY 29,PASS</p>	<p>Displays 29 in the readout display and lights the PASS indicator.</p>
<p>>2.25 DISPLAY (X*Y),PASS,WITHIN</p>	<p>Displays the value of the expression (X*Y) in the readout display and lights the PASS and WITHIN indicators.</p>

DISPLAYING TEST RESULTS AND INCREMENTING THE INTERNAL COUNTERS

The **SORT** statement performs the same function as the **DISPLAY** statement. **SORT** also has the additional capability of storing the number of times any of the **SORT** elements are accessed.

The **SORT** statement format is:

SORT expression [,PASS
,FAIL
,ABOVE
,WITHIN
,BELOW] . . . [,PASS
,FAIL
,ABOVE
,WITHIN
,BELOW]

expression and the test result indicators are discussed in the **DISPLAY** statement.

The system contains 105 internal counters for each test station. These counters correspond to the 105 possible values of the **SORT** statement elements. There are 100 counters for the numeric values 00 through 99, and 5 counters for the test result indicators **PASS**, **FAIL**, **ABOVE**, **WITHIN**, and **BELOW**.

Each time a **SORT** statement is encountered in the test program, the counters corresponding to its elements are incremented by 1. These counters may be displayed or graphed on the terminal screen by using the **HIST** program (see the **Processing Data** manual).

To clear the **SORT** counters, reboot the system or use the **HIST** program **CLEAR** command or the **HSTSUB** subroutine **CLRHST**. Rebooting clears all the counters. **CLEAR** and **CLRHST** clear all or selected counters.

Example:

>11.3 SORT 50,PASS

Displays 50 in the readout display, lights the PASS indicator, and increments the internal counter for 50 and for PASS.

>27.8 SORT (x),FAIL,ABOVE

Displays the value of the expression (X) in the readout display, lights the FAIL and ABOVE indicators, and increments the internal counter for the value of (X), for FAIL, and for ABOVE.

SECTION 8: DATA LOGGING

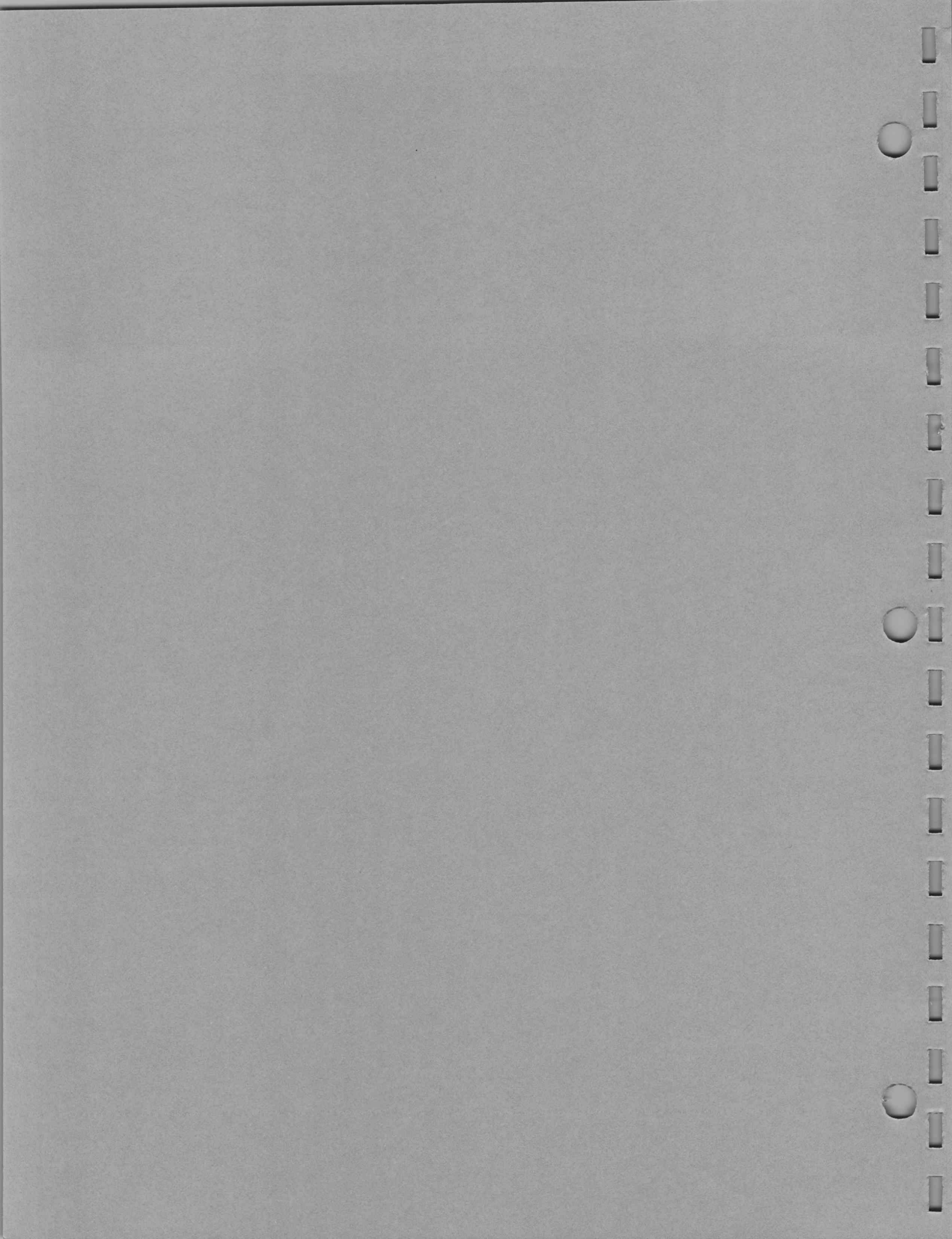
If the LOG* program has enabled data logging, the logging statements cause data to be logged into files while the test program is being executed. A test program may log the following types of data:

- The state of the sector-card error flip-flops.
- The state of the sector-card force flip-flops.
- The values of expressions and arrays evaluated during program execution.
- The DUT functional error data.

