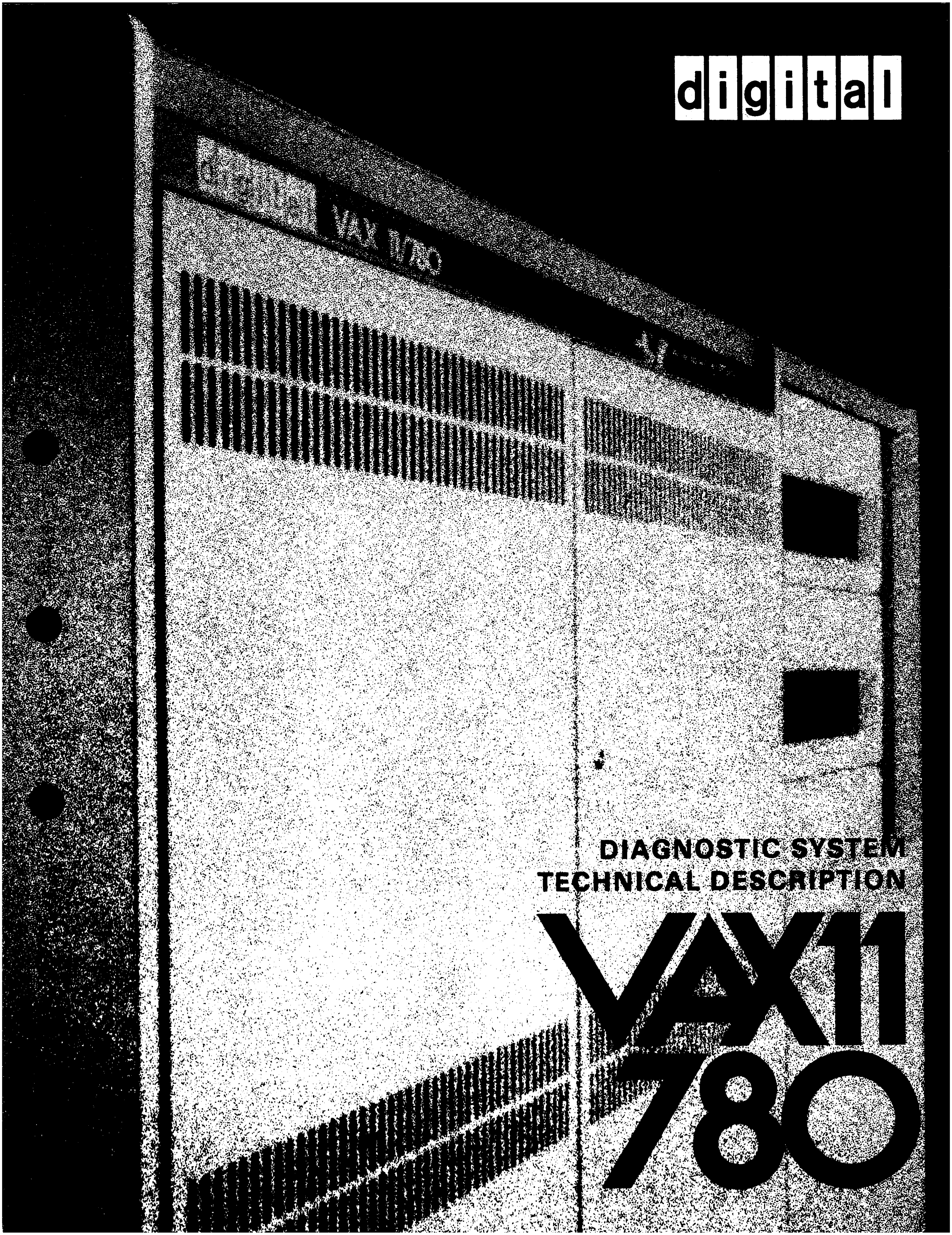
digital



DIAGNOSTIC SYSTEM TECHNICAL DESCRIPTION

VAX11 780

EK-DS780-TD-001

VAX-11/780 Diagnostic System Technical Description

digital equipment corporation • maynard, massachusetts

First Edition, February 1979

The material in this manual is for informational purposes and is subject to change without notice. Digital Equipment Corporation assumes no responsibility for any errors which may appear in this manual.

Printed in U.S.A.

8/84-15

CONTENTS

Page

CONTENTS (Cont)

CONTENTS (Cont)

Page

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

FIGURES

FIGURES (Cont)

TABLES

TABLES (Cont)

EXAMPLES

EXAMPLES (Cont)

CHAPTER 1
INTRODUCTION

1.1 MANUAL SCOPE

This manual provides a comprehensive description of the functional and operational characteristics of the VAX-11/78Ø diagnostic system. The level of detail presented provides a resource for appropriate branch courses of the field service training program and for a field reference. Table 1-1 provides a list of related documents. Note that a glossary of diagnostic software terms is provided in Appendix A.

Table 1-1 VAX-11/78Ø System Manuals

Document Title	Control Number	Form
VAX-11/78Ø Power System Technical Description	EK-PS78Ø-TD-ØØ1	In Microfiche Library
VAX-11/78Ø System Installation Manual	EK-SI78Ø-IN-ØØ1	Available in hard copy*
DS78Ø Diagnostic System User's Guide	EK-DS78Ø-UG-ØØ1	Available in hard copy*
DS78Ø Diagnostic System Technical Description	EK-DS78Ø-TD-ØØ1	In Microfiche Library
FP78Ø Floating-point Processor Technical Description	EK-FP78Ø-TD-ØØ1	In Microfiche Library
REPØ5/REPØ6 Subsystem Technical Description	EK-REPØ6-TD-ØØ1	In Microfiche Library
VAX-11 KA78Ø Central Processor Technical Description	EK-MS78Ø-TD-ØØ1	In Microfiche Library
VAX-11 MS78Ø Memory System Technical Description	EK-MS78Ø-TD-ØØ1	In Microfiche Library
DW78Ø Unibus Adapter Technical Description	EK-DW78Ø-TD-ØØ1	In Microfiche Library
KC78Ø Console Interface Technical Description	EK-KC78Ø-TD-ØØ1	In Microfiche Library
VAX-11/78Ø Architecture Handbook	EBØ7466	Available in hard copy*

Table 1-1 VAX-11/780 System Manuals (Cont)

Document Title	Control Number	Form
VAX-11/780 Software Handbook	EB08126	Available in hard copy*
VAX-11/780 Hardware Handbook	EB09987	Available in hard copy*
VAX/VMS Primer	AA-D030A-TE	Available in hard copy*
VAX/VMS Command Language User's Guide	AA-D023A-TE	Available in hard copy*
VAX-11 MACRO User's Guide	AA-D033A-TE	Available in hard copy*
VAX-11 Linker Reference Manual	AA-D019A-TE	Available in hard copy*
VAX-11 Symbolic Debugger Reference Manual	AA-D026A-TE	Available in hard copy*

*These documents can be ordered from:

Digital Equipment Corporation
444 Whitney Street
Northboro, MA 01532
Attn: Printing and Circulation Services (NR2/M15)
Customer Services Section

For information concerning microfiche libraries, contact:

Digital Equipment Corporation
Micropublishing Group, PK3-2/T12
129 Parker Street
Maynard, MA 01754

1.2 DIAGNOSTIC SYSTEM CAPABILITIES

The VAX-11/780 diagnostic system is a set of software components integrated as a system to provide a wide range of error detection and isolation capabilities for the VAX-11/780 hardware. The diagnostic levels range from system functional tests to dedicated microprogram techniques capable of identifying a faulty module (printed circuit board) or group of modules. In addition, the diagnostic control functions provide substantial selection and execution options.

The overall diagnostic strategy satisfies the major field service goals of:

a. High quality and efficiency of system installation, by providing formal installation procedures, automated test package configurations, and a system exerciser program that can be configured for specific VAX-11/78Ø systems.

b. Reduction of fault isolation and repair times, by providing high visibility diagnostic programs (programs accessible to the operator) and procedures keyed to the field service troubleshooting and repair philosophy.

The diagnostic system is supported by a PDP-11/VØ3 (LSI-11) microcomputer console system. In addition to providing for local (on-site) diagnostic execution, the diagnostic system allows for diagnosis from a remote diagnostic center.

1.3 DIAGNOSTIC SYSTEM OVERVIEW

The diagnostic system consists of programs that are organized hierarchically (from general to specific capabilities) in six levels. Each level contains one or more categories, as follows:

Level 1 -- Operating system (VMS) based diagnostic programs (using queue I/O)

^ System exerciser program

Level 2 -- Diagnostic supervisor--based diagnostic programs that can be run either under VMS or in the standalone mode (using queue I/O)

^ Bus interaction program
^ Formatter and reliability level peripheral diagnostic programs

Level 2R -- Diagnostic supervisor--based diagnostic programs that can be run only under VMS.

^ Certain peripheral diagnostic programs

Level 3 -- Diagnostic supervisor--based diagnostic programs that can be run in standalone mode only (using direct I/O)

^ Functional level peripheral diagnostic programs
^ Repair level peripheral diagnostic programs
^ Cluster diagnostic programs

Level 4 -- Standalone macrodiagnostic programs that run without the supervisor.

^ Hardcore instruction test

Console Level-- Console-based diagnostics that can be run in the standalone mode only

- Microdiagnostics
- Console program
- Octal Debugging Technique (ODT)
- ROM resident power-up tests
- LSI-11 diagnostics

The diagnostic programs can be used for preventive maintenance checks to ensure proper computer operation; or, if system malfunctions have been detected, specific programs or groups of programs can be run to isolate the fault.

Figure 1-1 shows the relation of the six levels to four diagnostic program operating environments. The console environment requires exclusive use of the VAX-11/780 system (standalone mode). It includes only the console level programs. In this environment, program control is exercised by the LSI-11 processor in the console subsystem.

In the cluster environment, the system environment, and the user environment, control is exercised in the VAX-11/780 CPU. The cluster environment supports only standalone diagnostic programs. It includes level 4 programs and some level 3 programs. The level 3 programs supported are those that test the CPU and the channel adapters.

The system environment supports peripheral diagnostic programs that can run in the standalone mode. These include level 2 programs and level 3 peripheral programs.

The user environment supports only programs that can be run under VMS, namely levels 2R, 2, and 1.

In general, the diagnostic system uses a building block approach to testing (and subsequent fault detection and isolation). When the diagnostic programs are executed in the standard system checkout sequence, they will initially test a minimum (basic) set of logical functions to ensure their proper operation. After these basic operations are verified, a larger and more complex block is tested, using the previously tested block as a base. This sequence is implemented consistently from the ROM resident power-up tests (which check the console) to interactive system tests executed as user mode tasks under the VMS operating system, as shown in Figure 1-2.

It may be that a diagnostic program will indicate an error in a hardware component which is more easily diagnosed by another program. For instance, the bus interaction program may indicate a failure of a tape drive. The tape reliability program may also detect the same failure or a related failure, but the problem may

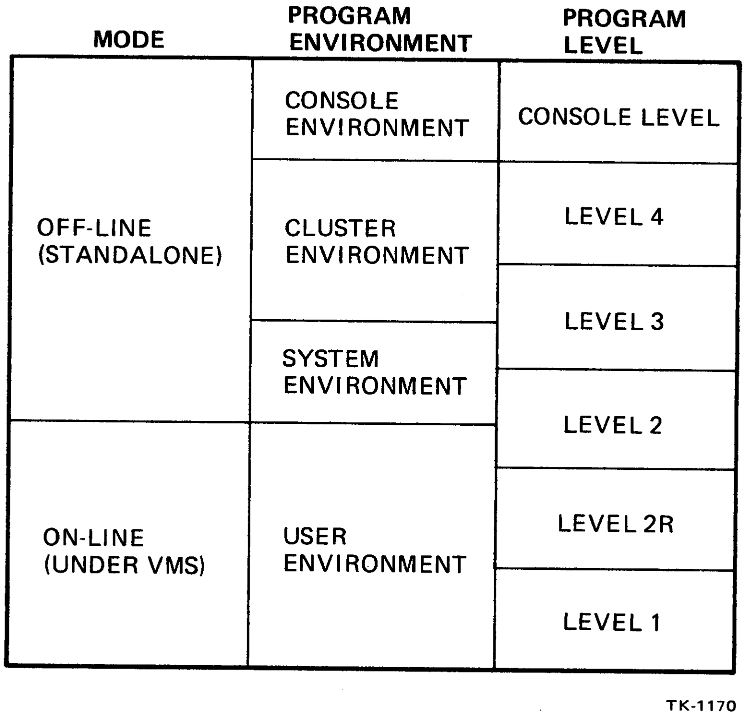


Figure 1-1 Diagnostic Program Mode, Environment and Levels

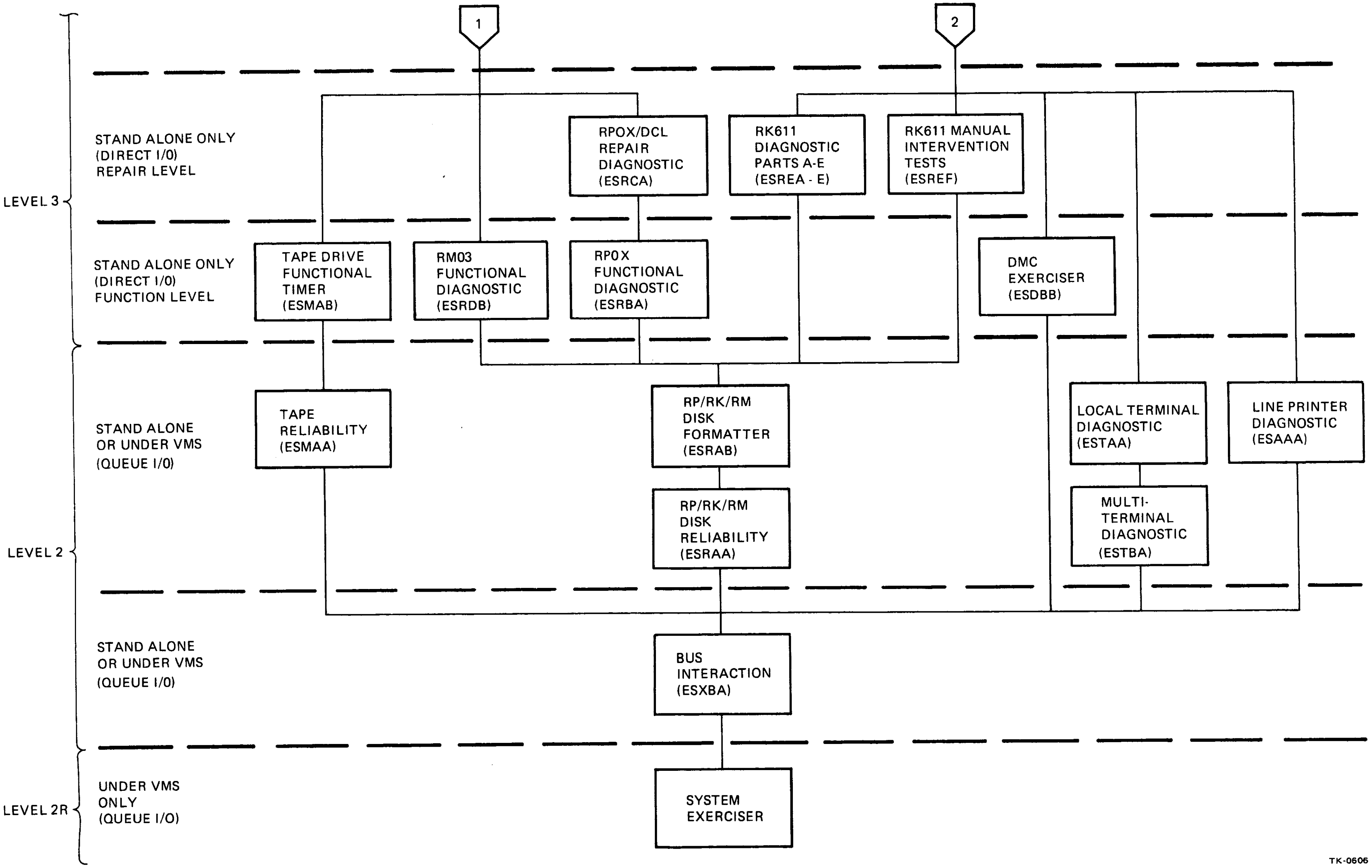

1
2
LEVEL 3
STAND ALONE ONLY (DIRECT I/O) REPAIR LEVEL
STAND ALONE ONLY (DIRECT I/O) FUNCTION LEVEL
LEVEL 2
STAND ALONE OR UNDER VMS (QUEUE I/O)
STAND ALONE OR UNDER VMS (QUEUE I/O)
LEVEL 2R
UNDER VMS ONLY (QUEUE I/O)
RPOX/DCL REPAIR DIAGNOSTIC (ESRCA)
RK611 DIAGNOSTIC PARTS A-E (ESREA - E)
RK611 MANUAL INTERVENTION TESTS (ESREF)
TAPE DRIVE FUNCTIONAL TIMER (ESMAB)
RM03 FUNCTIONAL DIAGNOSTIC (ESRDB)
RPOX FUNCTIONAL DIAGNOSTIC (ESRBA)
DMC EXERCISER (ESDBB)
TAPE RELIABILITY (ESMAA)
RP/RK/RM DISK FORMATTER (ESRAB)
LOCAL TERMINAL DIAGNOSTIC (ESTAA)
LINE PRINTER DIAGNOSTIC (ESAAA)
RP/RK/RM DISK RELIABILITY (ESRAA)
MULTI-TERMINAL DIAGNOSTIC (ESTBA)
BUS INTERACTION (ESXBA)
SYSTEM EXERCISER
TK-0606

Figure 1-2 VAX-11/78Ø Diagnostic System Program Hierarchy (Sheet 1 of 2)

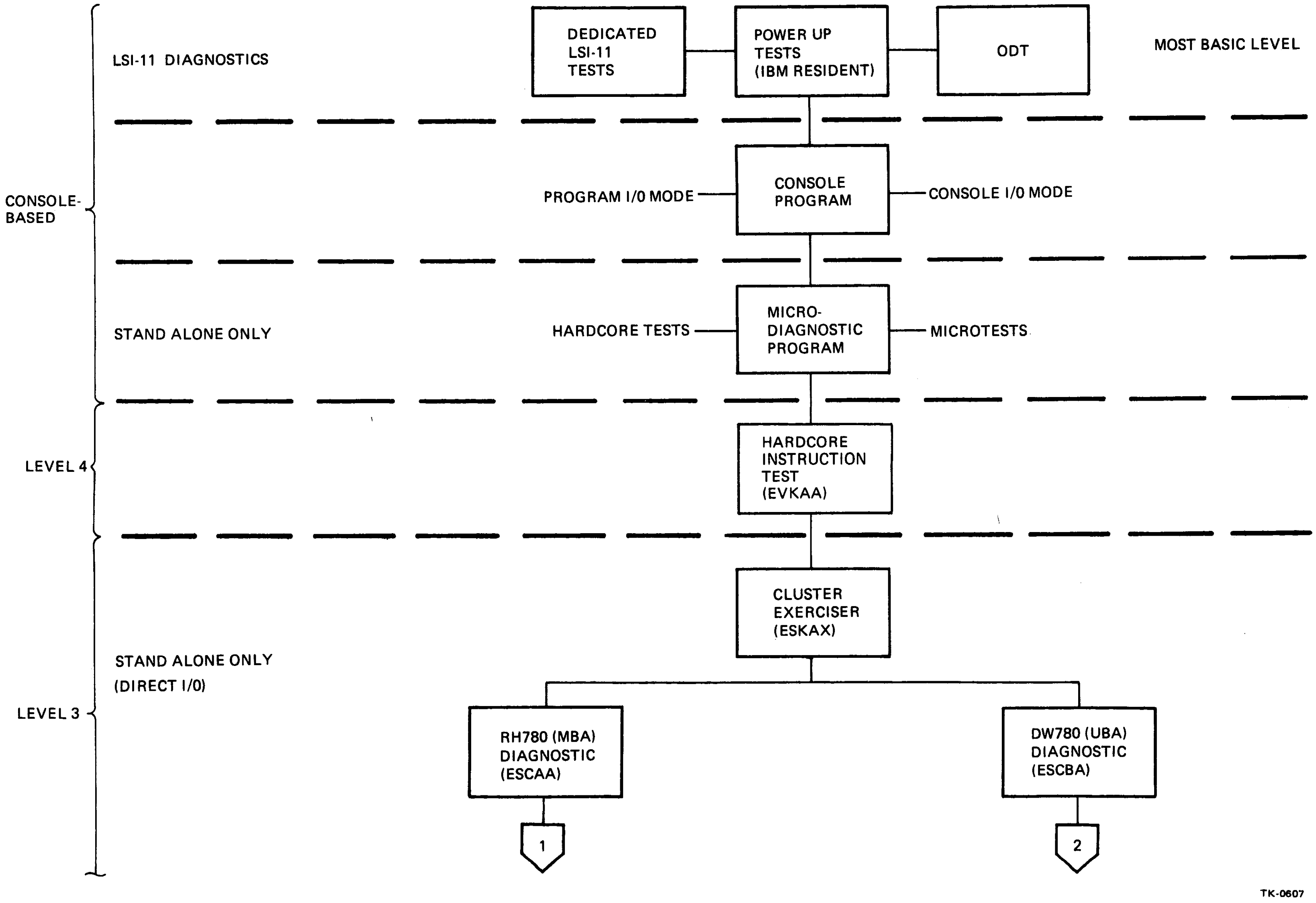


Figure 1-2 VAX-11/780 Diagnostic System Program Hierarchy (Sheet 2 of 2)

be on the tape drive controller, the RH780 (MBA), or the KA780 (CPU). Proper use of the six levels of diagnostic programs should enable the field service engineer to identify the failure quickly and accurately.

1.4 DIAGNOSTIC SYSTEM EXECUTION ENVIRONMENTS

Most of the diagnostic programs must be run off-line (standalone). In other words, they require exclusive use of the VAX-11/780 computer system and will not run under the VMS operating system. Diagnostic programs in levels 3, 4, and the console level are of this type (Figure 1-2). The diagnostic programs in level 2 can be run off-line or on-line (under VMS).

- Off-line diagnostics must be run from the console terminal.

- On-line diagnostics may be run from any terminal on the system and will share the computer system with other user mode programs. Figure 1-3 shows the execution environments required by the various diagnostic programs.

1.5 CONSOLE DIAGNOSTICS

On power up, a set of ROM resident tests verifies the proper functioning of the LSI-11 within the console subsystem before the console program is booted from the floppy disk. If the console program cannot be booted, the ROM resident tests, together with ODT, can be used to isolate the fault. For details see the VAX-11/780 Diagnostic System User's Guide (EK-DS780-UG-001), Appendix D. In addition, a set of dedicated LSI-11 diagnostics may be used to perform in-depth tests on each component of the console subsystem.

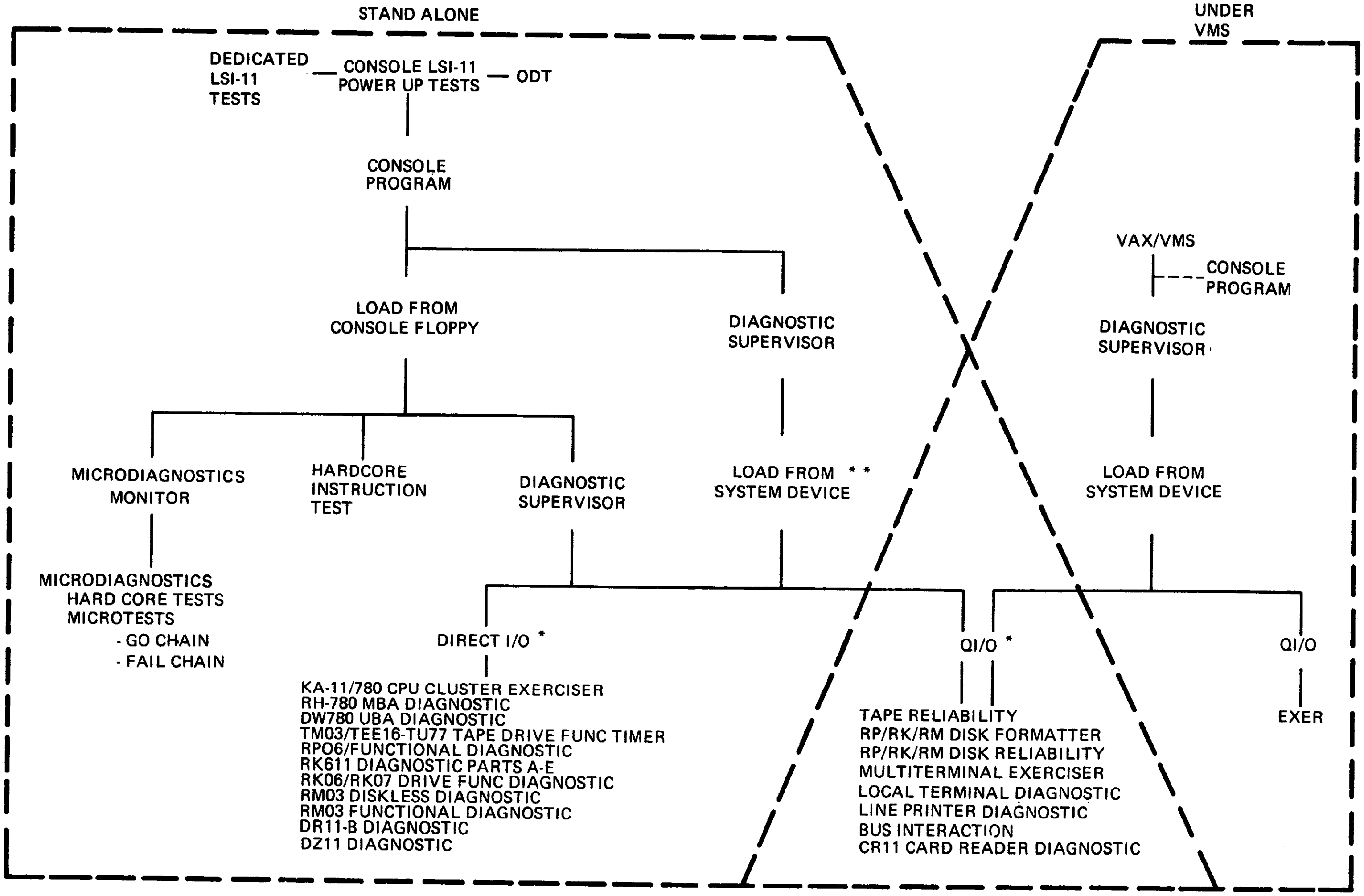
The console subsystem, in connection with the console program, provides the basis for the diagnostic system with the following functions:

- Traditional lights and switch functions such as EXAMINE, DEPOSIT, HALT, START, and single instruction

- Diagnostic and maintenance functions, including the capability to load diagnostic microcode into Writable Control Store (WCS), control execution, control single step clock functions, and examine key system points via a serial diagnostic visibility bus (V Bus), and to deposit and examine data in locations in the VAX-11/780 main memory and I/O space

- Operator communication with the VAX-11/780 software.

The console program enables the operator to run microdiagnostics, to load and run the diagnostic supervisor (in the standalone mode) and the standalone macrodiagnostic programs (using VAX-11/780 native code), and to boot the operating system.



* THE NUMBER OF I/O DIAGNOSTIC PROGRAMS WILL GROW.

Figure 1-3 VAX-11/780 Diagnostic System, Execution Environments

Note that when the console program is running in the LSI-11, it will always be in one of two modes, console I/O mode or program I/O mode. With the exception of the Control P (^P) command, the console commands (console command language) listed in the help files are available only when the console program is in the console I/O mode.

In the console I/O mode, the console program interprets the characters typed on the console terminal as console commands. In the program I/O mode, however, the console program is transparent to the operator. The console program passes characters from the console terminal directly to the VAX-11/78Ø CPU for use by VMS or the diagnostic supervisor.

Type Control P to switch from program I/O mode to console I/O mode.

Type SET TERMINAL PROGRAM to switch from console I/O mode to program I/O mode.

1.6 MICRODIAGNOSTIC PROGRAM

The microdiagnostic program provides module isolation for logic failures within the CPU, floating-point accelerator and MOS memory controllers. The program will detect stuck high/low logic functions and open or grounded etch and wire interconnections. The microdiagnostics are organized in a bootstrapping test sequence (i.e., building blocks) of the console interface, CPU hardware, cache-translation buffer, I-stream buffer, Synchronous Backplane Interconnect (SBI), and memory controller and array. All detected faults result in an error typeout indicating the smallest set of modules to which the diagnostic can isolate the failure.

The microdiagnostic program is initiated by one console command and executed from the CPU cluster test facility. The test facility consists of the console subsystem, console interface, Writable Control Store (WCS), and the V Bus.

The microdiagnostic package consists of two major test divisions: console adapter and hardcore, and microtests. Each test division is controlled by an associated monitor that provides nondiagnostic services to that division. Both test division monitors are serviced by the console-resident microdiagnostic monitor. In addition to loading the test monitors, the microdiagnostic monitor retrieves microtest overlays from the floppy disk, loads test sequences into WCS, performs test dispatching and sequencing, performs error reporting, and manages fault isolation. The microdiagnostic monitor also allows the operator microdiagnostic test selection and execution options (Chapter 4). Figure 1-4 shows overall monitor relationships and test sequencing.

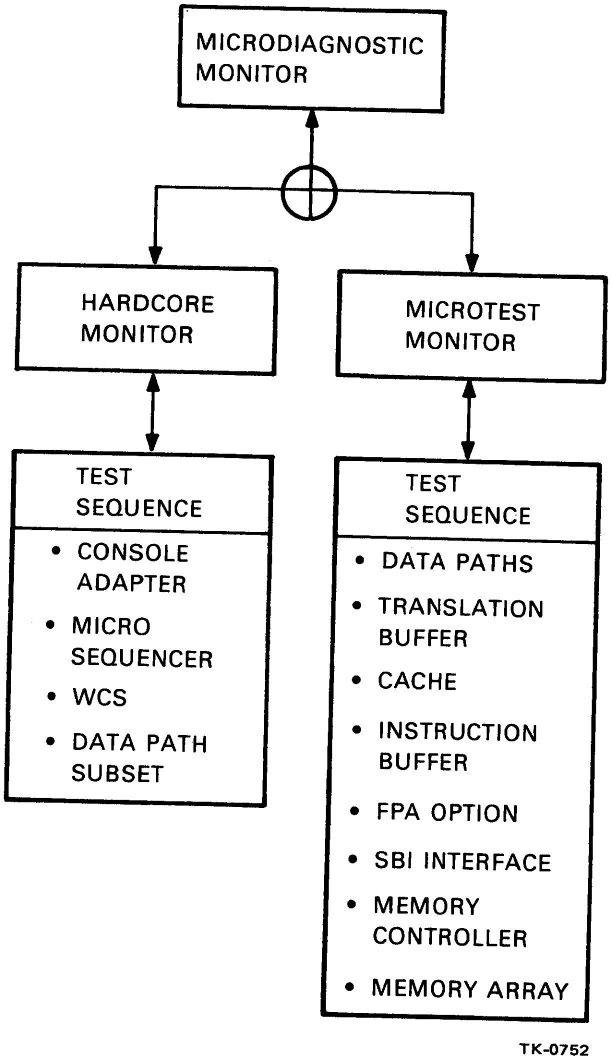


Figure 1-4 Monitor Relationships and Test Sequencing

1.6.1 Console Adapter and Hardcore Division

The adapter and hardcore division microdiagnostic is composed of a test stream of pseudo-instructions and test data located on the console floppy disk. Note that the pseudo-instructions are defined specifically for the test stream. This division tests the console adapter (CIB module), microsequencer, WCS, and a subset of the data paths. The hardcore monitor is called into the console memory by the microdiagnostic monitor. The hardcore monitor, in turn, retrieves small blocks (±1.5K bytes) of test data from the floppy into a console buffer, and then controls execution. When the current block has been completed, the hardcore monitor overlays this block with a new test block. The test data portion of the test stream is comprised of data words and lists of VAX-11/78Ø microinstructions. The microinstructions are loaded into the WCS and executed in single bus cycle or single time-state modes.

When an error is detected, an error header message is typed. Then, if the HALTD flag is not set, a trace message is typed and additional code is executed to isolate the fault. This additional testing will normally consist of V Bus compare instructions. Figure 1-5 shows monitor residency and the basic flow of the console adapter and hardcore tests.

1.6.2 Microtest Division

The microtest division completes testing of the CPU not covered by the hardcore division, and provides isolation to a failing module. The microtests, which are executed under control of the microtest monitor, are divided into two subdivisions: GO chain and FAIL chain. The GO chain consists of microtests loaded into WCS and executed at full speed. The purpose of the GO chain is to detect an error. If an error is detected, control is passed to the FAIL chain, which isolates the error and reports the failing module through the microdiagnostic monitor. Note that the FAIL chain is executed only on detection of an error.

The GO chain consists of a series of WCS overlays. Each overlay is approximately 1K microwords in length and will contain one or more microtests. Initially, the microdiagnostic monitor loads the first overlay into WCS; that overlay is then executed. If no error is detected, the next overlay is loaded into WCS and executed. This sequence continues until each test in each overlay has been executed, or until an error is detected.

When the GO chain detects an error, execution of the microtest that detected the error is suspended. The error microtest address is saved and used by the FAIL chain to restart microtest execution to recreate the conditions that detected the error.

The FAIL chain reenters the failing microtest and begins fault isolation. The microtest is clocked a specific number of ticks from the error address and then certain V Bus signals are processed. If the V Bus signals identify the faulty module, an error report is made through the microdiagnostic monitor. If these

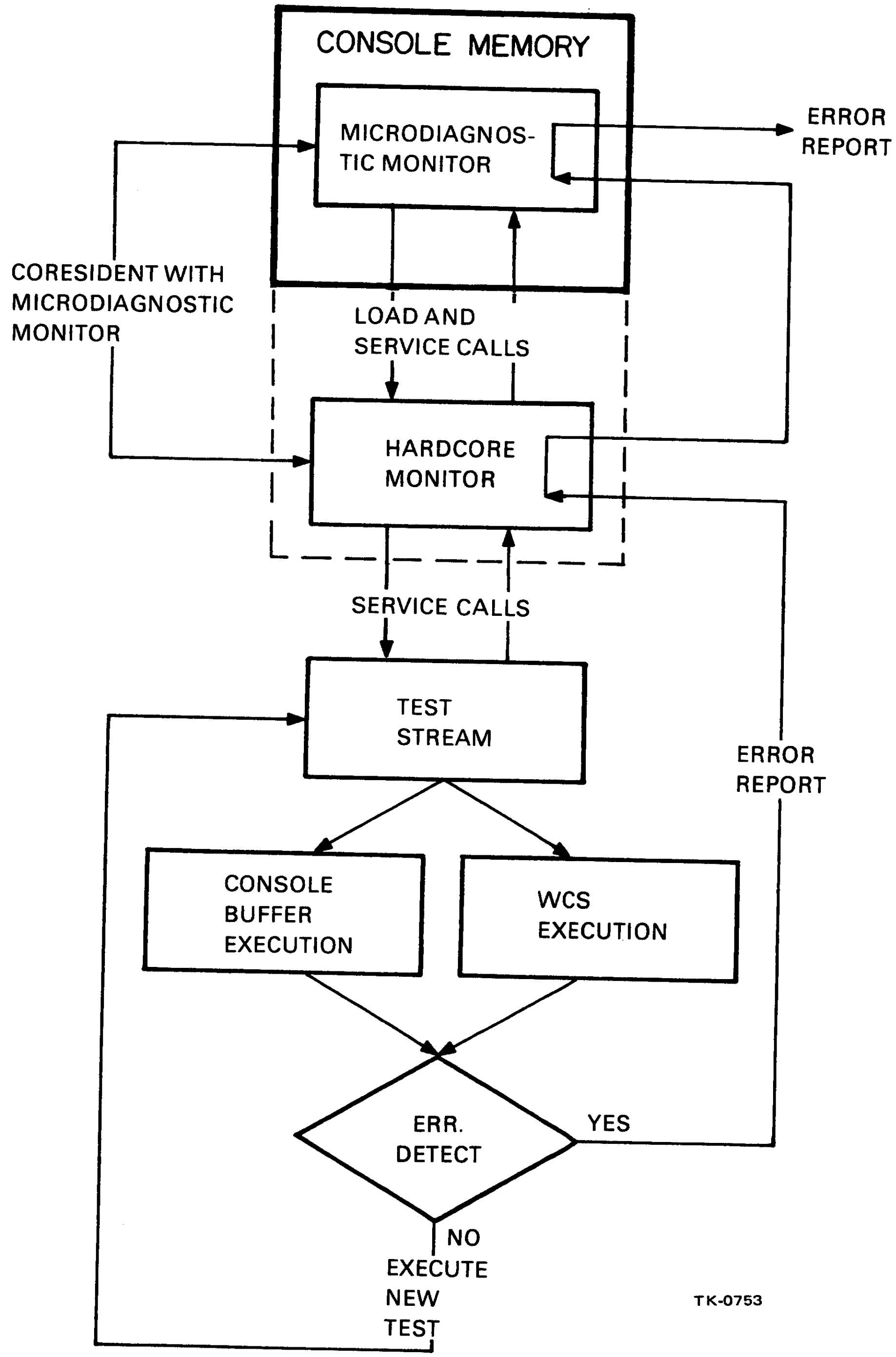


Figure 1-5 Hardcore Monitor Residency/Test Flow

signals do not identify the error, additional V Bus signals are processed. In the case of an intermittent error which is not reproduced during FAIL chain execution, a report is printed that lists the modules involved in the failing GO chain microtest. Figure 1-6 shows monitor residency and the basic test flow.

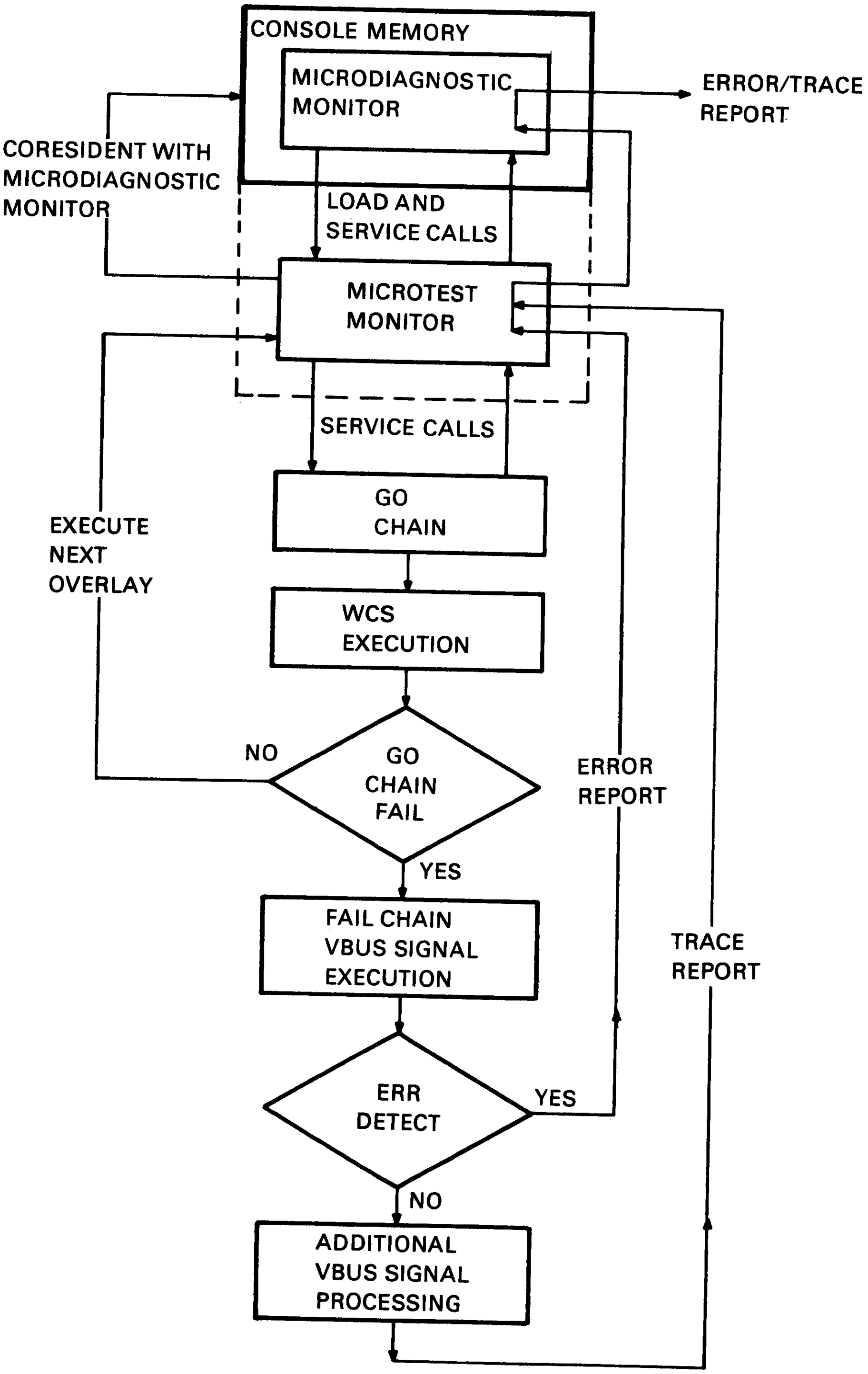
1.7 MACRODIAGNOSTIC PROGRAMS

The macrodiagnostic programs are written in VAX-11 MACRO and assembled in VAX-11 native code. Level 2, 2R, and 3 programs do not run independently; they must always be loaded and executed with the diagnostic supervisor.

1.7.1 Diagnostic Supervisor

The diagnostic supervisor provides a framework that supports each of the macrodiagnostic programs, one at a time. It operates in three environments and provides two major functions. Two of these environments, cluster environment (CE) and system environment (SE), constitute the standalone mode. The diagnostic supervisor operates in the user environment (USE) when it runs under the VMS operating system. In each of these environments different modules within the diagnostic supervisor are activated. The first major function of the diagnostic supervisor is the interpretation of the command line typed on the operator's terminal. The command line interpreter (CLI) portion of the supervisor performs this function, enabling the operator to control the loading, sequencing, and execution of diagnostic test programs. The program interface (PGI) performs the second major function of the supervisor, providing a set of common services required by some or all diagnostic programs. The PGI services include operator interaction routines, error message formatting, memory management, and I/O request handling. Notice that the operator can communicate with the diagnostic program only through the CLI and the PGI message service in the supervisor.

The supervisor supports programs that provide their own device interfaces (direct I/O) and programs that require I/O services. The direct I/O diagnostic programs must be run in the standalone mode (cluster environment and system environment), since VMS inhibits direct access to peripheral devices. Programs that do not directly access the peripheral devices under test rely on queue I/O system services. Both VMS and the diagnostic supervisor provide queue I/O system services, so that these programs can run in either the standalone mode (in the system environment) or the user mode (user environment, under VMS). When the diagnostic programs requiring queue I/O services are run in the user mode, the supervisor passes the queue I/O requests directly to VMS. When queue I/O diagnostic programs are run standalone, the supervisor emulates the VMS operating system, providing the queue I/O system services. Figure 1-7 shows the functions of the diagnostic supervisor in the three macrodiagnostic operating environments.



TK-0738

Figure 1-6 Microtest Monitor Residency/Test Flow

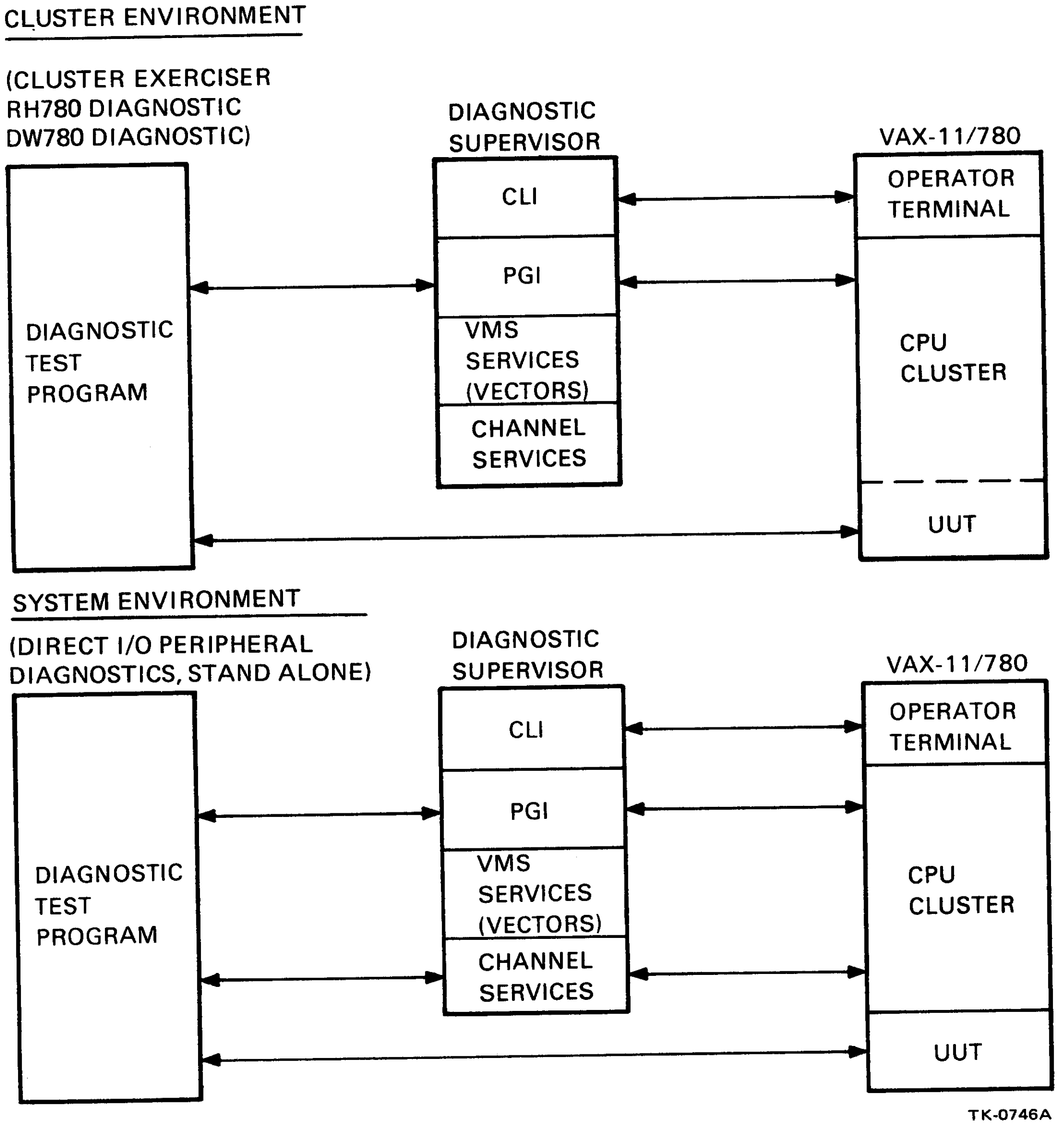


Figure 1-7 Functions of the Diagnostic Supervisor Environments (Sheet 1 of 2)

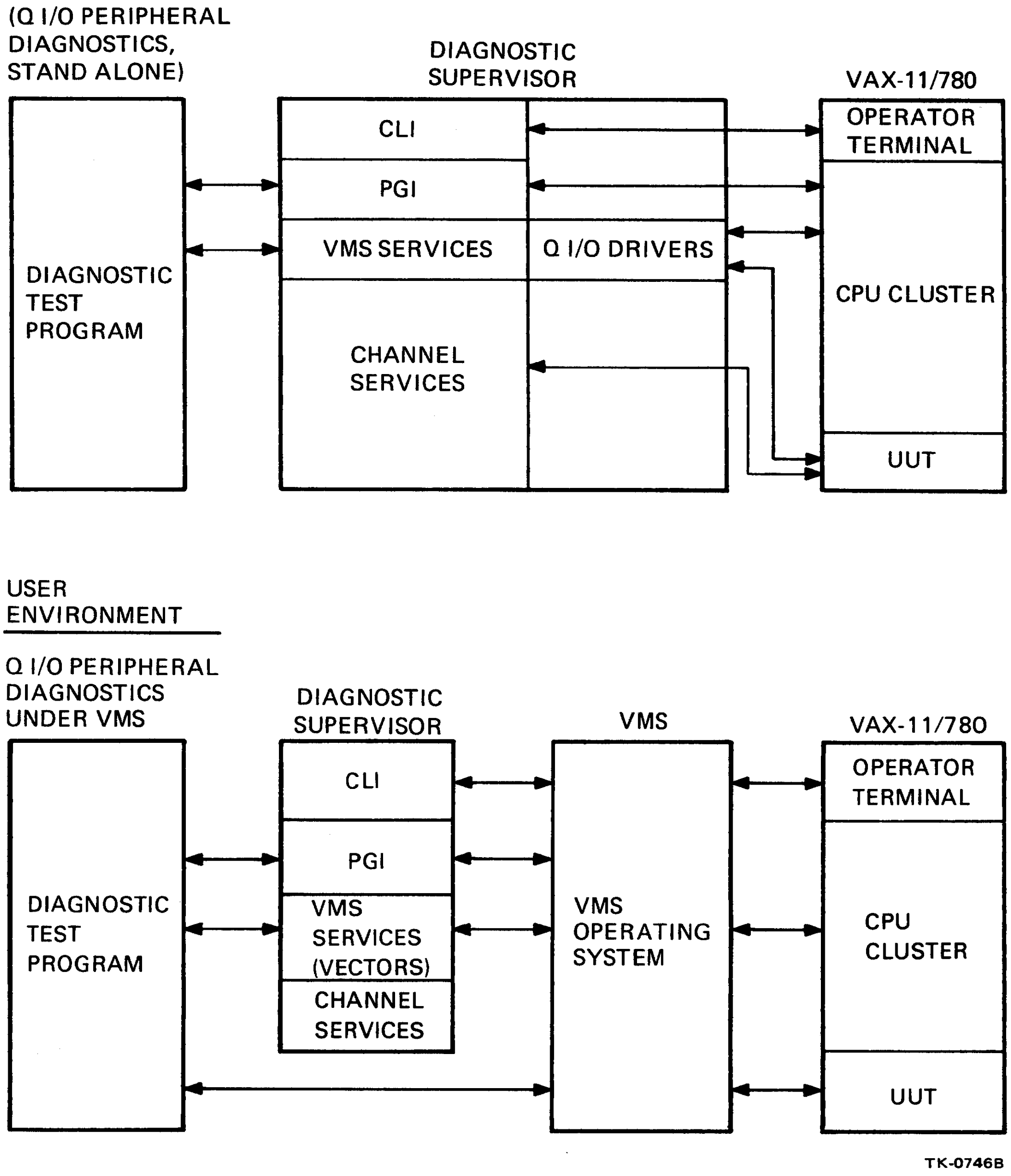


Figure 1-7 Functions of the Diagnostic Supervisor Environments (Sheet 2 of 2)

1.7.2 Cluster Diagnostic Programs

The CPU cluster exerciser, the RH78Ø (MBA) diagnostic program, and the DW78Ø (UBA) diagnostic program test the VAX-11/78Ø cluster hardware. They run under the cluster environment portion of the diagnostic supervisor, in the standalone mode.