A log file may contain data from more than one DUT, from any test station, and from any test number. You may log markers to separate test data into data sets.

The Processing Data manual discusses the logging statements along with information on how to read the logged data. Therefore, this section supplies a summary of the logging statements and a description of the data logging subprograms READREG and SREAD.

*See the *Processing Data* manual.



LOGICAL UNIT NUMBERS

The data logging statements are based on logical unit numbers (luns). The system associates the luns with peripheral devices.

At bootup, the system assigns all foreground luns to the terminal. To assign or change the device or file with which a lun is associated, use the LOG program ASSIGN command. If the lun selected by a logging statement is not assigned to a device or file, then that statement does not log any data.

Format Conventions

The following conventions are used in the formats of the logging statements.

Element	Defined As
key	LOG statement identifier: any expression is legal. If omitted, the statement line number is used as the identifier.
lun	The logical unit number: any expression is legal. The value of the expression is rounded to the nearest integer. The range of lun is from 0 to 15.
pinlist	A pinlist name.

SUMMARY OF DATA LOGGING STATEMENTS

Statement	Purpose
LOGARRAY [<lun[,key]>] arrayname	Logs all the elements of the selected ordinary array.
LOGERRORS [<lun[,key]>] ON pinlist	Logs the state of the error flip-flop for each sector card selected.
LOGFORCE [<lun[,key]>] ON pinlist	Logs the state of the force flip-flop for each sector card selected.
LOGIARRAY [<lun[,key]>] iarrayname	Logs all the elements of the selected integer array.
LOGMARKER [<lun>] expression	Marks a logical subdivision of a data set. expression must evaluate in the range from 1 to 128.
LOGPARAMETRIC [<lun[,key]>] exp,...,exp	Logs the values of each numeric expression.
LOGREGISTER [<lun[,key]>] ([exp1,]exp2) ON pinlist [WITH mode] [AND CHAIN (num)]	Logs the contents of the specified shift registers. The sector-card operating mode and chaining specified by LOGREGISTER must be the same as those specified in the MOVE statement.

Connections

The chart below shows the connections made and broken with the LOGREGISTER, LOGERROR, and LOGFORCE statements. Sector card (n) refers to the sector card specified in the pinlist. Sector cards (n+1) and (n+2) are programmed by serial chaining. No cards are connected in the counterclockwise direction.

SECTOR CARD (n)		CONNECTIONS MADE (1) OR BROKEN (0)															
DESTINATION →		EBUS				REGISTER				COMPUTER				SECTOR			
SOURCE →		ERRNOT	ERROR	DATA	REGISTER	REGISTER	COMPUTER	EBUS OUT	EBUS IN	ERROR	CARD I. D.	REGISTER	FORCE	EXTERNAL	COMPUTER	PRAM	REGISTER
INSTRUCTION EXECUTED ↓																	
LOGREGISTER		0	0	1	0	1	0	0	0	0	0	1	0	1	0	0	0
	CHAIN (2)																
LOGERROR		1	0	0	0					1	0	0	0				
LOGFORCE		0	0	0	1					0	0	0	1				

SECTOR CARD (n + 1)		CONNECTIONS MADE (1) OR BROKEN (0)															
DESTINATION →		EBUS			REGISTER			COMPUTER			SECTOR						
SOURCE →		ERRNOT	ERROR	DATA	REGISTER	REGISTER	COMPUTER	EBUS OUT	EBUS IN	ERROR	CARD I. D.	REGISTER	FORCE	EXTERNAL	COMPUTER	PRAM	REGISTER
INSTRUCTION EXECUTED ↓																	
LOGREGISTER	CHAIN (2)	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0
LOGERROR																	
LOGFORCE																	

SECTOR CARD (n + 2)		CONNECTIONS MADE (1) OR BROKEN (0)															
DESTINATION →		EBUS			REGISTER			COMPUTER			SECTOR						
SOURCE →		ERRNOT	ERROR	DATA	REGISTER	REGISTER	COMPUTER	EBUS OUT	EBUS IN	ERROR	CARD I. D.	REGISTER	FORCE	EXTERNAL	COMPUTER	PRAM	REGISTER
INSTRUCTION EXECUTED ↓																	
LOGREGISTER	CHAIN (2)	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0
LOGERROR																	
LOGFORCE																	

READING SAVED ERROR INFORMATION

The **READREG** function reads one cell of data from the specified shift register. **READREG** reads saved error information and clears the error flip-flops. The value returned by **READREG** is:

- one if the cell is set and
- zero if the cell is clear.

The **READREG** function call is:

READREG(n,pin1)

n specifies the cell in the shift register to be read. **n** must be from 1 to 4104. If **n** is outside of this range, **READREG** returns error message C2.

The function declaration is:

FUNCTION READREG(V,T1):SECDAT

NOTE

To read the data, READREG recirculates the shift register. Therefore, if the data was saved while the shift register was serially chained, then using READREG breaks the chain. READREG clears the error flip-flops.

READING PATTERN DATA

The **SREAD** subroutine reads pattern data from the sector-card shift registers into a bit array. The **TEK-TEST** statements and the subprograms in the **BARRAY** file* can then access and modify the pattern data.

The **SREAD** subroutine call is:

```
SREAD(pinlist,n1,n2,srloc,num,barray,mode)
```

The subroutine declaration is:

```
SUBROUTINE SREAD(P,V,V,V,V,I,V):SREAD
```

pinlist specifies a previously defined pinlist. **n1** and **n2** are subscripts to the pinlist. **pinlist**, **n1**, and **n2** define the group of sector cards from which the data is read. If both **n1** and **n2** are 0, then **SREAD** reads data from all the sector cards associated with **pinlist**.

srloc selects the first shift register location to be read. **num** specifies the number of DUT cycles to be read. Both **srloc** and **num** must have values in the range from 1 to 4104.

barray specifies the bit array where the data is stored. This array must have been previously dimensioned with the **BARRAY** subroutine. The value of the first dimension of the bit array (**x**) must be greater than or equal to **mode**. The value of the second dimension of the bit array (**y**) must be greater than or equal to the number of sector cards read. The value of the third dimension of the bit array (**z**) must be equal to or greater than the number of DUT cycles read (**num**).

mode specifies the clock pulse mode used in the **LOAD** and **MOVE** statements when the data was stored. **mode** must be 1, 2, or 4.

Clock Pulse Mode	Sector Card Mode
1	1 (F,I,C,M)
2	2 and 3 (FC and FI,CM)
4	4 (FICM)

*See the *General Purpose Processing Data Subprograms* manual.

mode multiplied by **num** equals the number of bits read from each shift register specified.

SREAD can produce three error messages:

- 85 – Illegal mode.
- 86 – Illegal shift register subscript.
- 87 – Pinlist error caused by illegal subscript.

Example:

```

>2.01 PINLIST INS = I1,I2,I3,I4,I5,I6,I7,I8/
>2.02 I9,I10,I11,I12,I13,I14,I15,I16/
>2.03 I17,I18,I19,I20,I21,I22,I23,I24

>3.01 SUBROUTINE BARRAY(I,V,V,V):BARRAY
>3.02 SUBROUTINE SREAD(P,V,V,V,V,I,V):SREAD

>4.01 IARRAY A(35)
>4.02 BARRAY(A,2,16,16)

>5.01 LOAD CORE PATT TO INS WITH FI

>6.01 SREAD(INS,3,18,20,35,A,2)
  
```

SREAD reads data from the sector cards for pins I3 through I18 for DUT cycles 20 through 35. SREAD stores the data in bit array A. Since the LOAD statement loads the shift registers with a clock mode 2 pattern, the mode argument of SREAD is 2. That is, SREAD selects clock mode 2, 16 pins, and 16 DUT cycles. Note that the dimensions of A are (2,16,16). Therefore, the bit array is large enough to contain all of the specified shift-register data.

BARRAY Location (x,y,z)	Data Stored		
	Type	Sector	DUT Cycle
(1,1,1)	FORCE	I3	20
(2,1,1)	INHIBIT	I3	20
(1,2,1)	FORCE	I4	20
(1,1,2)	FORCE	I3	21

SREAD clears the error flip-flops.

Example:

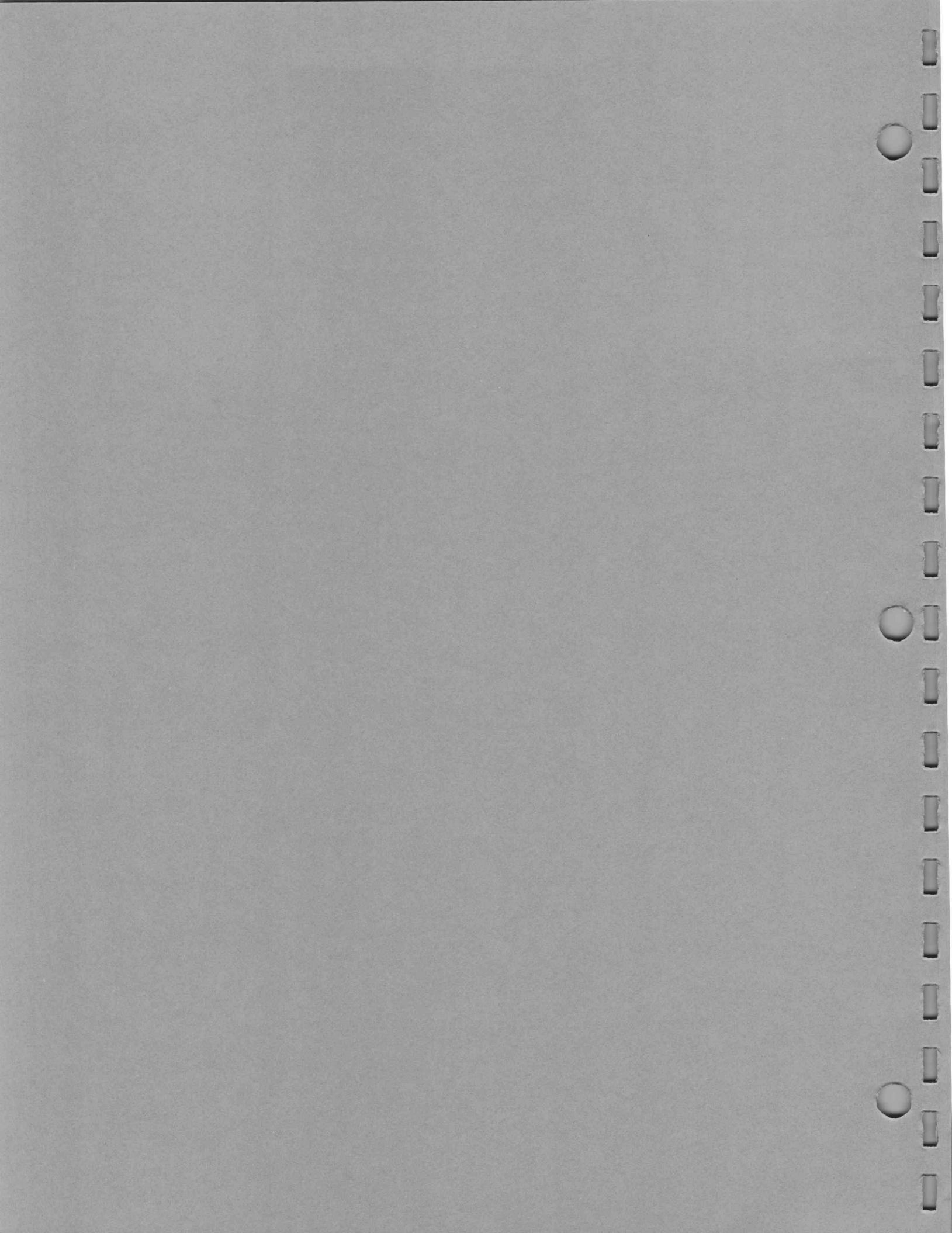
- >2.01 PINLIST INS = I1,I2,I3,I4,I5,I6,I7,I8
- >3.01 SUBROUTINE BARRAY(I,V,V,V):BARRAY
- >3.02 SUBROUTINE SREAD(P,V,V,V,V,I,V):SREAD
- >4.01 IARRAY A(35)
- >4.02 BARRAY(A,4,8,16)
- >5.01 LOAD CORE PATT TO INS WITH FICM
- >6.01 SREAD(INS,0,0,1,16,A,4)