1.7.2.1 CPU Cluster Exerciser Package -- The cluster exerciser package consists of three diagnostic programs. The package provides a comprehensive functional test of the CPU cluster, including the CPU, the Unibus and Massbus adapters, and memory. The first program (ESKAX) is the quick verify portion of the CPU cluster exerciser package. The second program (ESKAY) tests the native mode instruction set of the VAX-11/78Ø. EXKAZ, the third program in the package, checks memory management and the PDP-11 instruction set (in compatability mode).

The CPU cluster exerciser programs identify failing functions and failing subsystems. For further fault isolation the operator should run the microdiagnostic program or restrict the desired CPU cluster exerciser program to the minimum number of modules which will detect the failure, through commands to the diagnostic supervisor.

1.7.2.2 RH78Ø (MBA) Diagnostic Program -- The RH78Ø (MBA) diagnostic program tests the majority of the MBA logic regardless of the type of peripheral device attached to the Massbus. Although the program does not provide explicit component level fault isolation, every detectable error is associated with an operator-selectable scope loop. Diagnosis of attached devices is not attempted. Verification of the Massbus transceivers and cables is possible with a Massbus exerciser (MBE, RH11-TB) attached to the Massbus. Use of an MBE on the Massbus also allows verification of the MBA ability to perform high speed block transfers. Note that either a device or a bus terminator must be attached to the Massbus to enable program execution. The program tests the MBA at three levels.

1. The first level checks basic functions. The functions tested are those which are necessary for subsequent, detailed fault detection. The objective is to locate functional failures prior to testing for explicit bit failures. Map register access, virtual address register access, and correct data input buffer byte selection are tested at this level.

2. The second level of testing locates bit failures (stuck high/low). The program toggles bits directly accessible to the CPU, and it sets and clears bits indirectly by setting up specific commands and conditions.

3. The third level determines the ability of the MBA to meet system demands. The program performs block transfers using the MBA wraparound features. These block transfers are executed in the maintenance mode and ensure that the MBA will support data transfers typically associated with system software. In addition, the program tests the ability of the MBA to interrupt the CPU under all legal conditions.

1.7.2.3 DW780 (UBA) Diagnostic Program -- Like the RH780 (MBA) diagnostic program, the DW780 (UBA) diagnostic program tests most of the UBA logic. Every detectable error is associated with an operator-selectable scope loop. The program does not attempt to test devices attached to the Unibus. However, if a Unibus exerciser is attached to the Unibus, the program will verify the integrity of the Unibus transceivers and the ability of the UBA to respond to device-initiated functions. The program tests the UBA at seven levels.

1. The program tests the basic functions necessary for subsequent fault detection: the addressability of the UBA registers, their initial states, and whether they can be read and written.

2. The program tests the RAM addressing capability of the UBA logic (accessing map registers, data path registers, and BRSVRs).

3. Power-fail and interrupt functions of the UBA are tested next.

4. The program creates and tests all error conditions.

5. Extensive data transfer tests check the map registers, the direct data path, the buffered data paths, the data path registers, the Unibus address and data lines, and the microsequencer.

6. The device tests check all types of data transfer on the Unibus: DATI, DATIP, DATO, DATOB initiated by the UBA and by the UBE. Interrupts from the UBE to the CPU are also tested at the four BR levels.

7. The contention logic test checks for race conditions when the four microsequencer select lines (UBATT SEL, SB SEL, DMA SEL, FILE WRITE SEL) are asserted at about the same time.

1.8 PERIPHERAL DIAGNOSTIC PROGRAMS

In accordance with the structure of the diagnostic system as a whole, the peripheral diagnostic programs are organized in a hierarchy. Repair and functional level programs are designed to test specific peripheral devices. These programs (with the exception of the line printer and terminal diagnostics) must be executed in the standalone mode under the system environment (SE) services of the supervisor, since they provide their own access (direct I/O) to the devices under test. The diagnostic programs which rely on VMS, or the supervisor, for access to the units under test (queue I/O) are each designed to test a range of peripheral device types. For example, the disk reliability program (ESRAA) will test all disk drive types supported by the VMS operating system.

On error detection, the repair level diagnostic programs will call out both the failing device controller module and the failing function, dump the contents of relevant registers, and list expected and received data patterns. The functional level programs provide register dumps and call out the failing function when an error is detected. The reliability and formatter level programs provide more detailed information on the failing function in addition to the register dumps.

The system exerciser program tests the integrity of the major system buses (i.e., SBI, Massbus, Unibus) under heavy I/O activity, and it highlights any interaction problems that result. The program should be run as a dedicated process under VMS. No other program may run concurrently or compete for system resources, since the program requires the use of all system resources.

1.9 OPERATOR/VAX-11/780 COMMUNICATION

The operator communicates with the VAX-11/780 computer through the console subsystem. The console subsystem provides a programmed interface between the console terminal and the VAX-11/780 hardware and software, including the diagnostic system. The console subsystem hardware consists of an LSI-11 microprocessor (11/03), a single floppy disk drive and controller, a terminal and two serial line units, a VAX-11/780 CPU console interface (CIB), and a control panel on the VAX-11/780 CPU cabinet. The console program includes a console command language and the software utilities that provide operator console functions. These functions are required for VMS and diagnostic support. The paragraphs that follow introduce the basic console functions. Refer to Chapter 2 for a detailed description of the console command language.

1.9.1 Console Terminal Modes

The console terminal serves as the console program's I/O device and as a VMS operator terminal. The console program has two operating modes: console I/O mode and program I/O mode.

In console I/O mode, the terminal serves as the operator interface to the console panel functions, CPU debug functions, and CPU kernel test functions. In this mode console terminal input is not passed to the VAX ISP-level software. All terminal input is interpreted by the LSI-11, and appropriate console functions are invoked.

In program I/O mode the terminal serves as a VMS operator terminal. All terminal input is passed, character by character, to the ISP-level software. All validity checking, etc. is performed by VMS. The console program is transparent to the VAX-11/78Ø software. All terminal output from the software is passed directly to the console terminal.

1.9.2 Console Panel Equivalent Functions

The functions in this group are those normally available through a traditional console panel. These functions include ISP-level program and CPU clock controls, and display and modification of memory elements.

1.9.2.1 **Program Control** -- The console can initialize the CPU by setting certain logic to a defined state. It can initiate ISP-level instruction execution at a point specified by the program counter, as well as terminate instruction execution. In addition, the console can bootstrap the system by loading memory with a specific file from the system load device, and initiate instruction execution at a predefined address after the load. The console can also stop ISP level instruction execution.

1.9.2.2 **Memory Element Display and Modification** -- The console allows display and modification of memory elements in the VAX-11/78Ø including main memory, I/O, general, and internal register addressing space. The address spaces can be accessed, read, and written in the quantities specified below:

a. Main memory elements: byte, word, longword, quadword quantities

b. CPU general registers (RØ--R13, SP, PC), and processor register space: longword quantity

c. CPU processor register space: longword quantity

d. I/O registers: byte, word, longword quantities depending on register data length

e. ID bus registers: longword quantity

f. VAX-11/78Ø V Bus (Visibility Bus) channels can be displayed (V Bus channels are read-only)

g. VAX-11/78Ø main memory and/or Writable Control Store (WCS) can be loaded from files on the console subsystem floppy disk.

1.9.2.3 **Clock Control** -- The CPU clock can be controlled by the console to provide single step clock mode for use in hardware or software debugging. The control modes available include single instruction step, single SBI bus cycle step, and single SBI time state step modes.

Single instruction step mode allows ISP-level programs to execute one instruction at a time. This mode causes the CPU to enter the halt state after the instruction execution.

Single SBI bus cycle step mode causes the CPU clock to stop each time SBI time state Ø (TØ) is asserted. TØ remains asserted until a control signal from the console causes the clock to resume operation. The clock ticks until the next SBI TØ.

Single SBI time state step mode causes the CPU clock to assert and hold a time state (TØ, T1, T2, or T3) until a control signal from the console causes the next time state to be asserted and held.

1.9.3 Console Control Functions

The console control functions allow control of numeric radices, addressing modes, and data length, and provide for displaying console and CPU status. Functions are also provided that repeat commands and link multiple commands into a single executable command list. In addition, the console provides a means to control the number of fill characters to be added after special characters are sent to the console terminal.

1.9.3.1 **Default Settings** -- The console allows specification of defaults for addressing modes, radix of numeric input and output, and the data length of addressable memory elements. Any default setting can be overridden within the context of a console command.

a. The default addressing modes can be set for virtual, physical, ID Bus, V Bus, general register, or internal (processor) register.

b. Default radices for console numeric input and output can be set to octal, decimal, or hexadecimal radix.

c. Defaults for memory element data lengths can be set for byte, word, longword, and quadword.

d. Power-up defaults --

Address Type	=	Physical
Radix	=	Hexadecimal
Data Length	=	Longword (32 Bits)

1.9.3.2 Status Displays -- The console provides a means to display CPU and console subsystem status. The CPU status includes the stop/run state of the instruction set processor, the current clock step mode, and the state of the Stop on Microbreak Match Enable (SOMM). Console subsystem status includes the current setting of all console defaults and the number of terminal fill characters.

1.9.3.3 Command Linking and Repeating -- The console provides a facility that allows multiple commands to be linked into a single executable list. Commands to be linked are entered into an internal console queue. The console operator can specify execution of the command queue one pass at a time. Or, the queue may be executed continuously. This facility allows the diagnostic user to create short routines of console commands for use in hardware debugging operations.

The console also provides a facility to continuously execute a single command or list of commands. Once initiated, command execution continues until terminated by the operator. The repeat facility allows maintenance personnel to scope the operation of CPU and subsystem logic invoked by console commands.

1.9.3.4 Real-Time Delays -- The console provides a facility for introducing real-time delays of varying duration between the execution of console commands linked with the command linking facility. This function has no effect on the CPU, but only delays the console's processing of the next sequential command in the command queue. The delay facility is provided for use after console commands that invoke CPU functions which require time to complete (e.g., initialization).

CHAPTER 2
CONSOLE PROGRAM AND CONTROL DESCRIPTION

This chapter describes the console command language and associated command facilities. Where appropriate, examples of command usage are included. Also included are all applicable console error messages.

2.1 CONSOLE PROGRAM OVERVIEW

The following paragraphs provide a basic overview of the console software modules. Note that the services provided by the console are contained in the LSI-11 4K ROM and 8K RAM. The console provides services for console control, operator interface, microdiagnostic execution, VMS support functions and remote diagnosis.

2.1.1 Command Getter

The basic functions of this module are to retrieve a command line from the console terminal (get a command line routine), and provide a check point (or wait) loop for the console program (console null loop).

The program spends the majority of its time in the null loop, which consists of a series of test points and conditional branches (e.g., bootstrap initiated, VAX-11/78Ø CPU halted, etc.). Should any of these functions be active (i.e., flag set) the program performs a branch to the routine required to service the request initiated by that flag.

2.1.2 Parser and Parser Tables

The parser module decodes the command typed on the console terminal and provides a pointer to the appropriate routine to execute the command. The parser manipulates the command line to condition it for decoding (e.g., discarding leading blanks and checking for a delimiter in the command input string). The command is decoded through a set of syntax check trees that provide pointers to the appropriate execution routine within the command execution module. Any data required for command execution has been set up in tables included as part of the parser.

2.1.3 Command Executor Module

The command execution pointer from the parser is passed to the command execution module entry point. This entry point provides a pointer (i.e., starting address) to the appropriate command execution routine (e.g., DO BOOT, PERFORM QUAD CLEAR). The basic sequence of module action is:

a. Apply switches or defaults for radix, address space, and data length.
b. Execute command routine.
c. Test for repeat function.

If a repeat function is specified, the routine monitors the console for the control character (^C) required to terminate the loop.

The module also supplies the required subroutines to support the execution functions (e.g., open a file, load a file). Following command execution, control is passed to the console null loop within the command getter module.

2.1.4 Additional Services

In addition to the command decoding and execution functions, the console provides several other services.

Remote support is provided to allow console access from a remote terminal or computer. The facility also enables communication between local and remote operators, as well as transfer of console control between local and remote operators.

VMS services are also provided. These services include routines for terminal support and the associated drivers, as well as a file service for the floppy drive and its associated drivers. The code for some of these services is contained in the ROM as well as the RAM. These services are provided through emulator traps.

The console software also includes the basic LSI-11 processor and memory tests, that are executed on each power-up and bootstrap, and the primary bootstrap routine for the floppy.

2.2 COMMAND TERMS AND SYMBOLS

Table 2-1 provides a summary of terms and symbols used to describe the syntax of the console commands.

Table 2-1 Term and Symbol Definitions

Term/Symbol	Definition
< >	Used to denote a category name (label) e.g., category name <address> represents a valid address
!	Used to indicate the Exclusive OR operation (i.e., selection of parameters within a command line). For example, <A>!<B> means either <A> or <B> but not both is to be selected
( )	Used to indicate that one of the syntactic units of the expression is to be selected
[ ]	Used to indicate the part of an expression that is optional e.g., WAIT [ ] indicates that the wait command takes an optional count argument

Table 2-1 Term and Symbol Definitions (Cont)

Term/Symbol	Definition
<blank>	Represents one or more spaces or tabs
<count>	Represents a numeric count
<XYZ -list>	Indicates one or more occurrences from the category indicated by XYZ
<address>	Represents an address argument
<data>	Represents a numeric argument
<qualifier>	Represents a command modifier (switch)
<input prompt>	Represents the console's input prompt string '>>>'
<reverse prompt>	Represents the linking prompt '<<<'
<CR>	Represents a console terminal carriage return
<LF>	Represents a console terminal line feed
/	Delimits a command from its qualifiers
+	Represents the default address when used as an address argument in an examine or deposit command (The default address is the last address used plus the current data length in bytes.)
*	Used as an address argument in an examine or deposit command and represents the last address referenced.

2.2.1 Notation Examples

EXAMINE [<qualifier - list>] [<blank> <address>]

An examine command explanation follows.

a. An examine command may optionally contain a list of one or more qualifiers.

b. An examine command may optionally contain an address argument. If the address is specified it must be preceded by one or more spaces or tabs.

Following is a list of valid examine commands:

```
EXAMINE
EXAMINE/BYTE/VIRTUAL
EXAMINE <space> 123456
EXAMINE <tab> 123456
EXAMINE/WORD <space> 123456
EXAMINE/WORD <space> <tab> 123456
```

2.2.2 Command Abbreviations

Console command words may be abbreviated by typing only enough characters to identify each command word. The minimum abbreviation for each command is specified in parentheses in each command description paragraph title.

Example

```
EXAMINE/VIRTUAL/BYTE 1234
```

may be abbreviated to:

```
E/V/B 1234
```

2.3 CONSOLE COMMAND DESCRIPTIONS

Each console command description is divided into three, four, or five descriptive segments, depending on the particular command. The descriptive segments are:

a. Syntax: describes the command structure

b. Command description: a brief paragraph describing command operation, general restrictions, or available options

c. Response: a description of the console program response to the specified command

d. Qualifiers: a list of applicable command modifiers

e. Options: a list of applicable command options.

The descriptive segments use the terms and symbols defined in Table 2-1. Note that every command (or command line) must be terminated with a <CR>.

2.3.1 Boot Command (B)

Syntax: BOOT [<device name>] <CR>

The boot command initiates a VAX-11/780 system bootstrap sequence. The command may support bootstrap operations from a set of alternate system devices.

<device name> has the following format: DDn, where DD is a two-letter device mnemonic (e.g., DX for floppy), and n is a one-digit unit number.

If no <device name> is given with the boot command, the console will perform the boot sequence for the default system device by executing an indirect command file named DEFBOO.CMD. This indirect file contains the necessary console commands to boot from whichever device is chosen to be the default system device.

If <device name> is given with the command, the console will execute an indirect command file named DDNBOO.CMD, where DDN is the <device name> given.

Example

> BOOT RPØ -- will cause the console to execute an indirect command file named RPØBOO.CMD (console enters program I/O mode after executing the command file).

Bootstraps from devices other than the system default device are performed by indirect command files containing the console commands necessary to boot an alternate device. After successful CPU bootstrap completion, a response from VMS will be displayed on the console terminal.

2.3.2 Clear Command (CL)

Syntax: Clear<BLANK>(SOMM ! Step)

CLEAR SOMM — The Stop on Microbreak Match (SOMM) enable on the console interface board is cleared -- (disabled).

CLEAR STEP — Any existing clock step mode (single bus cycle, single time state, single instruction) is cleared, and the VAX-11/78Ø CPU clock will be in the normal (free-running) mode.

Response: <CR><LF><CONSOLE-PROMPT>

2.3.3 Continue Command (C)

Syntax: CONTINUE <CR>

The continue command causes the VAX-11/78Ø CPU to begin instruction execution at the address currently contained in the CPU program counter (PC). Note that CPU initialization is not performed. The console enters the program I/O mode after issuing the continue command.

Response: <CR> <LF> (console enters program I/O mode)

2.3.4 Deposit Command (D)

Syntax: DEPOSIT [<qualifier - list>] <blank> <address> <blank> <data> <CR>

Qualifiers: /BYTE, /WORD, /LONG, /QUAD, /NEXT, /VIRTUAL, /PHYSICAL, /V BUS, /INTERNAL, /GENERAL. (Refer to Paragraph 2.7 for description of defaults.)

The deposit command writes (deposits) <data> into the address specified. The address space used will depend on the qualifiers specified with the command. If no qualifiers are used, the current address type default will determine the address space to be used.

Response: <CR> <LF> <input prompt>

The <address> argument may also be one of the symbolic addresses defined in Table 2-2.

Table 2-2 Deposit Symbolic Addresses

Symbol	Definition
PSL	Deposits to the processor status longword
PC	Deposits to the program counter (general register F)
SP	Deposits to the stack pointer (general register E)
+	Deposits to the location immediately following the last location referenced. For physical and virtual references the location referenced will be the last address plus n, where n = 1 for byte, 2 for word, 4 for longword, 8 for quadword. For all other address spaces, n is always equal to 1.
-	Deposits to the location immediately preceding the last location referenced
*	Deposits to the location last referenced
@	Deposits to the address represented by the last data examined or deposited.
Example:	
E SP	Examines stack pointer
D @ <data>	Deposits <data> to the location specified by the contents of the stack pointer.

2.3.5 Enable DX1: Command

Syntax: Enable DX1:

Qualifiers: None

Enable console software to access floppy on those systems with dual floppies.

Response: <CR><LF> <INPUT-PROMPT>

2.3.6 Examine Command (E)

Syntax: EXAMINE [<qualifier - list>] [<blank> <address>] <CR>

Qualifiers: /BYTE, /WORD, /LONG, /QUAD, /NEXT, /VIRTUAL, /PHYSICAL, /ID BUS, /V BUS, /INTERNAL, /GENERAL. (Refer to Paragraph 2.7 for description of defaults.)

The examine command reads and displays the contents of the specified <address>. If no <address> is specified, the current <default address> is examined.

The <address> argument may also be one of the symbolic addresses defined in Table 2-3.

Table 2-3 Examine Symbolic Address

Symbol	Definition
PSL	Displays the processor status longword
PC	Displays the program counter (general register F)
SP	Displays the stack pointer (general register E)
+	Displays the location immediately following the last location referenced.
-	Displays the location immediately preceding the last location referenced
*	Displays the last location referenced
@	Displays the location whose <address> is the last data examined or deposited.

Response: <CR> <LF> <tab> <address space identifier> <address> <data> <CR> <LF> <input prompt>

Sample responses (console output underlined)

```
>>> EXAMINE/PHYSICAL 1234
P ØØØØ1234        ABCDEF89
>>> EXAMINE/VIRTUAL 1234
P ØØØØ5634        Ø1234567
```

NOTE

The translated physical address is displayed for virtual examines.

```
>>> EXAMINE/G Ø
G ØØØØØØØØ        98765432; GPR Ø
```

2.3.7 Halt Command (H)

Syntax: HALT <CR>

The halt command causes the VAX-11/780 CPU ISP to stop instruction execution after completing execution of the instruction being executed, when the console presents the halt request to the VAX-11/780 CPU.

Response: (VAX-11/780 CPU indicates it has stopped) <CR> <LF> <tab> HALTED AT <contents of VAX-11/780 CPU PC> <CR> <LF> <input prompt>

2.3.8 Help Command (HE)

Syntax: HELP <CR>

The console opens an indirect command file that displays a console help file, CONSOLE.HLP. The help file contains a description of all console commands and console abbreviation rules, and it lists the names of all other help files that may be displayed.

Response: Help file printed on console terminal.

2.3.9 Initialize Command (I)

Syntax: INITIALIZE <CR>

This command causes VAX-11/780 CPU system initialization.

Response: <CR> <LF> <tab> INIT SEQ DONE <cr> <LF> <input prompt>

2.3.10 LINK Command (LI)

Syntax: Link

Qualifiers: None

Link causes the console to begin command linking. Console prints reversed prompt to indicate linking. All commands typed by user are then stored in an indirect command file for later execution. Control C terminates linking.

Response: <CR><LF><REVERSE-PROMPT>

(Refer to Paragraph 2.9 for further details.)

2.3.11 Load command (LO)

Syntax: LOAD [<qualifier list>] <blank> <file specification> <CR>

The load command is used to read file data from the console's floppy disk to the VAX-11/780 main memory, or Writable Control Store (WCS). The applicable qualifiers are defined in Table 2-4. If no qualifier is given with the load command, physical main memory is loaded.

Table 2-4 Load Command Qualifiers

Qualifier	Definition
/START: <address>	This qualifier specifies a starting address for the load. If no start qualifier is given, the console will start loading at address Ø.
/WCS	This qualifier specifies that the WCS is to be loaded.

2.3.12 Perform Command (P)

Syntax: Perform

Qualifiers: None

The perform command executes a file of linked commands previously generated by a link command.

Response: <dependent on commands linked>

2.3.13 Quad Clear Command (Q)

Syntax: QCLEAR <blank> <physical address> <CR>

The quad clear command clears the quadword at the <address> specified. The command is used to clear an uncorrectable ECC error. The <address> is always interpreted as a physical main memory location. The <address> given is forced to a quadword boundary by the unconditional clearing of the three low-order address bits.

2.3.14 Reboot Command (REB)

Syntax: REBOOT

Qualifiers: None

This command causes a console software reload, without disturbing the VAX-11/78Ø.

Response: <console start-up display>

2.3.15 Repeat Command (R)

Syntax: REPEAT <console command> <CR>

A repeat command causes the console to repeatedly execute the specified <console command> until execution is terminated by a Control C (^C) (Paragraph 2.8). Any valid console command may be specified for <console command> with the exception of the repeat command.

Response: Dependent on command specified.

2.3.16 Set Command (SE)

Syntax:

```
SET <blank> DEFAULT [<blank> <default option>] <CR>
or
SET <blank> STEP [<blank> <step option>] <CR>
or
SET <blank> TERMINAL <blank> (fill: <count>! PROGRAM)
<CR>
or
SET <blank> SOMM <CR>
or
SET <blank> CLOCK [<blank> (SLOW! FAST! NORMAL)] <CR>
or
SET RELOCATION: <data> <CR>
```

Response: <CR> <LF> <input prompt> (for all commands)

The set default command sets console default for radix of console numeric input and output, address type, and data length. The console will apply defaults when a console command does not explicitly specify radix, address type, or data length. A set default command with no options specified will set all default settings to the power-up state. Applicable default options are listed in Table 2-5.

Table 2-5 Set Default Command Options

Option	Format	Specification
Address Default Options	Physical Virtual General Internal ID Bus V Bus	Sets default addressing mode as specified
Data Default Options	Long Byte Word Quad	Sets default data length as specified
Default Radix Options	Hex Octal	Sets default radix for terminal numeric I/O to radix specified

The set step command sets the VAX-11/78Ø CPU processor clock mode to the mode specified. The applicable modes are listed in Table 2-6.

Table 2-6 Set Step Command Options

Option	Specification
Step Step Instruction	Sets CPU clock mode to single instruction step mode
Set Step Bus	Sets CPU clock mode to single SBI cycle step mode
Set Step State	Sets CPU clock mode to single SBI time state step mode

The set terminal command allows the selection of two parameters.

a. Set Terminal Fill: <count> = The count specifies the number of fill characters to be added after special characters (e.g., prompts) are transmitted to the console terminal.

b. Set Terminal Program = The console terminal enters the program I/O mode.

The set clock command sets the VAX-11/78Ø CPU clock to a frequency specified by one of the arguments (Fast, Slow, Normal) within the command where:

Fast = 1Ø.525 MHz
Slow = 8.925 MHz
Normal (or no argument) = 1Ø.Ø MHz

The set SOMM command sets the Stop on Microbreak Match (SOMM) enable on the console interface board (CIB). When SOMM is set, if the contents of the VAX-11/78Ø micro PC ever become equal to the contents of the microbreak match register (ID register 21), the CPU clock is stopped.

The set relocation command deposits <data> to the console's relocation register. The contents of the relocation register are added to the effective address of all virtual and physical memory examines and deposits.

Response: <CR><LF><input-prompt>

2.3.17 Show Command (SH)

Syntax: SHOW <CR>

The show command will cause the console terminal to display:

a. The current default settings for data length, address type, and radix of address and data inputs and outputs.

b. The terminal fill character count.

c. The VAX-11/78Ø CPU status including the run/halt state and current clock mode setting.

2.3.18 Start Command (S)

The two start command formats are described below.

Syntax: START <blank> <address> <CR>

This format performs the equivalent of the following sequence of console commands:

1. Performs a VAX-11/78Ø CPU initialization (>>> INIT).

2. <address> is deposited into the VAX-11/78Ø PC (>>> DEPOSIT PC <address>).

3. A continue function is issued to begin VAX-11/78Ø CPU instruction execution (>>> CONTINUE).

Response: <CR><LF> (console enters program I/O mode) (for START)

Syntax: START/WCS <blank> <address>

This format performs the equivalent of the following sequence of commands:

1. <address> is deposited to the VAX-11/78Ø micro PC.
2. CPU clock is started in free-running mode.

Response: <CR><LF> <input prompt> (for START/WCS)

2.3.19 Next Command (N)

Syntax: NEXT [<blank> <count>] <CR>

The next command causes the VAX-11/78Ø CPU clock to step the number of times indicated by <count>. The type of step performed by the clock is determined by the current state of the CPU clock mode, as set by a previous set step command. A next command issued while the VAX-11/78Ø CPU is in normal (free-running) mode will default to single instruction step mode for the duration of the command.

The console enters program I/O mode immediately before issuing the step, and reenters console I/O mode as soon as the step is completed. Step-dependent responses are displayed on the console terminal after the completion of each of the count steps as specified below.

a. Single instruction step:
   <CR> <LF> <tab> HALTED AT <contents of PC>

b. Single bus cycle step:
   <CR> <LF> <tab> CPTØ UPC = <contents of UPC>

c. Single time state step:
   <CR> <LF> <tab> CPTn (where n = 1, 2, or 3) or
   <CR> <LF> <tab> CPTØ UPC = <contents of UPC>

If no <count> is specified, one step is performed, and the console enters the space bar step mode. While in this mode, each depression of the space bar causes one execution of the step option currently enabled (i.e., bus cycle, time state, instruction).

A next command with an argument will not enable the space bar feature. For example, NEXT 2 will cause two steps to be executed; the console will then prompt for another command.

An input of any character except SPACE will cause an exit from the space bar step mode.

2.3.2Ø Test Command (T)
Syntax: TEST [/COMMAND] <CR>

This command invokes the microdiagnostic monitor program. If no /COMMAND qualifier is issued with the command, microdiagnostic execution begins immediately. If microdiagnostic testing is completed successfully (i.e., no errors detected) the console program is invoked automatically.

The COMMAND qualifier is used to cause the microdiagnostic monitor to enter its command mode and wait for operator input before initiating microdiagnostic execution.

2.3.21 Unjam Command (U)
Syntax: UNJAM <CR>

This command initiates an SBI unjam operation.

Response: >>>

2.3.22 Wait Command (WA)
Syntax: WAIT <blank> DONE

The wait command has no effect unless it is executed from an indirect command file. When it is executed from an indirect command file, it causes further execution of the command file to be suspended until one of the following occurs:

a. A DONE signal is received from a program running in the VAX-11/780 CPU. On receipt of DONE, the console will resume execution of the command file.

b. If the VAX-11/780 CPU halts (or if the clock stops) and no DONE signal has been received, the console prints <@EXIT> and aborts execution of the remainder of the command file.

c. A Control C (^C) is entered on the console terminal, which causes the console to abort execution of the command file.

Response: <CR> <LF> <input prompt>

2.3.23 Indirect (@) Command

Syntax: @ <filename> <CR> or @ DX1: <filename> <CR>

This command causes the console to open the file specified by <filename> and begin executing console commands from the file. Execution continues until one of the following occurs:

a. A WAIT DONE command is read from the file (Paragraph 2.3.22).

b. The end of the indirect file is reached. In this case the console prints <@EOF> and prompts for normal command input.

c. A ^C is entered on the console causing it to abort execution of the indirect file.

2.3.24 WCS Command (W)

Syntax: WCS

This command invokes the control store debugger, overlaying the console program. The console help file, WCSMON.HLP, contains a summary of control store debugger commands. To print out this file, type @ WCSMON.HLP.

Response: WCS> (Control store debugger prompt)

2.4 COMMANDS PERFORMED WITH THE VAX-11/780 CPU RUNNING

Most console commands require that the VAX-11/780 CPU be halted to allow the command to be executed. However, some console commands do not require interaction with the VAX-11/780 CPU, and may be executed with the VAX-11/780 CPU running. These commands include:

a. Show
b. Help
c. Set commands
d. Examine /V Bus
e. Halt
f. Clear
g. Wait Done

Specifying any other console command while the VAX-11/780 CPU is running will cause the console adapter to reject the command and type out the following error message on the console terminal:

```
<CR> <LF> ? CPU NOT IN CONSOLE WAIT LOOP, COMMAND ABORTED
<CR> <LF> <INPUT PROMPT>
```

2.5 COMMENTS WITHIN COMMANDS

The console allows comments, preceded by an exclamation mark (!), to appear in any command line. When the console detects an exclamation mark, any remaining text in the command line is ignored.

A comment may begin in any character position within a command line, including the first.

Example (console output underlined)

```
    >>> ! THIS IS A VALID COMMENT <CR>
    >>> EXAMINE 1234 ! THIS IS ALSO A COMMENT <CR>
```

2.6 CONTROL CHARACTERS AND SPECIAL CHARACTERS

Table 2-7 contains a description of the control characters and special characters recognized by the console program.

Table 2-7 Control/Special Character Descriptions

Character	Description
CONTROL C (^C)	Causes the suspension of all repetitive console operations such as: a. Repeated command executions as a result of a repeat command b. Successive operations as a result of a /NEXT qualifier c. Delays resulting from a wait command d. Successive steps resulting from a next command e. Aborts further execution of an indirect command file after current instruction is completed.
CONTROL O (^O)	Suppresses or enables (on a toggle basis) console terminal output. Console terminal output is always enabled at the next console terminal input prompt.

Table 2-7 Control/Special Character Descriptions (Cont)

Character	Description
CONTROL U (^U)	^U typed before a line terminator causes the deletion of all characters typed since the last line terminator. The console echoes: ^U <CR> <LF>
RUBOUT	Typing RUBOUT deletes the last character typed on an input line. Only characters typed since the last line terminator can be rubbed out. Several characters can be deleted in sequence by typing successive rubouts. The first rubout echoes as a backslash (\) followed by the character that has been deleted. Subsequent rubouts cause only the deleted character to be echoed. The next character typed that is not a rubout causes another backslash (\) to be printed, followed by the new character to be echoed.
Carriage Return <CR>	Terminates a console command line.

2.7 COMMAND QUALIFIERS AND DEFAULTS

Qualifiers are used within a command to specify the type of addressing and the length of data arguments. Defaults are applied by the console when a command does not contain a qualifier specifying address-type or data length. An operator can specify the radix of a numeric argument by the use of a <local radix override> prefixed to the argument. The console will interpret numeric arguments in the current default radix when an argument is not prefixed by a <local radix override>.

Certain commands permit an address argument to be defaulted. The <default address> used by the console is the next address following the last virtual, physical, or register address accessed by an examine or deposit command. Note that the next address is dependent upon data length, since a byte reference updates the <default address> by 1, while a longword reference updates the <default address> by 4.

The /NEXT qualifier allows an examine or deposit command to operate on more than one address.

2.7.1 Address Type Qualifiers

Address type qualifiers are used within a command line to specify the type of address argument as virtual, physical, ID Bus, V Bus or register address. The qualifiers for the respective types are: /VIRTUAL, /PHYSICAL, /ID BUS, /V BUS, /GENERAL, /INTERNAL.

Virtual addresses that reference nonexistent or nonresident pages will cause the console to abort execution of the console command that referenced the virtual address. In each case an appropriate error message will be typed out on the console terminal.

Example

To examine virtual address 1234, type:

```
EXAMINE/VIRTUAL 1234 <CR>
```

Note that since some register addresses have mnemonic names that are unique and unambiguous, the /GENERAL qualifier need not be specified when mnemonic addresses such as PC, or SP, are referenced.

Example

To examine the VAX-11/78Ø PC, an operator could type either of the following statements:

```
EXAMINE/GENERAL PC <CR> or
E PC <CR> or
E GENERAL F <CR>
```

2.7.2 Address Type Defaults

The console applies an address type default to any command that requires an address argument and does not contain an address qualifier. The default applied can be set by using the set default command.

Example

The command:

```
SET DEFAULT VIRTUAL <CR>
```

will cause the console to default to virtual addressing for any console command that requires an address argument, but does not contain an address type qualifier. Thus, the command EXAMINE 1234 would type out the contents of virtual address 1234.

2.7.3 Data Length Qualifiers

Data length qualifiers are used within a command line to specify the length of the data quantity associated with the command. Data length may be specified as either byte, word, longword, or quadword by means of the /BYTE, /WORD, /LONG, /QUAD qualifiers, respectively.

Example

The command:

```
EXAMINE/BYTE 1234 <CR>
```

will type out the byte at address 1234.

Since VAX-11/780 CPU general and processor registers are longword quantities, all register references will default to longword data length, regardless of the current setting of the data length default.

2.7.4 Data Length Defaults

The console applies a data length default to any command that references data and does not contain a data length qualifier. The default applied can be set using the set default command.

Example

The command:

```
SET DEFAULT WORD <CR>
```

will cause the console to default to word data length.

The command:

```
EXAMINE 1234 <CR>
```

will then reference the word which has its first byte at address 1234.

Since all VAX-11/78Ø CPU general and processor registers are longword quantities, all register references will default to longword data length, regardless of the current setting of the data length default. Word length must be specified when accessing Unibus device registers.

2.7.5 Qualifiers for RADIX

The radix of console output for a command can be sepcified by a qualifier (/OCTAL or /HEX). The qualifier will override the current default radix.

2.7.6 Defaults for RADIX

The default radix for console numeric inputs and outputs is selectable as either HEX or OCTAL via the SET DEFAULT HEX or SET DEFAULT OCTAL command.

2.7.7 Local Radix Override

It is frequently convenient to specify an address or data argument in a radix different from the current default radix. The console allows the current default radix to be overridden by including a <local radix override> as a prefix to any numeric argument. A <local radix override> can be any one of the following:

% O (percent O) for octal arguments

% X (percent X) for hexadecimal arguments

The local radix override must appear as the two leftmost characters of the numeric argument it modifies, and must not be separated by spaces or tabs from that argument.

Example

Assuming that the current default radix is octal, the operator can deposit the octal value 3456 into the hexadecimal address 12A4 using <local radix override> as follows:

```
DEPOSIT %X12A4 3456
```

2.7.8 Default Address Facility

Each time an examine or deposit command is executed, the console computes the address of the next memory location following the location referenced by the command. The address of the next memory location is termed the <default address>, since an examine command that does not specify an address will reference the next address by default. The console computes the <default address> as follows:

<default address> = <address used by last examine or deposit
command> + n, where n is
1 for byte references
2 for word references
4 for longword references
8 for quadword references

The following example shows a sequence of console commands, and the value taken by the default address after each command is executed. Note that the next address is data length dependent, since a byte reference updates the <default address> by 1, while a longword reference updates the <default address> by 4.

Example of default address facility (all numbers are hex):

Command	**<default address> after execution**
EXAMINE/BYTE 2341	2342
EXAMINE/WORD (uses <default address> 2342)	2344
EXAMINE/LONG (uses <default address> 2344)	2348
EXAMINE/GENERAL Ø	general register 1 (R1)
EXAMINE/GENERAL D	general register E (SP)
EXAMINE PC	general register Ø (RØ)

Note that the <default address> is RØ following a PC reference.

2.7.8.1 **Specifying Default Address in a Command** -- The symbol (+) can be used as an address argument in a deposit or examine command to represent the <default address>. This symbol permits depositing to (or examining) successive location without typing the address argument after the first deposit.

Example

To toggle-in a program starting at address 123456, the following deposit commands can be used:

```
DEPOSIT      123456     <DATA>
DEPOSIT      +          <DATA>
DEPOSIT      +          <DATA>
```

Each deposit command, after the first, writes the <DATA> into the next successive memory location.

2.7.8.2 **Last Address Notation** -- The last address referenced (virtual, physical, or register) by an examine or deposit command is denoted by an asterisk (*). The LAST ADDRESS may be used as an argument to an examine or deposit command by typing an asterisk in place of the address argument.

Example

The command:

```
EXAMINE 1234 <CR>
```

will type out the contents of location 1234.

If the next command issued is

```
DEPOSIT * 356 <CR>
```

the console will deposit the number 356 into location 1234.

Examine and deposit commands to VAX-11/78Ø CPU general and processor registers will replace the <last address> with the register address. Mnemonic register names are translated into register addresses by the console.

2.7.8.3 Preceding Address Notation -- The symbol - (minus sign) can be used as an address in a deposit or examine command to specify the location immediately preceding the last location referenced.

2.7.8.4 Use of Last Data as an Address Argument -- The symbol @ can be used as an <address> in a deposit or examine command. The last <data> deposited or examined will be used as the address.

2.7.9 NEXT Qualifier

Syntax: SLASH NEXT [:<COUNT>]

The /NEXT qualifier permits examine and deposit commands to operate on multiple sequential addresses.

The <count> argument specifies the number of additional executions of the command to be performed after the initial execution. The default value for <count> is one.

Example 1

The command:

```
EXAMINE/BYTE 123Ø/NEXT:2
```

is evaluated by the console as follows:

1. The console initially evaluates the command and applies any applicable default values.

2. The command, with applied defaults, is executed. The console types out the contents of location 123Ø, and updates the <default address> to 1231.

3. The /NEXT switch is now evaluated by the console. The console repeats the command operation the number of times indicated by the <count> argument. Each execution uses the <default address> for its address argument and updates the <default address> afterwards. In this example, locations 1231 and 1232 are successively typed out. The final value of the <default address> will be 1233.

Example 2

If the command:

```
EXAMINE/NEXT:2 <CR>
```

is issued following the command in the previous example, the contents of locations 1233, 1234, and 1235 will be typed out. Since the examine command does not contain an address argument, the initial execution of the command will use the current <default address>, which was 1233, following the command in the previous example.

Note that when using the /NEXT qualifier to examine or deposit successive VAX-11/78Ø CPU general registers, the NEXT register after the PC is defined to be RØ.

Example 3

The command:

```
EXAMINE/NEXT:5 GENERAL D
```

will type out the contents of R13, SP, PC, RØ, R1, and R2, in that order.

2.8 COMMAND REPEAT FACILITY

The command repeat facility is provided to allow commands to be executed repeatedly so that CPU logic invoked by console commands can easily be scoped. The following paragraphs describe the repeat facility commands and capabilities.

2.8.1 Repeating Commands

Example

The command:

```
REPEAT EXAMINE 1234 <CR>
```

will continuously examine and display the contents of location 1234.

Once initiated, repeated execution continues until terminated by the operator typing Control C (^C) on the console keyboard.

2.9 COMMAND LINK FACILITY

The console's command link facility allows successive commands to be linked by the console into a single executable list of commands. Once a list of linked commands is constructed, the list can be executed one pass at a time, or executed continuously.

2.9.1 Link Facility Operation

Commands are linked by entering LINK on the console terminal, causing the console to enter the link mode. LINK is then followed by the desired commands. The LINK command is entered only at the beginning of the command string (i.e., at the beginning of the initial command line). Commands to be linked must be entered one-per-line, with each command line terminated by a <CR>. The console then returns a link mode prompt (<<<) requesting the next command. The linking operation is terminated by entering a Control C on the console.

As the command string is entered on the console, the commands are stored in dedicated sectors (limit of 1Ø sectors) on the floppy disk (RXØ1). When the command string is executed, the string is treated as if it were an indirect command file (i.e., command retrieved, parsed, executed, and the next command retrieved, etc.).

The console does not execute commands being linked until a PERFORM command is issued. Once the input of a list of linked commands has been terminated, no further commands can be added to the commands already linked.

The command string can be executed one pass at a time or continuously by means of the PERFORM command. If the PERFORM command is entered after the Control C to terminate the string, the string will execute only one pass. However, if PERFORM is entered before the Control C, that command becomes part of the command list and causes continuous execution of the command list.

Should a linked command be entered incorrectly, the console will output an appropriate error message when the command containing the syntactic error is executed. Typing a Control U (^U) while linking commands will cause the console to reject only the current command line.

2.9.2 Link Facility Usage

```
Syntax:  LINK
         COMMAND <CR>
         COMMAND <CR>
         <^C>
```

Response: Dependent on command list.

Example

The operator wishes to repeatedly examine the contents of a device register after VAX-11/78Ø CPU initialization. Since the CPU initialization requires a certain amount of time to complete, a delay must be inserted between the initialize and examine commands. The sequence of commands is as follows (console output is underlined).

```
>>> LINK INITIALIZE <CR>          LINK causes the console to
                                  enter the link mode and begin
                                  linking.

<<< DELAY 5 <CR>                  Delays five clock ticks to
                                  allow initialize time to
                                  complete.

<<< EXAMINE/LONG FFFFABBC <CR>    Examine command is entered into
                                  string.

<<< PERFORM <CR> <^C>             PERFORM is entered prior to
                                  linking termination.

>>> PERFORM <CR>                  Initiates execution.

P FFFFABBC  12A00123
P FFFFABBC  12A00123, etc.        Command string executed
                                  continuously.
```

2.10 CONSOLE MODE CHANGE

The console I/O mode escape sequence causes the console to switch from console to program I/O mode. The escape sequence to program I/O is:

```
SET TERMINAL PROGRAM <CR>
```

In addition, the console commands START, CONTINUE, and NEXT also enable program I/O mode.

The program I/O mode escape sequence causes the console to switch from program to console I/O mode. The escape sequence to console I/O is:

```
Control P (^P)
```

Note that Control P is not recognized by the console if the console power switch is in either the REMOTE DISABLE position or the LOCAL DISABLE position.

2.11 VMS COMMUNICATION WITH CONSOLE FLOPPY DISK

VMS must be able to read and write the console subsystem's floppy disk drive. These functions are available only when the console is in program I/O mode. The following set of commands are supported by the console software.

a. Write sector -- VMS supplies track, sector, and 128 bytes of data. Console returns status upon completion of write.

b. Read sector -- VMS supplies track and sector. Console returns 128 bytes of data, and status of read operation.

c. Read floppy status -- Console returns floppy status.

d. Write sector with deleted data mark -- VMS supplies track and sector (no data required). Console returns status upon completion of the write.

e. Cancel floppy function -- Console aborts current floppy function.

The following floppy functions will not be directly available to VMS: empty silo, fill silo, read error register, initialize.

While VMS initiated floppy functions are in progress, operator terminal I/O is not disabled. Terminal I/O may be interspersed with floppy I/O.

Once a floppy function is initiated, no other floppy commands will be issued by VMS until the function is complete. The only exception is the command cancel floppy function, which may be issued at any time.

The floppy functions described in this document will only be available to VMS when the console is in program I/O mode (i.e., the console terminal is being used as the system operator's terminal).

NOTE

In the following protocols, two hardware side-effects are implied:

1. **Each time VMS loads the transmit buffer (TXDB), the TX ready bit in the transmit status register (TXCS) is automatically cleared. TXDB is only loaded by VMS, and only when TX ready is set. TX ready is explicitly set by the console when the console is ready to accept another transfer through TXDB.**

2. Each time **VMS** reads the receiver buffer (RXDB), the RX done bit in the receiver status register (RXCS) will automatically clear. RXDB is only read by **VMS**, and only when **RX** done is set. RX done is explicitly set by the console each time the console has loaded RXDB with a character for **VMS**.