SREAD reads data from the shift registers of all sector cards associated with the pins in INS for DUT cycles 1 through 16. SREAD stores the data in bit array A. Since the LOAD statement loads the shift registers with a clock mode 4 pattern, the mode argument of SREAD is 4.

BARRAY Location (x,y,z)	Data Stored		
	Type	Sector	DUT Cycle
(1,1,1)	FORCE	I1	1
(2,1,1)	INHIBIT	I1	1
(3,1,1)	COMPARE	I1	1
(4,1,1)	MASK	I1	1

APPENDIX A:

RECIRCULATION AND PARALLEL CHAINING



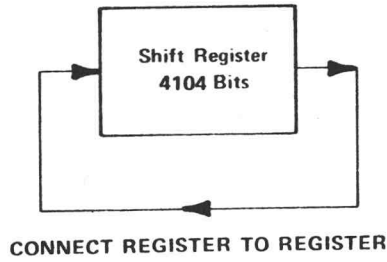
APPENDIX A: RECIRCULATION AND PARALLEL CHAINING

Recirculation

Data in any individual register or group of registers may be recirculated for repeated use without reloading from core or disk (see Figure A-1). Data sequences may thus be stored once and used repeatedly throughout the test.

Recirculation is compatible with all operating modes and with parallel and serial chaining. Recirculation of one register involving data sequences longer than 4104 bits per pin must be partitioned into 4104 (or less) bit groupings. Each group is used repeatedly by recirculation, followed by reloading of the register, repeated use of the next group, and so on. Serial chains may only be recirculated by strapping at the socket card, the driver at one end of the chain to the comparator at the other end of the chain.