2.11.1 Floppy Function Protocol

A. Write sector/write deleted data sector

1. VMS puts the write sector or the write deleted data sector command into TXDB.

2. The console takes the write command, and sets TX ready in TXCS.

3. VMS puts a sector number into TXDB.

4. The console takes the sector number and sets TX ready.

5. VMS puts a track number into TXDB.

6. The console takes the track number and sets TX ready.

7. VMS puts a byte of data into TXDB.

8. The console accepts a byte of data and sets TX ready. Steps 7 and 8 are done 128 times for write sector. Steps 7 and 8 are skipped for write deleted data sector.

9. The console initiates a floppy write function.

10. The floppy write is completed.

11. The console sends a floppy function complete message. The floppy function complete message consists of loading RXDB bits 8--11 with a select code of 2, and bits Ø--7 with the floppy status byte.

12. VMS receives the floppy function completed message.

B. Read sector

1. VMS puts the read sector command into TXDB.

2. The console takes the read command, and sets TX ready in TXCS.

3. VMS puts a sector number into TXDB.

4. The console takes the sector number and sets TX ready.

5. VMS puts a track number into TXDB.

6. The console takes the track number and sets TX ready.

7. The console initiates a floppy read function.

8. The floppy read is completed.

9. The console sends a floppy function complete message. The floppy function complete message consists of a select code of 2 in bits 8--11 of RXDB, and a floppy status byte in bits Ø--7 of RXDB.

1Ø. VMS receives the floppy function completed message.

11. The console puts one byte of data in RXDB and sets RX done.

12. VMS accepts one byte of data from RXDB. Steps 11 and 12 are done 128 times. When the 128th byte is accepted by VMS, the read is complete.

NOTE

If a floppy error occurs on Step 8, Steps 11 and 12 will be skipped.

C. Read status

1. VMS puts the read floppy status command in TXDB.

2. The console takes the read status command and sets TX ready in TXCS.

3. The console gets the floppy status from last floppy function performed.

4. The console puts a floppy function complete message; with the floppy status, into RXDB and sets RX done.

5. VMS reads the floppy status.

D. Terminate floppy function

1. VMS puts the cancel floppy function command in TXDB.

2. The console takes the cancel floppy function.

3. The console terminates the floppy function in progress, if any.

4. The console sets TX ready in TXCS.

2.12 MISCELLANEOUS CONSOLE COMMUNICATIONS

The console software will support certain additional functional communications from VMS and the diagnostic supervisor (VMS/DS).

A. **Examine console memory** -- VMS supplies an offset from the console-supplied base address of examinable memory space. The console returns the examine code and the contents of the requested byte.

Examinable Console Memory Space

(Octal Offset from 376ØØ(8) -- FIRSTF)

+145	Warm-start flag
+146	Cold-start flag
+147	APT-load flag
+15Ø	Last setting of remote and disable
+151	Auto-restart flag
+152	PCS version
+153	WCS primary version
+155	FPLA version

B. Software communication codes

1. Software done -- VMS issues this code to cause the console to resume execution of an indirect-command file that has been suspended due to a wait done command.

2. Warm restart boot command -- The console will boot the VAX-11/78Ø.

3. Clear warm-start and cold-start flags -- VMS/DS issues these codes when the VAX has restarted/rebooted successfully. The console clears the associated flags.

NOTE

The cold and warm restart flags are used by the console to prevent infinite loops when a warm restart results in a VAX-11/78Ø error halt.

2.13 COMMUNICATION REGISTER FORMATS AND SELECT CODES

The LSI-11 processor communicates with the VAX-11/78Ø CPU via two registers on the console interface board. Figure 2-1 shows the register formats and select codes.

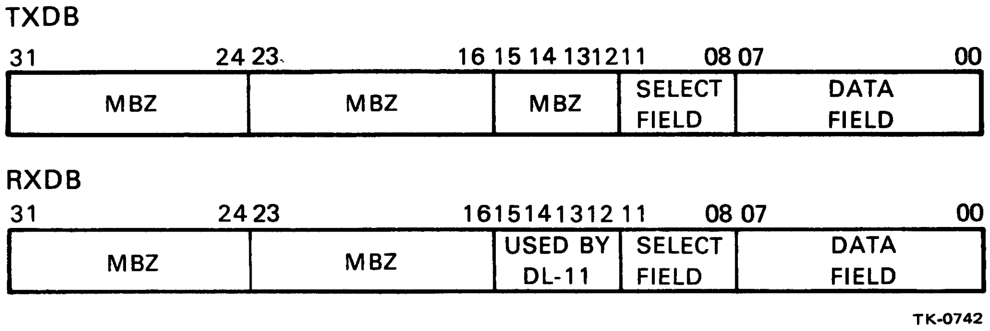


Figure 2-1 Communication Register Formats and Select Codes

Select Field Values (in hex)

Select Code	Device	Data Field Values
Ø	Operator's terminal	Ø through 7F ! ASCII data
1	Floppy drive Ø (data)	Ø through FF -- binary data
2	Floppy function complete	(floppy status)
3	Examine console memory	Address offset/contents of address
9	Floppy drive Ø (command)	Ø = read sector 1 = write sector 2 = read status 3 = write deleted data sector 4 = cancel floppy function 5 = protocol error
F	Misc. communication	1 = software done 2 = boot VAX-11/78Ø 3 = clear warm-start flag 4 = clear cold-start flag

NOTE

Code 5 (protocol error) is sent by the console when one of the following occurs:

1. **Another floppy command (except for cancel floppy function) is issued by VMS before a previous command is completed.**

2. **The console gets a floppy drive Ø code (DATA) when expecting a command.**

2.14 FLOPPY STATUS BYTE DEFINITION

The floppy status byte is used by VMS to determine the success or failure of a read or write operation. The floppy status byte is sent to VMS at the completion of a read, write, or read status operation. The select code is always the floppy function complete (code 2). The status bit assignments are shown in Figure 2-2.

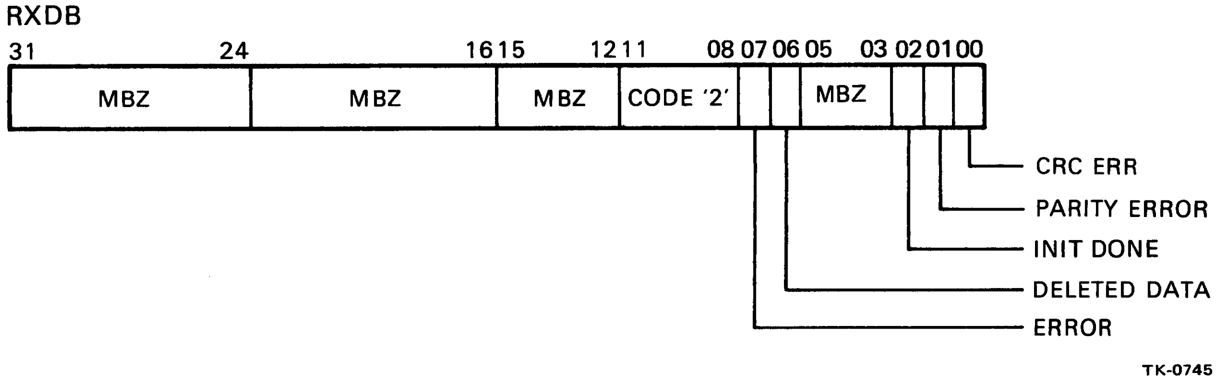


Figure 2-2 Floppy Status Bit Assignments

NOTE

The status bit assignments are identical to those supplied by the floppy controller, except for bit 7. Bit 7 corresponds to bit 15 of the floppy's RXCS register.

2.15 REMOTE CONSOLE ACCESS COMMAND SET

A special set of commands is included in the console command language of systems that use the remote diagnostic facility to facilitate console access from a remote terminal or computer. Commands can be initiated only on the terminal in control, according to the five-position key switch. The remote access command set provides for:

a. A talk mode, to allow communication between local and remote terminal operators (enable talk).

b. A copy control, to permit suppressing or enabling typeout on the local terminal while a remote operator is in control (enable/disable local copy).

c. A method of transferring control of the console between the local and remote operators (enable local control).

d. A method of controlling the echo of characters to the remote terminal while in talk mode (enable/disable echo).

e. A method of suppressing lost carrier error messages caused by a loss of carrier on the remote line (enable/disable carrier error).

f. A method of enabling and disabling use of the console subsystem floppy disk (enable/disable local floppy).

g. A method of enabling and disabling use of the remote floppy disk (enable/disable remote floppy).

2.15.1 Enable Talk Mode Command

Syntax: ENABLE <blank> TALK <CR>

The enable talk command puts the console into talk mode. While in talk mode, characters typed on the remote keyboard are typed on the local terminal, and vice versa. The console does not echo characters back to the originating keyboard, unless the talk mode echo feature has been enabled. No console commands are recognized while in talk mode.

Talk mode is terminated by typing the console escape character (^P) on the terminal in control. When talk mode is terminated, console mode is enabled.

Entering talk mode causes the console to enable the remote serial interface and assert the data terminal ready signal to the data set. All terminal I/O to a program running in the VAX-11/78Ø CPU is disabled while the console is in the talk mode.

2.15.2 Enable/Disable Echo Command

Syntax: (ENABLE ! DISABLE) <blank> ECHO <CR>

The enable echo command will cause the console to echo characters typed on either the remote or local keyboards while in talk mode. The disable echo command causes the console to suppress echo of characters typed on both keyboards.

Enable and disable echo are issued while the console is in console mode, but do not have any effect until talk mode is entered. A disable echo is automatically done each time the console keyswitch is put in the LOCAL/DISABLE position, and on power up of the console.

2.15.3 Enable/Disable Local Copy Command

Syntax: (ENABLE ! DISABLE) <blank. LOCAL <blank> COPY

The enable local copy command causes the local terminal to print a copy of all output directed to the remote terminal. The disable local copy command disables the local terminal from getting a copy of output directed to the remote terminal.

Local copy is automatically disabled each time the console keyswitch is turned to the LOCAL or LOCAL/DISABLE position and on power up of the console. Local copy is automatically enabled each time the console keyswitch is placed in the REMOTE/DISABLE position.

2.15.4 Enable Local Control Command

Syntax: ENABLE <blank> LOCAL <blank> CONTROL <CR>

An enable local control command, issued by the remote terminal operator while the console keyswitch is in the REMOTE position, transfers control of the console to the local terminal operator. (Normal remote operation locks out the local terminal.) This allows a local operator to take control of the console and the VAX-11/780 CPU, while the remote link is maintained. The remote operator may regain control of the console by typing a Control P on the remote keyboard.

An enable local control command issued from the local terminal has no effect. Local control is automatically enabled when the console keyswitch is placed in the LOCAL or LOCAL/DISABLE position, and also on console power up.

2.15.5 Enable/Disable Carrier Error Command

Syntax: (ENABLE ! DISABLE) <blank> CARRIER <blank> ERROR <CR>

The enable carrier error command causes the console to print the message ?CARRIER LOST each time a loss of carrier on the remote line is detected. Also, if the console keyswitch is in the LOCAL or REMOTE position, the console enters talk mode, enabling data terminal ready on the modem.

The disable carrier error command causes the console to inhibit the carrier lost message, and prevents a transition to talk mode. An enable carrier error is automatically done on console power up, and whenever the console keyswitch is placed in the LOCAL or LOCAL/DISABLE position.

2.15.6 ENABLE/DISABLE LOCAL Floppy Command

Syntax: (ENable!DIsable)<BLANK>Local<BLANK>FLoppy

Enable local floppy will cause the directory of the local floppy to be searched first, in an attempt to open a file. If the file is not found, the remote floppy directory is searched. Note that in terms of the console software, the remote floppy is a virtual device. It may be a floppy or it may be some other storage device.

Disable local floppy will cause only the directory of the remote floppy to be searched in an attempt to open a file.

An enable/disable local floppy command affects protocol operation only (transmission format).

2.16 CONSOLE ERROR MESSAGES

This paragraph describes all console error message formats and their interpretation. All console error messages are prefixed by a question mark to distinguish them from informational messages.

2.16.1 Syntactic Error Messages

?<TEXT STRING> IS INCOMPLETE
The <TEXT STRING> is not a complete console command.

?<TEXT STRING> IS INCORRECT
The <TEXT STRING> is not recognized as part of a console command.

?FILE NAME ERR
A <FILENAME> given with a LOAD or @ command cannot be translated to RAD5Ø. One of the characters is not translatable to RAD5Ø or the number of characters allowed is exceeded: six characters for file name, three for extension.

2.16.2 Command Generated Error Messages

?FILE NOT FOUND
A <FILENAME> given with a LOAD or @ command does not match any file on the current floppy disk. This error can also be generated by a HELP or BOOT command if the help file or boot file is missing from the floppy.

?NO CPU RESPONSE
The console timed out while waiting for a response from a VAX-11/78Ø CPU microroutine.

?CPU NOT IN CONSOLE WAIT LOOP, COMMAND ABORTED
A console command that required the assistance of the VAX-11/78Ø CPU was issued when the CPU was not in the console service loop.

?CPU CLOCK STOPPED, COMMAND ABORTED
A console command that requires the CPU clock to be running was issued with the clock stopped.

?IND-COM ERR
An indirect command file error was detected. This error is generated if:

a. An indirect command line exceeds 8Ø characters.
b. An indirect command line does not end with <CR> <LF>.

2.16.3 Microroutine Error Messages

The console uses various microcode routines in the VAX-11/78Ø CPU control store to perform console functions. The following errors are generated by microroutine failures.

?MEM-MAN FAULT, CODE = XX
A virtual examine or deposit caused an error in the memory management microroutine. XX is a one-byte error code supplied by the memory management routine and defined in Table 2-8.

NOTE

Bit positions are numbered from the right.

Table 2-8 Memory Management Error Code Definitions

Bit Position	Definition
Ø	Length violation
1	Fault was on a PTE reference
2	Write or modify intent
3	Access violation
4--7	Ignored

?MICROMACHINE TIME OUT
This message indicates that the VAX-11/78Ø micromachine has failed to strobe interrupts within the maximum time period allowed.

?MIC-ERR ON FUNCTION
An unspecified error occurred in the CPU while servicing a console request. Referencing nonexistent memory will cause this error.

?INT-REG ERR
An error occurred while referencing one of the VAX-11/78Ø CPU internal (processor) registers. Specifying a register address that is too large will cause this error.

?MICROERR, CODE = XX
An unrecognized microerror occurred. The code returned by the CPU is not in the range of recognized error codes. XX is the one-byte error code returned by the microroutine.

2.16.4 CPU Fault Generated Error Messages

?INT-STACK INVLD
The VAX-11/78Ø CPU interrupt stack was marked invalid.

?CPU DBLE-ERR HLT
The VAX-11/78Ø CPU has done a double error halt.

?ILL I/E VEC
An illegal interrupt/exception vector was encountered by the VAX-11/78Ø CPU.

?NO USR WCS
An interrupt/exception vector to WCS was encountered, and no WCS exists.

?CHM ERR
A change-mode instruction was attempted from the interrupt stack.

2.16.5 RXØ1 Error Messages

?FLOPPY ERR, CODE = X
The console floppy driver (a part of the console software) detected an error. X is an error code (in hexadecimal) and is defined in Table 2-9.

Table 2-9 RXØ1 Error Message Code Definitions

Code	Definition
Ø	Floppy hardware error (i.e., CRC, parity, or a floppy firmware detected error).
1	An open file command failed to find the specified file.
2	The floppy driver queue is full.
3	A floppy sector was referenced that is out of the legal range of sector numbers.

?FLOPPY NOT READY
The console floppy drive failed to become ready when booting.

?NO BOOT ON FLOPPY
The console attempted to boot from a floppy that does not contain a valid boot block.

?FLOPPY ERROR ON BOOT
A floppy error was detected while attempting a console boot.

2.16.6 Miscellaneous Error Messages

INT PENDING
This is not actually an error (note absence of ?). The message indicates that an error was pending at the time that a console requested halt was performed. Type CONTINUE to clear interrupt.

?WARNING-WCS and FPLA VER MISMATCH
The microcode in the WCS is not compatible with the FPLA. This message is printed on each ISP START or CONTINUE. However, no other action is taken by the console.

?FATAL-WCS and PCS VER MISMATCH
The microcode in the PCS is not compatible with that in the WCS. ISP START and CONTINUE are disabled by the console.

?REMOTE ACCESS NOT SUPPORTED
This message is printed when the console mode switch enters a REMOTE position, and the remote support software is not included in the console.

?TRAP -4, RESTARTING CONSOLE
The console took a time-out trap. Console will restart.

?UNEXPECTED TRAP
Console trapped to an unused vector. Console reboots when operator types Control C.

?QBLOCKED
Indicates that the console's output queue is blocked. Console will reboot.

CHAPTER 3
DIAGNOSTIC SUPERVISOR AND CONTROL

This chapter describes the basic structure and operating characteristics of the diagnostic supervisor. In addition, operator commands and execution control functions are described. This description is applicable to macro testing, and while many similarities exist for the micros, this chapter does not include them (refer to Chapter 4).

3.1 SUPERVISOR STRUCTURE OVERVIEW

The diagnostic supervisor provides operator control and utility support functions for three diagnostic runtime environments. The three runtime environments are:

a. Cluster Environment (CE): This environment supports the CPU cluster and repair level I/O bus adapter diagnostic programs. The CE consists of program modules that provide utility services (i.e., error reporting, scope loops, etc.), initialization and test dispatch, and operator terminal interface. Additional modules provide load and script management.

b. System Environment (SE): This environment supports the repair level I/O subsystem diagnostic programs, and the device functional test programs. The SE provides the same runtime support functions as the CE. Program modules provide CPU cluster hardware interface support (i.e., real-time clock control, interrupt system control, I/O bus adapter control, etc.).

c. User Environment (UE): This environment supports the I/O subsystem functional level diagnostic programs that run under the VMS operating system as a privileged user task.

I/O services are provided primarily for functional level programs. This allows programs that can execute in an operating system environment, which restricts I/O access, to perform equally well in a standalone environment.

The three supervisor environments are assembled into a common executable module that provides all necessary operator and program services. As shown in Figure 3-1, these services are implemented in two major functional areas: Command Line Interpreter (CLI) and Program Interface (PGI).

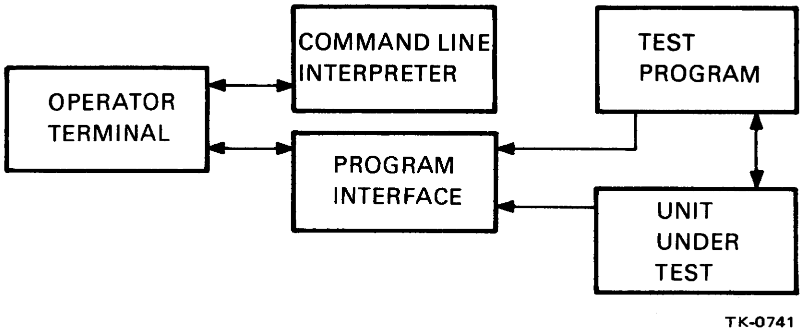


Figure 3-1 Basic Diagnostic Supervisor Structure

The CLI interfaces to an operator (controlling) terminal and enables the operator to control the loading, sequencing, and execution of diagnostic test programs. The CLI monitors all control information passing between the terminal and the supervisor. This information consists of supervisor commands from the operator which control either supervisor or test program operation. The CLI directs control to the appropriate supervisor service module according to the command supplied by the operator.

The PGI provides common services required by all diagnostic test programs. These services include operator interaction, program control, error message formatting, memory management, and I/O request handling. Note that the operator can communicate with the diagnostic program only through a PGI message service or with the CLI directly.

When the operator initiates diagnostic program execution (through the CLI), that program assumes control. Once program execution begins, the PGI handles all test information flowing between the terminal or the Unit Under Test (UUT) (i.e., QI/O-I/O driver interface) and a functional level program. For repair level programs, test information flows directly between the UUT and the diagnostic program (i.e., direct test program access to I/O registers).

Test control information flow between the terminal and the diagnostic program consists mainly of test parameter requests and responses, while test information flow between the UUT and the program consists mainly of test stimuli and responses.

The diagnostic program executes until the test sequence is completed, aborted, or the operator enters the appropriate control character, at which point control returns to the CLI.

System errors not directly related to the UUT are handled by the supervisor. Unless the program explicitly requests notification, these errors are transparent to the program and are reported directly to the operator.

3.2 CLI FUNCTIONAL MODULE DESCRIPTION

The CLI consists of a tree-structured command decoder and several service modules that execute the operator's commands (e.g., loading a diagnostic program from the system device; altering the operational characteristics of the program; or driving the CLI through a script file). The command syntax is a subset of the console command language.

Once a command line is interpreted, the CLI dispatches control to the appropriate service module. After the operator's command has been completed or aborted, control is returned to the CLI. Certain CLI service modules pass control to the diagnostic program, rather than back to the CLI. However, the CLI continues to monitor the operator's terminal for certain commands (e.g., ^C).

3.2.1 Image Loader Module

The image loader allows the operator to specify a load device and a file name for loading diagnostic programs. Depending on the environment (i.e., console, system, or user), the device media will be either the diagnostic load device (console floppy) or the system load device.

3.2.2 Test Sequence Control Module

The test sequence control module provides the operator with the capability to control the order in which tests within a program are executed. This is implemented by specifying test numbers in the run, start, and restart supervisor commands (Paragraph 3.4).

3.2.3 Script Processor Module

Automatic test sequence control is achieved through the use of a script file. This script file is a line-oriented ASCII file that contains standard CLI supervisor commands. To allow for commenting on a command line, any text following a (!) on a line is ignored by the script processor. Blank lines and extraneous spacing characters are also ignored.

A script file may contain CLI supervisor commands (Paragraph 3.5) only, or a combination of commands and program parameter responses. Generating script or parameter files is performed off-line using a standard editor system utility.

3.3 PGI FUNCTIONAL MODULE DESCRIPTION

The major function of the PGI module is to handle all information flowing between the operator's terminal or the UUT and the functional level I/O program. While a diagnostic program is executing, that program can call on the supervisor to supply various services. These services provide the program with the required common functions (e.g., memory allocation and mapping, I/O processing, operator terminal interfacing, error message formatting, and system error handling).

Several of the functions the PGI provides are a subset of the VMS system services. For example, the supervisor provides the VMS queue I/O service so that user mode diagnostics may be executed standalone as well as under VMS.

3.3.1 Memory Management and Adapter Services

All memory buffer allocation is performed by the diagnostic supervisor. This ensures system integrity throughout the various operating environments.

All necessary interfacing between the CPU and UUT will be handled by the diagnostic supervisor.

Running standalone, the supervisor provides I/O services similar to those available under VMS. This provides a smaller standalone environment for running user mode diagnostics. Only a small kernel subset of VMS system services is provided.

All functional level diagnostics perform device I/O as specified by the VMS operating system. However, in addition to the normal queue I/O functions, VMS provides special features that diagnostic programs can use if executing as privileged processes. On I/O completion, if requested and privilege permitting, raw status is deposited into a buffer specified by the program. This status contains all device registers and pertinent channel registers. A time stamp is also deposited into the status buffer.

3.3.2 Operator Terminal Services

Since the diagnostic programs do not interface directly with the operator's terminal, the supervisor provides all required operator communication services for the diagnostic program. The program can perform operator dialogue through a supervisor service to allow testing of mechanical devices that require operator interaction.

The terminal drivers within the service eliminate the need for the diagnostic programmer to be aware of the type of terminal currently used by the operator.

The output to the operator, including error reporting, uses formatted ASCII output to simplify the program's message-sending routines. Conversion of binary data to ASCII display is handled by the diagnostic supervisor instead of the programmer. The formatted ASCII output syntax is the same as that used by VMS.

3.3.3 System Error Handling

All system errors are intercepted and reported directly to the operator by the supervisor unless the program explicitly requests notification of exceptions or interrupts.

3.4 SUPERVISOR COMMAND DESCRIPTIONS

The following paragraphs describe the operator command and execution control functions provided by the diagnostic supervisor. Where appropriate, examples of command usage are included.

3.4.1 Command Terms and Symbols

Since the supervisor commands are a subset of the console commands, many of the console command terms and symbols are used in the symbolic supervisor command descriptions. The applicable characters are defined in Table 3-1.

Table 3-1 Term and Symbol Definitions

Term/Symbol	Definition
!	Used to indicate the Exclusive OR operation (i.e., selection of parameters within a command line)
( )	Used to indicate that one of the syntactic units of the expression is to be selected
< >	Used to indicate symbolic arguments, or program functions to/from the operator terminal
[ ]	Used to indicate that part of an expression is optional; e.g., WAIT [<blank> <count>] indicates that the wait command takes an optional <count> argument
<blank>	Represents one or more spaces
<tab>	Represents one or more tabs
<count>	Represents a numeric count
<XYZ - list>	Indicates one or more occurrences from the category indicated by XYZ
<address>	Represents an address argument
<data>	Represents a numeric argument
<qualifier>	Represents a command modifier (switch)
<input prompt>	Represents the console's input prompt string
>>>	Console program input prompt character
DS>	Diagnostic supervisor prompt
<CR>	Represents a console terminal carriage return
<LF>	Represents a console terminal line feed
/	Delimits a command from its qualifiers
,	Used as a separator within a list
:	Used as a separator within a command line.

3.4.2 Command Description Segments

Each supervisor command description is divided into three, four, or five descriptive segments, depending on the particular command. The descriptive segments are:

a. Syntax: describes the command structure.

b. Command description: a brief paragraph describing command operation, general restrictions, or available options.

c. Response: a description of the console program response to the specified command.

d. Qualifiers: a list of applicable command modifiers.

e. Options: a list of applicable command options.

The descriptive segments use the terms and symbols defined in Table 3-1. Note that every command (or command line) must be terminated with a <CR>.

3.4.3 Command Abbreviations

Supervisor command words, switches, and arguments may be abbreviated by typing only enough characters to uniquely identify each item. For example, the load command can be specified by L, while the start command requires a minimum entry of ST.

3.4.4 Command Overview

The supervisor operator commands are divided into the three following groups:

a. Load/test sequence control: provides the operator with the capability of loading and sequencing diagnostic programs.

b. Execution control: provides the operator with control of the operational characteristics of the diagnostic program and/or supervisor (e.g., looping, error reporting, etc.).

c. Debug/utility functions: provide the operator with debug and utility functions such as: breakpoints, examine, deposit, etc.

The supervisors also support operator terminal characteristics such as width, fill, etc. In addition, all control functions provided by the console (e.g., Control C) are also supported by the supervisor.

3.5 SEQUENCE CONTROL COMMANDS

The program/test sequence control commands provide the operator with the capability of loading and controlling the sequencing of diagnostic programs, as well as the capability of controlling the sequence of test execution within a program. The supervisor also provides for the execution of a single subtest, and if the pass count option is used, provides a loop-on-subtest capability.

The submit command allows an entire diagnostic test session to be predefined by the operator. The supervisor is then capable of performing the test session without operator assistance.

Note that the symbolic argument <file spec> as used in the following subsection is defined as:

```
        dev unit :[UIC] filename . ext
```

3.5.1 Load Command

Syntax: LOAD <file spec> [/PHYSICAL : <address>] <CR>

The load command causes the specified file to be loaded into memory. The supervisor obtains sufficient information from the program.

After a successful load, the supervisor prints out the following message: Progname-r.p LOADED.

> Progname is the program name. This is the internal name which the supervisor extracts from the program header section.
>
> --r.p is the release version number and the DEPO (patch) number of the program.

The optional PHYSICAL switch directs the image loader to attempt to load the program into physically contiguous memory starting at <address>. The <address> argument is normally accepted in hexadecimal format by default.

3.5.2 Start Command

Syntax: START [/SEC : <section name>] -- [/TEST : <first>] [:<last> !/SUBTEST : <number>]] -- [/PASS : <count>] <CR>

The start command causes the program in memory to begin execution. As execution begins, the supervisor enters into a dialogue with the operator to determine the program specific parameters. (e.g., which units to test). The command switches and certain arguments are optional.

The SECTION switch is program specific and not available for use with all programs. When a section is selected, only the tests that it contains will be executed.

The TEST switch is used in two distinctly different ways.

a. If the <first> and <last> arguments are specified, the supervisor sequentially passes control to tests <first> through <last> inclusively.

b. If the <first> argument is combined with the SUBTEST switch, program execution begins at the beginning of the <first> test and terminates at the end of SUBTEST <number>.

If the SUBTEST switch is used in conjunction with the PASS switch, the operator is provided with a loop-on-subtest capability. If the optional PASS switch is not specified, a default <count> of one is assumed.

If the TEST switch is not specified, all tests within the program are executed. If only the <first> argument is specified with the TEST switch, the <last> argument is assumed by default to be the highest numbered test within the program.

3.5.3 Restart Command

Syntax: RESTART [/SEC: <section name>] -- [/TEST : <first> [:<last> ! /SUBTEST : <number>]] -- [/PASS : <count>] <CR>

The restart command is similar to the start command; however, the supervisor does not enter into the parameter retrieval dialogue. This command requires that the program P-Tables have been previously setup with a start command. Switch syntax is identical to the start command switches.

3.5.4 Run Command

Syntax: RUN<file spec> [/SEC: <section-name>]! -- [/TEST : <first> [: <last>! -- /SUBTEST : <number>]] [/PASS : <count>]

The run command is equivalent to a load and start command sequence. (Refer to Paragraph 3.5.2 for a description of the optional switches.)

3.5.5 Control Characters and Special Characters

Table 3-2 contains a description of the control and special characters recognized by the supervisor.

Table 3-2 Control/Special Character Descriptions

Character	Description
Control C (^C)	Returns program control to the supervisor which enters a command wait state. The operator may then issue any valid supervisor command.
Control O (^O)	Suppresses or enables (on a toggle basis) console terminal output. Console terminal output is always enabled at the next console terminal supervisor input prompt. However, the supervisor will override ^O and reinstate an active output status to the operator terminal when it is servicing system errors, CLI prompts, or forced messages.
Control U (^U)	^U typed before a line terminator causes the deletion of all characters entered since the last line termination. The console echoes: ^U/<CR><LF>
Rubout	Typing rubout deletes the last character typed on the input line. Only characters entered since the last line terminator can be rubbed out. Several characters can be deleted in sequence by typing successive rubouts. The first rubout echoes as a backslash (\) followed by the character which has been deleted. Subsequent rubouts cause only the deleted character to be echoed. The next character typed that is not a rubout causes another (\) to be printed, followed by the new character to be echoed.
Carriage Return (CR)	Terminates a command line.

3.5.6 Continue Command
Syntax: CONTINUE <CR>

The continue command causes program execution to resume at the point at which the program was suspended. This command is used to proceed from a breakpoint, error halt, or Control C situation.

3.5.7 Summary Command
Syntax: SUMMARY <CR>

The summary command causes the execution of the program's summary report code section which prints statistical reports.

3.5.8 Abort Command

Syntax: ABORT <CR>

The abort command executes the program's cleanup code and returns control to the supervisor, which enters a command wait state. At this point the operator may issue any command except restart or continue.

3.5.9 Submit Command

Syntax: SUBMIT <file spec> [/LOG : ON ! OFF] [/CONSOLE : ON! OFF] <CR>

The submit command causes the supervisor to read a script file from any file-oriented device. The supervisor performs the functions outlined in the script file and then returns control to the operator at the console.

The script file may contain any valid operator commands, including a submit command. However, a submit command within a script file is considered a terminal command (i.e., the supervisor will close the current script and log files and open new ones as specified by the current command).

If the LOG switch is specified as ON, a transcript of the indirect terminal dialogue is maintained in a file of the same filename as the script file with an extension of .LOG on the device where the script file is located. The default for this switch is OFF.

If the CONSOLE switch is specified as ON, the terminal dialogue generated by the script file is printed on the operator's terminal. The default for this switch is ON.

3.6 EXECUTION CONTROL COMMANDS

This group of commands allows the operator to statically or dynamically alter the operational characteristics of the diagnostic program and/or the supervisor. These functions are implemented by flags that reside in both the supervisor and the program. The event flags are located within the diagnostic program and are supported by VMS and the supervisor.

These commands are used to control the printing of error messages, ringing the bell, halting and looping of the program, etc. Flags are provided that indicate to the supervisor which type of dialogue characteristics are desired by the operator. The operator also has access to a subset of the event flags that are available to the program.

3.6.1 Set Control Flag Command

Syntax: SET [FLAGS] <argument list> <CR>

This command sets the execution control flags specified by <argument list>; no other flags are affected. <argument list> is a string of flag mnemonics separated by commas. The applicable flags are described in Table 3-3.

Table 3-3 Control Flag Descriptions

Flag	Description
HALT	Halt on error detection. When the program detects a failure, with this flag set, the supervisor enters a command wait state after all error messages associated with the failure have been output. The operator may then continue, restart, or abort the program. This flag takes precedence over the LOOP flag.
LOOP	Loop on error. When set, this flag causes the program to enter a predetermined scope loop on a test or subtest that detects a failure. Looping will continue until the operator returns to the supervisor by using ^C. The operator may then continue, clear the flag and continue, restart, or abort the program.
BELL	Bell on error. When set, this flag will cause the supervisor to output a bell to the operator whenever the program detects a failure.
IE1	Inhibit error messages at level 1. When set, this flag suppresses all error messages except those that are forced by the program or supervisor.
IE2	Inhibit error messages at level 2. When set, this flag suppresses basic and extended information concerning the failure. Only the header information message (the first three lines) is output for each failure.
IE3	Inhibit error messages at level 3. When set, this flag suppresses extended information concerning the failure. The header and basic information messages are output for each failure.
IES	Inhibit summary report. When set, this flag suppresses statistical report messages.
QUICK	Quick verify. When set, this flag indicates to the program that the operator desires a quick verify mode of operation.

Table 3-3 Control Flag Descriptions (Cont)

Flag	Description
SPOOL	List error messages on line printer. When set, this flag causes the supervisor to direct all program messages to the line printer. In the VMS environment, the messages are not actually printed but entered into a file on disk (not yet implemented).
TRACE	Report the execution of each test. When set, this flag causes the supervisor to report the execution of each individual test within the program as the supervisor dispatches to that test.
LOCK	Lock in physical memory. When set, this flag disables the program relocation function. Self-relocating programs are then locked into their current physical memory space.
OPERATOR	Operator present. When set, this flag indicates to the supervisor that operator interaction is possible. When cleared, the supervisor takes appropriate actions to ensure that the test session bypasses any tests that require manual intervention.
PROMPT	Display long dialogue. When set, this flag indicates to the supervisor that the operator wants to see the limits and defaults for all questions printed by the program.
ALL	All flags in this list.

3.6.2 Clear Control Flag Command

Syntax: CLEAR [FLAGS] <argument list> <CR>

The clear command clears the flags specified by <argument list>; no other flags are affected. The <argument list> is a string of flag mnemonics separated by commas. The supported arguments are described in Table 3-3.

3.6.3 Set Control Flag Default Command

Syntax: SET FLAGS DEFAULT <CR>

This command returns all flags to their initial default status. The default flag settings are OPERATOR and PROMPT.

3.6.4 Show Control Flags Command

Syntax: SHOW FLAGS <CR>

This command causes the display of all execution control flags and their current status. The flags are displayed as two mnemonic lists: one for set flags, one for clear flags.

3.6.5 Set Event Flags Command

Syntax: SET EVENT [FLAGS] <argument list> ! ALL <CR>

This command sets those event flags specified by <argument list>; no other event flags are affected. The <argument list> is a string of flag numbers in the range 1--23, separated by commas. The optional ALL may be specified instead of <argument list>.

Event related services are provided by the supervisor to provide intraprocess synchronization and signaling by means of event flags. Event flags are located in clusters of 32 flags each.

The supervisor provides two event flag clusters. Event flags are specified by the numbers Ø--63. However, flags 24--31 are restricted for use by VMS. The operator has the capability to interactively set and clear flags 1--23.

Note that numbers 32--63 are for program use. Number Ø is used by the supervisor.

3.6.6 Clear Event Flags Command

Syntax: CLEAR EVENT [FLAGS] <argument list> ! ALL <CR>

This command clears those event flags specified by <argument list>; no other event flags are affected. The optional ALL may be specified instead of <argument list>.

3.6.7 Show Event Flags Command

Syntax: SHOW EVENT [FLAGS] <CR>

This command causes the display of a list of the event flags currently set.

3.7 DEBUG AND UTILITY COMMANDS

This group of commands provides the operator with the ability to alter diagnostic program code. The supervisor allows up to 15 simultaneous breakpoints within the program. The operator can also examine and/or modify the program image in memory. Optionally, a modified image can be written to a load device so that patching need occur only once.

Another feature allows the operator to unconditionally list any or all of the program error messages.

3.7.1 Set Base Command

Syntax: SET BASE <address> <CR>

This command loads the address specified into a software register. This number is then used as a base to which the address specified in the set breakpoint, clear breakpoint, examine, and deposit commands is added. The set base command is useful when referencing code in the diagnostic program listings. The base should be set to the base address (see the program link map) of the program section referenced. Then the PC numbers provided in the listings can be used directly in referencing locations in the program sections.

NOTE

Virtual address = physical address (normally) when memory management is turned off.

3.7.2 Set Breakpoint Command

Syntax: SET BREAKPOINT <address> <CR>

This command causes control to pass to the supervisor when program execution encounters the <address> previously specified by this command. A maximum of 15 simultaneous breakpoints can be set within the diagnostic program.

3.7.3 Clear Breakpoint Command

Syntax: CLEAR BREAKPOINT <address> ! ALL <CR>

This command clears the previously set breakpoint at the memory location specified by <address>. If no breakpoint existed at the specified address, no error message is given. An optional argument of all clears all previously defined breakpoints.

3.7.4 Show Breakpoints Command

Syntax: SHOW BREAKPOINTS <CR>

This command displays all currently defined breakpoints.

3.7.5 Set Default Command

Syntax: SET DEFAULT <argument list> <CR>

This command causes setting of default qualifiers for the examine and deposit commands. The <argument list> argument consists of a data length default and/or radix default qualifiers. If both qualifiers are present, they are separated by a comma. If only one default qualifier is specified, the other one is not affected. Default defaults are HEX and LONG. Default qualifiers are:

Data Length: Byte, Word, Long
Radix: Hexadecimal, Decimal, Octal

3.7.6 Examine Command

Syntax: EXAMINE [<qualifiers>] [<address>] <CR>

The examine command displays the contents of memory in the format described by the qualifiers. If no qualifiers are specified, the default qualifiers set by a previous default command are implemented. The applicable qualifiers are described in Table 3-4.

Table 3-4 Qualifier Descriptions

Qualifier	Description
/B	Address points to a byte
/W	Address points to a word
/L	Address points to a longword
/X	Display in hexadecimal radix
/D	Display in decimal radix
/O	Display in octal radix
/A	Display in ASCII bytes

When specified, the <address> argument is accepted in hexadecimal format unless some other radix has been set with the set default command. Optionally, <address> may be specified by immediately preceding the address argument with %D OR %O, respectively. <Address> may also be one of the following: RØ--R11, AP, FP, SP, PC, PSL.

3.7.7 Deposit Command

Syntax: DEPOSIT [<qualifiers>] <address> <data> <CR>

The command accepts data and writes it into the memory location specified by <address> in the format described by the qualifiers. If no qualifiers are specified, the default qualifiers are implemented. The applicable qualifiers are identical to those of the examine command and described in Table 3-4.

The <address> argument is accepted in hexadecimal format unless some other radix has been set with the set default command. Optionally, <address> may be specified as decimal or octal by immediately preceding <address> with %D or %O, respectively.

CHAPTER 4
MICRODIAGNOSTIC DESCRIPTION

4.1 MICRODIAGNOSTIC PROGRAM OVERVIEW

The microdiagnostic programs provide module isolation for logic failures within the CPU, floating-point, and MDS memory controllers. All detected failures result in an error printout indicating the module, or smallest set of modules, to which the microdiagnostics can isolate the failure.

The microdiagnostic package consists of two major test divisions: console adapter and hardcore, and microtests. Each test division is controlled by an associated monitor that provides non-diagnostic services to that division.

a. Hardcore Monitor -- Console Adapter and Hardcore Program
b. Microtest Monitor -- Microtest Program

Both test division monitors are serviced by the console-resident microdiagnostic monitor. In addition to loading the hardcore and microtest monitors, the microdiagnostic monitor allows the operator test selection and execution options (Paragraph 4.6). In order to reduce the address space required to execute the hardcore and microtest programs, the common code of both programs has been incorporated into the microdiagnostic monitor. That code, which is unique to either the hardcore tests or microtests, has been incorporated into the associated monitors.

The microdiagnostics reside on diskettes for the floppy drive. The basic test sequence is: 1) hardcore tests, 2) microtests. The hardcore tests verify the operation of the minimum logic required to reliably execute the microtests. The minimum logic consists of the basic hardware elements required for data transfer and error reporting.

The code, data, and structure required by the microdiagnostics prohibit them from being resident in the LSI-11 address space at any one time. The hardcore tests are executed out of a small buffer area in the LSI-11 memory. The microtests are executed out of the WCS of the VAX-11/78Ø CPU.

4.2 BASIC PROGRAM EXECUTION

With the console program resident (in LSI-11 memory), the operator can execute the entire microdiagnostic package by issuing the test command. The console program overlays itself with the microdiagnostic monitor from the console floppy. (However, the floppy and terminal software drivers are not overlaid since they provide utility service to each of the monitors.)

In turn, the microdiagnostic monitor transfers the hardcore monitor into the LSI-11 memory from the floppy. The hardcore tests are then executed sequentially out of the buffer in the LSI-11 memory. On completion of the hardcore tests, the hardcore monitor notifies the microdiagnostic monitor. The microdiagnostic monitor, in turn, transfers the microtest monitor into the LSI-11 memory (from floppy). The microtest monitor then executes the microtests out of the WCS in approximately 1K microword overlays. On completion of the microtests, control is returned to the console program.

Figure 4-1 illustrates program residency in the LSI-11 memory. Note that those items on a horizontal line are exclusive in memory; e.g., the console program or the microdiagnostic monitor may be resident, but not both. As previously mentioned, the floppy and terminal drivers (and software bootstrap) are always resident.

4.3 BASIC TEST STRATEGY

The basic test strategy is to transfer data from a test source and load it into the logic element under test. The next step is to retrieve the data from that element and compare it with the original data loaded. Depending on the test requirements, logic element structure, and functional location, the retrieved data may or may not have a true compare. In either case, the fail/no fail decision is based on the expected results. In some tests, the same logic is tested using an array of data patterns.

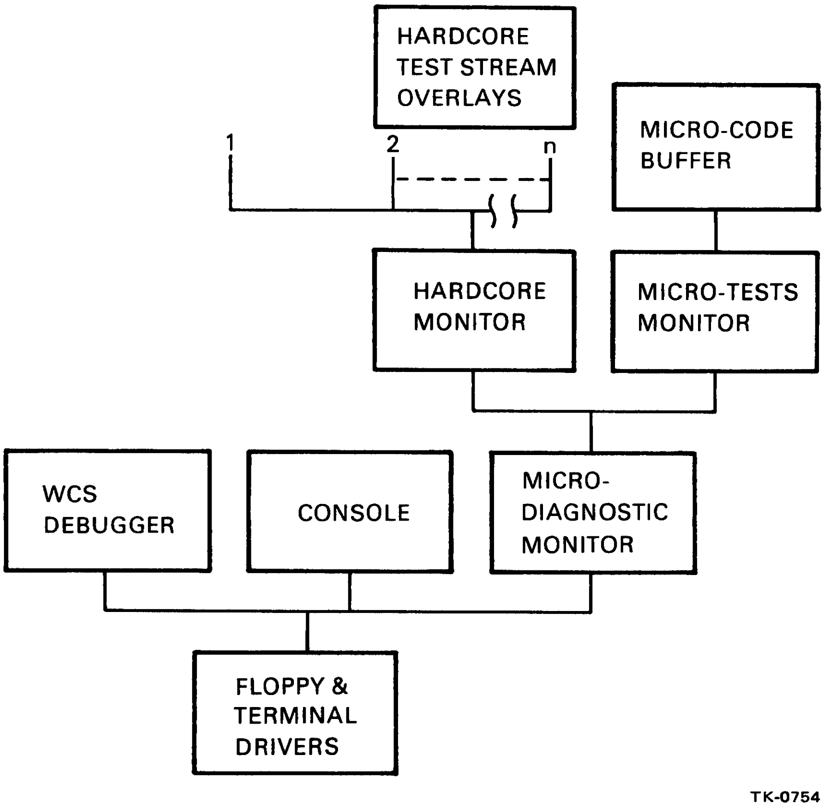


Figure 4-1 LSI-11 Memory Program Residency

It is essential to the test strategy that the basic load and error reporting paths are initially tested for reliable operation. In an error-free situation, the microdiagnostic can notify the user when a test is completed. In the case of error detection, the microdiagnostic can identify for the user: the failed module and test, the data pattern used, and the expected test result.

A simplified test procedure is illustrated in Figure 4-2. Note that a true compare is not necessarily the expected result.

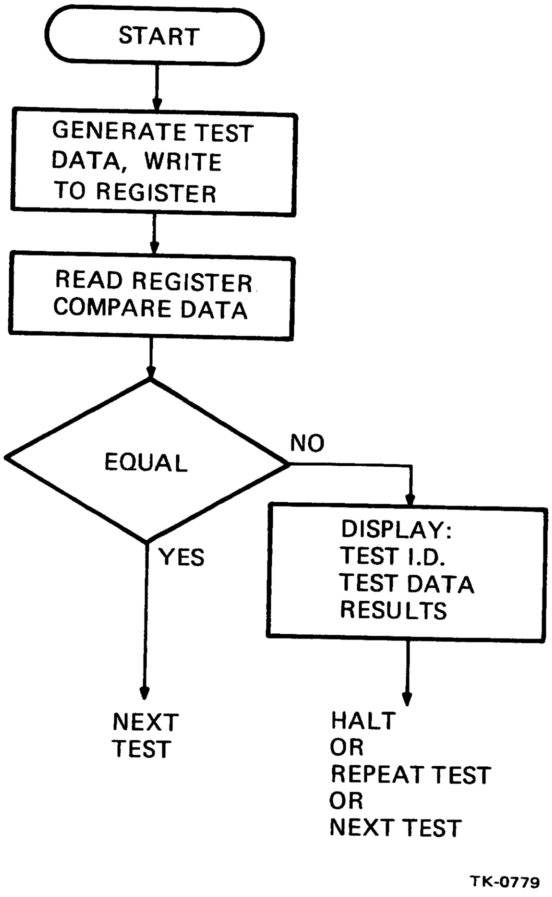

START
GENERATE TEST DATA, WRITE TO REGISTER
READ REGISTER COMPARE DATA
EQUAL
NO
YES
DISPLAY: TEST I.D. TEST DATA RESULTS
NEXT TEST
HALT OR REPEAT TEST OR NEXT TEST
TK-0779

Figure 4-2 Simplified Microdiagnostic Test Procedure

4.4 HARDCORE TEST DESCRIPTION

The hardcore tests are the initial set of microdiagnostic tests executed. Paragraph 4.4.1 describes the hardcore test structure.

The hardcore tests initially check the control and data registers of the Console Interface Board (CIB). This ensures test access to the VAX ID Bus. After the CIB tests, the clock board is tested. The clock is turned on and off, single-stepped, and certain clock function status is retrieved over the Visibility Bus (V Bus).

The next element tested is the microsequencer. For example, data is transferred onto the microstack and then retrieved. An address is placed on the microstack. The maintenance return feature is then used to pop the address off the microstack and load it into the micro PC. This allows the microaddress paths to be tested in small segments.