Recirculation for One Register



Recirculation for Multiple Registers

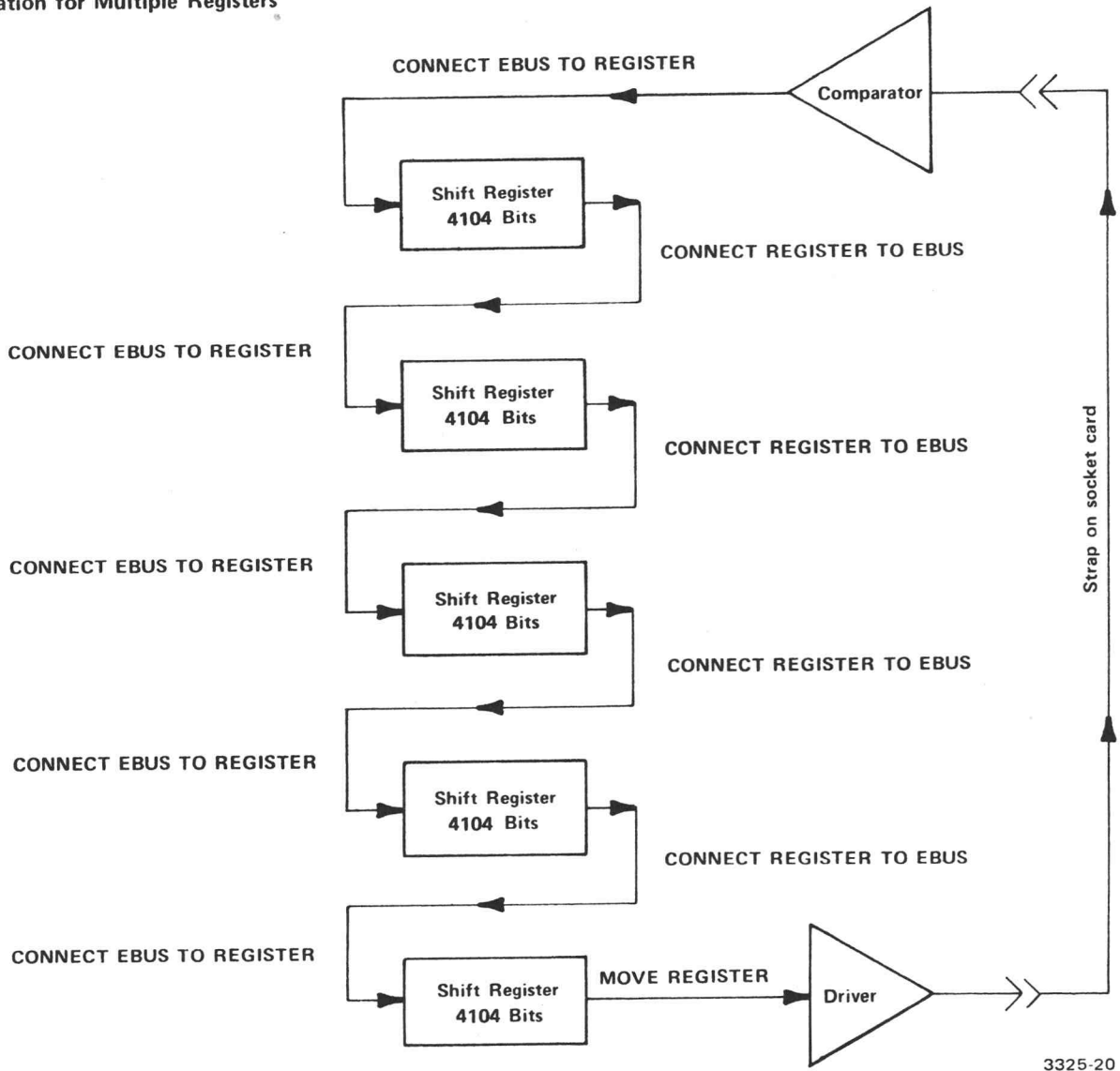


Figure A-1. CONNECT Statements for Recirculation

Parallel Chaining

Parallel chaining is that functional test mode where a pin card at sector N-1 provides Inhibit and/or Mask pattern data for its neighboring pin card at N. The neighboring pin card at N can pass this Inhibit/Mask pattern through itself for use on the pin card at N+1. Each pin card can, in turn, do the same so that pattern from N-1 can provide Inhibit/Mask data to a chain of adjacent pin cards for testing devices with I/O buses. Each pin card through which the Inhibit/Mask data passes delays that data by 10 ns. One card at N-1 can be chained to one card at N to test one I/O bus pin at cycle times down to 48 ns. So that one more I/O bus pin can be tested, 10 ns must be added to the minimum cycle time for each pin card which is added to the chain. When using Pram, you cannot chain more than 8 sector cards.

There are no statements which specifically and only program the parallel chain pattern paths. The D70 Card assumes that, if its pattern is not being used to Force or Compare the DUT, the pattern should be connected to the adjacent card via the Dbus. The D70 Card assumes that, if the pattern is being used to Force or Compare the DUT, the pattern is not useful on the Dbus; so the Dbus-in-to-Dbus-out connection is made. These assumptions mean that in most cases the parallel chain will be established correctly without any specific statements.

1.11 FORCE pins WITH PATTERN

1.12 COMPARE pins WITH PATTERN

Either of these statements cause the Dbus in to be connected to Dbus out. It is not possible to use the PATTERN in this way and also connect it to the Dbus out.

3.09 INITIALIZE

This statement causes the sector pattern source to be connected to the Dbus out. This is also the condition which exists at the beginning of a test.

2.11 FORCE pins WITH $\left\{ \begin{array}{l} \text{ONE} \\ \text{ZERO} \\ \text{TOGGLE} \end{array} \right.$

2.12 COMPARE pins WITH $\left\{ \begin{array}{l} \text{ONE} \\ \text{ZERO} \\ \text{TOGGLE} \end{array} \right.$

Both of these statements or initial conditions are required to cause the sector pattern source to be connected to the Dbus.

3.12 MASK pins WITH ONE

These statements may be written following the **FORCE** and **COMPARE** statement if needed to prevent the Driver from forcing or the Comparator from generating functional errors.

MASK Patterns

The D70 card uses **MASK** pattern bits in an inverse manner when compared with D1B pin cards. This inverted **MASK** pattern logic first appeared on the D1B/E pin card. The inversion was done so that a single parallel chained bit stream could control both the Inhibit and Mask functions. The pattern bits for Mask and Inhibit have the following effect:

Pattern Bit	
1	Compare and/or Inhibit driver
0	Mask Errors and/or Enable driver

Alternate Data

The D70 pin card has a feature which allows test pattern generators such as the PRAM to select, at clock rate, one of two pattern sources. The signal which can cause a pin card to use the alternate source of pattern data is connected in common to all pin cards. Each pin card can be programmed to obey or ignore the switch pattern signal. The choice of sector normal data source is done independently for each pin card. The alternate data source is determined by the choice of primary source.

7.22 CONNECT TO ALTERNATE ON pins

This statement causes the named pin card to obey the alternate data signal.

7.33 INITIALIZE

7.34 DISCONNECT FROM ALTERNATE ON pins

Either of these statements cause the pin cards to ignore the alternate data signal.

8.22 CONNECT source TO SECTOR ON pins

This statement chooses the normal data source.

8.33 MOVE source TO pinlist

This statement also chooses a normal data source.

Normal	Alternate
REGISTER	PRAM
EXTERNAL	REGISTER
PRAM	REGISTER or EXTERNAL, determined by strap option
COMPUTER { CORE DISK }	REGISTER

The statement which causes the PRAM to send the alternate data signal to the pin cards is **USE ALTERNATE PATTERN**. Conversely, **USE NORMAL PATTERN**, the default condition, tells the pin cards to use the normal source.

When used with the PRAM, the statement below is also valid.

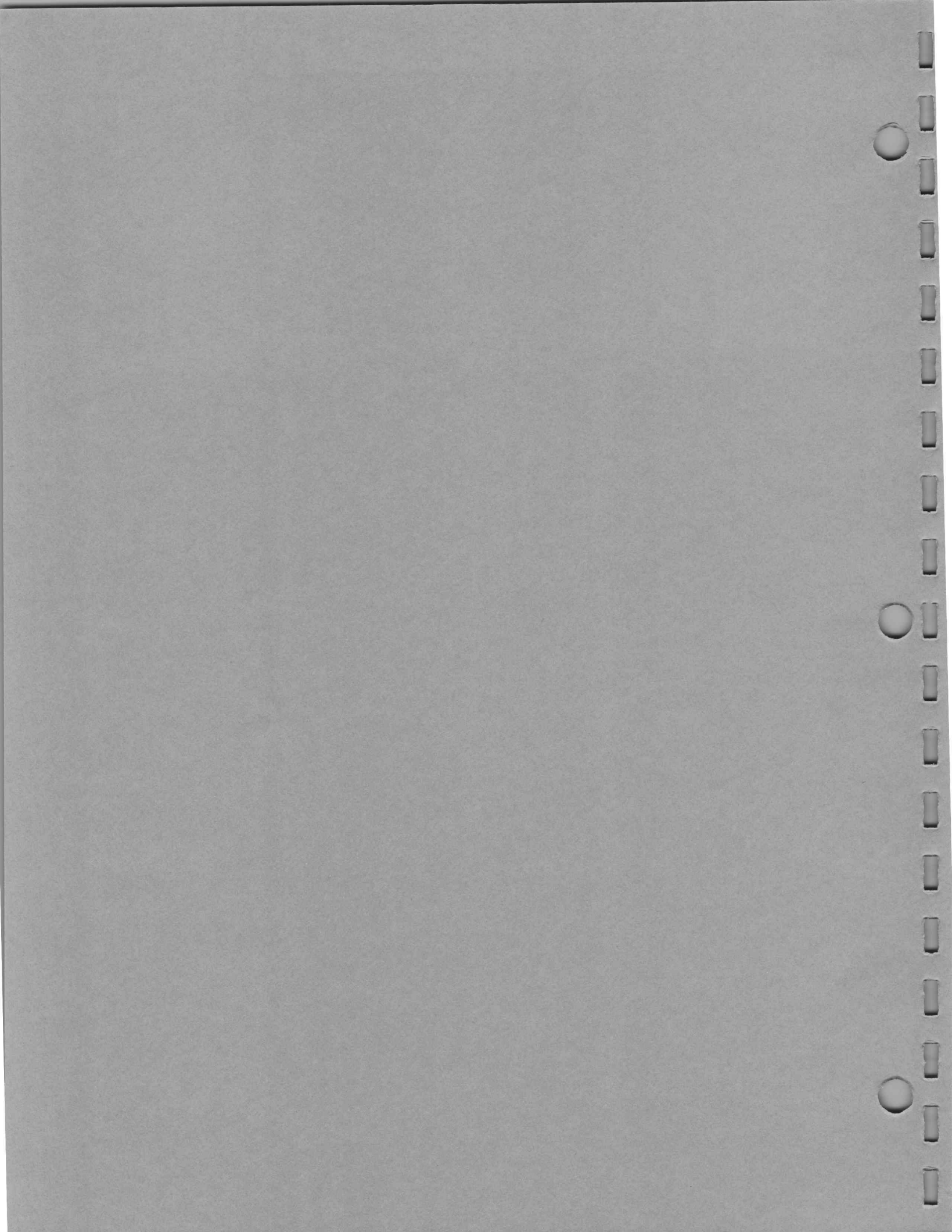
{ FORCE
COMPARE
MASK } pins WITH ALTERNATE PATTERN OR { ZERO
ONE }

PATTERN is then the normal source, and **ZERO/ONE** is the alternate source. **PATTERN** is defined by **CONNECT source TO SECTOR ON pins** and **MOVE source TO pins** statements. This **ALTERNATE** mode supersedes the one which is programmed by **CONNECT TO ALTERNATE ON pin** statements.



APPENDIX B:

**TRANSLATOR ERROR MESSAGES
FOR TEKTEST II/III
HARDWARE-ORIENTED STATEMENTS**

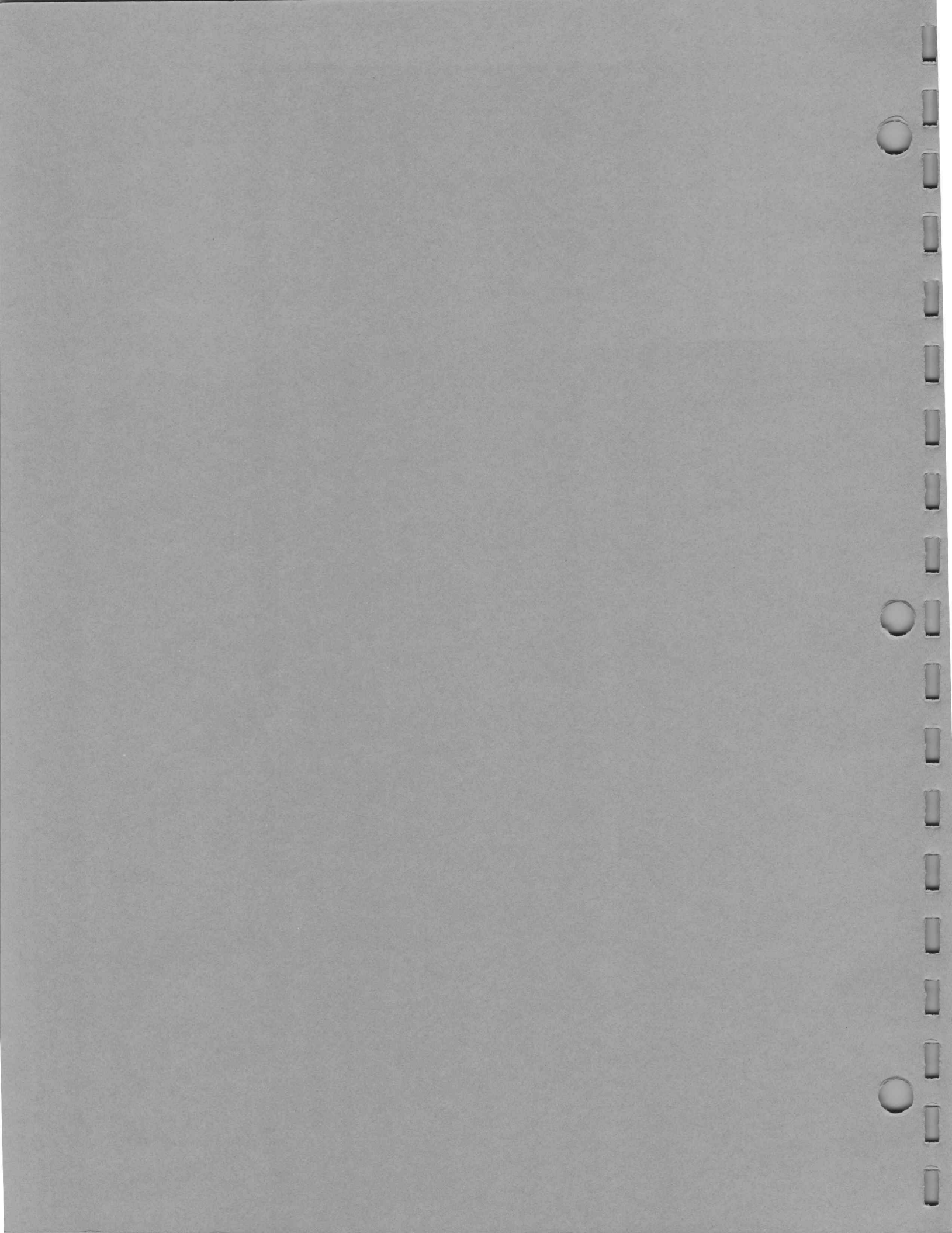


APPENDIX B: TRANSLATOR ERROR MESSAGES FOR TEKTEST II/III HARDWARE- ORIENTED STATEMENTS

EXTRA OPERATOR
FROM VS3 OR VS4
FROM VS3 VS4 OR DRIVER
ILLEGAL ARGUMENT TYPE
ILLEGAL CALL BY NAME ARGUMENT
ILLEGAL EQUAL SIGN
ILLEGAL IDENTIFIER
ILLEGAL ITEM
ILLEGAL LINE NUMBER
ILLEGAL LUN SPECIFICATION
ILLEGAL NUMBER
ILLEGAL RANGE OR LIMIT
ILLEGAL START OR DURATION TIME
ILLEGAL STATEMENT
ILLEGAL SUBROUTINE CALL
ILLEGAL SUBSCRIPT VALUE
ILLEGAL SYNTAX FOR THIS SYSTEM
ILLEGAL USE OF PARALLEL CHAINING
ILLEGAL VOLTAGE
ILLOGICAL CHAIN OR SAVE ERRORS OPTION
ITEM ALREADY DEFINED
I TYPE PIN IS ILLEGAL
MISSING COMMA
MISSING ENDING DELIMITER
MISSING EQUAL SIGN
MISSING LEFT PARENTHESIS
MISSING OPERAND
MISSING OPERATOR
MISSING PIN OR PINLIST
MISSING RIGHT PARENTHESIS
MISSING SUBSCRIPT
MIXED PIN TYPES IN PINLIST
MUST BE 5V OR 30V
NOT AN ARRAY
NOT AN INTEGER ARRAY NAME
NOT A PIN
NOT A PIN OR PINLIST
NOT LEGAL AS THE LAST STATEMENT OF A LOOP
ONLY COLON P IS LEGAL
ONLY PINLIST IS LEGAL
ONLY SIMPLE CONSTANT LEGAL
PIN OR SINGLE INDEXED PINLIST ONLY
TOO MANY PARAMETERS
TRIGGER CLAUSE OMITTED



APPENDIX C:
STATEMENT SUMMARY



APPENDIX C: STATEMENT SUMMARY

The following is an alphabetic summary of the TEKTEST II/III hardware-oriented statements, reserved system variables, and subprograms. Refer to the text for complete formats of the complex statements.

FORMAT	PURPOSE
ADVANCE	Provides user control of testing*
AUTOV	Initiates a voltage measurement*
BURST { ON OFF }	Determines the clock generator mode
CLEAR ERROR	Disables the WHEN ERROR and WHEN PASS statements and clears the error flip-flops
CLOCK	Reads a real-time counter*
COMPARE pin WITH sourcedata	Checks the level of DUT output pins against expected levels
CONNECT source TO destination ON pin	Connects the 1804 pin electronics card data paths

*Indicates a reserved system variable.