As shown in Figure 4-3, the remainder of the hardcore test sequence tests WCS, PROM Control Store (PCS), and basic elements of the data path. Generally, these are address integrity and parity checking tests. The WCS is tested by writing a variety of address and data patterns to it, and checking for good parity, or forced bad parity.

The basic elements of the data path are tested by writing data to certain registers, reading that data back, and comparing the results. The data path is tested for its ability to transmit and retain expected data. The basic capabilities of the Arithmetic Logic Unit (ALU) to transfer and compare data are tested. The scratch pads are also tested for retaining data; the scratch pads are used to hold error data in the case of error detection.

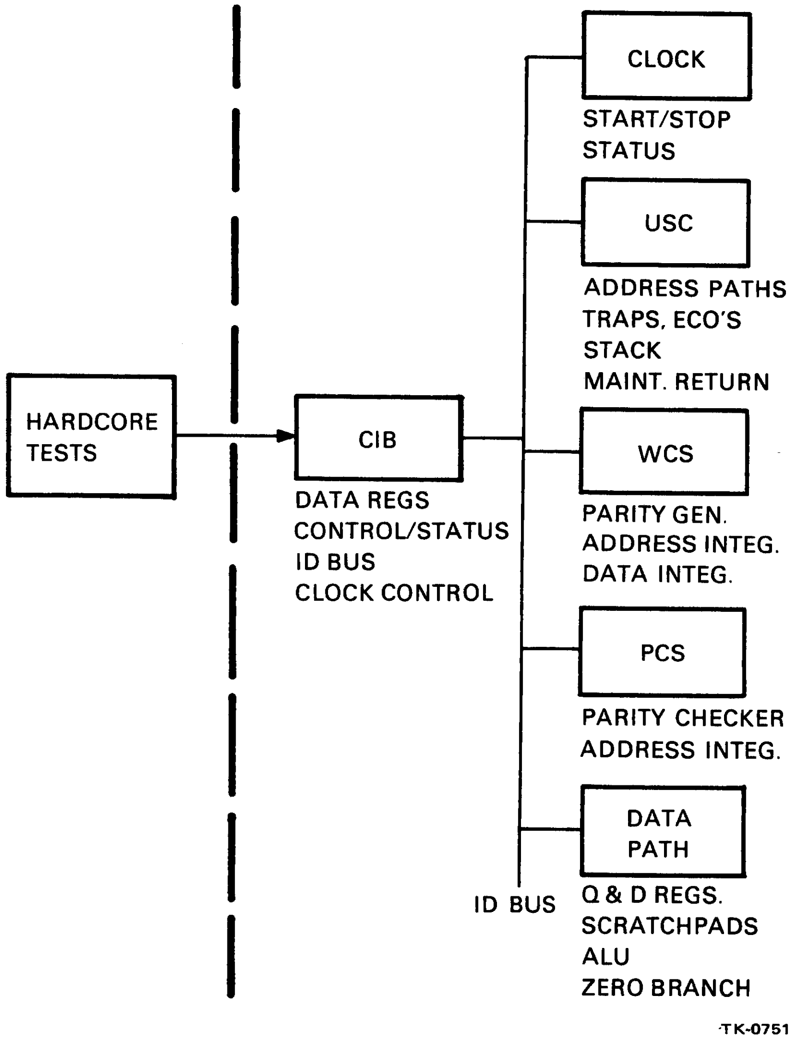


Figure 4-3 Hardcore Test Sequence

4.4.1 Hardcore Test Structure

Because of the limited LSI-11 memory address space, the hardcore tests are sequentially loaded from the floppy and executed out of a 1.5K byte buffer in the LSI-11 memory. The hardcore tests are implemented using special pseudo instructions. The pseudo instructions are actually functional statements, where each statement produces a table of parameters that resemble op codes and operand addresses.

The hardcore monitor contains a software PC which, in effect, is a pointer into the tables. Based on the content of the op codes and operands, the monitor calls subroutines that are written in PDP-11 code to perform the operation required by a specific test. Implementing test code in this manner allows a large test functionality to reside in a small address space.

4.4.2 Pseudo Instruction Description

The following paragraphs describe the hardcore test pseudo instructions and their associated statement formats. Table 4-1 defines the symbols and abbreviations used in describing the statement formats.

Table 4-1 Instruction Symbol/Abbreviation Definitions

Item	Definition
< >	Used to denote a category name or argument within a functional statement, e.g., <SCR ADDRESS> represents a valid source address.
[ ]	Used to indicate that part of a functional statement that is optional, e.g., [<WCS ADDRESS INDEX>], represents an address index value that may or may not be specified depending on the functional statement.
,	Used to separate category names or arguments within a functional statement.
SCR	Abbreviation for source.
DST	Abbreviation for destination.
I,J,K	Legal index names.

Each pseudo instruction description is divided into two descriptive segments. The format segment describes the statement format using the symbols defined in Table 4-1. The instruction description is a brief paragraph describing general command operation and the available options. Each instruction description is preceded by the instruction mnemonic in boldface type.

BLKMIC

```
BLKMIC  <SCR ADDRESS>, <SCR INDEX>, <WCS ADDRESS>,
        <WORD COUNT>, [<WCS ADDRESS INDEX>]
```

Move the <WORD COUNT> number of 96-bit microwords from the <SCR ADDRESS>, indexed by <SCR INDEX>, to the WCS starting at <WCS ADDRESS>, indexed by <WCS ADDRESS INDEX>. If an <SCR INDEX> is specified, the <SCR ADDRESS> is indexed by six PDP-11 words (i.e., 96 bits).

If the <WCS ADDRESS> starts with an alpha character, the <WCS ADDRESS> is used as a pointer to a table in the test data area of the test. Otherwise, it is used as a physical WCS address.

For example, if the current value of the index is 2, 14_8 (<SCR INDEX> * 6) would be added to the <SCR ADDRESS> to find the first 96-bit microword to load into the WCS.

CHKPNT

CHKPNT [<PASS ADDRESS>], [<FAIL ADDRESS>]

If the error flag, set during a compare instruction (see CMPXXX instructions), is zero, go to the <PASS ADDRESS>. If the error flag is not zero, go to the <FAIL ADDRESS>. If neither a pass nor a fail address is specified, go to the next instruction in line.

The address of the next instruction is typed. These addresses appear on the typed line named TRACE (Figure 4-1Ø).

CLOCK

CLOCK <TIMES>

Step the system clock <TIMES> number of single time states. If <TIMES> is an integral number of four, single bus cycles are executed for each four <TIMES>.

CMPCA

CMPCA [<MODE>], <REGISTER>, <DST ADDRESS>, [<DST ADDRESS INDEX>]

Compare the contents of the console register specified by <REGISTER> with the contents of the location specified by <DST ADDRESS>, indexed by <DST ADDRESS INDEX>. The <MODE> argument is generally EQUAL. If left blank, the default for <MODE> is EQUAL.

If the comparison is false, set the error flag. If the <MODE> argument is not specified, it defaults to EQUAL.

If the <REGISTER> argument is specified as IDREGLO or IDREGHI, the register used in the comparison is the ID Bus register that was read in the most recent READID instruction.

CMPCAD

CMPCAD [<MODE>], <REGISTER>, <DST ADDRESS>, [<DST ADDRESS INDEX>]

Compare the contents of the console registers specified by <REGISTER> and <REGISTER>+2 with the contents of the location specified by <DST ADDRESS> and <DST ADDRESS>+2, indexed by <DST ADDRESS INDEX>.

If the comparison is false, set the error flag. If the <MODE> argument is not specified, it defaults to EQUAL.

If the <REGISTER> argument is specified as IDREGLO or IDREGHI, the register used in the comparison is the ID Bus register that was read in the most recent READID instruction.

CMPCAM

```
CMPCAM  [<MODE>], <REGISTER>, <MASK ADDRESS>, [<MASK ADDRESS
        INDEX>], <DST ADDRESS>, [<DST ADDRESS INDEX>]
```

Take the content of the console register specified by <REGISTER>, mask it with the content of the <MASK ADDRESS>, indexed by <MASK ADDRESS INDEX>, and compare it with the content of <DST ADDRESS>, indexed by <DST ADDRESS INDEX>.

If the comparison is false, set the error flag. If the <MODE> argument is not specified, it defaults to EQUAL.

If the <REGISTER> argument is specified as IDREGLO or IDREGHI, the register used in the comparison is the ID Bus register that was read in the most recent READIN instruction.

The mask is performed by taking the content of <MASK ADDRESS>, indexed by <MASK ADDRESS INDEX>, complementing it, and bit-clearing the contents of <REGISTER> with it.

CMPCMD

```
CMPCMD  [<MODE>], <REGISTER>, <MASK ADDRESS>, [<MASK ADDRESS
        INDEX>], <DST ADDRESS>, [<DST ADDRESS INDEX>]
```

Take the content of the console registers specified by <REGISTER> and <REGISTER>+2, mask it with the contents of <MASK ADDRESS> and <MASK ADDRESS>+2, indexed by <MASK ADDRESS INDEX>, and compare it with the contents of <DST ADDRESS> and <DST ADDRESS>+2, indexed by <DST ADDRESS INDEX>.

If the <MODE> argument is false, set the error flag. If the <MODE> argument is not specified, it defaults to EQUAL.

If the <REGISTER> argument is specified as IDREGLO or IDREGHI, the register used in the comparison is the ID Bus register that was read in the most recent READIN instruction.

The mask is performed by taking the content of <MASK ADDRESS> and <MASK ADDRESS>+2, indexed by <MASK ADDRESS INDEX>, complementing it, and bit-clearing the contents of <REGISTER> and <REGISTER>+2.

CMPPCSV

```
CMPPCSV <DST ADDRESS>, [<DST ADDRESS INDEX>]
```

Compare the content of the PC save register with the content of the location specified by <DST ADDRESS>, indexed by <DST ADDRESS INDEX>. If the contents are not equal, set the error flag.

ENDLOOP

ENDLOOP <INDEX NAME>

Add the increment value of <INDEX NAME> (see loop instruction) to the current value of the index specified by <INDEX NAME>. Compare the current value with the last value (specified in the loop instruction). If the current value is less than or equal to the last value, go to the instruction following the associated (I, J, or K) loop instruction. Otherwise, go to the next sequential instruction.

ERRLOOP

ERRLOOP

Save the address of the next instruction. If an error is detected, and the loop or error flag is set (Paragraph 4.6), execution is restarted at this saved address after the IFERROR instruction is executed (Figure 4-11).

FETCH

FETCH <WCS ADDRESS>, [<WCS ADDRESS INDEX>], [<WCS ROM NOP>]

If <WCS ADDRESS> is a numeric string, execute a maintenance return to the location specified by <WCS ADDRESS>, indexed by <WCS ADDRESS INDEX>. If <WCS ADDRESS> is an alphanumeric string, execute a maintenance return to the location specified by the content of <WCS ADDRESS>, indexed by <WCS ADDRESS INDEX>. If <ROM NOP> is specified, clear bit 7 of the Machine Control Register (MCR) during the maintenance return.

FLTONE

FLTONE <DST ADDRESS>, <INDEX NAME>

Generate a 32-bit word of all zeros. Insert a logic one in the bit postion specified by the current value minus one of <INDEX NAME>, and load this word into the location specified by <DST ADDRESS> and <DST ADDRESS>+2.

FLTZRO

FLTZRO <DST ADDRESS>, <INDEX NAME>

Generate a 32-bit word of all logic ones. Insert a zero in the bit position specified by the current value minus one of <INDEX NAME>, and load this word into the location specified by <DST ADDRESS> and <DST ADDRESS>+2.

IFERROR

IFERROR [<MESSAGE NUMBER>], [<FAIL ADDRESS>]

If the error flag is nonzero, type the PC of this instruction, the test number, subtest number, and the good and bad data. Then, go to <FAIL ADDRESS> if the HALTD flag is not set (Paragraph 4.6).

If the error flag is zero, or the <FAIL ADDRESS> is not specified, go to the next instruction.

INITIALIZE

INITIALIZE

Set and clear the CPU initialize bit in the MCR, clear the single time state bit, set the single bus cycle bit, set the ROM NOP bit, and set the proceed bit in the MCR.

KMXGEN

KMXGEN <SRC ADDRESS>, <INDEX NAME>

Generate the KMUX address specified by the current value minus one of <INDEX NAME> and load it into the KMUX field of the microinstruction specified by <SRC ADDRESS>. <SRC ADDRESS> points to a six word table in the test data section of the test that contains the microinstruction.

LDIDREG

LDIDREG <REGISTER>, <SRC ADDRESS>, [<SRC ADDRESS INDEX>]

Load the ID Bus register specified by <REGISTER> with the contents of the locations specified by <SCR ADDRESS> and <SCR ADDRESS>+2, indexed by <SRC ADDRESS INDEX>.

If <REGISTER> is the microstack, microbreak, or WCS address, the content of <SCR ADDRESS> is taken to be 16 bits. Otherwise, it is taken to be 32 bits.

LOADCA

LOADCA <REGISTER>, <SRC ADDRESS>, [<SRC ADDRESS INDEX>]

Load the console register specified by <REGISTER> with the content of the location specified by <SRC ADDRESS>, indexed by <SRC ADDRESS INDEX>. This instruction loads 16 bits of data.

LOOP

LOOP <INDEX NAME>, <START>, <END>, [<SIZE DEPENDENT>]

Initialize the loop parameter specified by <INDEX NAME> to the value specified by <START>. Save the value specified by <END> for the ENDLOOP instruction. Calculate and save the increment value for the ENDLOOP instruction with the following algorithm:

> If <START> is less than or equal to <END>, set the increment value to +1; otherwise, set it to -1.

If <END> is an <INDEX NAME>, save the current value of that index name as the <END> value of this index name.

If <SIZE DEPENDENT> is specified, and there is only one WCS module on the system, divide the larger of <START> and <END> by two. Otherwise, leave them unchanged.

> **NOTE**
>
> **The tests are written for two WCS modules. This argument allows the loop parameters to be modified at run time if the system only has one module.**

MASK

MASK <DST ADDRESS>, <MASK ADDRESS>

Take the content of location <MASK ADDRESS>, complement it, and bit-clear the content of location <DST ADDRESS> with it.

MOVE

MOVE <SRC ADDRESS>, [<SRC ADDRESS INDEX>], <DST ADDRESS>

Move the content of <SRC ADDRESS INDEX> (indexed by <SRC ADDRESS INDEX>) to the location specified by <DST ADDRESS>.

NEWTST

NEWTST <TEST NAME>, [<TEST DESCRIPTION>], [<LOGIC DESCRIPTION>], [<ERROR DESCRIPTION>], [<SYNC POINT DESCRIPTION>]

This instruction creates a test header document for the specified arguments. It clears the error flag and saves the PC of the next instruction for looping on test.

READID

READID <REGISTER>

Read the ID Bus register specified by <REGISTER> and load the content into locations IDREGLO and IDREGHI.

RESET

RESET

Execute an LSI-11 reset instruction.

REPORT

REPORT <MODULE NAME STRING>

Type out the module numbers of the modules specified by <MODULE NAME STRING>. If the HALTI flag is set, return to the microdiagnostic monitor.

SETPSW

SETPSW <DATA>

Load the LSI processor status word with the value specified by <DATA>.

SETVEC

SETVEC <VECTOR ADDRESS>

Set the LSI-11 address specified by <VECTOR ADDRESS> to the expected trap routine.

SKIP

SKIP [<DST ADDRESS>]

Go to the <DST ADDRESS>. If <DST ADDRESS> is not specified, go to the next test. If <DST ADDRESS> starts with the alpha character S, go to the next subtest.

SUBTEST

SUBTEST

Increment the subtest counter.

TSTVB

TSTVB <SRC TABLE ADDRESS>, [<SRC TABLE ADDRESS INDEX>]

Load and read the V Bus. Compare the contents of the data at <SRC TABLE ADDRESS>, indexed by <SRC TABLE ADDRESS INDEX>, with the V Bus data just read. The <SRC TABLE> has the following format:

```
     1$:  .WORD    <NUMBER OF BITS TO CHECK>
          VBUSG    <CHANNEL NUMBER>, <BIT NUMBER>, <EXPECTED BIT
                   VALUE>
          .
     2$:  .WORD    <NUMBER OF BITS TO CHECK>
          VBUSG    <CHANNEL NUMBER>, <BIT NUMBER>, <EXPECTED BIT
                   VALUE>
          .
          .
          .
```

VBUSG is a MACRO name that encodes the three arguments into one 16-bit word as follows:

BITS <07:00> = <CHANNEL NUMBER>
BITS <14:08> = <BIT NUMBER)
BIT <15> = <EXPECTED BIT VALUE>

The following is an example of the <SRC TABLE ADDRESS INDEX>:

> TSTVB 1$,I
> If the current value of the <SRC TABLE ADDRESS INDEX> is 2, and the <SRC TABLE> looks like the preceding table, the physical <SRC TABLE ADDRESS> would be 2$.

TYPSIZE

TYPSIZE

Use the content of location BADDATA, which contains the value of the WCS data register when it was read, to determine the WCS module configuration, and type a message and the number of WCS modules that will be tested. If any of the following conditions exist, the test stream is aborted and the NER (No Error Report) flag is set.

a. WCS module count is zero
b. bits 3--0 are nonzero
c. fifth K of WCS is not present

These conditions mean that the WCS is either configured incorrectly or the WCS data register cannot be read correctly.

4.5 MICROTEST DESCRIPTION

On completion of the hardcore tests, the microdiagnostic monitor overlays the hardcore monitor with the microtest monitor. Microtest sequencing and execution are then controlled by the microtest monitor.

The microtest monitor begins to load the microtests from the floppy into the same buffer area used by the hardcore tests. However, in the case of microtests, this area is strictly a buffer. Since the microtests are implemented in system microcode, the tests are transferred from the buffer and loaded into and executed out of the WCS.

The monitor references a table that contains the WCS addresses of the first instruction of every test in the overlay (section) that was just loaded in order to locate the address of the first test (first entry in the table). The address is loaded onto the microstack. A maintenance return is performed, popping the address from the microstack into the micro PC and initiating execution of the first test. At the end of each test, the microtest monitor is interrupted. This allows the monitor to check that the microtests are being executed in the correct order.

The monitor then initiates the next test with another maintenance return. This sequence continues until the original 1K microword overlay has been executed. At this point, the microtest monitor loads another 1K microword overlay into WCS.

Because of the microtest package size, more than one diskette is required for storage. When the monitor executes the last test on a diskette, it determines whether it is the last test of the entire package. In the case where it is the last test, the monitor prints out a message to the operator with instructions to load the next sequential diskette and enter a command to continue microtest execution (Paragraph 4.8).

4.5.1 Microtest Structure

The initial microtests complete the data path testing started by the hardcore tests. The microtests then begin to test the Translation Buffer (TB) and cache without using memory. The tests check the TB and cache for their ability to retain correct address and data information, and to check parity correctly.

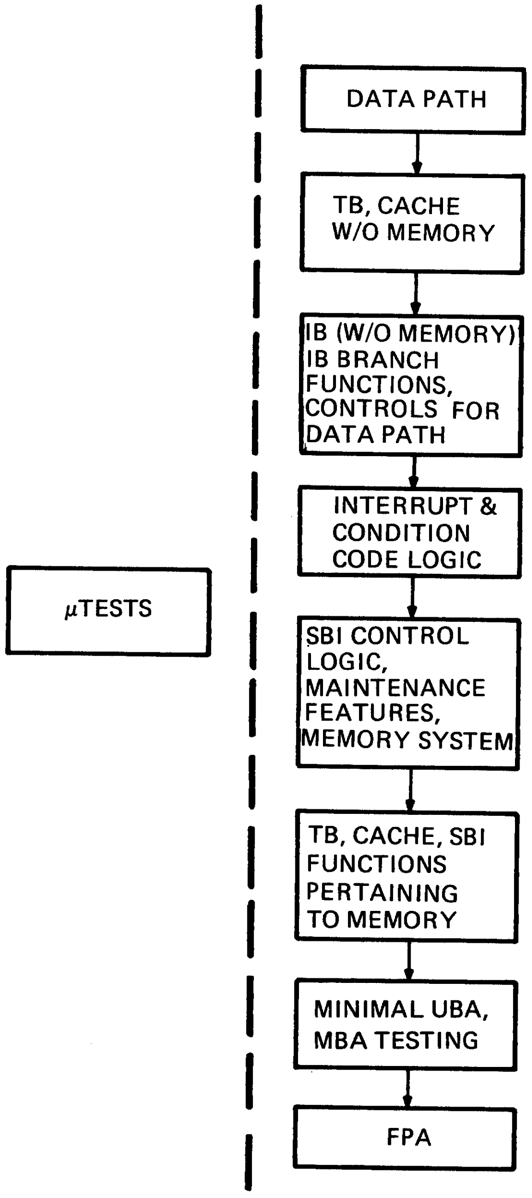
The Instruction Buffer (IB) tests are then executed, again without using memory. The IB test data is loaded into cache. The microtests cause instruction test patterns to be retrieved from cache, and check the IB branching functions and controls for the data path.

The interrupt and condition code logic is checked in a similar manner (i.e., test data loaded into and subsequently retrieved from cache.)

The next test segment covers the SBI control logic and its maintenance functions, and the memory system. After performing these tests, the microtests go back and test those functions of the TB, cache, and SBI subsystem that depend on retrieving data from memory (e.g., cache, SBI faults, etc.). A minimal amount of testing is performed on the Unibus and Massbus adapters. These tests force selected errors on the SBI and determine the adapters' capability to detect and react to the forced errors correctly.

The floating-point accelerator is tested last.

Figure 4-4 shows the microtest sequence.


μTESTS
DATA PATH
TB, CACHE W/O MEMORY
IB (W/O MEMORY) IB BRANCH FUNCTIONS, CONTROLS FOR DATA PATH
INTERRUPT & CONDITION CODE LOGIC
SBI CONTROL LOGIC, MAINTENANCE FEATURES, MEMORY SYSTEM
TB, CACHE, SBI FUNCTIONS PERTAINING TO MEMORY
MINIMAL UBA, MBA TESTING
FPA

TK-0778

Figure 4-4 Microtest Structure

On completion of the microtests, control is returned through the microdiagnostic monitor to the console program. The console reboots, sends the relevant bootstrap header information to the console terminal, and prompts for operator input.

4.6 MICRODIAGNOSTIC MONITOR CONTROLS

The following paragraphs describe the operator command execution control functions provided by the microdiagnostic monitor. Where appropriate, examples of command and program control flag usage are included. Also included is a description of microdiagnostic related error messages.

The majority of the commands available in the microdiagnostic monitor are not used in the normal course of execution. Normally the operator enters the test command and executes the entire microdiagnostic package. The command mode is usually used following error detection. Following the error message printout, testing stops and control is returned to the monitor command mode. At this point, the operator executes those microdiagnostic commands he decides would be most helpful.

Symbols used in the command syntax are the comma and < >. The comma is used to separate items within a list. < > denotes an argument, that is, either an address, pass count value, or a V Bus channel. Note that every command (or command line) must be terminated with a carriage return (CR).

Control C (^C) is the user interrupt control character. If Control C is entered during test execution, the current test will complete, further testing is suspended, and control is returned to the monitor command mode. If Control C is entered while a test is looping on an error, the loop will be suspended and control returned to the command mode. Any command may be aborted if a Control C is entered in that command line.

Table 4-2 describes the monitor commands. Note that although all commands, keywords, qualifiers, and flags are spelled out, they can be abbreviated to the first two characters. The only exceptions are the halt on error detection and halt on error isolation flags, which must be typed HD and HI, respectively.

Table 4-2 Microdiagnostic Command/Flag Descriptions

Command/Flag	Description
DIAGNOSE	Initializes the program control flags, and starts microdiagnostic execution at test number one. Valid qualifiers are: /TEST: <NUMBER> -- Dispatch to the test number specified (do not execute any prior tests), and loop on the test indefinitely. /SECTION: <NUMBER> -- Dispatch to the section number specified (do not execute any prior sections), and loop on the section indefinitely. /PASS: <NUMBER> -- Execute the microdiagnostics and the specified number of passes before returning to the console. If the number is -1, execute the microdiagnostics indefinitely. /CONTINUE -- Used with the /TEST or /SECT switch to automatically continue after the specified test or section has been reached. /TEST: <N> <M> -- Dispatch to test <N>, execute tests <N> through <M> (inclusive), and return to command mode. /SECT: <N> <M> -- Dispatch to section <N>, execute sections <N> through <M> (inclusive), and return to command mode.

NOTE

In the preceding variations of the /TEST and /SECTION qualifiers, the value of <N> must be less than or equal to <M>. If <M> is less than <N>, testing will start at <N> and continue to the end.

/TEST and /SECT cannot be specified simultaneously.

Examples

DIAG/TEST:2F
Dispatch to test number 2F and execute it indefinitely.

Table 4-2 Microdiagnostic Command/Flag Descriptions (Cont)

Command/Flag	Description
	DIAG/SECT:B Dispatch to section number B and execute it indefinitely.
	DIAG/PASS:--1 Execute all of the microdiagnostics indefinitely.
	DIAG/TEST:2F/CONT Dispatch to test 2F and start execution of the remaining tests.
CONTINUE	Continues microdiagnostic execution without changing the program control flags.
Set and Clear Flags	
SET/CLEAR FLAG HD	Sets (or clears) the halt on error detection flag.
SET/CLEAR FLAG HI	Sets (or clears) the halt on error isolation flag.
SET/CLEAR FLAG LOOP	Sets (or clears) the loop on error flag.
SET/CLEAR FLAG NER	Sets (or clears) the no error report flag.
SET/CLEAR FLAG BELL	Sets (or clears) the bell on error flag.
SET/CLEAR FLAG ERABT	Sets (or clears) the error abort flag.
CLEAR FLAG LS	Clears the loop on special section flag. (Note that this flag cannot be set.)
CLEAR LT FLAG	Clears the loop on special test flag. (Note that this flag cannot be set.)
SET/CLEAR FLAG ALL	Sets (or clears) all of the previous flags.
SET/CLEAR SOMM	Sets (or clears) the stop on micromatch bit.
SET/CLEAR SOMM:<ADDRESS>	Loads address into the CPU microsync register, and sets (or clears) the stop on micromatch bit.

Table 4-2 Microdiagnostic Command/Flag Descriptions (Cont)

Command/Flag	Description
SET/CLEAR FPA:<ADDRESS>	Loads <ADDRESS> into the FPA microsync register.
SET STEP STATE	Sets the CPU clock to single time state.
SET STEP BUS	Sets the CPU clock to single bus cycle. Both the SET STEP STATE and SET STEP BUS commands cause the monitor to enter step mode. Step mode types the current clock state or the UPC value, and waits for terminal input. If a space is typed, the clock is triggered and the current UPC value is typed out. If any other character is entered, step mode is exited.
SET STEP INSTRUCTION	Sets the software single instruction flag and returns to the monitor. When the hardcore tests are invoked, the current value of the Test PC (TPC) is typed. The monitor waits for terminal input. If a space is typed, the current pseudo instruction is executed and the current value of the TPC is typed. If any other character is typed, step mode is exited.
SET CLOCK FAST	Sets the CPU clock speed to the fast margin.
SET CLOCK SLOW	Sets the CPU clock speed to the slow margin.
SET CLOCK NORMAL	Sets the CPU clock speed to normal.
SET CLOCK EXTERNAL	Sets the CPU clock for an external oscillator.
SHOW	Causes a display of the HD, HI, LOOP, NER, BELL, ERABT, LS, and LT flags.
LOOP	Clears the HD and HI flags. Sets the LOOP and NER flags and executes a continue command.
RETURN	Returns control to the console program.

Table 4-2 Microdiagnostic Command/Flag Descriptions (Cont)

Command/Flag	Description
Examine Commands	The following examine commands cause the current microinstruction to be executed before the examine is performed, if it is the first examine since entering the monitor command mode. All successive examines do not execute any additional microinstructions. ID Bus registers T1--T8 are destroyed during examines, except for the V Bus examines. All of the following examines, except V Bus, advance the clock to CPTØ before executing the command.
EXAMINE ID:<ADDRESS>	Displays the content of the ID Bus register specified by <ADDRESS>.
EXAMINE VBUS:<CHANNEL>	Displays the content of the V BUS channel specified by <CHANNEL>. Bit Ø is at the right side of the display.
EXAMINE RA:<ADDRESS>	Displays the content of the RA scratch pad specified by <ADDRESS>.
EXAMINE RC:<ADDRESS>	Displays the content of the RC scratch pad specified by <ADDRESS>.
EXAMINE LA	Displays the content of the LA latch.
EXAMINE LC	Displays the content of the LC latch.
EXAMINE DR	Displays the content of the D register.
EXAMINE QR	Displays the content of the Q register.
EXAMINE SC	Displays the content of the SC register.
EXAMINE FE	Displays the content of the FE register.
EXAMINE VA	Displays the content of the VA register.
EXAMINE PC	Registers the content of the program counter.

Table 4-2 Microdiagnostic Command/Flag Descriptions (Cont)

Command/Flag	Description
Deposit Commands	The deposit command is the same as the examine command, except that the data to be deposited must be supplied by the user.
DEPOSIT ID: <ADDRESS> <DATA> DEPOSIT RA: <ADDRESS> <DATA> DEPOSIT RC: <ADDRESS> <DATA> DEPOSIT LA: <DATA> DEPOSIT LC: <DATA> DEPOSIT DR: <DATA> DEPOSIT QR: <DATA> DEPOSIT SC: <DATA> DEPOSIT FE: <DATA> DEPOSIT VA: <DATA> DEPOSIT PC: <DATA>	

4.6.1 Monitor Control Examples

The following paragraphs provide usage examples of selected monitor controls. These descriptions are brief and are intended only to indicate some of the capabilities of the microdiagnostic monitor.

4.6.1.1 HD/HI Flags -- In addition to testing, the microdiagnostics perform two basic functions: error detection and error isolation. Under normal circumstances, the user would set the HI flag. Setting the HI flag initiates the following microdiagnostic sequence:

a. Error detection

b. Call isolation routine to identify the error cause

c. Display an error message identifying the failed test, data pattern used, and the failing modules

d. Terminate test execution.

In a situation where the user does not require a scope loop, and wants to halt execution at the error detection point, the HD flag is set. This flag halts the test before the microdiagnostic calls the isolation routine overlay (e.g., V Bus compare).

4.6.1.2 Loop On Error Flag (LOOP) -- With this flag set, the microdiagnostic will revert to a tight program loop after error detection (assuming the NER flag is set). Note that the loop will continue even though the error is intermittent; the flag must be cleared to break the loop.

4.6.1.3 No Error Report Flag (NER) -- This flag suppresses the typing of error messages. The flag is especially useful in the case of looping on an error. Since the error printout takes time, the scope sync is lost during typing time. With error reports suppressed, the loop is tight and produces a reasonable sync.

4.6.1.4 Bell On Error Flag (BELL)

Hardcore Tests -- When running the hardcore tests, setting this flag causes the console terminal to ring its bell when an error occurs. This flag is useful in a situation where a manual adjustment could clear the error. In this situation, the user would set the LOOP flag and the NER flag, producing a tight loop.

However, with no error report, the user does not have an indication of where the error cleared during the adjustment. Setting the BELL flag is a compromise between a reasonably tight error loop (the BELL flag slows the loop somewhat) and an error indication during the adjustment. If a scope were used during the adjustment, the user would have an error indication without losing the scope trace.

Microtests -- In the microtests, one must loop on test (DI/TEST:n); then, after the error message has been printed, set the NER and BELL flags, clear the HI flag, and type CONTINUE.

4.6.1.5 Continue Command (CONT) -- This command allows the user to proceed from a microdiagnostic halt situation. For example, suppose that a hardware ECO, which has not been reflected in the diagnostic system, is incorporated into the computer. When the microdiagnostic halts following detection of the pseudo error, the user can bypass the failing test and continue execution at the next test by entering CONTINUE.

4.6.1.6 Error Abort Flag (ERABT) -- This flag allows the user to display more than one error report in certain hardcore tests (tests which exercise a particular piece of logic with more than one data pattern).

For example, consider the situation where the ERABT flag is set, and the test detects a type 2 error (ERROR2) on one of the initial data patterns. If the user were to enter CONTINUE, the flag would abort the remainder of the test and initiate execution of the next sequential test. However, with the flag cleared, CONTINUE will initiate execution of the same test with the next sequential data pattern.

4.7 MICRODIAGNOSTIC RELATED ERROR MESSAGES

The following paragraphs describe the microdiagnostic-related error message formats and their interpretation. All error messages are prefixed by a question mark to distinquish them from informational messages.

4.7.1 Syntax Error Messages

?USE DIAG COMMAND
Execution of a continue command was attempted before a diagnose command. This would only occur if TEST/COM were used to invoke the microdiagnostics from the console program.

?INVALID COMMAND
The previously entered command was not recognized.

?INVALID KEYWORD
The argument of a command was not recognized.

?NUMBER MUST BE HEX
A non-hexadecimal number was recognized.

4.7.2 System Error Messages

?OPEN FILE: <NUMBER>
An error was detected and identified while trying to open a floppy file. Error code is:

```
<NUMBER> = 1 = Floppy hardware error
<NUMBER> = 2 = File not found
<NUMBER> = 3 = Floppy not ready
```

?READ SECTOR: <NUMBER>
An error was detected and identified while trying to read a sector from the floppy. Error code is:

```
<NUMBER> = 4 = Sector number out of range
<NUMBER> = 3 = Floppy queue full
<NUMBER> = 1 = Floppy hardware error
```

?KEYBOARD ERROR: <NUMBER>
An error was detected and identified while trying to read the terminal. Error code is:

```
<NUMBER> = 5 = Terminal driver busy
<NUMBER> = 7 = Terminal hardware error
```

?UNEXPECTED TRAP TO 4 PC =
The LSI-11 trapped to 4 at the specified PC.

4.7.3 Go Chain Monitor Error Messages

?TIMEOUT IN TEST ... UPC =
Indicates that the microcode is hung. The monitor did not receive a call from the microcode in the last four seconds.

?EXECUTION OUT OF SEQUENCE UPC = SHOULD BE =
The microcode has not executed the tests within the overlay in sequential order.

?CLOCK STOPPED UNEXPECTEDLY
The clock stopped and the SOMM bit was not set.

?ILLEGAL MONITOR CALL: <NUMBER>
The microcode made a call to the monitor with a bad argument, which was <NUMBER>.

4.8 PROGRAM LISTING AND ERROR MESSAGE DESCRIPTIONS

The following paragraphs describe microdiagnostic program listings and error message formats. It is beyond the scope of this chapter to describe the various diagnostic program assemblers and their associated languages.

4.8.1 Monitor Listing Descriptions

The program listings for the microdiagnostic-associated monitors (i.e., microdiagnostic, hardcore, and microtest) share the same listing format. That is, since the three monitors operate out of the LSI-11, they are coded in MACRO-11 (PDP-11 assembly language) and are discussed as one in the general description.

Each listing is comprised of three general sections: Table of Contents, Program Definitions, and Program Code and Descriptions. Each page in a listing has a title containing the name of the program, the date the particular listing was generated, the page number, and a line item that indicates the content of that page (listing header) (Figure 4-5).

The Table of Contents is a list of the content of the program listing. The first (left) column contains a hyphenated number. The number preceding the hyphen specifies the page number of the listing on which the line appears. The number following the hyphen is a listing line number. This number specifies the starting line (within the listing) of the associated listing content contained in the right column (e.g., definitions, utility routines, test sections, etc.)

The Program Definition section specifies register address assignments, bit definitions, module and bus name assignments, and other constants that are used throughout the program.

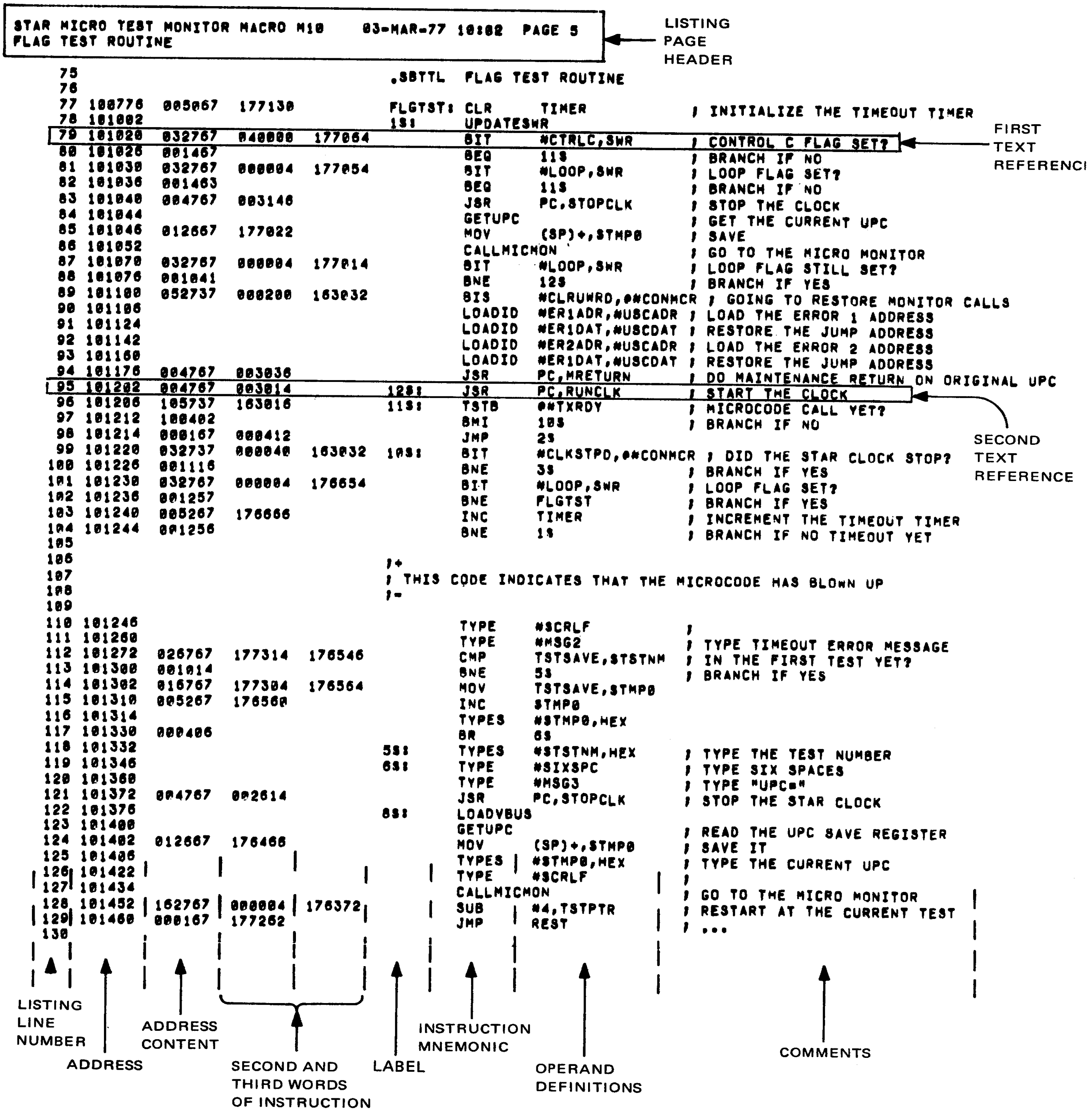
The remainder of the listing (and the largest section by far) is the Program Code and Description section (Figure 4-5). The format is described on a per column basis from left to right. Note that the address and data radix for all monitor listings is octal.

^ Column 1, Listing Line Number -- Each line in the listing is assigned a unique decimal number to allow easy referencing from the Table of Contents.

^ Column 2, Address -- The address of the instruction.

^ Column 3, Content of the Address listed in Column 2 -- This is usually an instruction (e.g., reference line number 79 in Figure 4-5). The address is 1Ø1Ø2Ø, and its content is Ø32767, which is the octal code for a Bit Test (BIT) instruction.

^ Column 4 and 5 -- If the instruction is a two-word or three-word instruction, the second and third words are specified in columns 4 and 5, respectively (e.g., reference line number 79 in Figure 4-5). This BIT



```
STAR MICRO TEST MONITOR MACRO M10    03-MAR-77 10:02  PAGE 5
FLAG TEST ROUTINE

   75                                                .SBTTL  FLAG TEST ROUTINE
   76
   77 100776  005067  177130                         FLGTST: CLR     TIMER                   ; INITIALIZE THE TIMEOUT TIMER
   78 101002                                         1$:     UPDATESWR
   79 101020  032767  040000  177064                         BIT     #CTRLC,SWR              ; CONTROL C FLAG SET?
   80 101026  001467                                         BEQ     11$                     ; BRANCH IF NO
   81 101030  032767  000004  177054                         BIT     #LOOP,SWR               ; LOOP FLAG SET?
   82 101036  001463                                         BEQ     11$                     ; BRANCH IF NO
   83 101040  004767  003146                                 JSR     PC,STOPCLK              ; STOP THE CLOCK
   84 101044                                                 GETUPC                          ; GET THE CURRENT UPC
   85 101046  012667  177022                                 MOV     (SP)+,STMP0             ; SAVE
   86 101052                                                 CALLMICMON                      ; GO TO THE MICRO MONITOR
   87 101070  032767  000004  177014                         BIT     #LOOP,SWR               ; LOOP FLAG STILL SET?
   88 101076  001041                                         BNE     12$                     ; BRANCH IF YES
   89 101100  052737  000200  163032                         BIS     #CLRUWRD,@#CONMCR ; GOING TO RESTORE MONITOR CALLS
   90 101106                                                 LOADID  #ER1ADR,#USCADR ; LOAD THE ERROR 1 ADDRESS
   91 101124                                                 LOADID  #ER1DAT,#USCDAT ; RESTORE THE JUMP ADDRESS
   92 101142                                                 LOADID  #ER2ADR,#USCADR ; LOAD THE ERROR 2 ADDRESS
   93 101160                                                 LOADID  #ER1DAT,#USCDAT ; RESTORE THE JUMP ADDRESS
   94 101176  004767  003036                                 JSR     PC,MRETURN              ; DO MAINTENANCE RETURN ON ORIGINAL UPC
   95 101202  004767  003014                         12$:    JSR     PC,RUNCLK               ; START THE CLOCK
   96 101206  105737  163016                         11$:    TSTB    @#TXRDY                 ; MICROCODE CALL YET?
   97 101212  100402                                         BMI     10$                     ; BRANCH IF NO
   98 101214  000167  000412                                 JMP     2$
   99 101220  032737  000040  163032                 10$:    BIT     #CLKSTPD,@#CONMCR ; DID THE STAR CLOCK STOP?
  100 101226  001116                                         BNE     3$                      ; BRANCH IF YES
  101 101230  032767  000004  176654                         BIT     #LOOP,SWR               ; LOOP FLAG SET?
  102 101236  001257                                         BNE     FLGTST                  ; BRANCH IF YES
  103 101240  005267  176666                                 INC     TIMER                   ; INCREMENT THE TIMEOUT TIMER
  104 101244  001256                                         BNE     1$                      ; BRANCH IF NO TIMEOUT YET
  105
  106                                                ;+
  107                                                ; THIS CODE INDICATES THAT THE MICROCODE HAS BLOWN UP
  108                                                ;-
  109
  110 101246                                                 TYPE    #$CRLF                  ;
  111 101260                                                 TYPE    #MSG2                   ; TYPE TIMEOUT ERROR MESSAGE
  112 101272  026767  177314  176546                         CMP     TSTSAVE,$TSTNM          ; IN THE FIRST TEST YET?
  113 101300  001014                                         BNE     5$                      ; BRANCH IF YES
  114 101302  016767  177304  176564                         MOV     TSTSAVE,$TMP0
  115 101310  005267  176560                                 INC     $TMP0
  116 101314                                                 TYPES   #$TMP0,HEX
  117 101330  000406                                         BR      6$
  118 101332                                         5$:     TYPES   #$TSTNM,HEX             ; TYPE THE TEST NUMBER
  119 101346                                         6$:     TYPE    #$SIXSPC                ; TYPE SIX SPACES
  120 101360                                                 TYPE    #MSG3                   ; TYPE "UPC="
  121 101372  004767  002614                                 JSR     PC,STOPCLK              ; STOP THE STAR CLOCK
  122 101376                                         8$:     LOADVBUS
  123 101400                                                 GETUPC                          ; READ THE UPC SAVE REGISTER
  124 101402  012667  176466                                 MOV     (SP)+,$TMP0             ; SAVE IT
  125 101406                                                 TYPES   #$TMP0,HEX              ; TYPE THE CURRENT UPC
  126 101422                                                 TYPE    #$CRLF                  ;
  127 101434                                                 CALLMICMON                      ; GO TO THE MICRO MONITOR
  128 101452  162767  000004  176372                         SUB     #4,TSTPTR               ; RESTART AT THE CURRENT TEST
  129 101460  000167  177262                                 JMP     REST                    ; ...
  130
```

Figure 4-5 Monitor Listing Sample

instruction happens to be a three-word instruction, the second word is Ø4ØØØØ, and the third word is 177Ø64. Thus, this instruction is testing bit 14 (Ø4ØØØØ) to determine if the Control C flag is set at address 177Ø64. (Note that this flag was defined in the definition section.)

- Column 6, Label -- This symbol is the name used by the program mnemonics to reference this instruction (e.g., reference line number 95 in Figure 4-5). The label in this case is 12$.
- Column 7, Instruction Mnemonic -- This is the assembler language mnemonic for the instruction (e.g., Bit Test Instruction = BIT).
- Column 8, Operand Definitions -- These symbols and mnemonics are the assembler language mnemonic definitions for the operands.
- Column 9, Comments -- A brief description (following the semicolon) of the instruction operation.

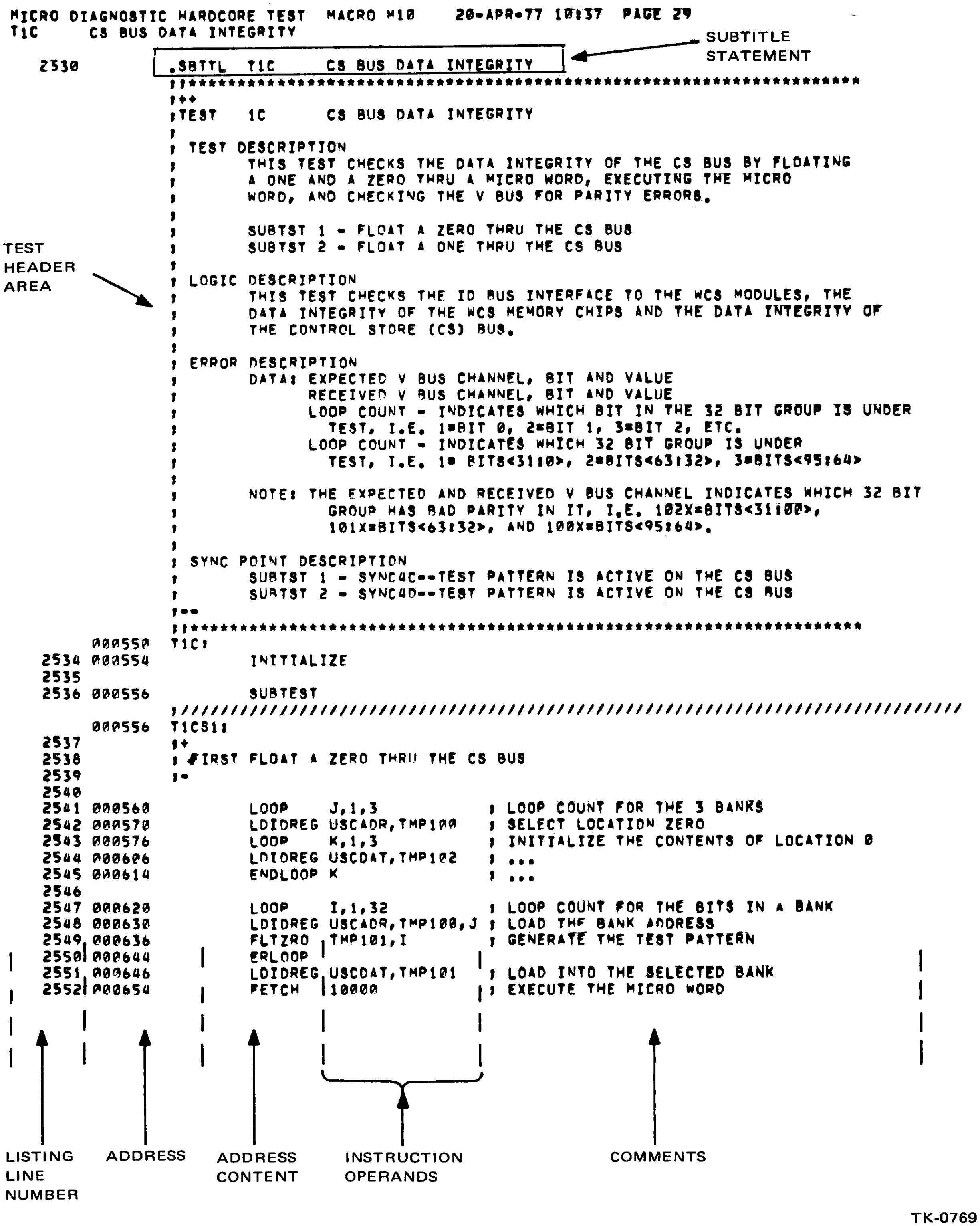
4.8.2 Hardcore Listing Description

The general format of the hardcore listing is similar to that of the monitors (i.e., Table of Contents, Program Definitions, and Program Code and Descriptions). The left column of the Table of Contents contains a hyphenated number. The number preceding the hyphen specifies the page number of the listing on which the line appears. The number following the hyphen is the listing line number, indicating the starting line of the associated listing contents. The definition section is similar to the monitor listings, i.e., address, module and bus assignments, bit definitions, and other constants used in the program.

The remainder of the listing is the Program Code and Descriptions. As indicated in the Table of Contents, the hardcore tests are composed of sections and tests. The section number represents a 1.5K byte segment. The section number is displayed on the console terminal during hardcore test execution. The test number identifies a test on a particular logic area or function. The subtest number (which is not referenced in the Table of Contents) identifies a particular portion of a test. For example, Subtest 1 floats a logic one through each bit of a register; Subtest 2 floats a logic zero through the same register.

As shown in Figure 4-6, the program code is preceded by an outlined test header area. A subtitle statement (.SBTTL) generates the test number and title above the header area.

The header area consists of five descriptive segments. The first line within the outlined header repeats the test number and test title. The test description segment is a brief paragraph describing the general logic area tested and method of test. The logic description segment describes the test in more detail.



```
MICRO DIAGNOSTIC HARDCORE TEST  MACRO M10    20-APR-77 10:37  PAGE 29
T1C      CS BUS DATA INTEGRITY

   2530        .SBTTL  T1C      CS BUS DATA INTEGRITY
               ;;*********************************************************************
               ;+
               ;TEST   1C      CS BUS DATA INTEGRITY
               ;
               ; TEST DESCRIPTION
               ;       THIS TEST CHECKS THE DATA INTEGRITY OF THE CS BUS BY FLOATING
               ;       A ONE AND A ZERO THRU A MICRO WORD, EXECUTING THE MICRO
               ;       WORD, AND CHECKING THE V BUS FOR PARITY ERRORS.
               ;
               ;       SUBTST 1 - FLOAT A ZERO THRU THE CS BUS
               ;       SUBTST 2 - FLOAT A ONE THRU THE CS BUS
               ;
               ; LOGIC DESCRIPTION
               ;       THIS TEST CHECKS THE ID BUS INTERFACE TO THE WCS MODULES, THE
               ;       DATA INTEGRITY OF THE WCS MEMORY CHIPS AND THE DATA INTEGRITY OF
               ;       THE CONTROL STORE (CS) BUS.
               ;
               ; ERROR DESCRIPTION
               ;       DATA: EXPECTED V BUS CHANNEL, BIT AND VALUE
               ;             RECEIVED V BUS CHANNEL, BIT AND VALUE
               ;             LOOP COUNT - INDICATES WHICH BIT IN THE 32 BIT GROUP IS UNDER
               ;               TEST, I.E. 1=BIT 0, 2=BIT 1, 3=BIT 2, ETC.
               ;             LOOP COUNT - INDICATES WHICH 32 BIT GROUP IS UNDER
               ;               TEST, I.E. 1= BITS<31:0>, 2=BITS<63:32>, 3=BITS<95:64>
               ;
               ;       NOTE: THE EXPECTED AND RECEIVED V BUS CHANNEL INDICATES WHICH 32 BIT
               ;               GROUP HAS BAD PARITY IN IT, I.E. 102X=BITS<31:00>,
               ;               101X=BITS<63:32>, AND 100X=BITS<95:64>.
               ;
               ; SYNC POINT DESCRIPTION
               ;       SUBTST 1 - SYNC4C--TEST PATTERN IS ACTIVE ON THE CS BUS
               ;       SUBTST 2 - SYNC4D--TEST PATTERN IS ACTIVE ON THE CS BUS
               ;--
               ;;*********************************************************************
        000550 T1C:
   2534 000554         INITTIALIZE
   2535
   2536 000556         SUBTEST
               ;//////////////////////////////////////////////////////////////////////////////
        000556 T1CS1:
   2537        ;+
   2538        ; FIRST FLOAT A ZERO THRU THE CS BUS
   2539        ;-
   2540
   2541 000560         LOOP    J,1,3             ; LOOP COUNT FOR THE 3 BANKS
   2542 000570         LDIDREG USCADR,TMP100     ; SELECT LOCATION ZERO
   2543 000576         LOOP    K,1,3             ; INITIALIZE THE CONTENTS OF LOCATION 0
   2544 000606         LDIDREG USCDAT,TMP102     ; ...
   2545 000614         ENDLOOP K                 ; ...
   2546
   2547 000620         LOOP    I,1,32            ; LOOP COUNT FOR THE BITS IN A BANK
   2548 000630         LDIDREG USCADR,TMP100,J ; LOAD THE BANK ADDRESS
   2549 000636         FLTZRO  TMP101,I          ; GENERATE THE TEST PATTERN
   2550 000644         ERLOOP
   2551 000646         LDIDREG USCDAT,TMP101     ; LOAD INTO THE SELECTED BANK
   2552 000654         FETCH   10000             ; EXECUTE THE MICRO WORD
```

Figure 4-6 Hardcore Listing Sample

The error description segment specifies test parameters. For example, in the error description of Figure 4-6, the first line specifies what is expected during the test; the second line specifies what is received. The third line indicates which bit in the 32-bit array is under test; the fourth line indicates the 32-bit group under test. The sync point segment specifies critical points in the listing around which an error loop or scope loop might be set up (Paragraph 4.9.2).

Following the test header is the program code. The hardcore listings are described on a per column basis below.

^ Column 1, Listing Line Number -- Each line is assigned a unique decimal number to allow easy referencing.

^ Column 2, Address -- The relative address (PC) of the instruction.

^ Column 3, Address Content -- Content of the address listed in Column 2. (Note that the contents are the pseudo instructions described in Paragraph 4.4.2.)

^ Columns 4 and 5, Instruction Operands -- The operands are the instruction source, destination, or index values. The mnemonics appearing in these columns have been defined in the definition section of the listing.

^ Column 6, Comments -- A brief descriptive note concerning the instruction operation.

4.8.3 Microtest Listing Description

The general format of the microtest listing is somewhat similar to the other microdiagnostic listings, i.e., a Table of Contents, Program Definitions, and Program Code and Descriptions. However, since the microtests are executed out of the WCS, they are written in system microcode and, therefore, are similar to the system firmware listings. Unlike the hardcore listings that are assembled in one listing, the microtests are assembled into separate listings by 1K microword test sections and identified by those section numbers. Note also that the address and data radix for these listings is hexadecimal.

The Table of Contents is similar to those of the other listings; i.e., it contains a line number entry and the corresponding listing content description. Since the first column does not contain assembler directives, only the line number appears. The Program Definition section describes all macro definitions associated with the listing.

The Program Code and Description section format is similar to that of the system firmware listing. As in the hardcore listing, the program code is identical to the hardcore format and content described in Paragraph 4.8.2.

The microtest listing is described on a per column basis (Figure 4-7).

- Column 1, UPC -- This column specifies the address contained in the UPC at that particular microstate.

- Column 2, Microword -- This column describes the microword content of the address specified in column 1.

- Column 3, Listing Line Number -- Decimal number assigned to allow easy referencing.

- Column 4, Microstate Operation -- This column specifies the operation during a particular microstate. The notations used to describe the operation have been defined in the program definition section.

- Column 5, Comments -- A brief descriptive note concerning the microstate operation. (A detailed firmware description is provided in the VAX-11/78Ø Central Processor Technical Description, e.g., field definitions, coding conventions, etc.)

4.8.4 Microdiagnostic Execution

The entire microdiagnostic package may be executed by entering TEST on the console terminal. Other operation options are described in the detailed diagnostic operating procedures in The VAX-11/78Ø Diagnostic System User's Guide (EK-DS78Ø-UG-ØØ1). Following microdiagnostic identification the monitors initiate hardcore and microtest execution. Figure 4-8 illustrates typical console terminal output during error-free microdiagnostic execution.

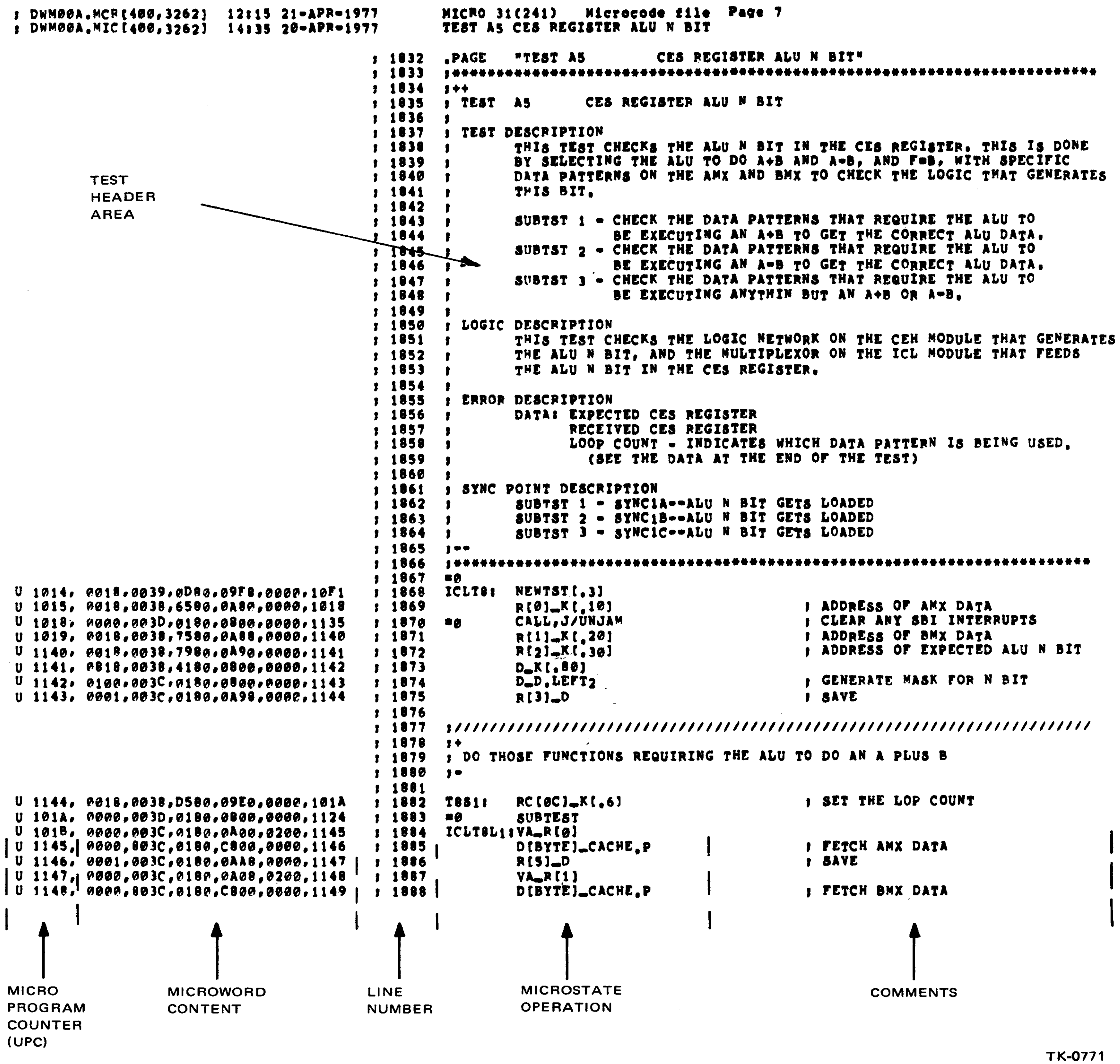
The microtests and hardcore tests are numbered sequentially (with no duplication of test numbers). As shown in Figure 4-8, there is no differentiation between hardcore and microtests. A differentiation is required only in the case of an error (Paragraph 4.8.5).

The monitor loads the test, and the test section number is printed on the console terminal. Test execution is then initiated. The section number is printed (in hexadecimal) prior to execution to allow the operator to identify the exact failing section in the case of an error.

The entire microdiagnostic package requires two diskettes. As indicated in Figure 4-8, the microdiagnostic monitor instructs the operator when to mount the second diskette, and prompts for the command required to initiate execution of those diagnostics.

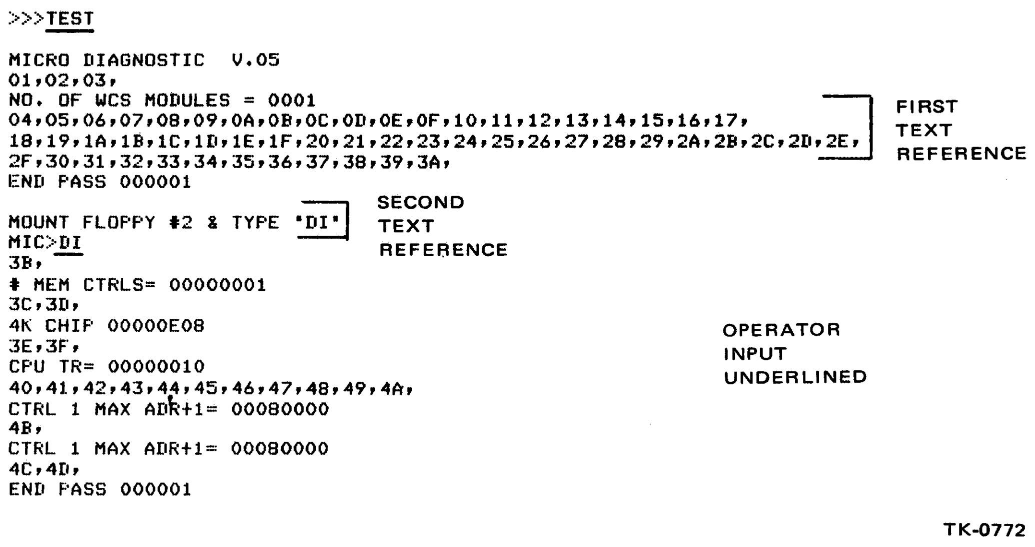
4.8.5 Error Message Format

The general error message format for both types of microdiagnostics is shown in Figure 4-9.



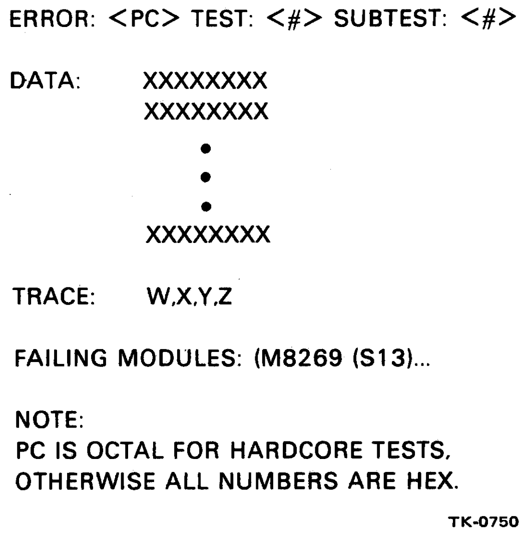
```
; DWM00A.MCR[400,3262]  12:15 21-APR-1977        MICRO 31(241)   Microcode file  Page 7
; DWM00A.MIC[400,3262]  14:35 20-APR-1977        TEST A5 CES REGISTER ALU N BIT

                                      ; 1832  .PAGE   "TEST A5        CES REGISTER ALU N BIT"
                                      ; 1833  ;***********************************************************************
                                      ; 1834  ;++
                                      ; 1835  ; TEST  A5      CES REGISTER ALU N BIT
                                      ; 1836  ;
                                      ; 1837  ; TEST DESCRIPTION
                                      ; 1838  ;       THIS TEST CHECKS THE ALU N BIT IN THE CES REGISTER. THIS IS DONE
                                      ; 1839  ;       BY SELECTING THE ALU TO DO A+B AND A-B, AND F=B, WITH SPECIFIC
                                      ; 1840  ;       DATA PATTERNS ON THE AMX AND BMX TO CHECK THE LOGIC THAT GENERATES
                                      ; 1841  ;       THIS BIT.
                                      ; 1842  ;
                                      ; 1843  ;       SUBTST 1 - CHECK THE DATA PATTERNS THAT REQUIRE THE ALU TO
                                      ; 1844  ;                  BE EXECUTING AN A+B TO GET THE CORRECT ALU DATA.
                                      ; 1845  ;       SUBTST 2 - CHECK THE DATA PATTERNS THAT REQUIRE THE ALU TO
                                      ; 1846  ;                  BE EXECUTING AN A-B TO GET THE CORRECT ALU DATA.
                                      ; 1847  ;       SUBTST 3 - CHECK THE DATA PATTERNS THAT REQUIRE THE ALU TO
                                      ; 1848  ;                  BE EXECUTING ANYTHIN BUT AN A+B OR A-B.
                                      ; 1849  ;
                                      ; 1850  ; LOGIC DESCRIPTION
                                      ; 1851  ;       THIS TEST CHECKS THE LOGIC NETWORK ON THE CEH MODULE THAT GENERATES
                                      ; 1852  ;       THE ALU N BIT, AND THE MULTIPLEXOR ON THE ICL MODULE THAT FEEDS
                                      ; 1853  ;       THE ALU N BIT IN THE CES REGISTER.
                                      ; 1854  ;
                                      ; 1855  ; ERROR DESCRIPTION
                                      ; 1856  ;       DATA: EXPECTED CES REGISTER
                                      ; 1857  ;             RECEIVED CES REGISTER
                                      ; 1858  ;             LOOP COUNT - INDICATES WHICH DATA PATTERN IS BEING USED.
                                      ; 1859  ;               (SEE THE DATA AT THE END OF THE TEST)
                                      ; 1860  ;
                                      ; 1861  ; SYNC POINT DESCRIPTION
                                      ; 1862  ;       SUBTST 1 - SYNC1A--ALU N BIT GETS LOADED
                                      ; 1863  ;       SUBTST 2 - SYNC1B--ALU N BIT GETS LOADED
                                      ; 1864  ;       SUBTST 3 - SYNC1C--ALU N BIT GETS LOADED
                                      ; 1865  ;--
                                      ; 1866  ;***********************************************************************
                                      ; 1867  =0
U 1014, 0018,0039,0D80,09F8,0000,10F1 ; 1868  ICLT8:  NEWTST[.3]
U 1015, 0018,0038,6580,0A80,0000,1018 ; 1869          R[0]_K[.10]                   ; ADDRESS OF AMX DATA
U 1018, 0000,003D,0180,0800,0000,1135 ; 1870  =0      CALL,J/UNJAM                  ; CLEAR ANY SBI INTERRUPTS
U 1019, 0018,0038,7580,0A88,0000,1140 ; 1871          R[1]_K[.20]                   ; ADDRESS OF BMX DATA
U 1140, 0018,0038,7980,0A90,0000,1141 ; 1872          R[2]_K[.30]                   ; ADDRESS OF EXPECTED ALU N BIT
U 1141, 0818,0038,4180,0800,0000,1142 ; 1873          D_K[.80]
U 1142, 0100,003C,0180,0800,0000,1143 ; 1874          D_D.LEFT2                     ; GENERATE MASK FOR N BIT
U 1143, 0001,003C,0180,0A98,0000,1144 ; 1875          R[3]_D                        ; SAVE
                                      ; 1876
                                      ; 1877  ;///////////////////////////////////////////////////////////////////////
                                      ; 1878  ;+
                                      ; 1879  ; DO THOSE FUNCTIONS REQUIRING THE ALU TO DO AN A PLUS B
                                      ; 1880  ;-
                                      ; 1881
U 1144, 0018,0038,D580,09E0,0000,101A ; 1882  T8S1:   RC[0C]_K[.6]                  ; SET THE LOP COUNT
U 101A, 0000,003D,0180,0800,0000,1124 ; 1883  =0      SUBTEST
U 101B, 0000,003C,0180,0A00,0200,1145 ; 1884  ICLT8L1:VA_R[0]
U 1145, 0000,803C,0180,C800,0000,1146 ; 1885          D[BYTE]_CACHE,P               ; FETCH AMX DATA
U 1146, 0001,003C,0180,0AA8,0000,1147 ; 1886          R[5]_D                        ; SAVE
U 1147, 0000,003C,0180,0A08,0200,1148 ; 1887          VA_R[1]
U 1148, 0000,803C,0180,C800,0000,1149 ; 1888          D[BYTE]_CACHE,P               ; FETCH BMX DATA
```

Figure 4-7 Microtest Listing Sample



```
>>>TEST

MICRO DIAGNOSTIC  V.05
01,02,03,
NO. OF WCS MODULES = 0001
04,05,06,07,08,09,0A,0B,0C,0D,0E,0F,10,11,12,13,14,15,16,17,
18,19,1A,1B,1C,1D,1E,1F,20,21,22,23,24,25,26,27,28,29,2A,2B,2C,2D,2E,
2F,30,31,32,33,34,35,36,37,38,39,3A,
END PASS 000001

MOUNT FLOPPY #2 & TYPE 'DI'
MIC>DI
3B,
# MEM CTRLS= 00000001
3C,3D,
4K CHIF 00000E08
3E,3F,
CPU TR= 00000010
40,41,42,43,44,45,46,47,48,49,4A,
CTRL 1 MAX ADR+1= 00080000
4B,
CTRL 1 MAX ADR+1= 00080000
4C,4D,
END PASS 000001
```

Figure 4-8 Typical Error-Free Terminal Output



```
ERROR: <PC> TEST: <#> SUBTEST: <#>

DATA:   XXXXXXXX
        XXXXXXXX
            •
            •
            •
        XXXXXXXX

TRACE:  W,X,Y,Z

FAILING MODULES: (M8269 (S13)...

NOTE:
PC IS OCTAL FOR HARDCORE TESTS,
OTHERWISE ALL NUMBERS ARE HEX.
```

Figure 4-9 Error Message Format

The first line items are ERROR, TEST, and SUBTEST. ERROR is the address (PC) of the failing test. In the case of a hardcore test error, the PC is displayed as a six-digit octal address, since these tests are executed out of the LSI-11. In the case of a microtest error, the PC is displayed as a four-digit hexadecimal address since it executes out of WCS.

TEST is the failing test number. Note that this is different from the section number sent to the console terminal during error-free execution. SUBTEST is the failing subtest number. These three first line items are important in referencing the program listings (Paragraph 4.9.2).

The DATA line item represents data used during the particular test. The number of data words displayed depends on the particular test. Generally, in the hardcore tests two words are displayed; the first word is the expected (or good) data, the second word is the received (or bad) data. However, as described in Paragraphs 4.8.2 and 4.8.3, the program listings contain a header describing the data patterns used.

The TRACE line item is involved in the fault isolation procedure in determining the set of modules responsible for the failure.

The last line item is FAILING MODULES. The output of this item represents the failing module and its backplane slot number. In some cases, the output will be several module numbers listed in the order of failure probability. However, in other cases the output will not be a module number. For example, consider the situation of a grounded ID Bus bit. The failure could appear to extend across all boards on the bus. Rather than printing out all related module numbers, the program would print out ID BUS.

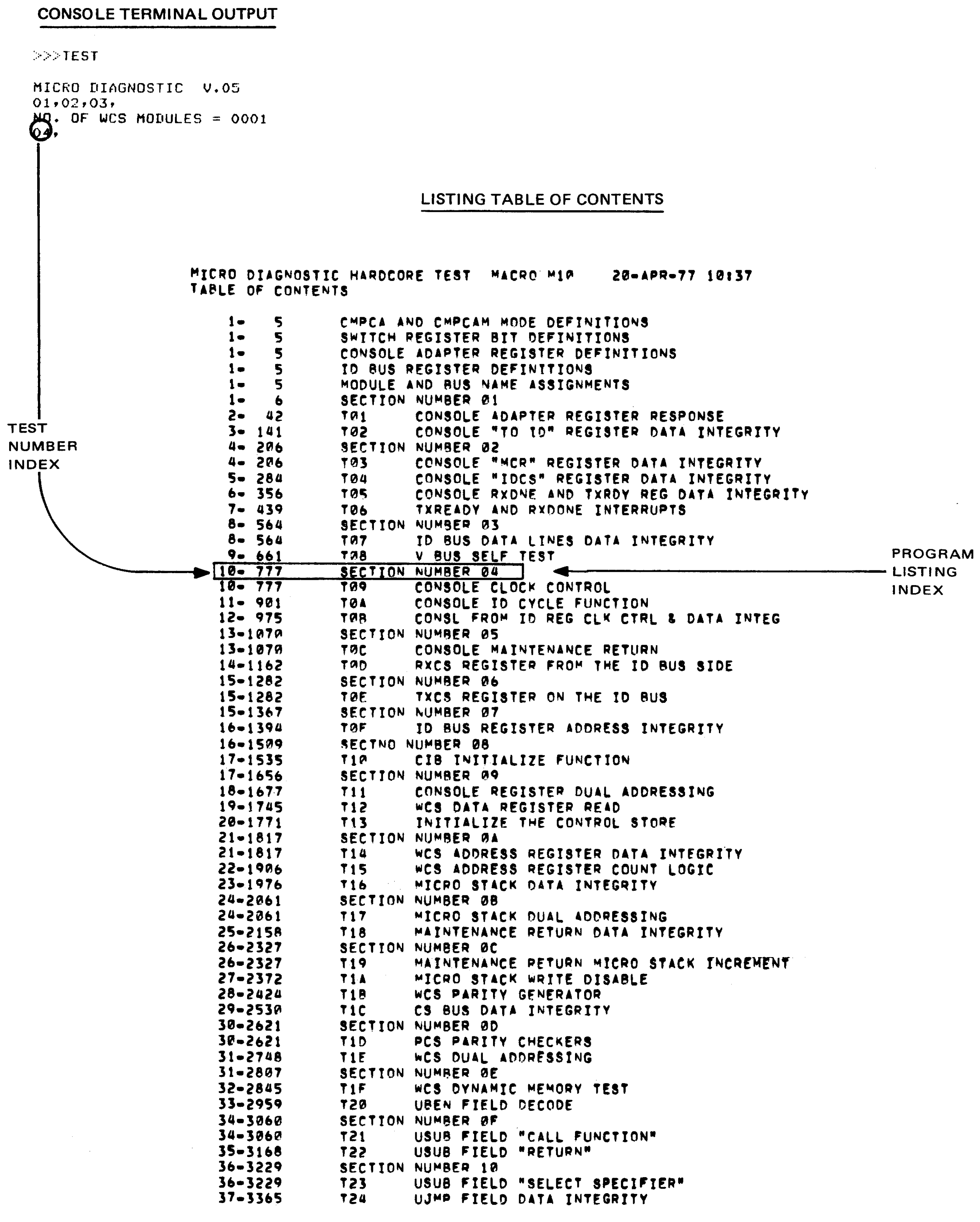
4.9 LISTING/ERROR MESSAGE CORRELATION

This subsection provides basic direction in the use of error message content and its relationship to the program listings. The examples are included mainly to illustrate basic microdiagnostic capabilities.

4.9.1 No Error Message Situation

Consider the situation where the operator has initiated microdiagnostic execution using the TEST command. For one reason or another execution stops in the hardcore tests, and an error message is not printed. As shown in Figure 4-1Ø, execution stops on test section Ø4.

The operator has a reasonable index into the hardcore test listings since section Ø4 is one of the initial sections executed. Referencing the section number in the hardcore listing Table of Contents, the operator finds that the section Ø4 description starts on listing line number 777.



CONSOLE TERMINAL OUTPUT

```
>>>TEST

MICRO DIAGNOSTIC  V.05
01,02,03,
NO. OF WCS MODULES = 0001
04,
```

LISTING TABLE OF CONTENTS

```
MICRO DIAGNOSTIC HARDCORE TEST  MACRO M1Ø    2Ø-APR-77 1Ø:37
TABLE OF CONTENTS

    1-   5      CMPCA AND CMPCAM MODE DEFINITIONS
    1-   5      SWITCH REGISTER BIT DEFINITIONS
    1-   5      CONSOLE ADAPTER REGISTER DEFINITIONS
    1-   5      ID BUS REGISTER DEFINITIONS
    1-   5      MODULE AND BUS NAME ASSIGNMENTS
    1-   6      SECTION NUMBER Ø1
    2-  42      TØ1     CONSOLE ADAPTER REGISTER RESPONSE
    3- 141      TØ2     CONSOLE "TO ID" REGISTER DATA INTEGRITY
    4- 2Ø6      SECTION NUMBER Ø2
    4- 2Ø6      TØ3     CONSOLE "MCR" REGISTER DATA INTEGRITY
    5- 284      TØ4     CONSOLE "IDCS" REGISTER DATA INTEGRITY
    6- 356      TØ5     CONSOLE RXDNE AND TXRDY REG DATA INTEGRITY
    7- 439      TØ6     TXREADY AND RXDONE INTERRUPTS
    8- 564      SECTION NUMBER Ø3
    8- 564      TØ7     ID BUS DATA LINES DATA INTEGRITY
    9- 661      TØ8     V BUS SELF TEST
   1Ø- 777      SECTION NUMBER Ø4
   1Ø- 777      TØ9     CONSOLE CLOCK CONTROL
   11- 9Ø1      TØA     CONSOLE ID CYCLE FUNCTION
   12- 975      TØB     CONSL FROM ID REG CLK CTRL & DATA INTEG
   13-1Ø7Ø      SECTION NUMBER Ø5
   13-1Ø7Ø      TØC     CONSOLE MAINTENANCE RETURN
   14-1162      TØD     RXCS REGISTER FROM THE ID BUS SIDE
   15-1282      SECTION NUMBER Ø6
   15-1282      TØE     TXCS REGISTER ON THE ID BUS
   15-1367      SECTION NUMBER Ø7
   16-1394      TØF     ID BUS REGISTER ADDRESS INTEGRITY
   16-15Ø9      SECTNO NUMBER Ø8
   17-1535      T1Ø     CIB INITIALIZE FUNCTION
   17-1656      SECTION NUMBER Ø9
   18-1677      T11     CONSOLE REGISTER DUAL ADDRESSING
   19-1745      T12     WCS DATA REGISTER READ
   2Ø-1771      T13     INITIALIZE THE CONTROL STORE
   21-1817      SECTION NUMBER ØA
   21-1817      T14     WCS ADDRESS REGISTER DATA INTEGRITY
   22-19Ø6      T15     WCS ADDRESS REGISTER COUNT LOGIC
   23-1976      T16     MICRO STACK DATA INTEGRITY
   24-2Ø61      SECTION NUMBER ØB
   24-2Ø61      T17     MICRO STACK DUAL ADDRESSING
   25-2158      T18     MAINTENANCE RETURN DATA INTEGRITY
   26-2327      SECTION NUMBER ØC
   26-2327      T19     MAINTENANCE RETURN MICRO STACK INCREMENT
   27-2372      T1A     MICRO STACK WRITE DISABLE
   28-2424      T1B     WCS PARITY GENERATOR
   29-253Ø      T1C     CS BUS DATA INTEGRITY
   3Ø-2621      SECTION NUMBER ØD
   3Ø-2621      T1D     PCS PARITY CHECKERS
   31-2748      T1E     WCS DUAL ADDRESSING
   31-28Ø7      SECTION NUMBER ØE
   32-2845      T1F     WCS DYNAMIC MEMORY TEST
   33-2959      T2Ø     UBEN FIELD DECODE
   34-3Ø6Ø      SECTION NUMBER ØF
   34-3Ø6Ø      T21     USUB FIELD "CALL FUNCTION"
   35-3168      T22     USUB FIELD "RETURN"
   36-3229      SECTION NUMBER 1Ø
   36-3229      T23     USUB FIELD "SELECT SPECIFIER"
   37-3365      T24     UJMP FIELD DATA INTEGRITY
```

TK-0770

Figure 4-1Ø Listing Indexing Example

4.9.2 Hardcore Loop and Single Step Setup

During microdiagnostic execution the error message shown in Figure 4-11 is displayed on the console terminal. Since the error PC is a six-digit number (ØØØ67Ø), it is an octal address and indicates a hardcore test. Referencing TEST: 1C in the hardcore Table of Contents indicates that the test begins on line 253Ø. Referencing the error PC of ØØØ67Ø in the program code shows the PC to be at an IFERROR statement on line 2554.

The function of the IFERROR statement (Paragraph 4.4.2) is to produce an error report if a failure is encountered in the test. Usually the IFERROR statement is preceded by a check or compare function (in this case TSTVB). Basically this test is comparing V Bus signals. In this example, the received data did not match the expected data; consequently, an error was detected.

Since the hardcore tests execute out of the LSI-11, a scope loop may be too slow to be of practical use. An alternative is to use the set step instruction and loop commands of the microdiagnostic monitor. As indicated in Figure 4-11, the operator sets the single instruction and loop flags. In this case the loop range is between the statement following the previous ERLOOP statement (line 2551) and the IFERROR statement (line 2554). As shown in Figure 4-11, each time the operator types SPACE, the current PC is displayed. At TPC = ØØØ662 the operator reaches sync point SYNC4C, at which time the operator could scope the CS Bus data bits in an attempt to detect the failing bit. (At this point in the test the microword has just been fetched from WCS and is driving the CS Bus.)

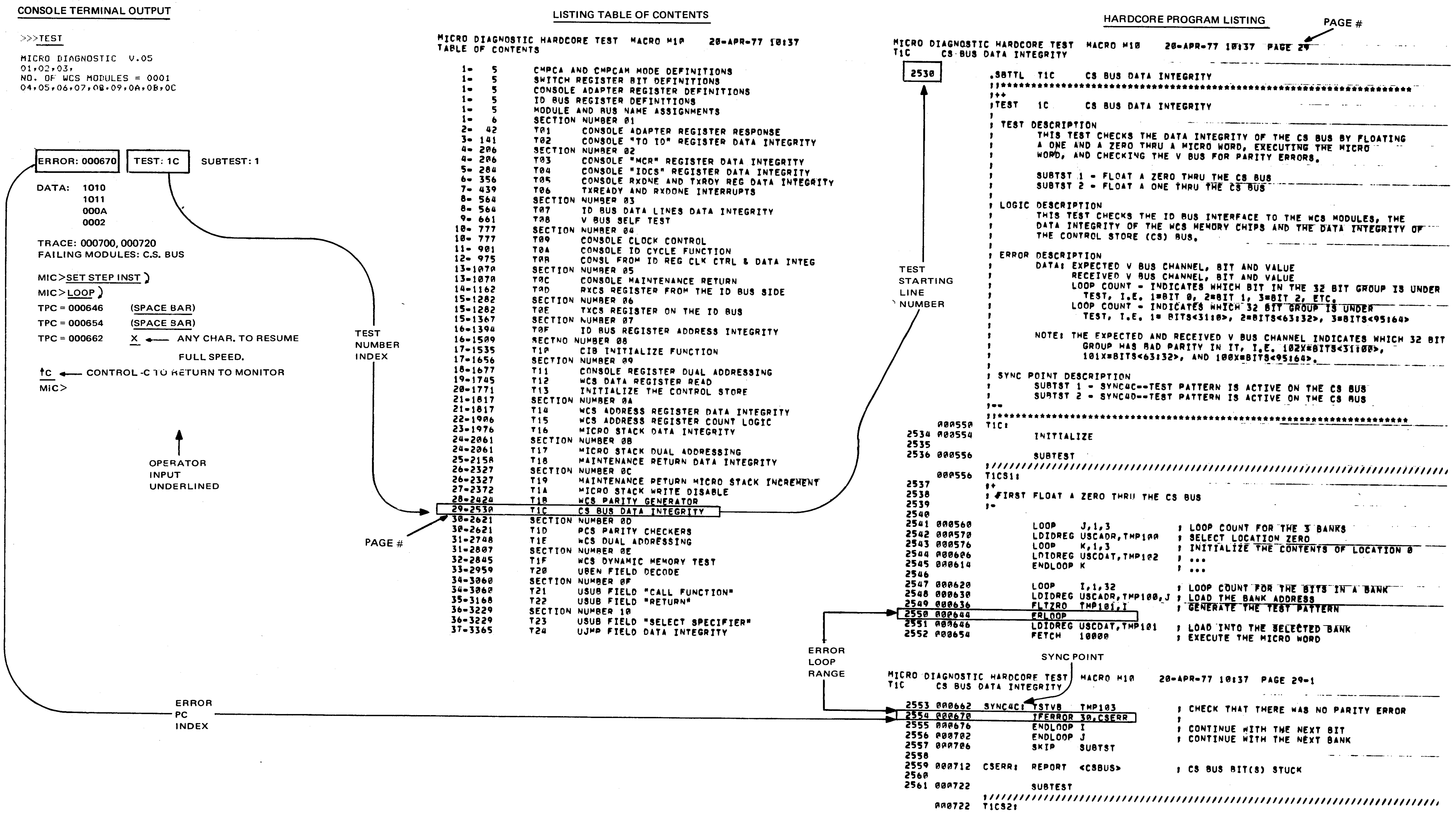
The operator can exit from the step mode by typing any character except SPACE. In the example, Control C has been typed and control returned to the microdiagnostic monitor command mode.

4.9.3 Microtest Scope Loop Setup

During microdiagnostic execution the error message shown in Figure 4-12 is sent to the console terminal. Note that execution stopped on test section 3A.

Since the error PC is a four-digit number (1Ø1E), it is a hexadecimal address and indicates a microtest. Using the test section number of 3A, and referencing the Table of Contents for that section, test A5 starts on line 1832. A look at the PC column (of the microtest listing) shows that the error PC is on line 19Ø6.

By scanning back through the microcode, select a sync point, in this case SYNC1A on line 19Ø1. Control is returned to the microdiagnostic monitor via Control C. The operator enters a CLEAR SOMM: <1153> command. This command will clear the stop on micromatch bit, and produce a sync pulse when the UPC equals the content of the microbreak register (i.e., 1153). The operator then enters a loop command. This sequence causes the test to begin looping and produce a sync pulse each time the UPC = 1153.

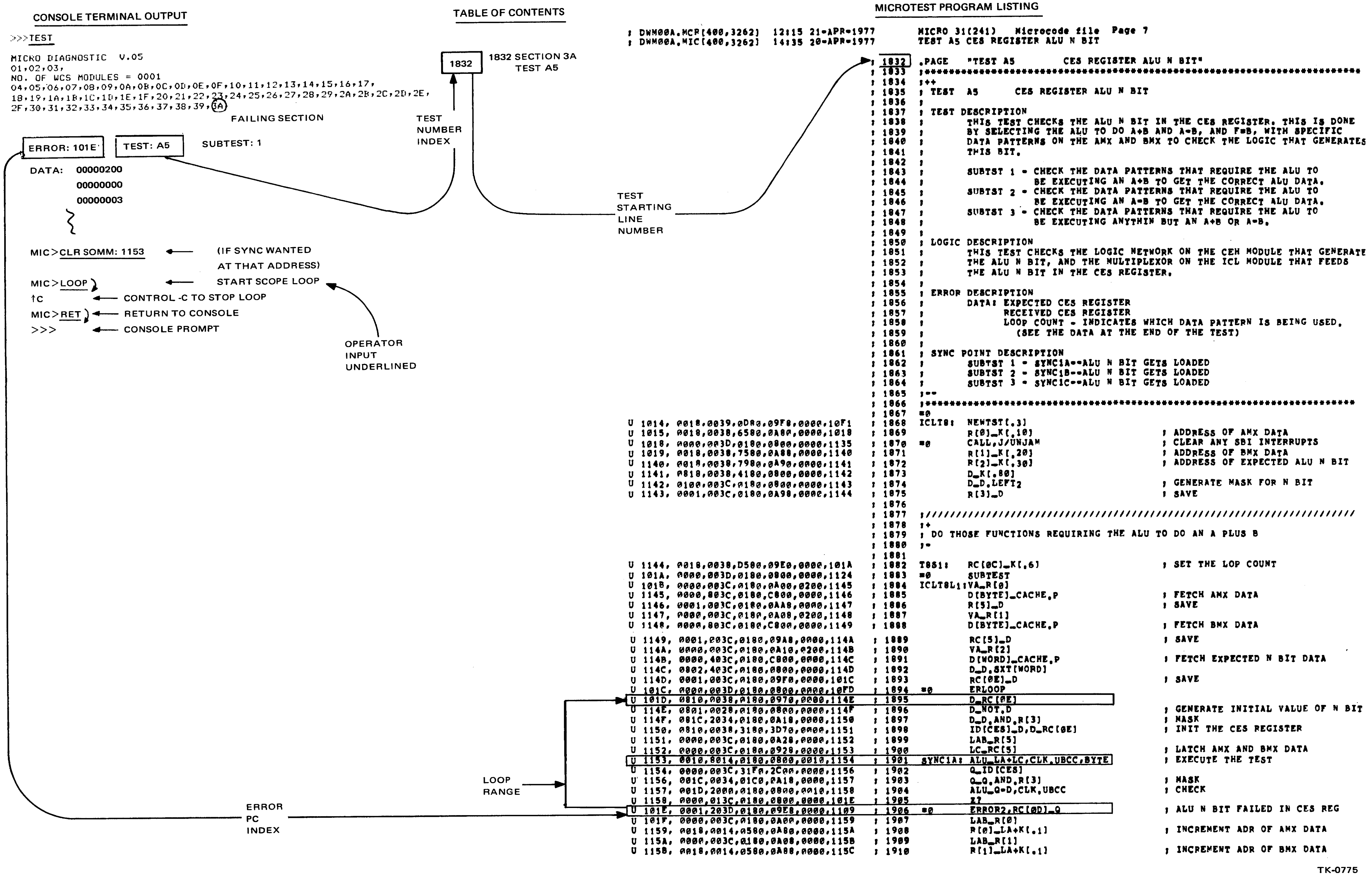


```
>>>TEST

MICRO DIAGNOSTIC  V.05
01,02,03,
NO. OF WCS MODULES = 0001
04,05,06,07,08,09,0A,0B,0C


ERROR: 000670   TEST: 1C   SUBTEST: 1

DATA:   1010
        1011
        000A
        0002

TRACE: 000700, 000720
FAILING MODULES: C.S. BUS

MIC>SET STEP INST )
MIC>LOOP )
TPC = 000646      (SPACE BAR)
TPC = 000654      (SPACE BAR)
TPC = 000662      X <--- ANY CHAR. TO RESUME
                         FULL SPEED.
↑C <--- CONTROL -C TO RETURN TO MONITOR
MIC>
```

```
MICRO DIAGNOSTIC HARDCORE TEST  MACRO M10    20-APR-77 10:37
TABLE OF CONTENTS

   1-   5      CMPCA AND CMPCAM MODE DEFINITIONS
   1-   5      SWITCH REGISTER BIT DEFINITIONS
   1-   5      CONSOLE ADAPTER REGISTER DEFINITIONS
   1-   5      ID BUS REGISTER DEFINITIONS
   1-   5      MODULE AND BUS NAME ASSIGNMENTS
   1-   6      SECTION NUMBER 01
   2-  42      T01      CONSOLE ADAPTER REGISTER RESPONSE
   3- 141      T02      CONSOLE "TO ID" REGISTER DATA INTEGRITY
   4- 206      SECTION NUMBER 02
   4- 206      T03      CONSOLE "MCR" REGISTER DATA INTEGRITY
   5- 284      T04      CONSOLE "IDCS" REGISTER DATA INTEGRITY
   6- 356      T05      CONSOLE RXDNE AND TXRDY REG DATA INTEGRITY
   7- 439      T06      TXREADY AND RXDONE INTERRUPTS
   8- 564      SECTION NUMBER 03
   8- 564      T07      ID BUS DATA LINES DATA INTEGRITY
   9- 661      T08      V BUS SELF TEST
  10- 777      SECTION NUMBER 04
  10- 777      T09      CONSOLE CLOCK CONTROL
  11- 901      T0A      CONSOLE ID CYCLE FUNCTION
  12- 975      T0B      CONSL FROM ID REG CLK CTRL & DATA INTEG
  13-1070      SECTION NUMBER 05
  13-1070      T0C      CONSOLE MAINTENANCE RETURN
  14-1162      T0D      RXCS REGISTER FROM THE ID BUS SIDE
  15-1282      SECTION NUMBER 06
  15-1282      T0E      TXCS REGISTER ON THE ID BUS
  15-1367      SECTION NUMBER 07
  16-1394      T0F      ID BUS REGISTER ADDRESS INTEGRITY
  16-1509      SECTNO NUMBER 08
  17-1535      T10      CIB INITIALIZE FUNCTION
  17-1656      SECTION NUMBER 09
  18-1677      T11      CONSOLE REGISTER DUAL ADDRESSING
  19-1745      T12      WCS DATA REGISTER READ
  20-1771      T13      INITIALIZE THE CONTROL STORE
  21-1817      SECTION NUMBER 0A
  21-1817      T14      WCS ADDRESS REGISTER DATA INTEGRITY
  22-1906      T15      WCS ADDRESS REGISTER COUNT LOGIC
  23-1976      T16      MICRO STACK DATA INTEGRITY
  24-2061      SECTION NUMBER 0B
  24-2061      T17      MICRO STACK DUAL ADDRESSING
  25-2158      T18      MAINTENANCE RETURN DATA INTEGRITY
  26-2327      SECTION NUMBER 0C
  26-2327      T19      MAINTENANCE RETURN MICRO STACK INCREMENT
  27-2372      T1A      MICRO STACK WRITE DISABLE
  28-2424      T1B      WCS PARITY GENERATOR
  29-2530      T1C      CS BUS DATA INTEGRITY
  30-2621      SECTION NUMBER 0D
  30-2621      T1D      PCS PARITY CHECKERS
  31-2748      T1E      WCS DUAL ADDRESSING
  31-2807      SECTION NUMBER 0E
  32-2845      T1F      WCS DYNAMIC MEMORY TEST
  33-2959      T20      UBEN FIELD DECODE
  34-3060      SECTION NUMBER 0F
  34-3060      T21      USUB FIELD "CALL FUNCTION"
  35-3168      T22      USUB FIELD "RETURN"
  36-3229      SECTION NUMBER 10
  36-3229      T23      USUB FIELD "SELECT SPECIFIER"
  37-3365      T24      UJMP FIELD DATA INTEGRITY
```

```
MICRO DIAGNOSTIC HARDCORE TEST  MACRO M10    20-APR-77 10:37  PAGE 29
T1C      CS BUS DATA INTEGRITY

2530           .SBTTL  T1C      CS BUS DATA INTEGRITY
               ;;*************************************************************
               ;++
               ;TEST   1C       CS BUS DATA INTEGRITY
               ;
               ; TEST DESCRIPTION
               ;        THIS TEST CHECKS THE DATA INTEGRITY OF THE CS BUS BY FLOATING
               ;        A ONE AND A ZERO THRU A MICRO WORD, EXECUTING THE MICRO
               ;        WORD, AND CHECKING THE V BUS FOR PARITY ERRORS.
               ;
               ;        SUBTST 1 - FLOAT A ZERO THRU THE CS BUS
               ;        SUBTST 2 - FLOAT A ONE THRU THE CS BUS
               ;
               ; LOGIC DESCRIPTION
               ;        THIS TEST CHECKS THE ID BUS INTERFACE TO THE WCS MODULES, THE
               ;        DATA INTEGRITY OF THE WCS MEMORY CHIPS AND THE DATA INTEGRITY OF
               ;        THE CONTROL STORE (CS) BUS.
               ;
               ; ERROR DESCRIPTION
               ;        DATA: EXPECTED V BUS CHANNEL, BIT AND VALUE
               ;              RECEIVED V BUS CHANNEL, BIT AND VALUE
               ;              LOOP COUNT - INDICATES WHICH BIT IN THE 32 BIT GROUP IS UNDER
               ;                TEST, I.E. 1=BIT 0, 2=BIT 1, 3=BIT 2, ETC.
               ;              LOOP COUNT - INDICATES WHICH 32 BIT GROUP IS UNDER
               ;                TEST, I.E. 1= BITS<31:0>, 2=BITS<63:32>, 3=BITS<95:64>
               ;
               ;        NOTE: THE EXPECTED AND RECEIVED V BUS CHANNEL INDICATES WHICH 32 BIT
               ;              GROUP HAS BAD PARITY IN IT, I.E. 102X=BITS<31:00>,
               ;              101X=BITS<63:32>, AND 100X=BITS<95:64>.
               ;
               ; SYNC POINT DESCRIPTION
               ;        SUBTST 1 - SYNC4C--TEST PATTERN IS ACTIVE ON THE CS BUS
               ;        SUBTST 2 - SYNC4D--TEST PATTERN IS ACTIVE ON THE CS BUS
               ;--
               ;;*************************************************************
      000550   T1C:
2534  000554           INITIALIZE
2535
2536  000556           SUBTEST
               ;//////////////////////////////////////////////////////////////////
      000556   T1CS1:
2537           ;+
2538           ; FIRST FLOAT A ZERO THRU THE CS BUS
2539           ;-
2540
2541  000560           LOOP    J,1,3                ; LOOP COUNT FOR THE 3 BANKS
2542  000570           LDIDREG USCADR,TMP100        ; SELECT LOCATION ZERO
2543  000576           LOOP    K,1,3                ; INITIALIZE THE CONTENTS OF LOCATION 0
2544  000606           LDIDREG USCDAT,TMP102        ; ...
2545  000614           ENDLOOP K                    ; ...
2546
2547  000620           LOOP    I,1,32               ; LOOP COUNT FOR THE BITS IN A BANK
2548  000630           LDIDREG USCADR,TMP100,J      ; LOAD THE BANK ADDRESS
2549  000636           FLTZRO  TMP101,I             ; GENERATE THE TEST PATTERN
2550  000644           ERLOOP
2551  000646           LDIDREG USCDAT,TMP101        ; LOAD INTO THE SELECTED BANK
2552  000654           FETCH   10000                ; EXECUTE THE MICRO WORD

MICRO DIAGNOSTIC HARDCORE TEST  MACRO M10    20-APR-77 10:37  PAGE 29-1
T1C      CS BUS DATA INTEGRITY

2553  000662   SYNC4C: TSTVB   TMP103               ; CHECK THAT THERE WAS NO PARITY ERROR
2554  000670           IFERROR 30,CSERR             ;
2555  000676           ENDLOOP I                    ; CONTINUE WITH THE NEXT BIT
2556  000702           ENDLOOP J                    ; CONTINUE WITH THE NEXT BANK
2557  000706           SKIP    SUBTST
2558
2559  000712   CSERR:  REPORT  <CSBUS>              ; CS BUS BIT(S) STUCK
2560
2561  000722           SUBTEST
               ;//////////////////////////////////////////////////////////////////
      000722   T1CS2:
```