CONNECT TO { PHASE
DATAPHASE } ON pins[;ND]

Connects the pin electronics cards to a clock phase

CURRENT

Initiates a current measurement*

CYCLE=n

Establishes the period in which data is sent to and received from the DUT

DATAPHASE=starttime FOR width

Programs the clock generator data strobe

DISCONNECT source FROM destination ON pin[;ND]

Disconnects the pin electronics card data paths

DISPLAY expression[,indicator] ... [,indicator]

Displays test results on the TSCU

DRIVE(pin1)

Reads the pin electronics card force flip-flops†

ENVIRONMENT IS { 1803
1804
1805
1843
BACKGROUND }

Selects the test station environment

ERROR

Indicates the status of functional testing*

* Indicates a reserved system variable.

† Indicates a subprogram (a function or subroutine).

FORCE pin WITH sourcedata	$\left[\begin{array}{l} ;RZ[,INVERT] \\ ;RC \end{array} \right]$	Selects the source data forced between specific logic levels on DUT input pins
HICOMPARE=starttime FOR width		Sets phase times
HICOMPARE=voltagevalue [AT range] ONpins		Sets compare voltages
HIDRIVE=voltagevalue ON pins		Sets force voltages
IDENTIFY(pin1)		Reads the identification of the specified pin electronics card [†]
INDEX		Prints to the shift-register pattern rows presently available at the DUT [*]
INDEXP expression		Renumbers the rows in the shift registers
INHIBIT pin WITH sourcedata		Disconnects the driver associated with DUT input pins
INITIALIZE		Initializes the hardware
ISn=currentvalue AT voltage limit		Sets the 1140A current supply levels and voltage limits

*Indicates a reserved system variable.

[†]Indicates a subprogram (a function or subroutine).

LOAD { CORE
DISK } filnam [(expression1,) expression2] TO pinlist [WITH mode]
[AND option AND option]

Transfers blocks of data from a source file to the pin electronics card shift registers

LOCOMPARE=starttime FOR width

Sets phase times

LOCOMPARE=voltagevalue [AT range] ON pins

Sets compare voltages

LODRIVE=voltagevalue ON pins

Sets force voltages

LOGARRAY[<lun[,key]>] arrayname

Logs all the elements of the selected ordinary array

LOGERRORS[<lun[,key]>] ON pinlist

Logs the state of the error flip-flop for each pin electronics card selected

LOGFORCE[<lun[,key]>] ON pinlist

Logs the state of the force flip-flop for each pin electronics card selected

LOGIARRAY[<lun[,key]>] iarrayname

Logs all the elements of the selected integer array

LOGMARKER[<lun>] expression

Marks a logical subdivision of a data set. **expression** must evaluate in the range from 1 to 128.

LOGPARAMETRIC[<lun[,key]>] exp, ...,exp

Logs the values of each numeric expression

LOGREGISTER[<lun[,key]>] ([exp1,]exp2)
ON pinlist [**WITH** mode][**AND CHAIN** (num)]

Logs the contents of the specified shift registers. The pin electronics card operating mode and chaining specified by LOGREGISTER must be the same as those specified in the MOVE statement.

MASK pin **WITH** sourcedata

Disconnects the comparator output associated with DUT output pins

MOVE {
REGISTER
CORE filnam
DISK filnam
} [[expression1,]expression2] **TO** pinlist [**WITH** mode]

[**AND** option ... **AND** option]

Moves the pattern to and from the DUT

PHASE n=starttime **FOR** width

Programs the clock phases

PINERR(pins)

Reads the sector-card error flip-flops[†]

{
PINLIST
LET
} name {
=
BE
} {
pinname
pinlist
pinlist (num)
pinlist (num1,num2)
X
} ,..., {
pinname
pinlist
pinlist (num)
pinlist (num1,num2)
X
}

Groups under one name individual pins defined in the pin assignment table

PINPAS(pins)

Reads the pin electronics card error flip-flops[†]

[†]Indicates a subprogram (a function or subroutine).

READREG(n,pin1)	Reads saved error information and clears the error flip-flops [†]
SETUP FOR measurement ON pin FROM source	Sets the DC and ΔT Subsystems in the test station
SORT expression [,indicator] ... [,indicator]	Displays test results on the TSCU and increments the internal counters
SREAD(pinlist,n1,n2,srloc,num,barray,mode)	Reads pattern data from the pin electronics card shift registers into a bit array [†]
STATN	Identifies the active test station [†]
TIME	Initiates a time measurement*
TIMOUT	Returns the current state of the TIMEOUT system flag [†]
TRIGGER exp[,REPEAT]	Causes a 2943 clock generator trigger pulse at any DUT cycle during a clock sequence
TRIGGER2 exp[,REPEAT]	Causes a 2944 clock generator trigger pulse at any DUT cycle during a clock sequence
TSTRIG	Causes a 1 μs pulse to occur at the rear panel of the test station
UNDERSOCKET [#] data	Routes data via the undersocket card
UNSET FOR measurement ON pin	Unsets the DC and ΔT Subsystems in the test station

* Indicates a reserved system variable.

[†] Indicates a subprogram (a function or subroutine).

VOLTAGE

Initiates a voltage measurement*

VS_n=voltagevalue AT currentlimit

Sets the 1140A voltage levels and current limits

WHEN ERROR linenumber

Tests for an error condition and clears the error flip-flops

WHEN PASS linenumber

Tests for a pass condition and clears the error flip-flops

*Indicates a reserved system variable.