Figure 4-11 Loop and Single Example



```
>>>TEST

MICRO DIAGNOSTIC  V.05
01,02,03,
NO. OF WCS MODULES = 0001
04,05,06,07,08,09,0A,0B,0C,0D,0E,0F,10,11,12,13,14,15,16,17,
18,19,1A,1B,1C,1D,1E,1F,20,21,22,23,24,25,26,27,28,29,2A,2B,2C,2D,2E,
2F,30,31,32,33,34,35,36,37,38,39,3A

ERROR: 101E   TEST: A5     SUBTEST: 1
 DATA:   00000200
         00000000
         00000003

 MIC>CLR SOMM: 1153
 MIC>LOOP
 ↑C
 MIC>RET
 >>>
```

```
; DWM00A.MCR[400,3262]  12:15 21-APR-1977        MICRO 31(241)   Microcode file  Page 7
; DWM00A.MIC[400,3262]  14:35 20-APR-1977        TEST A5 CES REGISTER ALU N BIT

                                          ; 1832  .PAGE   "TEST A5        CES REGISTER ALU N BIT"
                                          ; 1833  ;*********************************************************************
                                          ; 1834  ;++
                                          ; 1835  ; TEST  A5      CES REGISTER ALU N BIT
                                          ; 1836  ;
                                          ; 1837  ; TEST DESCRIPTION
                                          ; 1838  ;       THIS TEST CHECKS THE ALU N BIT IN THE CES REGISTER. THIS IS DONE
                                          ; 1839  ;       BY SELECTING THE ALU TO DO A+B AND A-B, AND F=B, WITH SPECIFIC
                                          ; 1840  ;       DATA PATTERNS ON THE AMX AND BMX TO CHECK THE LOGIC THAT GENERATES
                                          ; 1841  ;       THIS BIT.
                                          ; 1842  ;
                                          ; 1843  ;       SUBTST 1 - CHECK THE DATA PATTERNS THAT REQUIRE THE ALU TO
                                          ; 1844  ;                  BE EXECUTING AN A+B TO GET THE CORRECT ALU DATA.
                                          ; 1845  ;       SUBTST 2 - CHECK THE DATA PATTERNS THAT REQUIRE THE ALU TO
                                          ; 1846  ;                  BE EXECUTING AN A-B TO GET THE CORRECT ALU DATA.
                                          ; 1847  ;       SUBTST 3 - CHECK THE DATA PATTERNS THAT REQUIRE THE ALU TO
                                          ; 1848  ;                  BE EXECUTING ANYTHIN BUT AN A+B OR A-B.
                                          ; 1849  ;
                                          ; 1850  ; LOGIC DESCRIPTION
                                          ; 1851  ;       THIS TEST CHECKS THE LOGIC NETWORK ON THE CEH MODULE THAT GENERATE
                                          ; 1852  ;       THE ALU N BIT, AND THE MULTIPLEXOR ON THE ICL MODULE THAT FEEDS
                                          ; 1853  ;       THE ALU N BIT IN THE CES REGISTER.
                                          ; 1854  ;
                                          ; 1855  ; ERROR DESCRIPTION
                                          ; 1856  ;       DATA: EXPECTED CES REGISTER
                                          ; 1857  ;             RECEIVED CES REGISTER
                                          ; 1858  ;             LOOP COUNT - INDICATES WHICH DATA PATTERN IS BEING USED.
                                          ; 1859  ;               (SEE THE DATA AT THE END OF THE TEST)
                                          ; 1860  ;
                                          ; 1861  ; SYNC POINT DESCRIPTION
                                          ; 1862  ;       SUBTST 1 - SYNC1A--ALU N BIT GETS LOADED
                                          ; 1863  ;       SUBTST 2 - SYNC1B--ALU N BIT GETS LOADED
                                          ; 1864  ;       SUBTST 3 - SYNC1C--ALU N BIT GETS LOADED
                                          ; 1865  ;--
                                          ; 1866  ;*********************************************************************
                                          ; 1867  =0
U 1014, 0018,0039,0D80,09F8,0000,10F1     ; 1868  ICLT8:  NEWTST[,3]
U 1015, 0018,0038,6580,0A80,0000,1018     ; 1869          R[0]_K[,10]                   ; ADDRESS OF AMX DATA
U 1018, 0000,003D,0180,0800,0000,1135     ; 1870  =0      CALL,J/UNJAM                  ; CLEAR ANY SBI INTERRUPTS
U 1019, 0018,0038,7580,0A88,0000,1140     ; 1871          R[1]_K[,20]                   ; ADDRESS OF BMX DATA
U 1140, 0018,0038,7980,0A90,0000,1141     ; 1872          R[2]_K[,30]                   ; ADDRESS OF EXPECTED ALU N BIT
U 1141, 0818,0038,4180,0800,0000,1142     ; 1873          D_K[,80]
U 1142, 0100,003C,0180,0800,0000,1143     ; 1874          D_D.LEFT2                     ; GENERATE MASK FOR N BIT
U 1143, 0001,003C,0180,0A98,0000,1144     ; 1875          R[3]_D                        ; SAVE
                                          ; 1876
                                          ; 1877  ;/////////////////////////////////////////////////////////////////////
                                          ; 1878  ;+
                                          ; 1879  ; DO THOSE FUNCTIONS REQUIRING THE ALU TO DO AN A PLUS B
                                          ; 1880  ;-
                                          ; 1881
U 1144, 0018,0038,D580,09E0,0000,101A     ; 1882  T8S1:   RC[0C]_K[,6]                  ; SET THE LOP COUNT
U 101A, 0000,003D,0180,0800,0000,1124     ; 1883  =0      SUBTEST
U 101B, 0000,003C,0180,0A00,0200,1145     ; 1884  ICLT8L1:VA_R[0]
U 1145, 0000,803C,0180,C800,0000,1146     ; 1885          D[BYTE]_CACHE.P               ; FETCH AMX DATA
U 1146, 0001,003C,0180,0AA8,0000,1147     ; 1886          R[5]_D                        ; SAVE
U 1147, 0000,003C,0180,0A08,0200,1148     ; 1887          VA_R[1]
U 1148, 0000,803C,0180,C800,0000,1149     ; 1888          D[BYTE]_CACHE.P               ; FETCH BMX DATA
U 1149, 0001,003C,0180,09A8,0000,114A     ; 1889          RC[5]_D                       ; SAVE
U 114A, 0000,003C,0180,0A10,0200,114B     ; 1890          VA_R[2]
U 114B, 0000,403C,0180,C800,0000,114C     ; 1891          D[WORD]_CACHE.P               ; FETCH EXPECTED N BIT DATA
U 114C, 0802,403C,0180,0800,0000,114D     ; 1892          D_D.SXT[WORD]
U 114D, 0001,003C,0180,09F0,0000,101C     ; 1893          RC[0E]_D                      ; SAVE
U 101C, 0000,003D,0180,0800,0000,10FD     ; 1894  =0      ERLOOP
U 101D, 0810,0038,0180,0970,0000,114E     ; 1895          D_RC[0E]
U 114E, 0801,0028,0180,0800,0000,114F     ; 1896          D_NOT.D                       ; GENERATE INITIAL VALUE OF N BIT
U 114F, 081C,2034,0180,0A18,0000,1150     ; 1897          D_D.AND.R[3]                  ; MASK
U 1150, 0810,0038,3180,3D70,0000,1151     ; 1898          ID[CES]_D,D_RC[0E]            ; INIT THE CES REGISTER
U 1151, 0000,003C,0180,0A28,0000,1152     ; 1899          LAB_R[5]
U 1152, 0000,003C,0180,0928,0000,1153     ; 1900          LC_RC[5]                      ; LATCH AMX AND BMX DATA
U 1153, 0010,8014,0180,0800,0010,1154     ; 1901  SYNC1A: ALU_LA+LC,CLK.UBCC,BYTE       ; EXECUTE THE TEST
U 1154, 0000,003C,31F0,2C00,0000,1156     ; 1902          Q_ID[CES]
U 1156, 001C,0034,01C0,0A18,0000,1157     ; 1903          Q_Q.AND.R[3]                  ; MASK
U 1157, 001D,2000,0180,0800,0010,1158     ; 1904          ALU_Q-D,CLK,UBCC              ; CHECK
U 1158, 0000,013C,0180,0800,0000,101E     ; 1905          Z?
U 101E, 0001,203D,0180,09E8,0000,1109     ; 1906  =0      ERROR2,RC[0D]_Q               ; ALU N BIT FAILED IN CES REG
U 101F, 0000,003C,0180,0A00,0000,1159     ; 1907          LAB_R[0]
U 1159, 0018,0014,0580,0A80,0000,115A     ; 1908          R[0]_LA+K[.1]                 ; INCREMENT ADR OF AMX DATA
U 115A, 0000,003C,0180,0A08,0000,115B     ; 1909          LAB_R[1]
U 115B, 0018,0014,0580,0A88,0000,115C     ; 1910          R[1]_LA+K[.1]                 ; INCREMENT ADR OF BMX DATA
```

Figure 4-12 Microtest Scope Loop Example

As shown in Figure 4-12, the operator has entered a Control C followed by a RETURN (RET) command. This sequence breaks the test loop and returns control to the console program.

4.9.4 Microtest Single Bus Step Setup

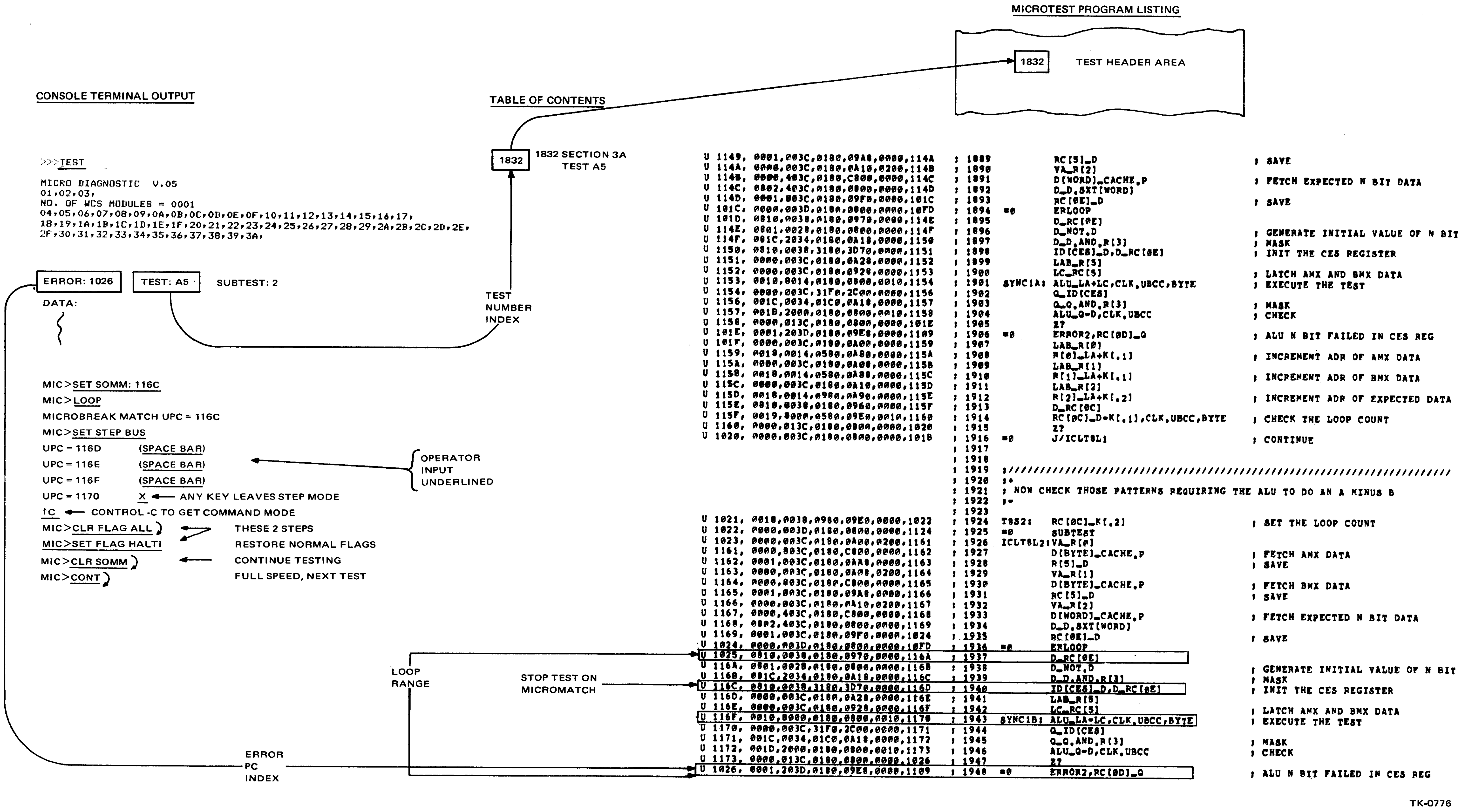
During microdiagnostic execution the error message shown in Figure 4-13 is printed on the console terminal. As in the previous example, execution stopped on test section 3A.

As in Paragraph 4.9.3, the error PC is a four-digit number indicating a microtest. A look at the section 3A listing indicates that the error PC is on line 1948. At this point it is decided to use the single bus step capability.

A scan backward through the microcode indicates a possible loop between SYNC1B (line 1943) and ERLOOP (line 1936). A point in the loop is chosen to stop the microtest, in this case UPC 116C (line 194Ø). The operator enters SET SOMM: 116C, which sets the stop on micromatch bit and loads 116C into the microbreak register. A loop command is then entered which initiates execution of the loop. When the loop reaches UPC 116C, the microtest halts and prints the UPC on the console terminal.

At this point, the operator enters the bus cycle mode (set step bus command). Each time the operator types SPACE, a single bus cycle is executed and the UPC is displayed on the console terminal. At any point in the loop the operator may scope the current conditions.

As shown in Figure 4-13, the operator has exited from step mode by typing any character other than SPACE. The program control flags previously set are cleared. The HI flag is set to restore the normal default case. A CLEAR SOMM is then performed to clear the stop on micromatch bit and the microbreak register, then a CONTINUE is performed to begin normal test execution at the next sequential test (i.e., A6). If the operator feels that the problem has been cleared, it is probably more practical to start the tests over rather than to begin at the next test.



```
>>>TEST

MICRO DIAGNOSTIC  V.05
01,02,03,
NO. OF WCS MODULES = 0001
04,05,06,07,08,09,0A,0B,0C,0D,0E,0F,10,11,12,13,14,15,16,17,
18,19,1A,1B,1C,1D,1E,1F,20,21,22,23,24,25,26,27,28,29,2A,2B,2C,2D,2E,
2F,30,31,32,33,34,35,36,37,38,39,3A,

ERROR: 1026    TEST: A5    SUBTEST: 2
DATA:

MIC>SET SOMM: 116C
MIC>LOOP
MICROBREAK MATCH UPC = 116C
MIC>SET STEP BUS
UPC = 116D        (SPACE BAR)
UPC = 116E        (SPACE BAR)
UPC = 116F        (SPACE BAR)
UPC = 1170        X
↑C
MIC>CLR FLAG ALL
MIC>SET FLAG HALTI
MIC>CLR SOMM
MIC>CONT
```

```
U 1149, 0001,003C,0180,09A8,0000,114A   ; 1889          RC[5]_D                         ; SAVE
U 114A, 0000,003C,0180,0A10,0200,114B   ; 1890          VA_R[2]
U 114B, 0000,403C,0180,C800,0000,114C   ; 1891          D[WORD]_CACHE.P                 ; FETCH EXPECTED N BIT DATA
U 114C, 0802,403C,0180,0800,0000,114D   ; 1892          D_D.SXT[WORD]
U 114D, 0001,003C,0180,09F0,0000,101C   ; 1893          RC[ØE]_D                        ; SAVE
U 101C, 0000,003D,0180,0800,0000,10FD   ; 1894  =Ø      ERLOOP
U 101D, 0810,0038,0180,0970,0000,114E   ; 1895          D_RC[ØE]
U 114E, 0801,0028,0180,0800,0000,114F   ; 1896          D_NOT.D                         ; GENERATE INITIAL VALUE OF N BIT
U 114F, 081C,2034,0180,0A18,0000,1150   ; 1897          D_D.AND.R[3]                    ; MASK
U 1150, 0810,0038,3180,3D70,0000,1151   ; 1898          ID[CES]_D,D_RC[ØE]              ; INIT THE CES REGISTER
U 1151, 0000,003C,0180,0A28,0000,1152   ; 1899          LAB_R[5]
U 1152, 0000,003C,0180,0928,0000,1153   ; 1900          LC_RC[5]                        ; LATCH AMX AND BMX DATA
U 1153, 0010,8014,0180,0800,0010,1154   ; 1901  SYNC1A: ALU_LA+LC,CLK.UBCC,BYTE         ; EXECUTE THE TEST
U 1154, 0000,003C,31F0,2C00,0000,1156   ; 1902          Q_ID[CES]
U 1156, 001C,0034,01C0,0A18,0000,1157   ; 1903          Q_Q.AND.R[3]                    ; MASK
U 1157, 001D,2000,0180,0800,0010,1158   ; 1904          ALU_Q-D,CLK.UBCC                ; CHECK
U 1158, 0000,013C,0180,0800,0000,101E   ; 1905          Z?
U 101E, 0001,203D,0180,09E8,0000,1109   ; 1906  =Ø      ERROR2,RC[ØD]_Q                 ; ALU N BIT FAILED IN CES REG
U 101F, 0000,003C,0180,0A00,0000,1159   ; 1907          LAB_R[Ø]
U 1159, 0018,0014,0580,0A80,0000,115A   ; 1908          R[Ø]_LA+K[.1]                   ; INCREMENT ADR OF AMX DATA
U 115A, 0000,003C,0180,0A08,0000,115B   ; 1909          LAB_R[1]
U 115B, 0018,0014,0580,0A88,0000,115C   ; 1910          R[1]_LA+K[.1]                   ; INCREMENT ADR OF BMX DATA
U 115C, 0000,003C,0180,0A10,0000,115D   ; 1911          LAB_R[2]
U 115D, 0018,0014,0980,0A90,0000,115E   ; 1912          R[2]_LA+K[.2]                   ; INCREMENT ADR OF EXPECTED DATA
U 115E, 0810,0038,0180,0960,0000,115F   ; 1913          D_RC[ØC]
U 115F, 0019,8000,0580,09E0,0010,1160   ; 1914          RC[ØC]_D-K[.1],CLK.UBCC,BYTE    ; CHECK THE LOOP COUNT
U 1160, 0000,013C,0180,0800,0000,1020   ; 1915          Z?
U 1020, 0000,003C,0180,0800,0000,101B   ; 1916  =Ø      J/ICLT8L1                       ; CONTINUE
                                        ; 1917
                                        ; 1918
                                        ; 1919  ;////////////////////////////////////////////////////////////////////
                                        ; 1920  ;+
                                        ; 1921  ; NOW CHECK THOSE PATTERNS REQUIRING THE ALU TO DO AN A MINUS B
                                        ; 1922  ;-
                                        ; 1923
U 1021, 0018,0038,0980,09E0,0000,1022   ; 1924  T8S2:   RC[ØC]_K[.2]                    ; SET THE LOOP COUNT
U 1022, 0000,003D,0180,0800,0000,1124   ; 1925  =Ø      SUBTEST
U 1023, 0000,003C,0180,0A00,0200,1161   ; 1926  ICLT8L2:VA_R[Ø]
U 1161, 0000,803C,0180,C800,0000,1162   ; 1927          D[BYTE]_CACHE.P                 ; FETCH AMX DATA
U 1162, 0001,003C,0180,0AA8,0000,1163   ; 1928          R[5]_D                          ; SAVE
U 1163, 0000,003C,0180,0A08,0200,1164   ; 1929          VA_R[1]
U 1164, 0000,803C,0180,C800,0000,1165   ; 1930          D[BYTE]_CACHE.P                 ; FETCH BMX DATA
U 1165, 0001,003C,0180,09A8,0000,1166   ; 1931          RC[5]_D                         ; SAVE
U 1166, 0000,003C,0180,0A10,0200,1167   ; 1932          VA_R[2]
U 1167, 0000,403C,0180,C800,0000,1168   ; 1933          D[WORD]_CACHE.P                 ; FETCH EXPECTED N BIT DATA
U 1168, 0802,403C,0180,0800,0000,1169   ; 1934          D_D.SXT[WORD]
U 1169, 0001,003C,0180,09F0,0000,1024   ; 1935          RC[ØE]_D                        ; SAVE
U 1024, 0000,003D,0180,0800,0000,10FD   ; 1936  =Ø      ERLOOP
U 1025, 0810,0038,0180,0970,0000,116A   ; 1937          D_RC[ØE]
U 116A, 0801,0028,0180,0800,0000,116B   ; 1938          D_NOT.D                         ; GENERATE INITIAL VALUE OF N BIT
U 116B, 001C,2034,0180,0A18,0000,116C   ; 1939          D_D.AND.R[3]                    ; MASK
U 116C, 0810,0038,3180,3D70,0000,116D   ; 1940          ID[CES]_D,D_RC[ØE]              ; INIT THE CES REGISTER
U 116D, 0000,003C,0180,0A28,0000,116E   ; 1941          LAB_R[5]
U 116E, 0000,003C,0180,0928,0000,116F   ; 1942          LC_RC[5]                        ; LATCH AMX AND BMX DATA
U 116F, 0010,8000,0180,0800,0010,1170   ; 1943  SYNC1B: ALU_LA-LC,CLK.UBCC,BYTE         ; EXECUTE THE TEST
U 1170, 0000,003C,31F0,2C00,0000,1171   ; 1944          Q_ID[CES]
U 1171, 001C,0034,01C0,0A18,0000,1172   ; 1945          Q_Q.AND.R[3]                    ; MASK
U 1172, 001D,2000,0180,0800,0010,1173   ; 1946          ALU_Q-D,CLK.UBCC                ; CHECK
U 1173, 0000,013C,0180,0800,0000,1026   ; 1947          Z?
U 1026, 0001,203D,0180,09E8,0000,1109   ; 1948  =Ø      ERROR2,RC[ØD]_Q                 ; ALU N BIT FAILED IN CES REG
```

Figure 4-13 Microtest Single Bus Example

CHAPTER 5
MACRODIAGNOSTIC PROGRAM DESCRIPTIONS

5.1 DEFINITION OF TERMS

Module -- The diagnostic programs are written in a modular format. Each module (file) is a part of the program assembled separately. Modular programming allows the development of large programs in which separate parts share data and routines.

Assembler -- The MARS assembler (which runs on a PDP-11) and the VAX-11 Macro assembler (which runs on a VAX-11) are programs that accept one or more source modules written in MACRO assembly language and produce relocatable object modules and symbol tables.

Linker -- The VAX/VMS linker and the cross linker accept as input one or more native code object modules produced by the assembler. Linking consists of three basic operations.

1. Allocation of virtual memory addresses

2. Resolution of intermodule symbolic references (global symbols)

3. Initialization of the contents of a memory image.

Program Defined Symbols -- Program defined symbols (and labels) are either internal or external (global) to a source program module. An internal symbol definition (and reference) is limited to the module in which it appears. Internal symbols used by the diagnostics are temporary definitions that are resolved by the assembler.

A global symbol can be defined in one source program module and referenced by another. Global symbols are preserved in the object module and are not resolved until the object modules are linked into an executable program by the linker.

Program Sections -- The assembler creates a number of program sections (.PSECT) within a module, according to directives by the program developer. In addition, any code that precedes the first defined program section is placed in the BLANK program section by the assembler.

Through program sectioning the program developer controls the virtual memory allocation of a program. Any program attributes established by the program section directive are passed on to the linker. Thus program sections can be declared as read-only, non-executable, etc. Refer to the VAX-11 MACRO Language Reference Manual for an explanation of the various program section attribute functions.

In the diagnostic programs, each test is given a separate program section.

5.2 OVERVIEW OF THE MACRODIAGNOSTIC PROGRAM

The macrodiagnostic programs and the diagnostic supervisor are written in VAX-11 native code. Each of the programs (and the supervisor) consists of modules. These modules are separate files, which are assembled separately and then linked by the linker program. Each module contains one or more program sections. The program sections and routines are organized according to a common format and a set of conventions that enable them to interact with the supervisor. Note that the listings described in this chapter are those assembled by the MARS assembler and linked by the cross linker in compatibility mode. The format will change when the native assembler and linker are used.

5.3 MACRODIAGNOSTIC PROGRAM LISTING DESCRIPTION

This section describes the program listings in general terms. Illustrations and examples are taken from the MBA RH78Ø diagnostic program. The formats of the other listings are similar.

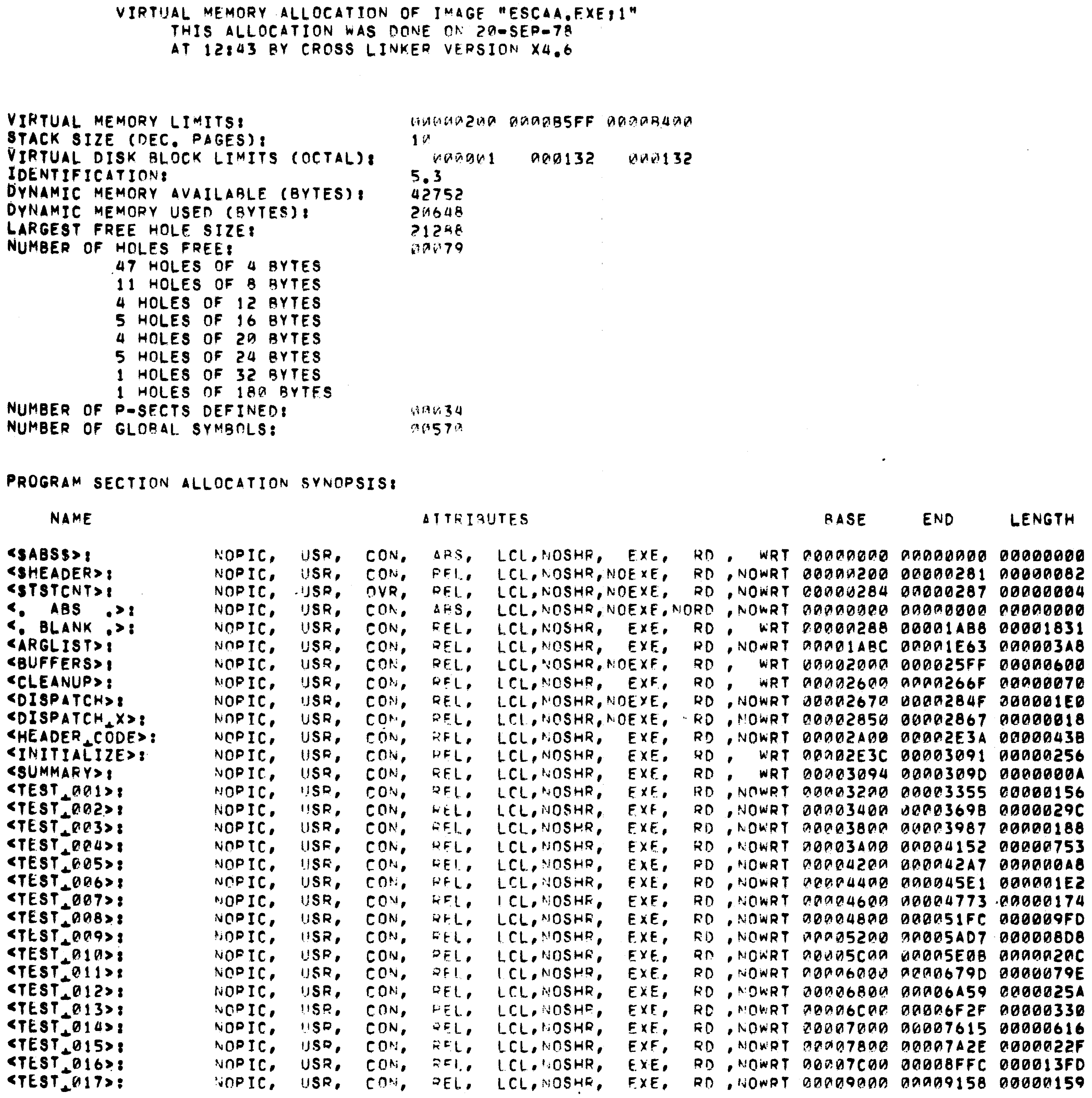
Each program listing begins with user information and a link map created by the linker program. The separate modules that make up the program constitute the rest of the listing. The first module, called the header, defines symbols and labels and provides routines that are called by other modules. The modules which follow contain the test routines.

User Information -- The user information, which comes at the beginning of the listing, includes the following items.

Program identification
Copyright statement
Program abstract
Hardware and software requirements
Prerequisites to running the program
Load and start instructions
Program description
History of program maintenance

Link Map -- The link map shows the virtual memory allocation of the total program image in the program section allocation synopsis. Figure 5-1 shows this synopsis for the MBA diagnostic listing.

The program section allocation synopsis lists the program sections according to the order in which they appear in memory. A list of attributes and the base, end, and length are given for each program section. The base address is the virtual address of the first location in each program section assigned at link time. This number must be added to the relative addresses given in the module listings to determine the virtual addresses of specific instructions because the assembly addresses are all relative to the base of the program section.



```
                  VIRTUAL MEMORY ALLOCATION OF IMAGE "ESCAA.EXE;1"
                       THIS ALLOCATION WAS DONE ON 20-SEP-78
                       AT 12:43 BY CROSS LINKER VERSION X4.6


VIRTUAL MEMORY LIMITS:                     00000200 0000B5FF 00008400
STACK SIZE (DEC. PAGES):                   10
VIRTUAL DISK BLOCK LIMITS (OCTAL):          000001   000132   000132
IDENTIFICATION:                            5.3
DYNAMIC MEMORY AVAILABLE (BYTES):          42752
DYNAMIC MEMORY USED (BYTES):               20648
LARGEST FREE HOLE SIZE:                    21288
NUMBER OF HOLES FREE:                      00079
          47 HOLES OF 4 BYTES
          11 HOLES OF 8 BYTES
          4 HOLES OF 12 BYTES
          5 HOLES OF 16 BYTES
          4 HOLES OF 20 BYTES
          5 HOLES OF 24 BYTES
          1 HOLES OF 32 BYTES
          1 HOLES OF 180 BYTES
NUMBER OF P-SECTS DEFINED:                 00034
NUMBER OF GLOBAL SYMBOLS:                  00570


PROGRAM SECTION ALLOCATION SYNOPSIS:

    NAME                                  ATTRIBUTES                                  BASE     END      LENGTH

<$ABS$>:            NOPIC,  USR,  CON,  ABS,  LCL,NOSHR,  EXE,  RD ,  WRT 00000000 00000000 00000000
<$HEADER>:          NOPIC,  USR,  CON,  REL,  LCL,NOSHR,NOEXE,  RD ,NOWRT 00000200 00000281 00000082
<$TSTCNT>:          NOPIC,  USR,  OVR,  REL,  LCL,NOSHR,NOEXE,  RD ,NOWRT 00000284 00000287 00000004
<. ABS  .>:         NOPIC,  USR,  CON,  ABS,  LCL,NOSHR,NOEXE,NORD ,NOWRT 00000000 00000000 00000000
<. BLANK .>:        NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,  WRT 00000288 00001AB8 00001831
<ARGLIST>:          NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00001ABC 00001E63 000003A8
<BUFFERS>:          NOPIC,  USR,  CON,  REL,  LCL,NOSHR,NOEXE,  RD ,  WRT 00002000 000025FF 00000600
<CLEANUP>:          NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,  WRT 00002600 0000266F 00000070
<DISPATCH>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,NOEXE,  RD ,NOWRT 00002670 0000284F 000001E0
<DISPATCH_X>:       NOPIC,  USR,  CON,  REL,  LCL,NOSHR,NOEXE,  RD ,NOWRT 00002850 00002867 00000018
<HEADER_CODE>:      NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00002A00 00002E3A 0000043B
<INITIALIZE>:       NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,  WRT 00002E3C 00003091 00000256
<SUMMARY>:          NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,  WRT 00003094 0000309D 0000000A
<TEST_001>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00003200 00003355 00000156
<TEST_002>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00003400 0000369B 0000029C
<TEST_003>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00003800 00003987 00000188
<TEST_004>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00003A00 00004152 00000753
<TEST_005>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00004200 000042A7 000000A8
<TEST_006>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00004400 000045E1 000001E2
<TEST_007>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00004600 00004773 00000174
<TEST_008>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00004800 000051FC 000009FD
<TEST_009>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00005200 00005AD7 000008D8
<TEST_010>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00005C00 00005E0B 0000020C
<TEST_011>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00006000 0000679D 0000079E
<TEST_012>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00006800 00006A59 0000025A
<TEST_013>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00006C00 00006F2F 00000330
<TEST_014>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00007000 00007615 00000616
<TEST_015>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00007800 00007A2E 0000022F
<TEST_016>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00007C00 00008FFC 000013FD
<TEST_017>:         NOPIC,  USR,  CON,  REL,  LCL,NOSHR,  EXE,  RD ,NOWRT 00009000 00009158 00000159
```

TK-1118

Figure 5-1 Portion of the Program Section Synopsis, RH78Ø (MBA) Diagnostic Program

The link map also lists the global symbols and their assigned values. Note that symbols used as labels point to routines in the diagnostic supervisor, if their values are over 10000.

Figure 5-2 shows a portion of the global symbol table for the absolute program section in the header file of the MBA diagnostic program.

The link map is a part of the listing created by the linker, but not a part of the actual program in memory. It always precedes the first file in the macrodiagnostic program listings.

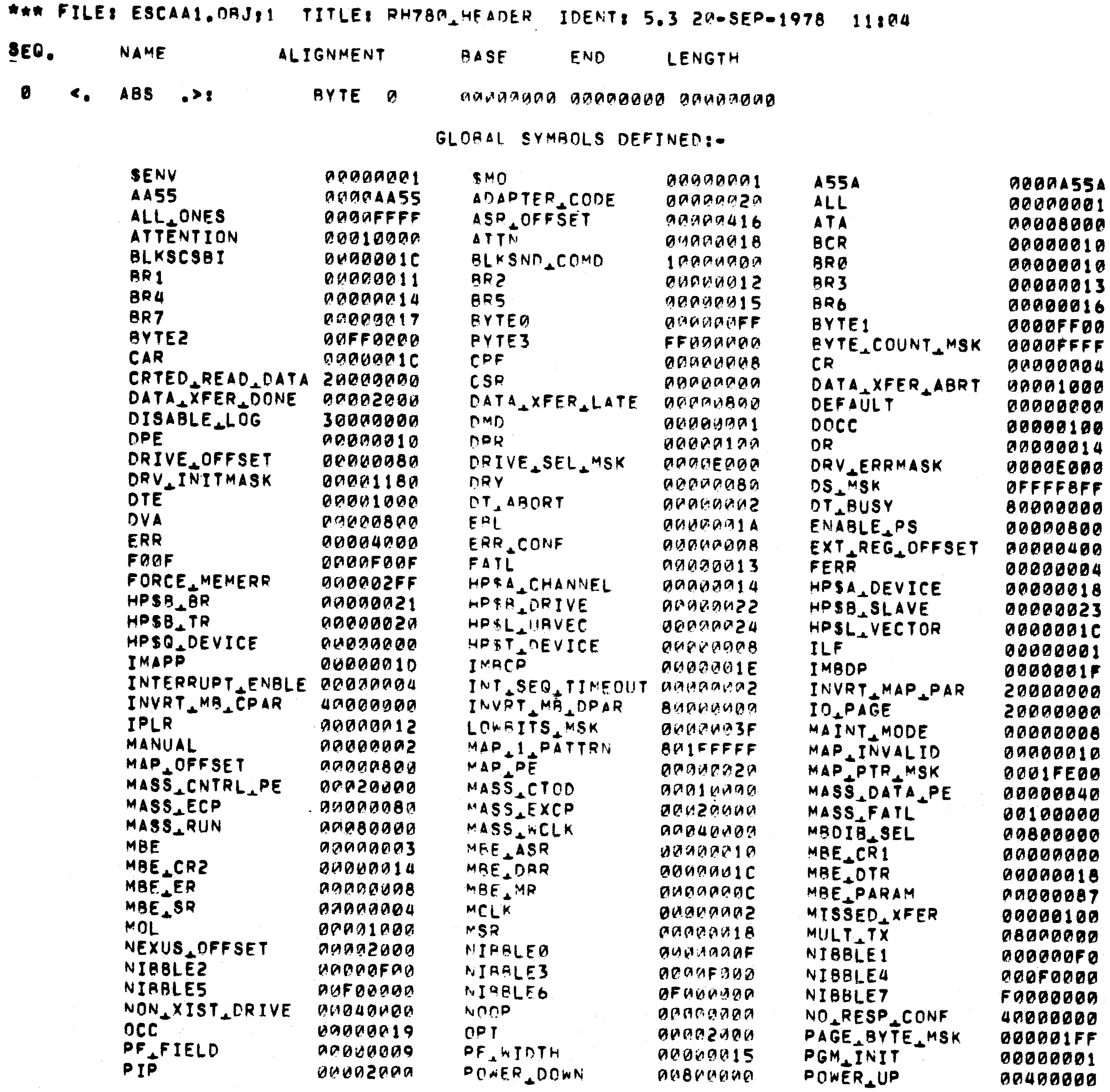
Header Module -- Like all of the modules in the program, the header module begins with a Table of Contents, a Copyright Statement, and a Revision History.

The declarations section in the header module contains global symbol definitions for register bit names, data patterns, masks referenced by the program, and Macro definitions. This section constitutes the beginning of the program code. The own storage section in the header module contains program labeled data, such as drive addresses, and program text and format statements, containing the ASCII texts of error and status messages.

The header module also contains code that generates the hardware and software parameter tables, report and print routines, initialization and clean up routines, and interrupt and exception service routines.

The assembler prints a symbol table and a program section synopsis for the entire module following the last program section in the module.

Test Modules -- The remaining modules in the program contain the tests, which are the main body of the program. Each module begins with a Table of Contents, Copyright Statement, and Program Maintenance History. The program code begins with macro definitions. A symbol table and program section synopsis are provided by the assembler following the last program section for each test module. Notice that each test begins a new program section.



```
*** FILE: ESCAA1.OBJ;1  TITLE: RH780_HEADER  IDENT: 5.3 20-SEP-1978  11:04

SEQ.      NAME            ALIGNMENT       BASE      END       LENGTH

 0   <.  ABS   .>:        BYTE  0       00000000 00000000 00000000

                                  GLOBAL SYMBOLS DEFINED:-

        $ENV                00000001   $MO                 00000001   A55A                0000A55A
        AA55                0000AA55   ADAPTER_CODE        00000020   ALL                 00000001
        ALL_ONES            0000FFFF   ASR_OFFSET          00000416   ATA                 00008000
        ATTENTION           00010000   ATTN                00000018   BCR                 00000010
        BLKSCSBI            0000001C   BLKSND_COMD         10000000   BR0                 00000010
        BR1                 00000011   BR2                 00000012   BR3                 00000013
        BR4                 00000014   BR5                 00000015   BR6                 00000016
        BR7                 00000017   BYTE0               000000FF   BYTE1               0000FF00
        BYTE2               00FF0000   BYTE3               FF000000   BYTE_COUNT_MSK      0000FFFF
        CAR                 0000001C   CPF                 00000008   CR                  00000004
        CRTED_READ_DATA     20000000   CSR                 00000000   DATA_XFER_ABRT      00001000
        DATA_XFER_DONE      00002000   DATA_XFER_LATE      00000800   DEFAULT             00000000
        DISABLE_LOG         30000000   DMD                 00000001   DOCC                00000100
        DPE                 00000010   DPR                 00000100   DR                  00000014
        DRIVE_OFFSET        00000080   DRIVE_SEL_MSK       0000E000   DRV_ERRMASK         0000E000
        DRV_INITMASK        00001180   DRY                 00000080   DS_MSK              0FFFF8FF
        DTE                 00001000   DT_ABORT            00000002   DT_BUSY             80000000
        DVA                 00000800   EBL                 0000001A   ENABLE_PS           00000800
        ERR                 00004000   ERR_CONF            00000008   EXT_REG_OFFSET      00000400
        F00F                0000F00F   FATL                00000013   FERR                00000004
        FORCE_MEMERR        000002FF   HP$A_CHANNEL        00000014   HP$A_DEVICE         00000018
        HP$B_BR             00000021   HP$B_DRIVE          00000022   HP$B_SLAVE          00000023
        HP$B_TR             00000020   HP$L_UBVEC          00000024   HP$L_VECTOR         0000001C
        HP$Q_DEVICE         00000000   HP$T_DEVICE         00000008   ILF                 00000001
        IMAPP               0000001D   IMBCP               0000001E   IMBDP               0000001F
        INTERRUPT_ENBLE     00000004   INT_SEQ_TIMEOUT     00000002   INVRT_MAP_PAR       20000000
        INVRT_MB_CPAR       40000000   INVRT_MB_DPAR       80000000   IO_PAGE             20000000
        IPLR                00000012   LOWBITS_MSK         0000003F   MAINT_MODE          00000008
        MANUAL              00000002   MAP_1_PATTRN        801FFFFF   MAP_INVALID         00000010
        MAP_OFFSET          00000800   MAP_PE              00000020   MAP_PTR_MSK         0001FE00
        MASS_CNTRL_PE       00020000   MASS_CTOD           00010000   MASS_DATA_PE        00000040
        MASS_ECP            00000080   MASS_EXCP           00020000   MASS_FATL           00100000
        MASS_RUN            00080000   MASS_WCLK           00040000   MBDIB_SEL           00800000
        MBE                 00000003   MBE_ASR             00000010   MBE_CR1             00000000
        MBE_CR2             00000014   MBE_DBR             0000001C   MBE_DTR             00000018
        MBE_ER              00000008   MBE_MR              0000000C   MBE_PARAM           00000087
        MBE_SR              00000004   MCLK                00000002   MISSED_XFER         00000100
        MOL                 00001000   MSR                 00000018   MULT_TX             08000000
        NEXUS_OFFSET        00002000   NIBBLE0             0000000F   NIBBLE1             000000F0
        NIBBLE2             00000F00   NIBBLE3             0000F000   NIBBLE4             000F0000
        NIBBLE5             00F00000   NIBBLE6             0F000000   NIBBLE7             F0000000
        NON_XIST_DRIVE      00040000   NOOP                00000000   NO_RESP_CONF        40000000
        OCC                 00000019   OPT                 00002000   PAGE_BYTE_MSK       000001FF
        PF_FIELD            00000009   PF_WIDTH            00000015   PGM_INIT            00000001
        PIP                 00002000   POWER_DOWN          00800000   POWER_UP            00400000
```

TK-1119

Figure 5-2 Portion of the Global Symbol Table for the Absolute PSECT of the Loader File of the RH780 (MBA) Diagnostic Program

5.4 DIAGNOSTIC PROGRAM AND SUPERVISOR INTERACTION

Whether a diagnostic program is executed in the user mode or in the standalone mode, its relation to the diagnostic supervisor is basically that shown in Figure 5-3. Once a diagnostic program has been loaded and the diagnostic supervisor has been loaded and started, program control moves to the boot routine of the supervisor. This routine clears vector space, flags, mail boxes, and sets up the processor registers to a known state. The boot routine checks to determine whether the operator has typed a Control C and sets up a map of memory and I/O addresses creating PØ and Pl page tables. It then initializes the system control block and the process control block, and then calls the begin routine.

The begin routine changes the processor mode to kernel and calls the CLI (the command flag should be set). The CLI types out the prompt symbol, DS>, indicating that the supervisor is ready for commands. When the operator types in a command (e.g. START), a parser routine in the supervisor is activated to decode the command and call the requisite action routines, clear the command flag, and then call the dispatch routine.

The dispatch routine forms the heart of the supervisor. It begins by clearing the error count and setting the pass zero flag and then calls the initialization routine in the diagnostic program to be executed.

The initialization routine initializes the unit under test and sets up conditions in the CPU and on the SBI which are necessary to the diagnostic program. The initialization routine then questions the operator concerning the unit to be tested, creates a hardware parameter table (P Table), tests for end of pass, and returns control to the dispatch routine in the supervisor.

The dispatch routine then calls the first test. At the end of each test, control returns to the dispatch routine. At the end of the last test in the program (or the last test selected by the operator), the dispatch routine in the supervisor calls the initialization routine in the diagnostic program. This routine determines whether or not the end of the current pass has been reached. If the end of the current pass has not been reached, the first test routine in the dispatch section of the supervisor is called, beginning another test sequence. If the end of the pass has been reached, the program calls the end of pass routine in the supervisor.

The end of pass routine in the supervisor determines whether or not the last pass to be run has been completed. If so, the cleanup and summary routines in the diagnostic program are called, the CLI command mode is set, and control passes to the begin routine which calls the CLI. The CLI prints out the DS> prompt symbol and waits for operator input.

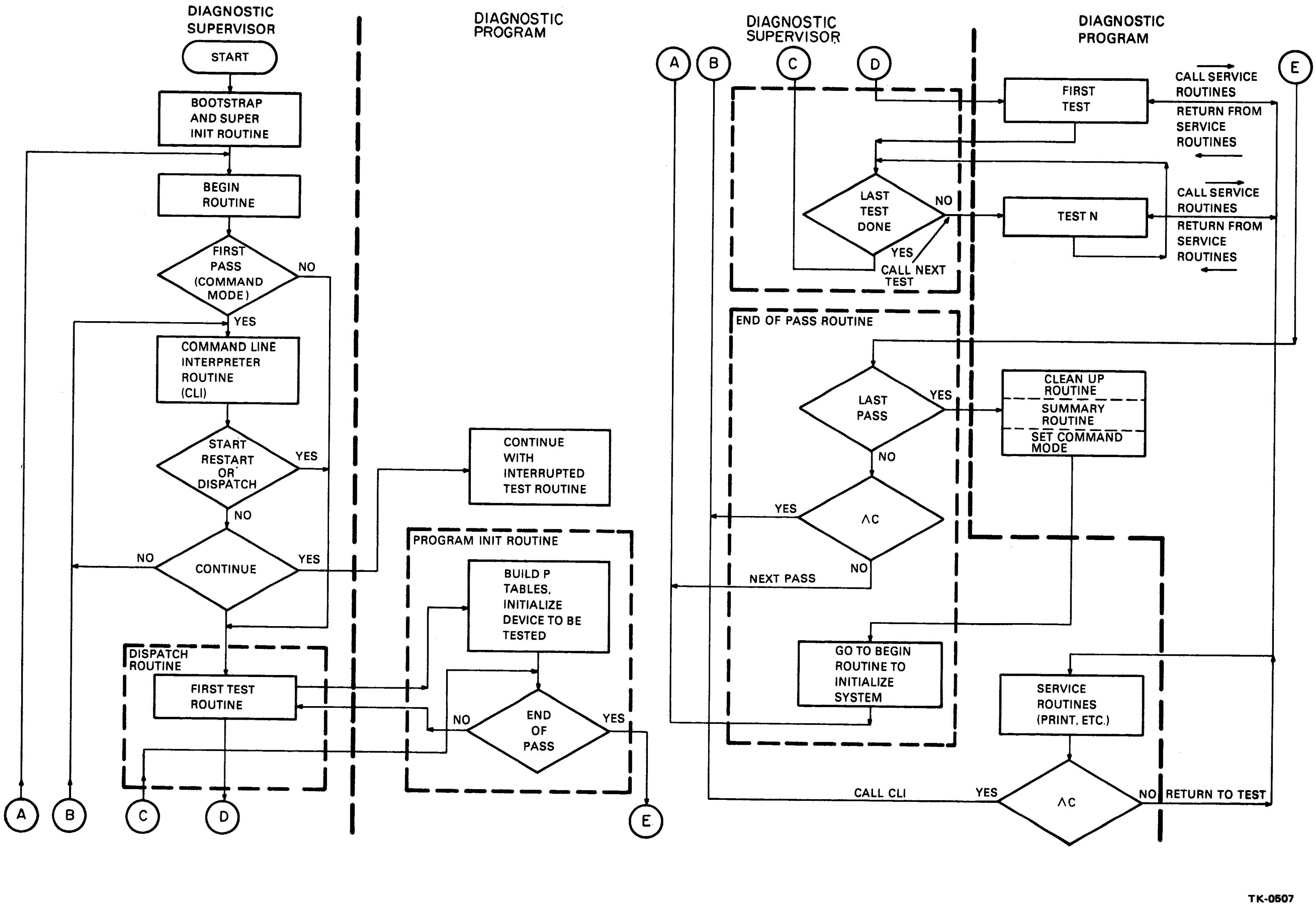


Figure 5-3 Diagnostic Program and Diagnostic Supervisor Interaction

If the last pass has not been completed, the end of pass routine checks to see whether the operator has typed control C. If the Control C flag is set, control returns to the CLI. Otherwise, the end of pass routine calls the begin routine in the supervisor to initialize the system and initiate the next pass of the diagnostic program.

Note that when the operator types Control C, he does not cause an interrupt routine to be called. The Control C merely sets a flag. The status of the flag is checked periodically when the tests in the diagnostic program call various service routines and at the end of a pass.

5.5 ANALYSIS OF A SAMPLE TEST: RH78Ø (MBA) TEST 3, SUBTEST 1

5.5.1 Listing Column Format Description

Figure 5-4 shows the program listing for the MBA RH78Ø diagnostic program (ESCAA), test 3, subtest 1. The sixth column from the left contains the relative address of each instruction. These numbers begin at Ø with the beginning of each program section. Note that the address offset of the program section containing Test 3 (36ØØ$_{16}$), found in the link map, must be added to the relative address to find the virtual memory address of the instruction.

The seventh column from the left contains the listing line numbers. These numbers begin at Ø for each module of the program. Note that the line number increments for each line of the source module. The sixth column shows the program counter, containing the relative address. The relative address increases according to the amount of memory space required for the instructions and operands. Line numbers are present only for lines entered by the program developer. Macro expansions do not have line numbers.

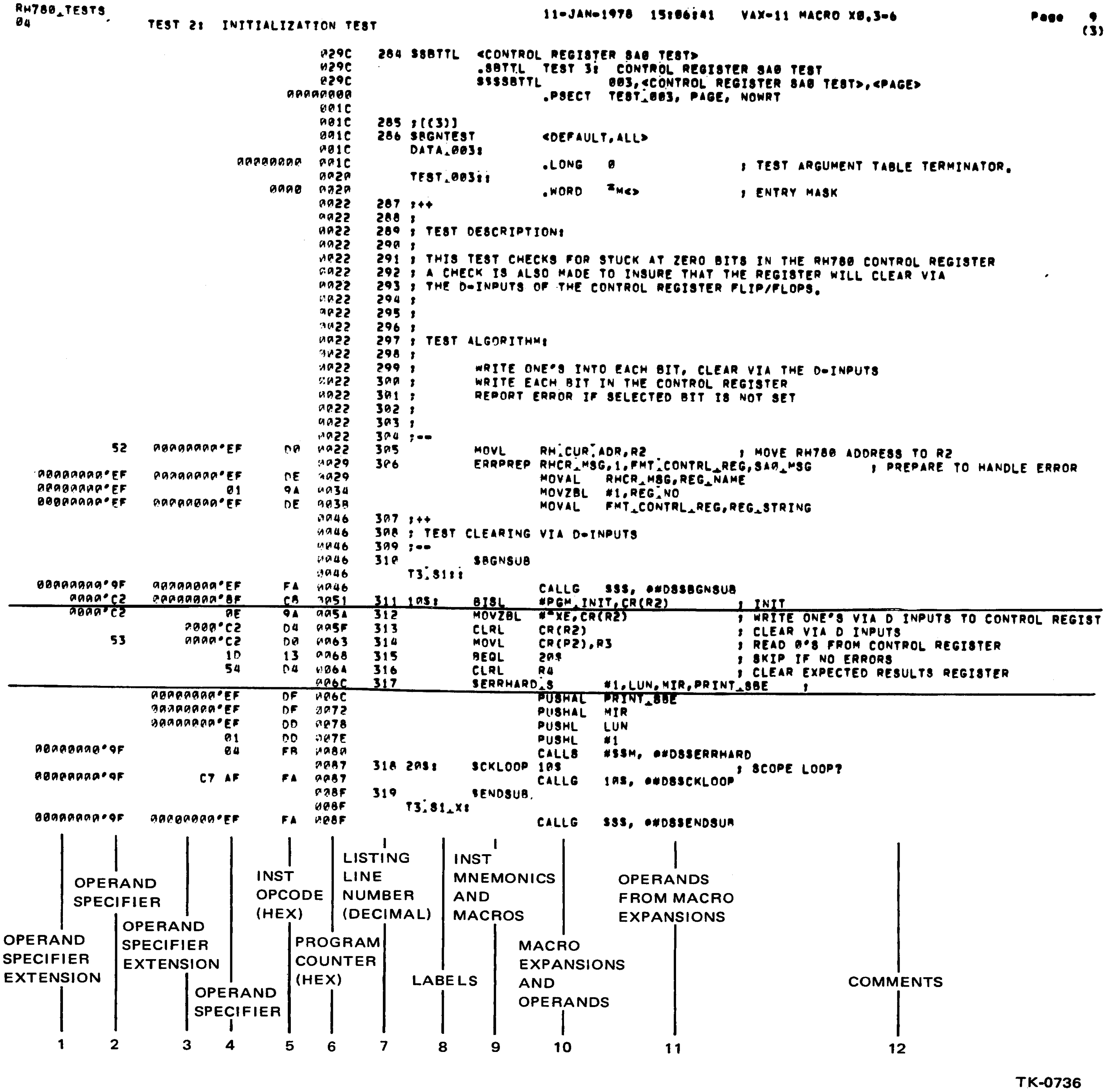
The eighth column from the left contains labels used by the programmer as symbolic addresses.

The ninth column from the left contains instruction mnemonics and Macro calls. Note that the Macro calls themselves require no memory space (the relative address does not change), and that in the Macro expansion which follows, the line number is not incremented (line 317). At assembly time, the assembler program responds to the Macro calls, expanding the Macro according to the definition listed at the beginning of the file.

Column ten contains operands for those instructions contained in column nine; and it contains instruction mnemonics and parameters for the Macro expansions.

The eleventh column from the left contains operands from the Macro expansions.

Column five contains the op codes (hex) for the instructions contained in columns seven and eight.



```
RH780_TESTS                                                 11-JAN-1978  15:06:41   VAX-11 MACRO X0.3-6            Page   9
04          TEST 2:  INITIALIZATION TEST                                                                                 (3)
                                         029C   284 $SBTTL  <CONTROL REGISTER SA0 TEST>
                                         029C                 .SBTTL  TEST 3:  CONTROL REGISTER SA0 TEST
                                         029C                 $$$SBTTL          003,<CONTROL REGISTER SA0 TEST>,<PAGE>
                                     00000000                         .PSECT  TEST_003, PAGE, NOWRT
                                         001C
                                         001C   285 ;[(3)]
                                         001C   286 $BGNTEST          <DEFAULT,ALL>
                                         001C       DATA_003:
                              00000000   001C                         .LONG   0                     ; TEST ARGUMENT TABLE TERMINATOR.
                                         0020       TEST_003::
                                  0000   0020                         .WORD   ^M<>                  ; ENTRY MASK
                                         0022   287 ;++
                                         0022   288 ;
                                         0022   289 ; TEST DESCRIPTION:
                                         0022   290 ;
                                         0022   291 ; THIS TEST CHECKS FOR STUCK AT ZERO BITS IN THE RH780 CONTROL REGISTER
                                         0022   292 ; A CHECK IS ALSO MADE TO INSURE THAT THE REGISTER WILL CLEAR VIA
                                         0022   293 ; THE D-INPUTS OF THE CONTROL REGISTER FLIP/FLOPS.
                                         0022   294 ;
                                         0022   295 ;
                                         0022   296 ;
                                         0022   297 ; TEST ALGORITHM:
                                         0022   298 ;
                                         0022   299 ;           WRITE ONE'S INTO EACH BIT, CLEAR VIA THE D-INPUTS
                                         0022   300 ;           WRITE EACH BIT IN THE CONTROL REGISTER
                                         0022   301 ;           REPORT ERROR IF SELECTED BIT IS NOT SET
                                         0022   302 ;
                                         0022   303 ;
                                         0022   304 ;--
            52   00000000'EF       D0    0022   305           MOVL     RH_CUR_ADR,R2              ; MOVE RH780 ADDRESS TO R2
                                         0029   306           ERRPREP RHCR_MSG,1,FMT_CONTRL_REG,SA0_MSG      ; PREPARE TO HANDLE ERROR
   00000000'EF   00000000'EF       DE    0029                         MOVAL    RHCR_MSG,REG_NAME
   00000000'EF            01       9A    0034                         MOVZBL   #1,REG_NO
   00000000'EF   00000000'EF       DE    0038                         MOVAL    FMT_CONTRL_REG,REG_STRING
                                         0046   307 ;++
                                         0046   308 ; TEST CLEARING VIA D-INPUTS
                                         0046   309 ;--
                                         0046   310           $BGNSUB
                                         0046       T3_S1::
   00000000'9F   00000000'EF       FA    0046                         CALLG   $$$, @#DS$BGNSUB
        0000'C2  00000000'8F       C8    0051   311 10$:      BISL     #PGM_INIT,CR(R2)           ; INIT
        0000'C2           0E       9A    005A   312           MOVZBL   #^XE,CR(R2)                ; WRITE ONE'S VIA D INPUTS TO CONTROL REGIST
                     0000'C2       D4    005F   313           CLRL     CR(R2)                     ; CLEAR VIA D INPUTS
            53       0000'C2       D0    0063   314           MOVL     CR(P2),R3                  ; READ 0'S FROM CONTROL REGISTER
                          1D       13    0068   315           BEQL     20$                        ; SKIP IF NO ERRORS
                          54       D4    006A   316           CLRL     R4                         ; CLEAR EXPECTED RESULTS REGISTER
                                         006C   317           $ERRHARD_S       #1,LUN,MIR,PRINT_SBE    ;
                 00000000'EF       DF    006C                         PUSHAL   PRINT_SBE
                 00000000'EF       DF    0072                         PUSHAL   MIR
                 00000000'EF       DD    0078                         PUSHL    LUN
                          01       DD    007E                         PUSHL    #1
   00000000'9F            04       FB    0080                         CALLS    #$$M, @#DS$ERRHARD
                                         0087   318 20$:      $CKLOOP 10$                         ; SCOPE LOOP?
   00000000'9F         C7 AF       FA    0087                         CALLG    10$, @#DS$CKLOOP
                                         008F   319           $ENDSUB.
                                         008F       T3_S1_X:
   00000000'9F   00000000'EF       FA    008F                         CALLG   $$$, @#DS$ENDSUB
```

OPERAND SPECIFIER EXTENSION
1
OPERAND SPECIFIER
2
OPERAND SPECIFIER EXTENSION
3
OPERAND SPECIFIER
4
INST OPCODE (HEX)
5
PROGRAM COUNTER (HEX)
6
LISTING LINE NUMBER (DECIMAL)
7
LABELS
8
INST MNEMONICS AND MACROS
9
MACRO EXPANSIONS AND OPERANDS
10
OPERANDS FROM MACRO EXPANSIONS
11
COMMENTS
12
TK-0736

Figure 5-4 RH780 (MBA) Diagnostic Program Test 3, Subtest 1, Listing

Columns one through four contain the hexadecimal code for the operands specified in columns ten and eleven. Columns two and four contain operand specifiers. Columns one and three contain operand specifier extensions. Numbers followed by an apostrophe (e.g., ØØØØØØØØ') are the machine code for symbolic operands. They are modified by the linker at link time. (MARS is a one pass assembler. Forward references, with the exception of branches within P sections, and global symbols cannot be resolved until link time.)

The twelfth column contains comments describing the functions of the instructions. Each comment is preceded by a semicolon.

5.5.2 Analysis Of Typical Lines

Line 311 -- The BISL instruction sets bits in the destination according to the mask provided. #PGM.INIT is the symbol for the mask. Its value (ØØØØØØØ1) can be found in the symbol table at the end of the module. CR is the symbol for the relative address (offset from the MBA base register) of the control register of the MBA under test. Its value (ØØØØØØØ4) is also listed in the symbol table. This value is added to the contents of R2, the base address of the MBA under test, to produce the physical address of the control register. The instruction thus sets bit zero of the control register. The comment, INIT, indicates the function that setting bit zero performs.

Line 317 -- $ERRHARDS, in line 317, is a Macro call. The symbols that follow it are arguments to be used in the call. The five lines that follow line 317 show the expansion of the Macro. These instructions push the arguments on the stack and call the DS$ERRHARD subroutine in the supervisor, which sets the error flag and prints an error message based on the stored arguments.

5.6 RH78Ø (MBA) DIAGNOSTIC SAMPLE SUBTEST (Direct I/O)

5.6.1 RH78Ø Diagnostic Detailed Flow

Each test in a given diagnostic program relies on subroutines provided by the diagnostic supervisor. The diagnostic program thus depends on the supervisor for services as well as initialization and test sequencing functions. The operator should be able to follow references and subroutine calls back and forth between the diagnostic program being run and the supervisor in order to use the listings.

The general strategy used throughout the diagnostic programs involves writing and then reading back data directly (with MOVE instructions) in order to exercise the logic circuits or device functions under test. Data retrieved is compared with data expected. If the comparison indicates a failure, an error routine takes appropriate action and sends a message to the operator. Subroutine 1 of test 3 of the MBA diagnostic is representative of this strategy. This subtest determines whether the control register of the MBA under test can be cleared after each bit in the register has been set.

When the diagnostic supervisor calls test 3 of the MBA diagnostic, the test initialization code moves the base address of the MBA under test to general register 2. This register is then used to index specific MBA registers. The ERRPREP Macro then stores information concerning test 3 in a buffer area for use in error messages.

Subtest 1 begins with a call to the DS$BGNSUB entry point in the supervisor, as shown in Figure 5-5. In order to find this entry point, look in the global symbol table in the program link map (Figure 5-6). DS$BGNSUB equals ØØØ1ØØ3Ø, an address in the supervisor. Note that the first two characters of the symbol (DS) indicate that the symbol points into the diagnostic supervisor. The global symbol table in the link map for the supervisor shows that the entry module (ESSAA11) defines the symbol (contains the code for the entry point) (Figure 5-7). The names of the supervisor modules suggest their functions (e.g., entry, loop, print).

The DS$BGNSUB entry point contains only one instruction, a jump to RBGNSUB, as shown in Figure 5-8. This subroutine is in the loop module of the supervisor (Figure 5-9). RBGNSUB checks the subtest sequence for correct order. A discrepancy causes the subroutine to call a print routine, which displays an error message, and then to return to the CLI. If the subtest sequence is correct, the RBGNSUB subroutine calls the KB-CHECK routine to check for Control C. If the operator has typed Control C, control returns to the CLI. Otherwise, control returns to subtest 1 which, at this point, begins testing the MBA logic.

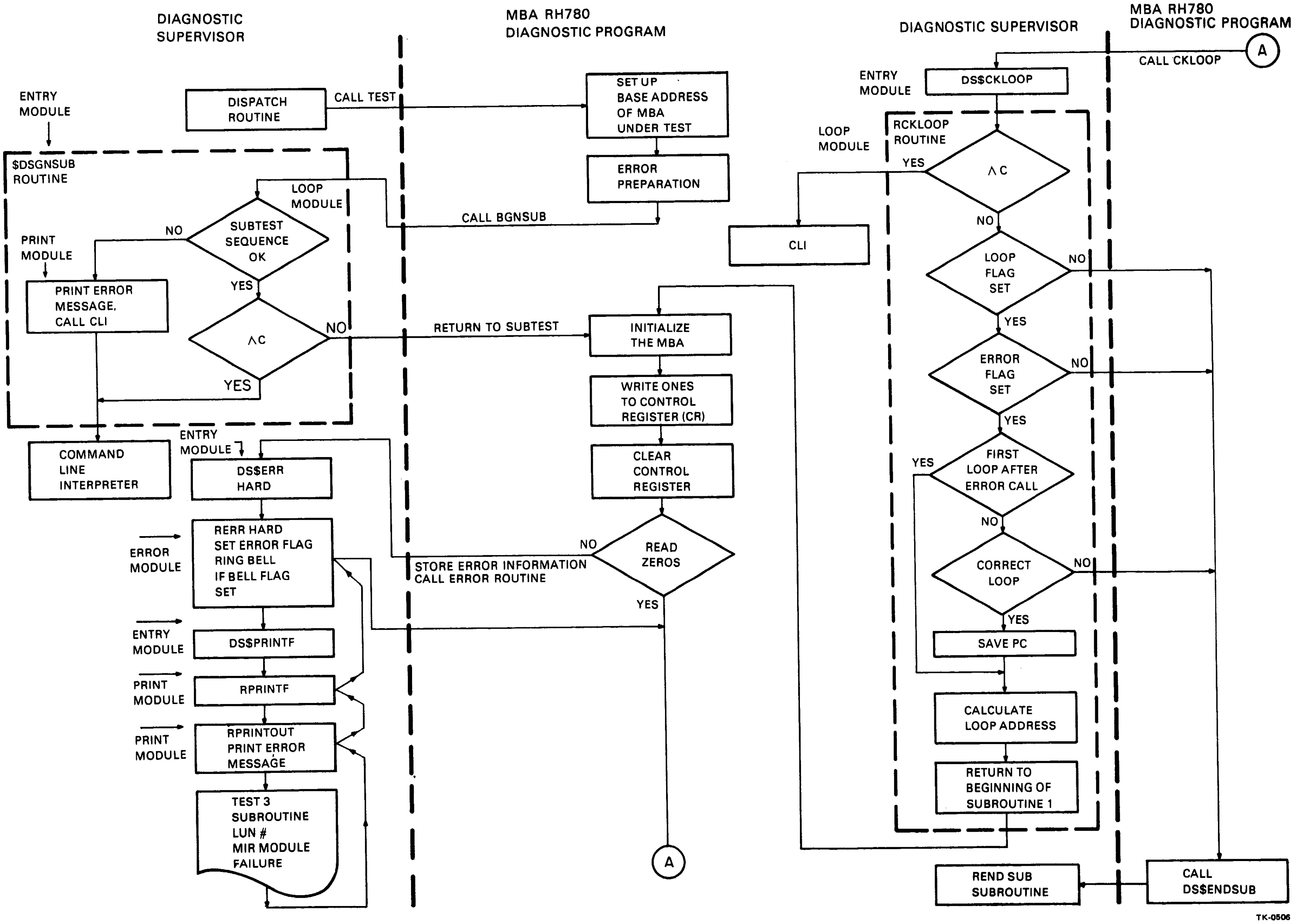
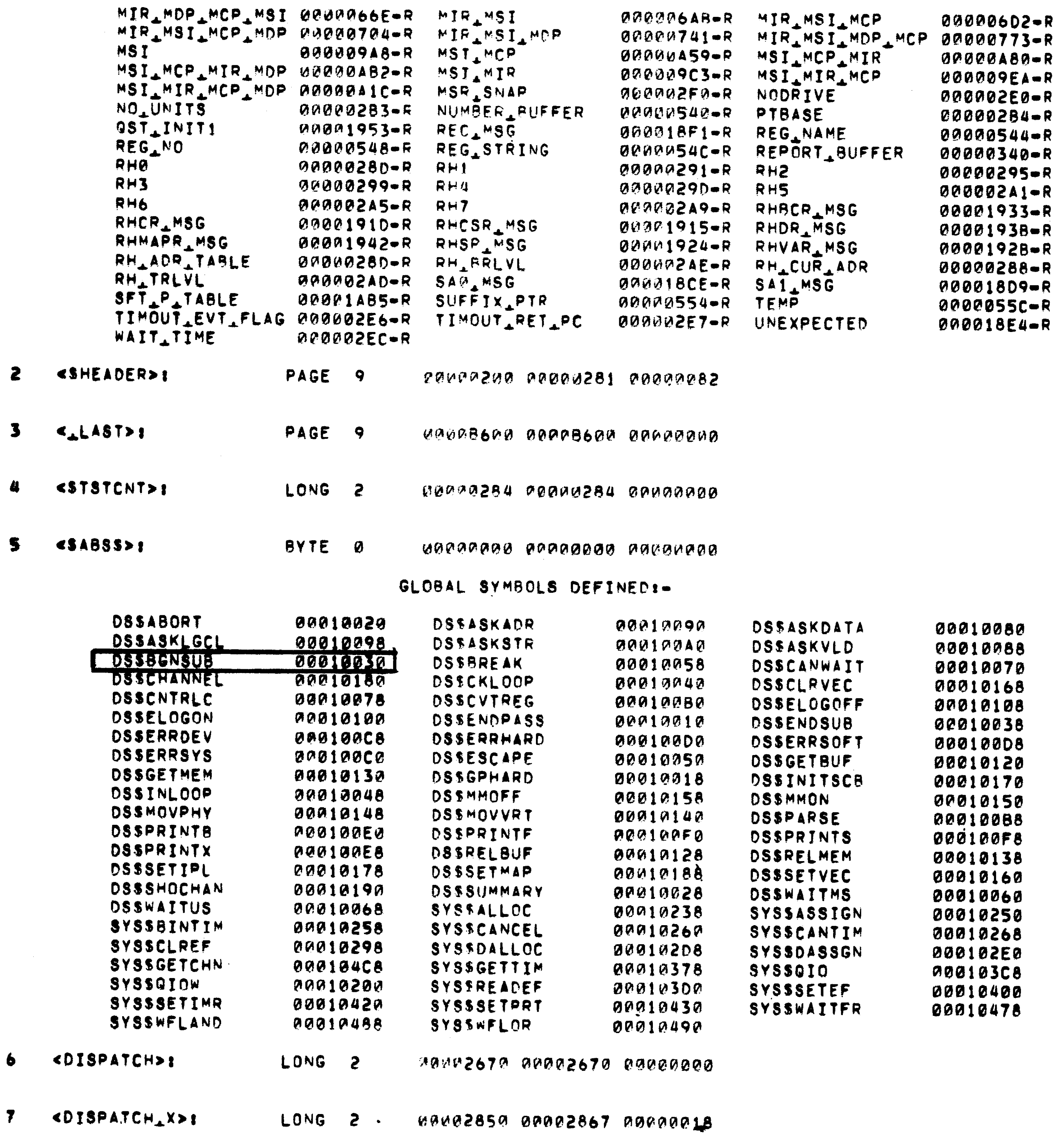


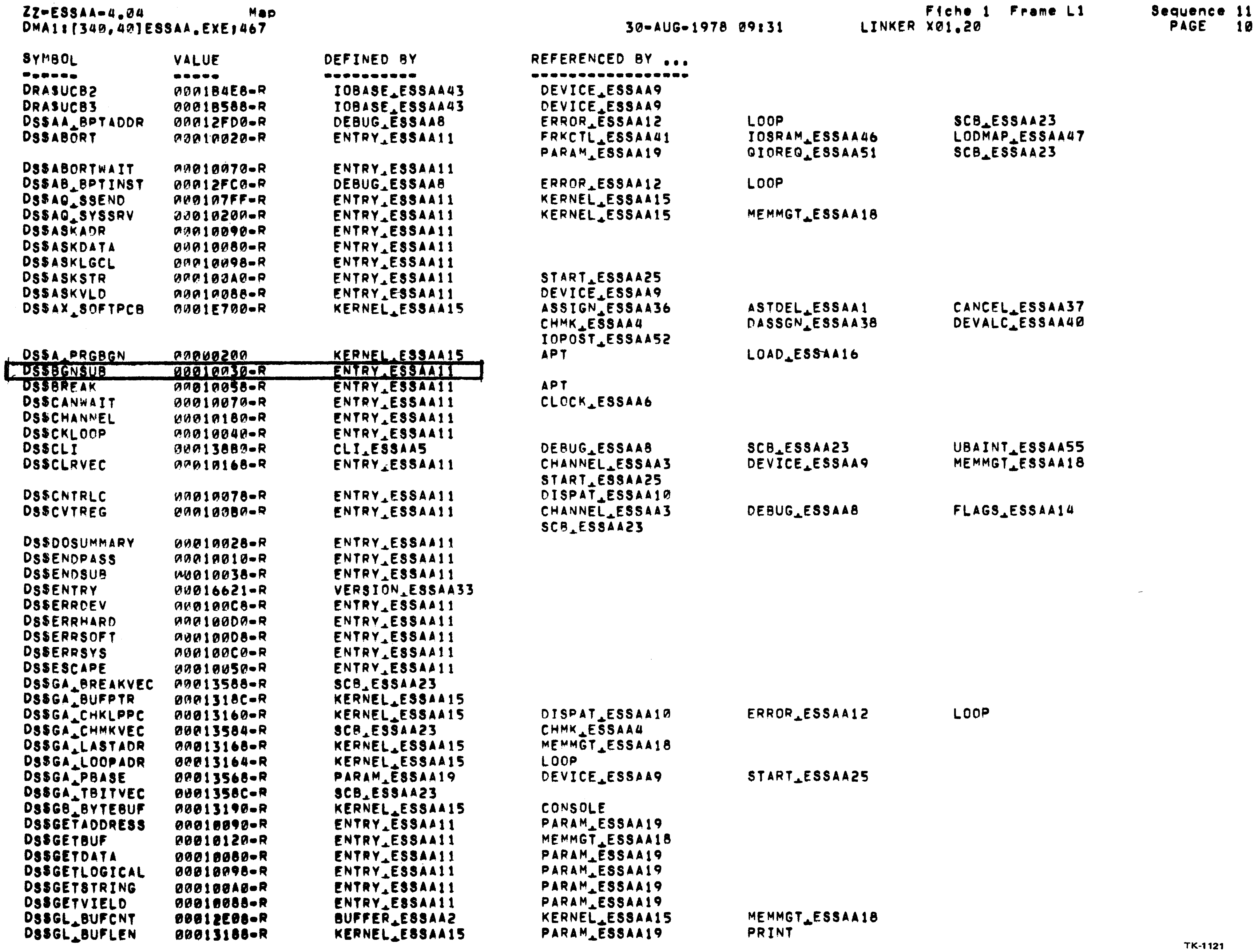
Figure 5-5 RH780 (MBA) Diagnostic Program Test 3, Subtest 1, Flowchart



```
                MIR_MDP_MCP_MSI 0000066E-R  MIR_MSI            000006AB-R  MIR_MSI_MCP         000006D2-R
                MIR_MSI_MCP_MDP 00000704-R  MIR_MSI_MDP        00000741-R  MIR_MSI_MDP_MCP     00000773-R
                MSI             000009A8-R  MSI_MCP            00000A59-R  MSI_MCP_MIR         00000A80-R
                MSI_MCP_MIR_MDP 00000AB2-R  MSI_MIR            000009C3-R  MSI_MIR_MCP         000009EA-R
                MSI_MIR_MCP_MDP 00000A1C-R  MSR_SNAP           000002F0-R  NODRIVE             000002E0-R
                NO_UNITS        000002B3-R  NUMBER_BUFFER      00000540-R  PTBASE              000002B4-R
                QST_INIT1       00001953-R  REC_MSG            000018F1-R  REG_NAME            00000544-R
                REG_NO          00000548-R  REG_STRING         0000054C-R  REPORT_BUFFER       00000340-R
                RH0             0000028D-R  RH1                00000291-R  RH2                 00000295-R
                RH3             00000299-R  RH4                0000029D-R  RH5                 000002A1-R
                RH6             000002A5-R  RH7                000002A9-R  RHBCR_MSG           00001933-R
                RHCR_MSG        0000191D-R  RHCSR_MSG          00001915-R  RHDR_MSG            0000193B-R
                RHMAPR_MSG      00001942-R  RHSP_MSG           00001924-R  RHVAR_MSG           0000192B-R
                RH_ADR_TABLE    0000028D-R  RH_BRLVL           000002AE-R  RH_CUR_ADR          00000288-R
                RH_TRLVL        000002AD-R  SA0_MSG            000018CE-R  SA1_MSG             000018D9-R
                SFT_P_TABLE     00001AB5-R  SUFFIX_PTR         00000554-R  TEMP                0000055C-R
                TIMOUT_EVT_FLAG 000002E6-R  TIMOUT_RET_PC      000002E7-R  UNEXPECTED          000018E4-R
                WAIT_TIME       000002EC-R

 2   <$HEADER>:             PAGE  9     00000200 00000281 00000082

 3   <_LAST>:               PAGE  9     00008600 00008600 00000000

 4   <STSTCNT>:             LONG  2     00000284 00000284 00000000

 5   <$ABSS>:               BYTE  0     00000000 00000000 00000000

                                     GLOBAL SYMBOLS DEFINED:-

               DS$ABORT         00010020    DS$ASKADR       00010090    DS$ASKDATA      00010080
               DS$ASKLGCL       00010098    DS$ASKSTR       000100A0    DS$ASKVLD       00010088
               DS$BGNSUB        00010030    DS$BREAK        00010058    DS$CANWAIT      00010070
               DS$CHANNEL       00010180    DS$CKLOOP       00010040    DS$CLRVEC       00010168
               DS$CNTRLC        00010078    DS$CVTREG       000100B0    DS$ELOGOFF      00010108
               DS$ELOGON        00010100    DS$ENDPASS      00010010    DS$ENDSUB       00010038
               DS$ERRDEV        000100C8    DS$ERRHARD      000100D0    DS$ERRSOFT      000100D8
               DS$ERRSYS        000100C0    DS$ESCAPE       00010050    DS$GETBUF       00010120
               DS$GETMEM        00010130    DS$GPHARD       00010018    DS$INITSCB      00010170
               DS$INLOOP        00010048    DS$MMOFF        00010158    DS$MMON         00010150
               DS$MOVPHY        00010148    DS$MOVVRT       00010140    DS$PARSE        000100B8
               DS$PRINTB        000100E0    DS$PRINTF       000100F0    DS$PRINTS       000100F8
               DS$PRINTX        000100E8    DS$RELBUF       00010128    DS$RELMEM       00010138
               DS$SETIPL        00010178    DS$SETMAP       00010188    DS$SETVEC       00010160
               DS$SHOCHAN       00010190    DS$SUMMARY      00010028    DS$WAITMS       00010060
               DS$WAITUS        00010068    SYS$ALLOC       00010238    SYS$ASSIGN      00010250
               SYS$BINTIM       00010258    SYS$CANCEL      00010260    SYS$CANTIM      00010268
               SYS$CLREF        00010298    SYS$DALLOC      000102D8    SYS$DASSGN      000102E0
               SYS$GETCHN       000104C8    SYS$GETTIM      00010378    SYS$QIO         000103C8
               SYS$QIOW         00010200    SYS$READEF      000103D0    SYS$SETEF       00010400
               SYS$SETIMR       00010420    SYS$SETPRT      00010430    SYS$WAITFR      00010478
               SYS$WFLAND       00010488    SYS$WFLOR       00010490

 6   <DISPATCH>:             LONG  2    00002670 00002670 00000000

 7   <DISPATCH_X>:           LONG  2 .  00002850 00002867 00000018
```

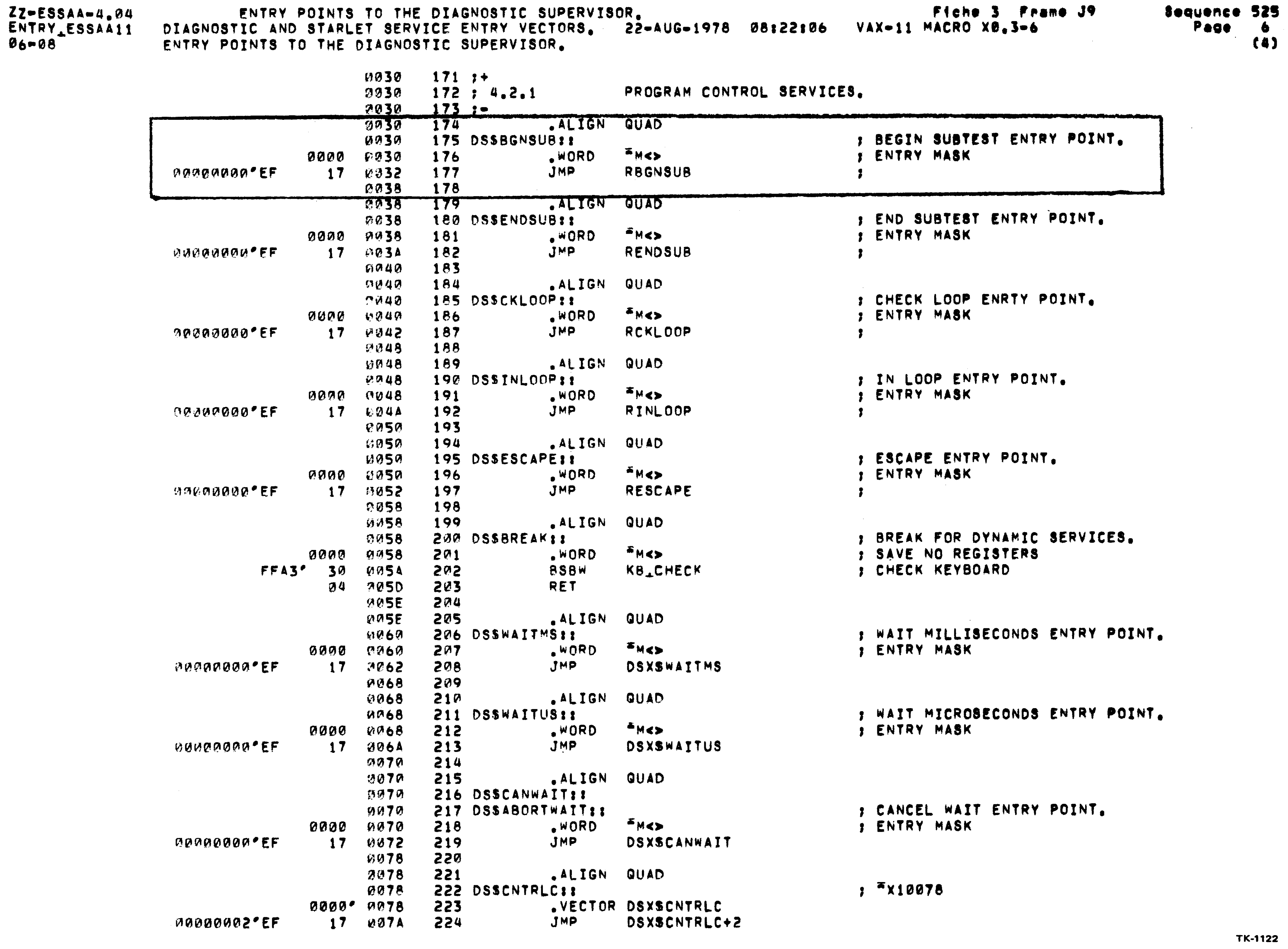
TK-1120

Figure 5-6 DS$BGNSUB Listed in the Symbol Table in the ESCAA Link Map



```
ZZ-ESSAA-4.04         Map                                                         Fiche 1  Frame L1       Sequence 11
DMA1:[340,40]ESSAA.EXE;467                        30-AUG-1978 09:31       LINKER X01.20                      PAGE   10

SYMBOL            VALUE             DEFINED BY          REFERENCED BY ...
------            -----             ----------          -----------------
DRA$UCB2          0001B4E8-R         IOBASE_ESSAA43      DEVICE_ESSAA9
DRA$UCB3          0001B588-R         IOBASE_ESSAA43      DEVICE_ESSAA9
DS$AA_BPTADDR     00012FD0-R         DEBUG_ESSAA8        ERROR_ESSAA12          LOOP                    SCB_ESSAA23
DS$ABORT          00010020-R         ENTRY_ESSAA11       FRKCTL_ESSAA41         IOSRAM_ESSAA46          LODMAP_ESSAA47
                                                         PARAM_ESSAA19          QIOREQ_ESSAA51          SCB_ESSAA23
DS$ABORTWAIT      00010070-R         ENTRY_ESSAA11
DS$AB_BPTINST     00012FC0-R         DEBUG_ESSAA8        ERROR_ESSAA12          LOOP
DS$AQ_SSEND       000107FF-R         ENTRY_ESSAA11       KERNEL_ESSAA15
DS$AQ_SYSSRV      00010200-R         ENTRY_ESSAA11       KERNEL_ESSAA15         MEMMGT_ESSAA18
DS$ASKADR         00010090-R         ENTRY_ESSAA11
DS$ASKDATA        00010080-R         ENTRY_ESSAA11
DS$ASKLGCL        00010098-R         ENTRY_ESSAA11
DS$ASKSTR         000100A0-R         ENTRY_ESSAA11       START_ESSAA25
DS$ASKVLD         00010088-R         ENTRY_ESSAA11       DEVICE_ESSAA9
DS$AX_SOFTPCB     0001E700-R         KERNEL_ESSAA15      ASSIGN_ESSAA36         ASTDEL_ESSAA1           CANCEL_ESSAA37
                                                         CHMK_ESSAA4            DASSGN_ESSAA38          DEVALC_ESSAA40
                                                         IOPOST_ESSAA52
DS$A_PRGBGN       00000200           KERNEL_ESSAA15      APT                    LOAD_ESSAA16
DS$BGNSUB         00010030-R         ENTRY_ESSAA11
DS$BREAK          00010058-R         ENTRY_ESSAA11       APT
DS$CANWAIT        00010070-R         ENTRY_ESSAA11       CLOCK_ESSAA6
DS$CHANNEL        00010180-R         ENTRY_ESSAA11
DS$CKLOOP         00010040-R         ENTRY_ESSAA11
DS$CLI            00013880-R         CLI_ESSAA5          DEBUG_ESSAA8           SCB_ESSAA23             UBAINT_ESSAA55
DS$CLRVEC         00010168-R         ENTRY_ESSAA11       CHANNEL_ESSAA3         DEVICE_ESSAA9           MEMMGT_ESSAA18
                                                         START_ESSAA25
DS$CNTRLC         00010078-R         ENTRY_ESSAA11       DISPAT_ESSAA10
DS$CVTREG         000100B0-R         ENTRY_ESSAA11       CHANNEL_ESSAA3         DEBUG_ESSAA8            FLAGS_ESSAA14
                                                         SCB_ESSAA23
DS$DOSUMMARY      00010028-R         ENTRY_ESSAA11
DS$ENDPASS        00010010-R         ENTRY_ESSAA11
DS$ENDSUB         00010038-R         ENTRY_ESSAA11
DS$ENTRY          00016621-R         VERSION_ESSAA33
DS$ERRDEV         000100C8-R         ENTRY_ESSAA11
DS$ERRHARD        000100D0-R         ENTRY_ESSAA11
DS$ERRSOFT        000100D8-R         ENTRY_ESSAA11
DS$ERRSYS         000100C0-R         ENTRY_ESSAA11
DS$ESCAPE         00010050-R         ENTRY_ESSAA11
DS$GA_BREAKVEC    00013588-R         SCB_ESSAA23
DS$GA_BUFPTR      0001318C-R         KERNEL_ESSAA15
DS$GA_CHKLPPC     00013160-R         KERNEL_ESSAA15      DISPAT_ESSAA10         ERROR_ESSAA12           LOOP
DS$GA_CHMKVEC     00013584-R         SCB_ESSAA23         CHMK_ESSAA4
DS$GA_LASTADR     00013168-R         KERNEL_ESSAA15      MEMMGT_ESSAA18
DS$GA_LOOPADR     00013164-R         KERNEL_ESSAA15      LOOP
DS$GA_PBASE       00013568-R         PARAM_ESSAA19       DEVICE_ESSAA9          START_ESSAA25
DS$GA_TBITVEC     0001358C-R         SCB_ESSAA23
DS$GB_BYTEBUF     00013190-R         KERNEL_ESSAA15      CONSOLE
DS$GETADDRESS     00010090-R         ENTRY_ESSAA11       PARAM_ESSAA19
DS$GETBUF         00010120-R         ENTRY_ESSAA11       MEMMGT_ESSAA18
DS$GETDATA        00010080-R         ENTRY_ESSAA11       PARAM_ESSAA19
DS$GETLOGICAL     00010098-R         ENTRY_ESSAA11       PARAM_ESSAA19
DS$GETSTRING      000100A0-R         ENTRY_ESSAA11       PARAM_ESSAA19
DS$GETVIELD       00010088-R         ENTRY_ESSAA11       PARAM_ESSAA19
DS$GL_BUFCNT      00012E08-R         BUFFER_ESSAA2       KERNEL_ESSAA15         MEMMGT_ESSAA18
DS$GL_BUFLEN      00013188-R         KERNEL_ESSAA15      PARAM_ESSAA19          PRINT
```

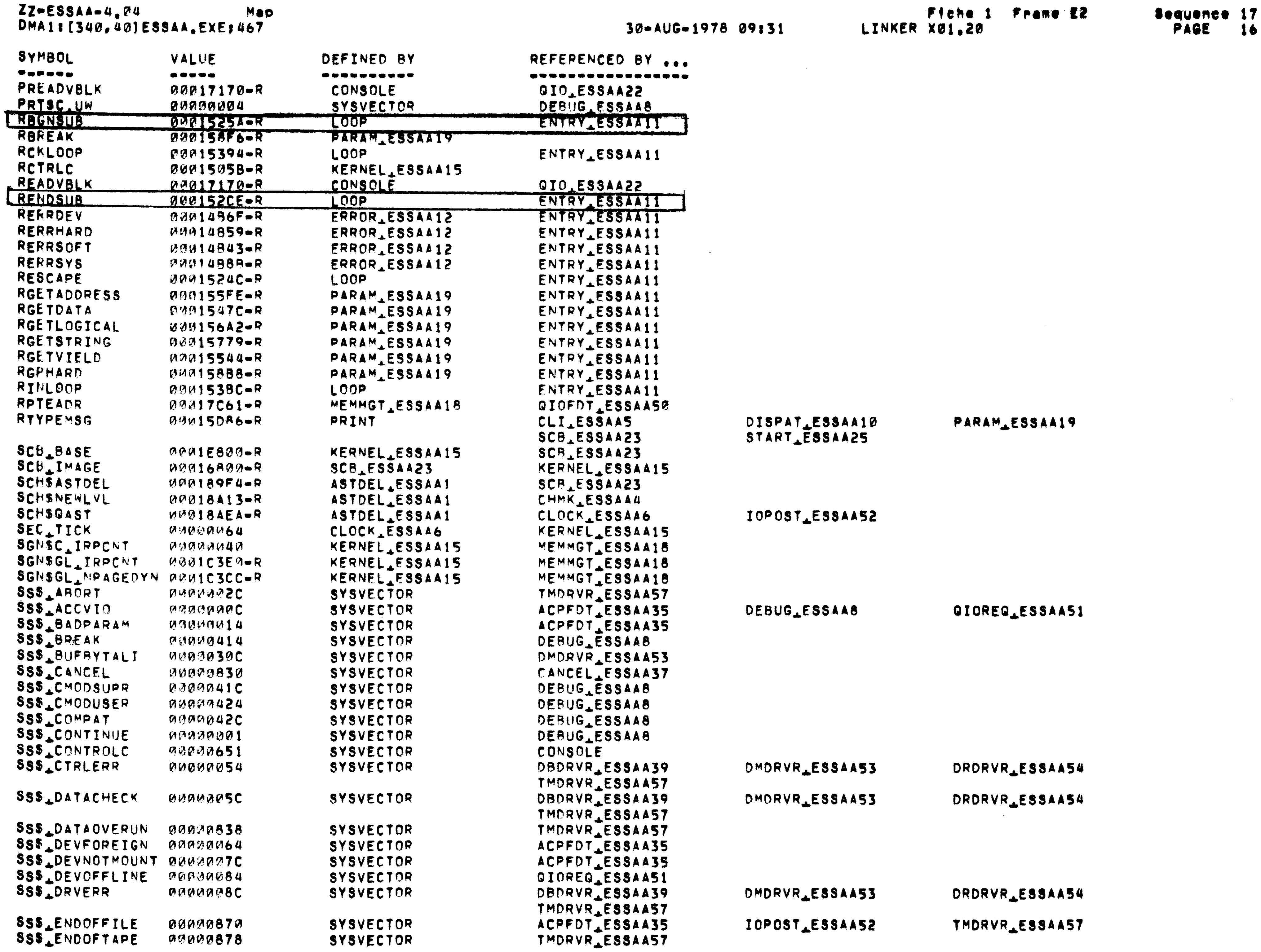
Figure 5-7 DS$BGNSUB Listed in the Symbol Table in the Supervisor Link Map



```
ZZ-ESSAA-4.04          ENTRY POINTS TO THE DIAGNOSTIC SUPERVISOR.                                     Fiche 3  Frame J9        Sequence 525
ENTRY_ESSAA11    DIAGNOSTIC AND STARLET SERVICE ENTRY VECTORS.    22-AUG-1978  08:22:06    VAX-11 MACRO X0.3-6                  Page   6
06-08            ENTRY POINTS TO THE DIAGNOSTIC SUPERVISOR.                                                                      (4)

                             0030    171 ;+
                             0030    172 ; 4.2.1            PROGRAM CONTROL SERVICES.
                             0030    173 ;-
                             0030    174            .ALIGN  QUAD
                             0030    175 DS$BGNSUB::                                    ; BEGIN SUBTEST ENTRY POINT.
                      0000   0030    176            .WORD   ^M<>                        ; ENTRY MASK
    00000000'EF         17   0032    177            JMP     RBGNSUB                     ;
                             0038    178
                             0038    179            .ALIGN  QUAD
                             0038    180 DS$ENDSUB::                                    ; END SUBTEST ENTRY POINT.
                      0000   0038    181            .WORD   ^M<>                        ; ENTRY MASK
    00000000'EF         17   003A    182            JMP     RENDSUB                     ;
                             0040    183
                             0040    184            .ALIGN  QUAD
                             0040    185 DS$CKLOOP::                                    ; CHECK LOOP ENRTY POINT.
                      0000   0040    186            .WORD   ^M<>                        ; ENTRY MASK
    00000000'EF         17   0042    187            JMP     RCKLOOP                     ;
                             0048    188
                             0048    189            .ALIGN  QUAD
                             0048    190 DS$INLOOP::                                    ; IN LOOP ENTRY POINT.
                      0000   0048    191            .WORD   ^M<>                        ; ENTRY MASK
    00000000'EF         17   004A    192            JMP     RINLOOP                     ;
                             0050    193
                             0050    194            .ALIGN  QUAD
                             0050    195 DS$ESCAPE::                                    ; ESCAPE ENTRY POINT.
                      0000   0050    196            .WORD   ^M<>                        ; ENTRY MASK
    00000000'EF         17   0052    197            JMP     RESCAPE                     ;
                             0058    198
                             0058    199            .ALIGN  QUAD
                             0058    200 DS$BREAK::                                     ; BREAK FOR DYNAMIC SERVICES.
                      0000   0058    201            .WORD   ^M<>                        ; SAVE NO REGISTERS
               FFA3'    30   005A    202            BSBW    KB_CHECK                    ; CHECK KEYBOARD
                        04   005D    203            RET
                             005E    204
                             005E    205            .ALIGN  QUAD
                             0060    206 DS$WAITMS::                                    ; WAIT MILLISECONDS ENTRY POINT.
                      0000   0060    207            .WORD   ^M<>                        ; ENTRY MASK
    00000000'EF         17   0062    208            JMP     DSX$WAITMS
                             0068    209
                             0068    210            .ALIGN  QUAD
                             0068    211 DS$WAITUS::                                     ; WAIT MICROSECONDS ENTRY POINT.
                      0000   0068    212            .WORD   ^M<>                        ; ENTRY MASK
    00000000'EF         17   006A    213            JMP     DSX$WAITUS
                             0070    214
                             0070    215            .ALIGN  QUAD
                             0070    216 DS$CANWAIT::
                             0070    217 DS$ABORTWAIT::                                 ; CANCEL WAIT ENTRY POINT.
                      0000   0070    218            .WORD   ^M<>                        ; ENTRY MASK
    00000000'EF         17   0072    219            JMP     DSX$CANWAIT
                             0078    220
                             0078    221            .ALIGN  QUAD
                             0078    222 DS$CNTRLC::                                    ; ^X10078
                     0000'   0078    223            .VECTOR DSX$CNTRLC
    00000002'EF         17   007A    224            JMP     DSX$CNTRLC+2
```

TK-1122

Figure 5-8 DS$BGNSUB Entry Point



```
ZZ-ESSAA-4.04           Map                                                                             Fiche 1  Frame E2        Sequence 17
DMA1:[340,40]ESSAA.EXE;467                                            30-AUG-1978 09:31        LINKER X01.20                          PAGE   16

SYMBOL             VALUE              DEFINED BY             REFERENCED BY ...
------             -----              ----------             -----------------
PREADVBLK          00017170-R          CONSOLE                QIO_ESSAA22
PRTSC_UW           00000004            SYSVECTOR              DEBUG_ESSAA8
RBGNSUB            0001525A-R          LOOP                   ENTRY_ESSAA11
RBREAK             000158F6-R          PARAM_ESSAA19
RCKLOOP            00015394-R          LOOP                   ENTRY_ESSAA11
RCTRLC             0001505B-R          KERNEL_ESSAA15
READVBLK           00017170-R          CONSOLE                QIO_ESSAA22
RENDSUB            000152CE-R          LOOP                   ENTRY_ESSAA11
RERRDEV            0001496F-R          ERROR_ESSAA12          ENTRY_ESSAA11
RERRHARD           00014859-R          ERROR_ESSAA12          ENTRY_ESSAA11
RERRSOFT           00014843-R          ERROR_ESSAA12          ENTRY_ESSAA11
RERRSYS            0001488B-R          ERROR_ESSAA12          ENTRY_ESSAA11
RESCAPE            0001524C-R          LOOP                   ENTRY_ESSAA11
RGETADDRESS        000155FE-R          PARAM_ESSAA19          ENTRY_ESSAA11
RGETDATA           0001547C-R          PARAM_ESSAA19          ENTRY_ESSAA11
RGETLOGICAL        000156A2-R          PARAM_ESSAA19          ENTRY_ESSAA11
RGETSTRING         00015779-R          PARAM_ESSAA19          ENTRY_ESSAA11
RGETVIELD          00015544-R          PARAM_ESSAA19          ENTRY_ESSAA11
RGPHARD            000158B8-R          PARAM_ESSAA19          ENTRY_ESSAA11
RINLOOP            0001538C-R          LOOP                   ENTRY_ESSAA11
RPTEADR            00017C61-R          MEMMGT_ESSAA18         QIOFDT_ESSAA50
RTYPEMSG           00015D86-R          PRINT                  CLI_ESSAA5             DISPAT_ESSAA10         PARAM_ESSAA19
                                                              SCB_ESSAA23            START_ESSAA25
SCB_BASE           0001E800-R          KERNEL_ESSAA15         SCB_ESSAA23
SCB_IMAGE          00016800-R          SCB_ESSAA23            KERNEL_ESSAA15
SCH$ASTDEL         000189F4-R          ASTDEL_ESSAA1          SCB_ESSAA23
SCH$NEWLVL         00018A13-R          ASTDEL_ESSAA1          CHMK_ESSAA4
SCH$QAST           00018AEA-R          ASTDEL_ESSAA1          CLOCK_ESSAA6           IOPOST_ESSAA52
SEC_TICK           00000064            CLOCK_ESSAA6           KERNEL_ESSAA15
SGN$C_IRPCNT       00000040            KERNEL_ESSAA15         MEMMGT_ESSAA18
SGN$GL_IRPCNT      0001C3E0-R          KERNEL_ESSAA15         MEMMGT_ESSAA18
SGN$GL_NPAGEDYN    0001C3CC-R          KERNEL_ESSAA15         MEMMGT_ESSAA18
SS$_ABORT          0000002C            SYSVECTOR              TMDRVR_ESSAA57
SS$_ACCVIO         0000000C            SYSVECTOR              ACPFDT_ESSAA35         DEBUG_ESSAA8           QIOREQ_ESSAA51
SS$_BADPARAM       00000014            SYSVECTOR              ACPFDT_ESSAA35
SS$_BREAK          00000414            SYSVECTOR              DEBUG_ESSAA8
SS$_BUFBYTALI      0000030C            SYSVECTOR              DMDRVR_ESSAA53
SS$_CANCEL         00000830            SYSVECTOR              CANCEL_ESSAA37
SS$_CMODSUPR       0000041C            SYSVECTOR              DEBUG_ESSAA8
SS$_CMODUSER       00000424            SYSVECTOR              DEBUG_ESSAA8
SS$_COMPAT         0000042C            SYSVECTOR              DEBUG_ESSAA8
SS$_CONTINUE       00000001            SYSVECTOR              DEBUG_ESSAA8
SS$_CONTROLC       00000651            SYSVECTOR              CONSOLE
SS$_CTRLERR        00000054            SYSVECTOR              DBDRVR_ESSAA39         DMDRVR_ESSAA53         DRDRVR_ESSAA54
                                                              TMDRVR_ESSAA57
SS$_DATACHECK      0000005C            SYSVECTOR              DBDRVR_ESSAA39         DMDRVR_ESSAA53         DRDRVR_ESSAA54
                                                              TMDRVR_ESSAA57
SS$_DATAOVERUN     00000838            SYSVECTOR              TMDRVR_ESSAA57
SS$_DEVFOREIGN     00000064            SYSVECTOR              ACPFDT_ESSAA35
SS$_DEVNOTMOUNT    0000007C            SYSVECTOR              ACPFDT_ESSAA35
SS$_DEVOFFLINE     00000084            SYSVECTOR              QIOREQ_ESSAA51
SS$_DRVERR         0000008C            SYSVECTOR              DBDRVR_ESSAA39         DMDRVR_ESSAA53         DRDRVR_ESSAA54
                                                              TMDRVR_ESSAA57
SS$_ENDOFFILE      00000870            SYSVECTOR              ACPFDT_ESSAA35         IOPOST_ESSAA52         TMDRVR_ESSAA57
SS$_ENDOFTAPE      00000878            SYSVECTOR              TMDRVR_ESSAA57
```

TK-1123

Figure 5-9 RBGNSUB Listed in the Symbol Table in the Diagnostic Supervisor Link Map

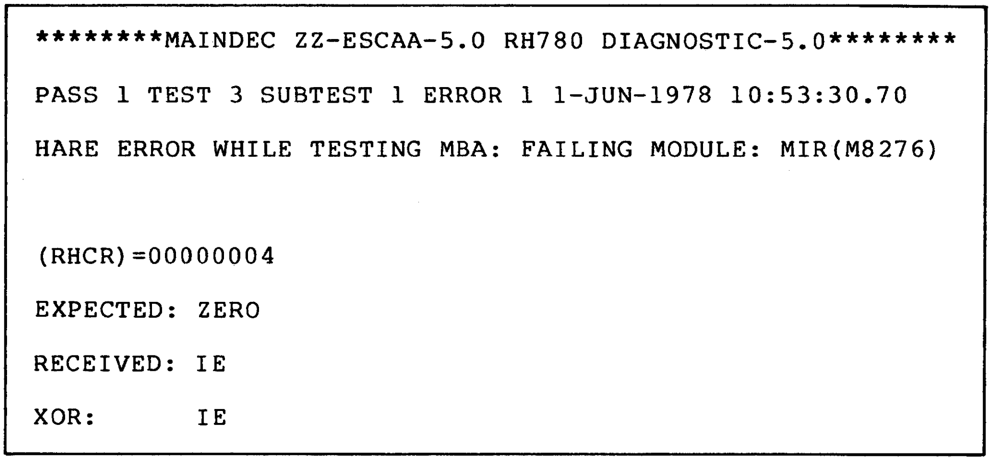
Subtest 1 initializes the MBA under test by writing a one to bit zero of the control register, the initialization bit. The subtest then writes ones to the control register, writes zeros, and then reads the register. If zeros are not returned from the control register, the subtest calls the DS$ERRHARD entry point in the supervisor. The DS$ERRHARD code is in the supervisor entry module, ESSAA11 (see the supervisor link map). DS$ERRHARD causes a jump to RERRHARD, which is located in the error module of the supervisor. RERRHARD sets the error flag, rings the bell if the bell on error flag is set, and calls the DS$PRINTF entry point. Like the other DS$ entry points, this one is located in the entry module and contains only a jump instruction. The jump transfers control to the RPRINTF routine in the print module (see the supervisor link map). RPRINTF, in turn, calls the RPRINTOUT routine, also in the print module. This routine prints out a message on the operator's terminal indicating the test and subtest numbers, the logical unit number (LUN) under test, and the failing module name (MIR). RPRINTOUT returns control to RPRINTF, which returns control to RERRHARD, which returns control to ESCAA test 3, subtest 1.

At this point, the testing and error reporting portions of subtest 1 have been completed. The subtest then calls DS$CKLOOP (in the control module of the supervisor). This entry point causes a jump to the RCKLOOP subroutine, which is located in the loop module. Unless the loop and error flags are both set, control returns to the subtest, which in turn calls DS$ENDSUB in the supervisor to terminate subtest 1 and start the next subtest.

If the loop and error flags are both set, the loop address is calculated and the RCLKLOOP routine causes a jump back to the beginning of subtest 1, at label 10$. Note that if the loop is the first loop after the error call, the test and subtest numbers are checked. If one of these numbers is wrong, control returns to the subtest as if the loop flag were not set. After the first loop, the subtest will be repeated continually.

5.6.2 RH780 Diagnostic Sample Error Message

The error messages generated by the RH780 diagnostic vary, depending on the type of error detected and the type of the failing test. However, in all cases the error message will identify the failing module (or bus signal) and the nature of the failure. Expected and received data are printed when meaningful. For example, if bit 2 (IE) of the MBA control register is set, the error message printed will look like that shown in Figure 5-10.



```
********MAINDEC ZZ-ESCAA-5.0 RH780 DIAGNOSTIC-5.0********

PASS 1 TEST 3 SUBTEST 1 ERROR 1 1-JUN-1978 10:53:30.70

HARE ERROR WHILE TESTING MBA: FAILING MODULE: MIR(M8276)


(RHCR)=00000004

EXPECTED: ZERO

RECEIVED: IE

XOR:      IE
```

TK-0780

Figure 5-1Ø ESRCA Sample Error Listing

Use the test, subtest, and error number to look up the relevant code in the program listing. Notice that test 3, subtest 1, error 1 is the portion of the program discussed in the previous paragraph. The program sets the maintenance bit in the control register. It then writes ones to the control register, clears the register, and reads the register. Since bit 2 is stuck at one, the received data and the expected data do not match.

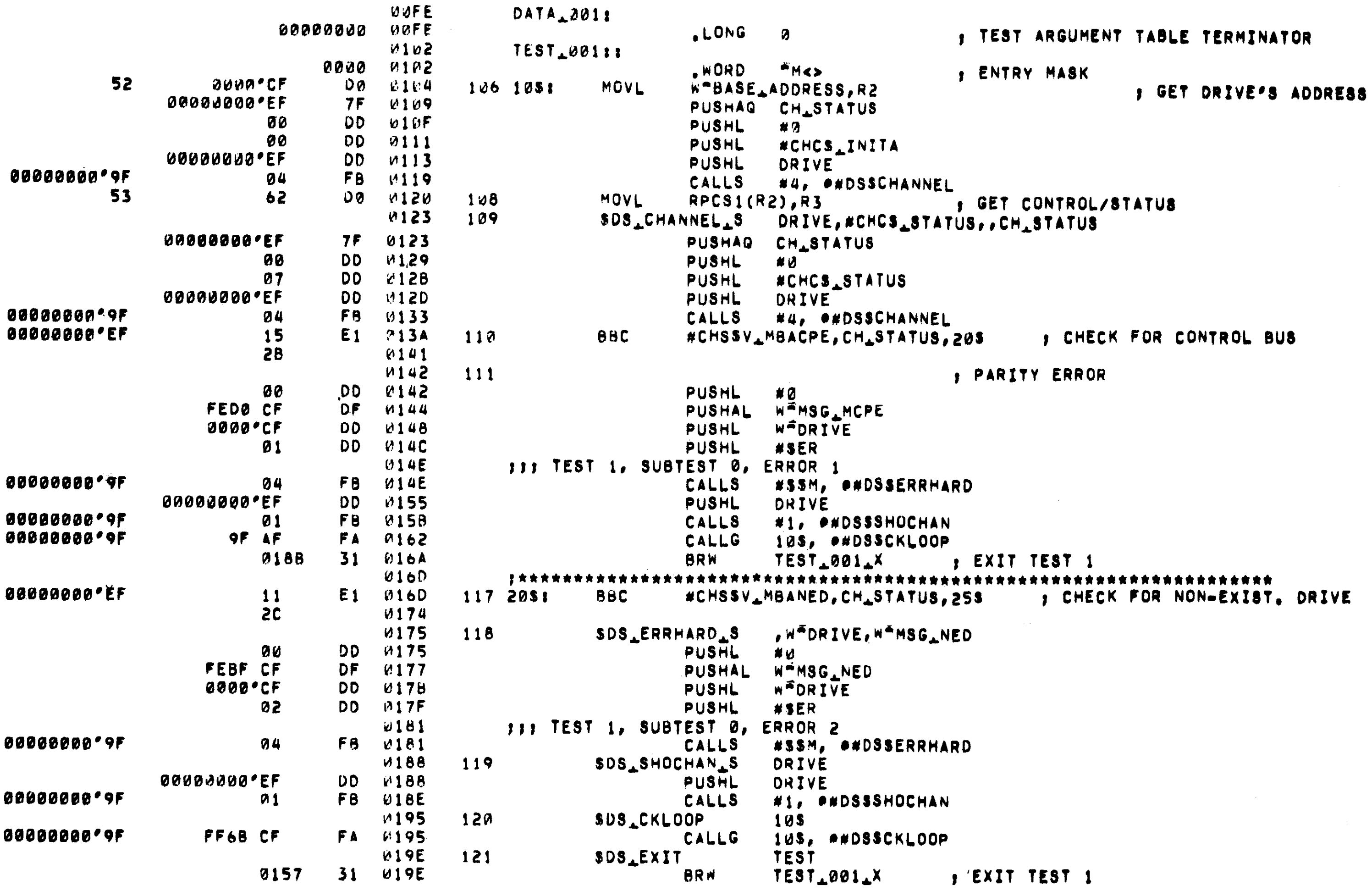
After listing the failure, the program continues with the succeeding tests. The operator may, at this point, shut down the computer to change the MIR board as directed, or use the diagnostic supervisor commands to set up a scope loop and monitor the failure more closely.

5.7 RPØX/DCL REPAIR DIAGNOSTIC (DIRECT I/O), SAMPLE SUBTEST

5.7.1 Detailed Flow

The RPØX/DCL Repair Diagnostic (ESRCA) is representative of the peripheral diagnostic programs that use direct I/O. Like the cluster diagnostic programs, the RPØX/DCL repair diagnostic accesses registers on the unit under test directly, with move and bit set instructions and the like. However, the RPØX/DCL repair diagnostic relies more heavily than the MBA diagnostic on services provided by the diagnostic supervisor. In particular, it uses the channel services of the supervisor to perform such functions as initializing a channel, aborting a function, enabling and disabling interrupts, setting map registers, defeating parity, and determining adapter and error status.

For example, when the dispatch routine in the supervisor calls the first test in the RPØX/DCL repair diagnostic, the test routine gets the address of the device under test and then, passing a list of arguments, calls for channel services through the DS$CHANNEL entry point in the supervisor. Figure 5-11 shows the code for test 1, subtest Ø, errors 1-3. Figure 5-12 shows the subtest flow. DS$CHANNEL calls DSX$CHANNEL, which in turn, calls the BLDCDB subroutine, which builds a channel data block containing the



```
                                 00FE        DATA_001:
                       00000000  00FE                           .LONG    0                ; TEST ARGUMENT TABLE TERMINATOR
                                 0102        TEST_001::
                           0000  0102                           .WORD    ^M<>             ; ENTRY MASK
       52       0000'CF     D0   0104   106 10$:     MOVL       W^BASE_ADDRESS,R2                                 ; GET DRIVE'S ADDRESS
            00000000'EF     7F   0109                           PUSHAQ   CH_STATUS
                     00     DD   010F                           PUSHL    #0
                     00     DD   0111                           PUSHL    #CHCS_INITA
            00000000'EF     DD   0113                           PUSHL    DRIVE
00000000'9F          04     FB   0119                           CALLS    #4, @#DS$CHANNEL
       53            62     D0   0120   108          MOVL       RPCS1(R2),R3              ; GET CONTROL/STATUS
                                 0123   109          $DS_CHANNEL_S   DRIVE,#CHCS_STATUS,,CH_STATUS
            00000000'EF     7F   0123                           PUSHAQ   CH_STATUS
                     00     DD   0129                           PUSHL    #0
                     07     DD   012B                           PUSHL    #CHCS_STATUS
            00000000'EF     DD   012D                           PUSHL    DRIVE
00000000'9F          04     FB   0133                           CALLS    #4, @#DS$CHANNEL
00000000'EF          15     E1   013A   110          BBC        #CHSSV_MBACPE,CH_STATUS,20$     ; CHECK FOR CONTROL BUS
                     2B          0141
                                 0142   111                                                    ; PARITY ERROR
                     00     DD   0142                           PUSHL    #0
                FED0 CF     DF   0144                           PUSHAL   W^MSG_MCPE
                0000'CF     DD   0148                           PUSHL    W^DRIVE
                     01     DD   014C                           PUSHL    #SER
                                 014E        ;;; TEST 1, SUBTEST 0, ERROR 1
00000000'9F          04     FB   014E                           CALLS    #SSM, @#DS$ERRHARD
            00000000'EF     DD   0155                           PUSHL    DRIVE
00000000'9F          01     FB   015B                           CALLS    #1, @#DS$SHOCHAN
00000000'9F       9F AF     FA   0162                           CALLG    10$, @#DS$CKLOOP
                   018B     31   016A                           BRW      TEST_001_X       ; EXIT TEST 1
                                 016D        ;*******************************************************************
00000000'EF          11     E1   016D   117 20$:     BBC        #CHSSV_MBANED,CH_STATUS,25$     ; CHECK FOR NON-EXIST. DRIVE
                     2C          0174
                                 0175   118          $DS_ERRHARD_S      ,W^DRIVE,W^MSG_NED
                     00     DD   0175                           PUSHL    #0
                FEBF CF     DF   0177                           PUSHAL   W^MSG_NED
                0000'CF     DD   017B                           PUSHL    W^DRIVE
                     02     DD   017F                           PUSHL    #SER
                                 0181        ;;; TEST 1, SUBTEST 0, ERROR 2
00000000'9F          04     FB   0181                           CALLS    #SSM, @#DS$ERRHARD
                                 0188   119          $DS_SHOCHAN_S      DRIVE
            00000000'EF     DD   0188                           PUSHL    DRIVE
00000000'9F          01     FB   018E                           CALLS    #1, @#DS$SHOCHAN
                                 0195   120          $DS_CKLOOP         10$
00000000'9F     FF6B CF     FA   0195                           CALLG    10$, @#DS$CKLOOP
                                 019E   121          $DS_EXIT           TEST
                   0157     31   019E                           BRW      TEST_001_X       ; EXIT TEST 1
```

TK-1124

Figure 5-11 ESRCA RPØX/DCL
Test 1, Subtest Ø,
Program Listing

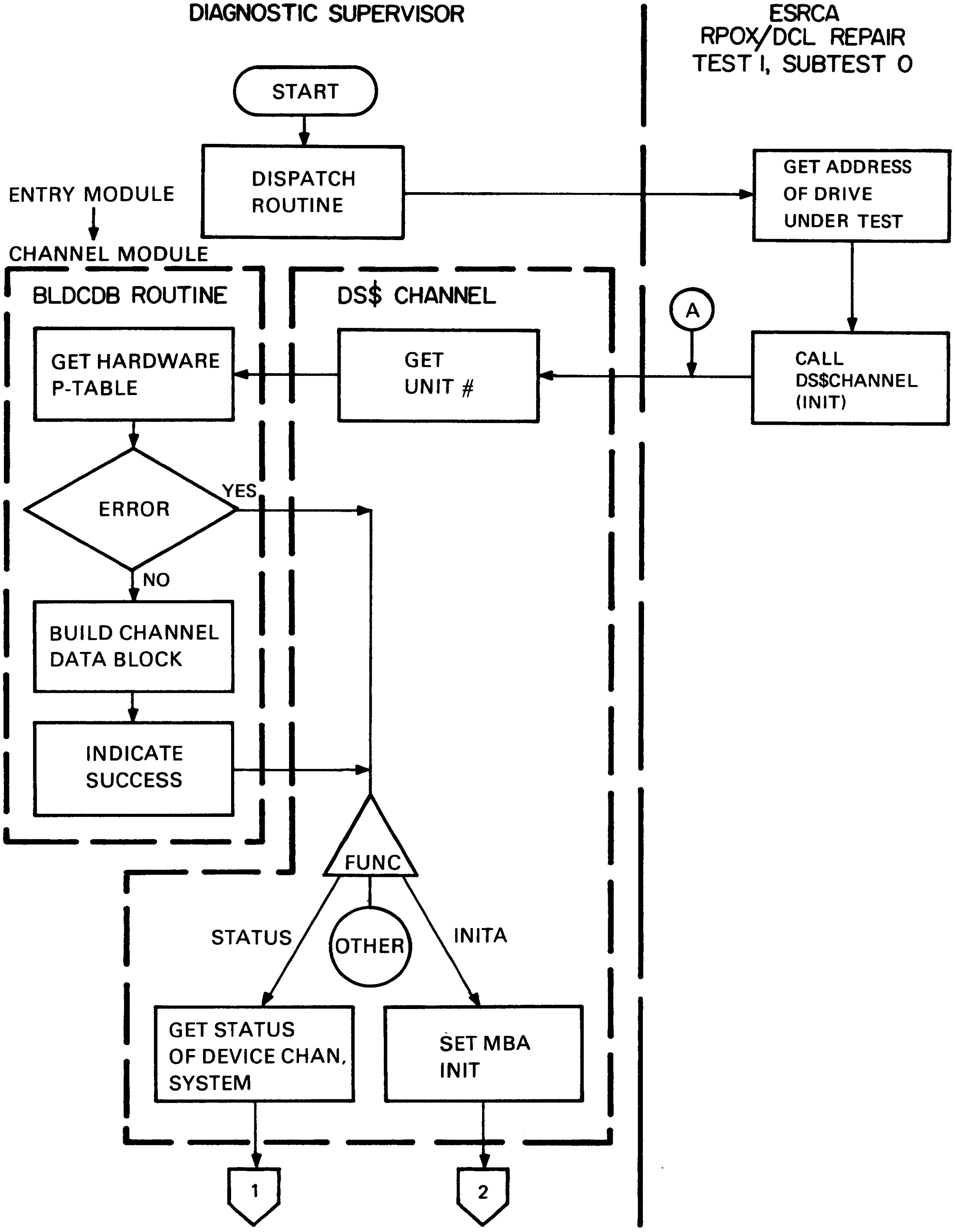
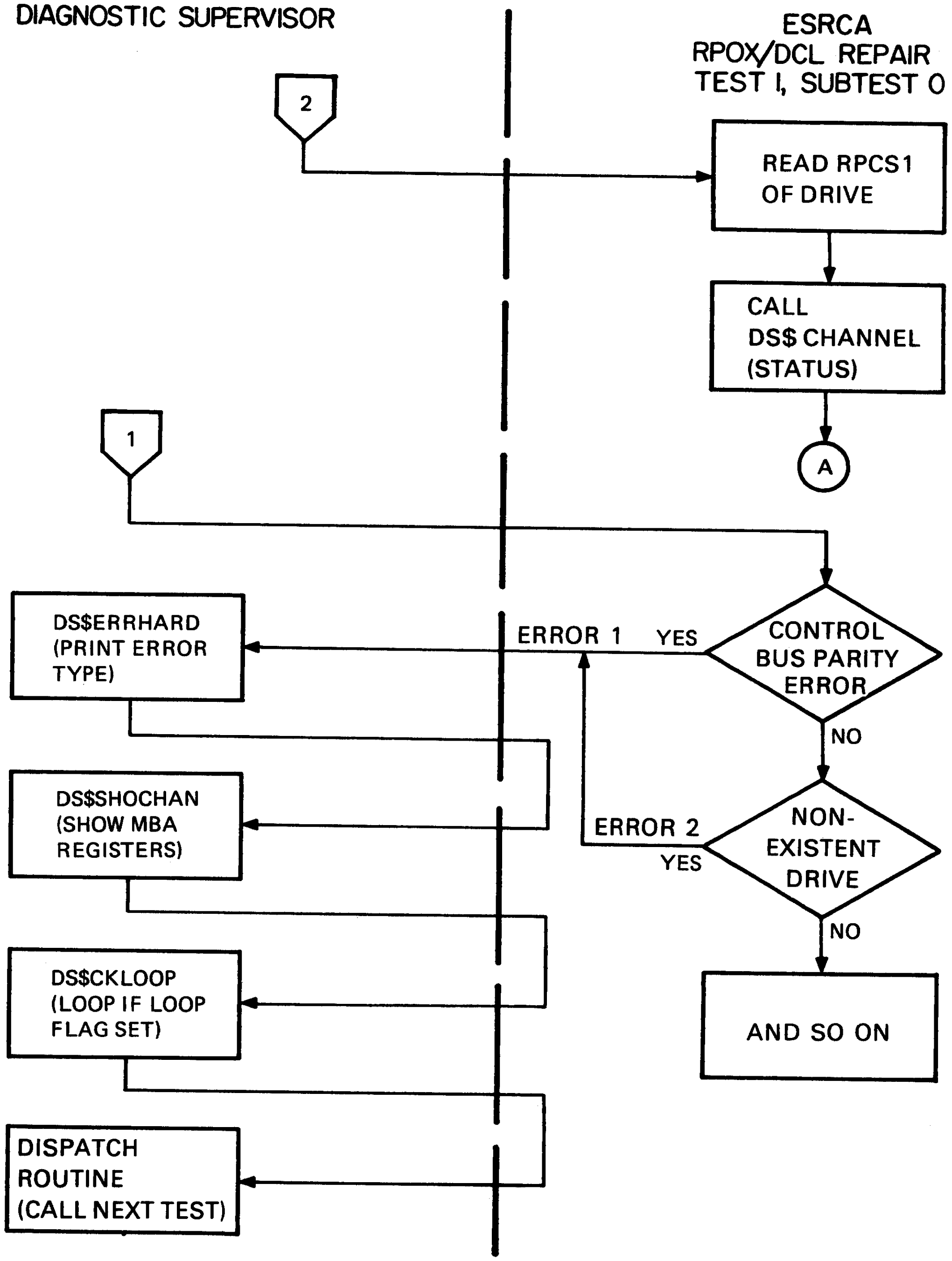


Figure 5-12   ESRCA RPØX/DCL Repair Diagnostic Test 1, Subtest Ø, Flowchart (Sheet 1 of 2)



TK-0748

Figure 5-12 ESRCA RPØX/DCL Repair Diagnostic Test 1, Subtest Ø, Flowchart (Sheet 2 of 2)

P-Table address, and the adapter address; clears the flag word; and determines whether the channel is an MBA or a UBA. When control returns to the DSX$CHANNEL routine, the function argument passed from the calling program (ESRCA) is evaluated, activating one of several function subroutines. In this case, the INITA subroutine sets the initialization bit in the MBA control register.

Control then returns to the calling program (ESRCA), which reads the RPCS1 (control status) register of the unit under test with a MOVE instruction (direct I/O). The test routine then calls DS$CHANNEL again, this time passing a different function argument (CHC$_STATUS). The DSX$CHANNEL routine is executed again, activating the CHC$_STATUS subroutine which stores the unit and adapter status in the location labeled CH_STATUS.

Then when control returns from the DSX$CHANNEL routine to the test routine, the data in location CH_STATUS is compared, bit by bit, with expected data patterns. If an error is detected, the test routine calls a series of supervisor routines (DS$ERRHARD, DS$SHOCHAN, DS$CKLOOP, and DISPATCH) to print out error information, loop if the loop flag is set, and return to the dispatch routine if the loop flag is not set.

5.7.2 RPØX/DCL Repair Diagnostic Sample Error Message

Test 1, subtest Ø, error 1 of the RPØX/DCL diagnostic identifies control bus parity failures on the Massbus (Figure 5-11). When the program detects this failure, the error message identifies the failure by test, subtest, error, failing unit, and error type. In addition, the message includes an MBA channel status dump, showing the contents of the pertinent registers, as shown in Figure 5-13.

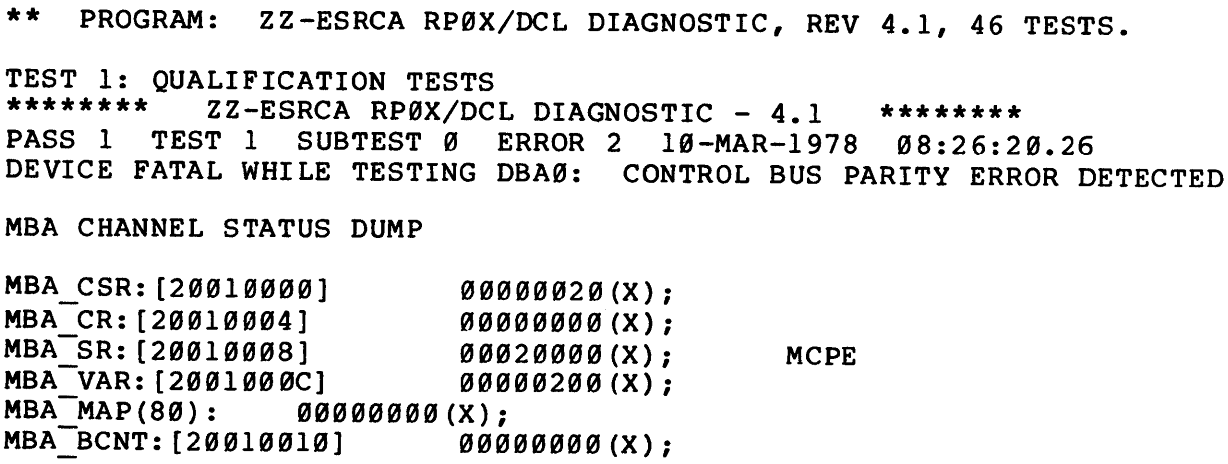
Bit 17 of the status register is set, indicating the Massbus control parity error.

Other error message formats display expected and received data and the contents of relevant registers in the RPØX/DCL, depending on the error and the failing test.

5.8 DISK RELIABILITY DIAGNOSTIC (QUEUE I/O), SAMPLE SUBTEST

5.8.1 Detailed Flow

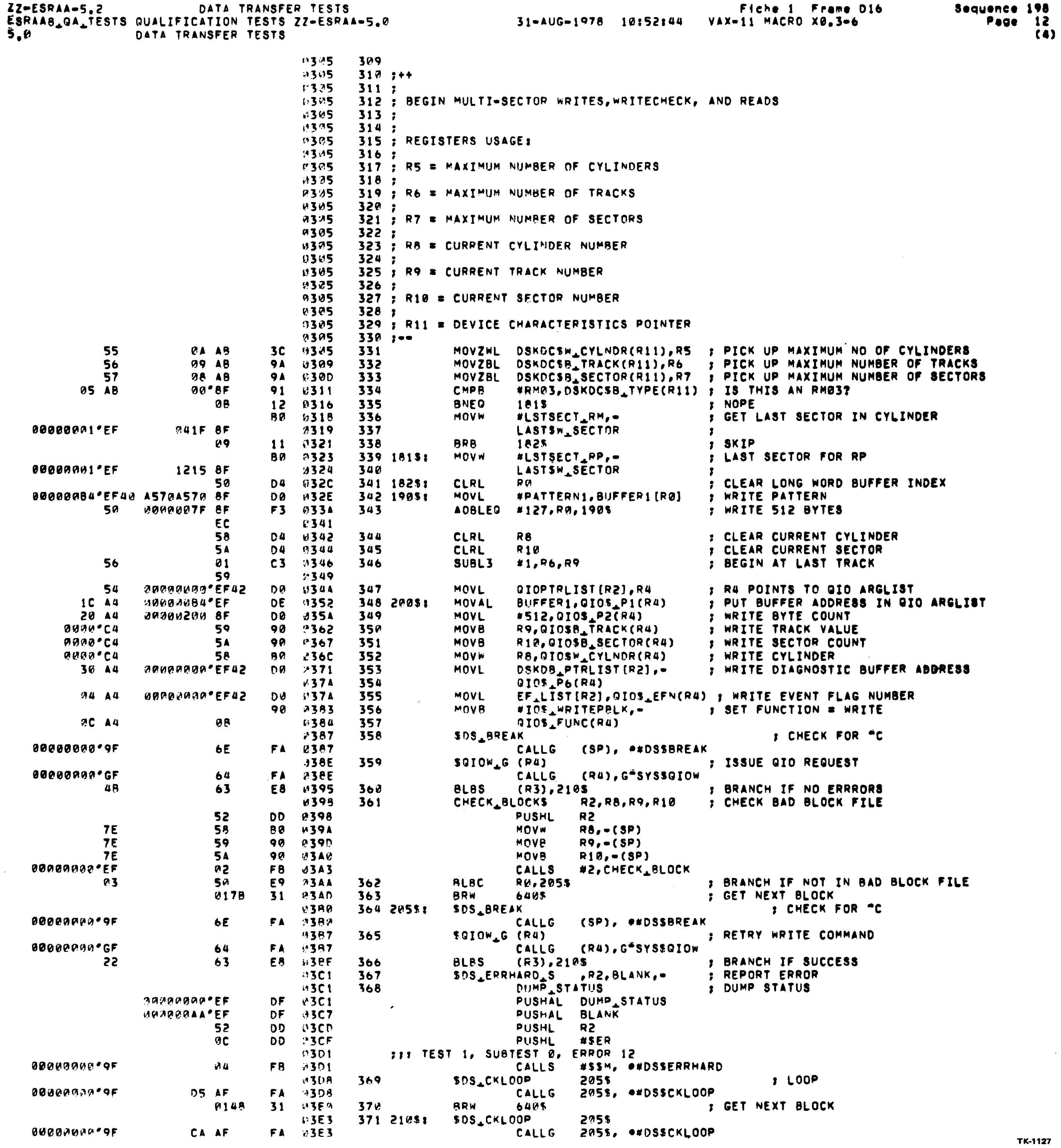
The Disk Reliability Diagnostic program (ESRAA) is representative of the queue I/O diagnostics. Instead of performing move instructions to read and write peripheral device registers directly (as the MBA diagnostic does), the program builds an argument list containing device and transfer parameters and pointing to the data to be transferred and the function to be performed. The program then calls the queue I/O services of VMS or the diagnostic supervisor. In this way, the program transfers information to and from the peripheral device under test without requiring exclusive use of the device, the channel, or the computer system. Figure 5-14 shows the listing for ESRAA test 1, subtest Ø, error 12, the first of the data transfer tests. Figure 5-15 shows the program flow for the same routine.



```
**  PROGRAM:  ZZ-ESRCA RPØX/DCL DIAGNOSTIC, REV 4.1, 46 TESTS.

TEST 1: QUALIFICATION TESTS
********   ZZ-ESRCA RPØX/DCL DIAGNOSTIC - 4.1    ********
PASS 1  TEST 1  SUBTEST Ø  ERROR 2  1Ø-MAR-1978  Ø8:26:2Ø.26
DEVICE FATAL WHILE TESTING DBAØ:  CONTROL BUS PARITY ERROR DETECTED

MBA CHANNEL STATUS DUMP

MBA_CSR:[2ØØ1ØØØØ]       ØØØØØØ2Ø(X);
MBA_CR:[2ØØ1ØØØ4]        ØØØØØØØØ(X);
MBA_SR:[2ØØ1ØØØ8]        ØØØ2ØØØØ(X);       MCPE
MBA_VAR:[2ØØ1ØØØC]       ØØØØØ2ØØ(X);
MBA_MAP(8Ø):    ØØØØØØØØ(X);
MBA_BCNT:[2ØØ1ØØ1Ø]      ØØØØØØØØ(X);
```

TK-1265

Figure 5-13 ESRCA Sample Error Message



```
ZZ-ESRAA-5.2           DATA TRANSFER TESTS                                          Fiche 1  Frame D16         Sequence 198
ESRAAB_QA_TESTS QUALIFICATION TESTS ZZ-ESRAA-5.0              31-AUG-1978  10:52:44   VAX-11 MACRO X0.3-6             Page  12
5.0              DATA TRANSFER TESTS                                                                                     (4)

                                  0305   309
                                  0305   310 ;++
                                  0305   311 ;
                                  0305   312 ; BEGIN MULTI-SECTOR WRITES,WRITECHECK, AND READS
                                  0305   313 ;
                                  0305   314 ;
                                  0305   315 ; REGISTERS USAGE:
                                  0305   316 ;
                                  0305   317 ; R5 = MAXIMUM NUMBER OF CYLINDERS
                                  0305   318 ;
                                  0305   319 ; R6 = MAXIMUM NUMBER OF TRACKS
                                  0305   320 ;
                                  0305   321 ; R7 = MAXIMUM NUMBER OF SECTORS
                                  0305   322 ;
                                  0305   323 ; R8 = CURRENT CYLINDER NUMBER
                                  0305   324 ;
                                  0305   325 ; R9 = CURRENT TRACK NUMBER
                                  0305   326 ;
                                  0305   327 ; R10 = CURRENT SECTOR NUMBER
                                  0305   328 ;
                                  0305   329 ; R11 = DEVICE CHARACTERISTICS POINTER
                                  0305   330 ;--
          55        0A AB     3C  0305   331         MOVZWL  DSKDC$W_CYLNDR(R11),R5  ; PICK UP MAXIMUM NO OF CYLINDERS
          56        09 AB     9A  0309   332         MOVZBL  DSKDC$B_TRACK(R11),R6   ; PICK UP MAXIMUM NUMBER OF TRACKS
          57        08 AB     9A  030D   333         MOVZBL  DSKDC$B_SECTOR(R11),R7  ; PICK UP MAXIMUM NUMBER OF SECTORS
       05 AB        00'8F     91  0311   334         CMPB    #RM03,DSKDC$B_TYPE(R11) ; IS THIS AN RM03?
                       0B     12  0316   335         BNEQ    181$                    ; NOPE
                              B0  0318   336         MOVW    #LSTSECT_RM,-           ; GET LAST SECTOR IN CYLINDER
 00000001'EF      041F 8F         0319   337                 LAST$W_SECTOR           ;
                       09     11  0321   338         BRB     182$                    ; SKIP
                              B0  0323   339 181$:   MOVW    #LSTSECT_RP,-           ; LAST SECTOR FOR RP
 00000001'EF      1215 8F         0324   340                 LAST$W_SECTOR           ;
                       50     D4  032C   341 182$:   CLRL    R0                      ; CLEAR LONG WORD BUFFER INDEX
 000000B4'EF40 A570A570 8F    D0  032E   342 190$:   MOVL    #PATTERN1,BUFFER1[R0]   ; WRITE PATTERN
          50    0000007F 8F   F3  033A   343         AOBLEQ  #127,R0,190$            ; WRITE 512 BYTES
                       EC         0341
                       58     D4  0342   344         CLRL    R8                      ; CLEAR CURRENT CYLINDER
                       5A     D4  0344   345         CLRL    R10                     ; CLEAR CURRENT SECTOR
          56           01     C3  0346   346         SUBL3   #1,R6,R9                ; BEGIN AT LAST TRACK
                       59         0349
          54    00000000'EF42 D0  034A   347         MOVL    QIOPTRLIST[R2],R4       ; R4 POINTS TO QIO ARGLIST
       1C A4    000000B4'EF   DE  0352   348 200$:   MOVAL   BUFFER1,QIO$_P1(R4)     ; PUT BUFFER ADDRESS IN QIO ARGLIST
       20 A4    00000200 8F   D0  035A   349         MOVL    #512,QIO$_P2(R4)        ; WRITE BYTE COUNT
     0000'C4           59     90  0362   350         MOVB    R9,QIO$B_TRACK(R4)      ; WRITE TRACK VALUE
     0000'C4           5A     90  0367   351         MOVB    R10,QIO$B_SECTOR(R4)    ; WRITE SECTOR COUNT
     0000'C4           58     B0  036C   352         MOVW    R8,QIO$W_CYLNDR(R4)     ; WRITE CYLINDER
       30 A4    00000000'EF42 D0  0371   353         MOVL    DSKDB_PTRLIST[R2],-     ; WRITE DIAGNOSTIC BUFFER ADDRESS
                                  037A   354                 QIO$_P6(R4)
       04 A4    00000000'EF42 D0  037A   355         MOVL    EF_LIST[R2],QIO$_EFN(R4) ; WRITE EVENT FLAG NUMBER
                              90  0383   356         MOVB    #IO$_WRITEPBLK,-        ; SET FUNCTION = WRITE
       0C A4           08         0384   357                 QIO$_FUNC(R4)
                                  0387   358         $DS_BREAK                               ; CHECK FOR ^C
 00000000'9F           6E     FA  0387                       CALLG   (SP), @#DS$BREAK
                                  038E   359         $QIOW_G (R4)                    ; ISSUE QIO REQUEST
 00000000'GF           64     FA  038E                       CALLG   (R4),G^SYS$QIOW
          4B           63     E8  0395   360         BLBS    (R3),210$               ; BRANCH IF NO ERRRORS
                                  0398   361         CHECK_BLOCKS    R2,R8,R9,R10    ; CHECK BAD BLOCK FILE
                       52     DD  0398                       PUSHL   R2
          7E           58     B0  039A                       MOVW    R8,-(SP)
          7E           59     90  039D                       MOVB    R9,-(SP)
          7E           5A     90  03A0                       MOVB    R10,-(SP)
 00000002'EF           02     FB  03A3                       CALLS   #2,CHECK_BLOCK
          03           50     E9  03AA   362         BLBC    R0,205$                 ; BRANCH IF NOT IN BAD BLOCK FILE
                     017B     31  03AD   363         BRW     640$                    ; GET NEXT BLOCK
                                  03B0   364 205$:   $DS_BREAK                               ; CHECK FOR ^C
 00000000'9F           6E     FA  03B0                       CALLG   (SP), @#DS$BREAK
                                  03B7   365         $QIOW_G (R4)                    ; RETRY WRITE COMMAND
 00000000'GF           64     FA  03B7                       CALLG   (R4),G^SYS$QIOW
          22           63     E8  03BF   366         BLBS    (R3),210$               ; BRANCH IF SUCCESS
                                  03C1   367         $DS_ERRHARD_S   ,R2,BLANK,-     ; REPORT ERROR
                                  03C1   368                 DUMP_STATUS             ; DUMP STATUS
                00000000'EF   DF  03C1                       PUSHAL  DUMP_STATUS
                000000AA'EF   DF  03C7                       PUSHAL  BLANK
                       52     DD  03CD                       PUSHL   R2
                       0C     DD  03CF                       PUSHL   #$ER
                                  03D1       ;;; TEST 1, SUBTEST 0, ERROR 12
 00000000'9F           04     FB  03D1                       CALLS   #$$M, @#DS$ERRHARD
                                  03D8   369         $DS_CKLOOP      205$                    ; LOOP
 00000000'9F        D5 AF     FA  03D8                       CALLG   205$, @#DS$CKLOOP
                     0148     31  03E0   370         BRW     640$                    ; GET NEXT BLOCK
                                  03E3   371 210$:   $DS_CKLOOP      205$
 00000000'9F        CA AF     FA  03E3                       CALLG   205$, @#DS$CKLOOP
```

TK-1127

Figure 5-14 Disk Reliability (ESRAA) Test 1, Subtest Ø, Error 12 Listing

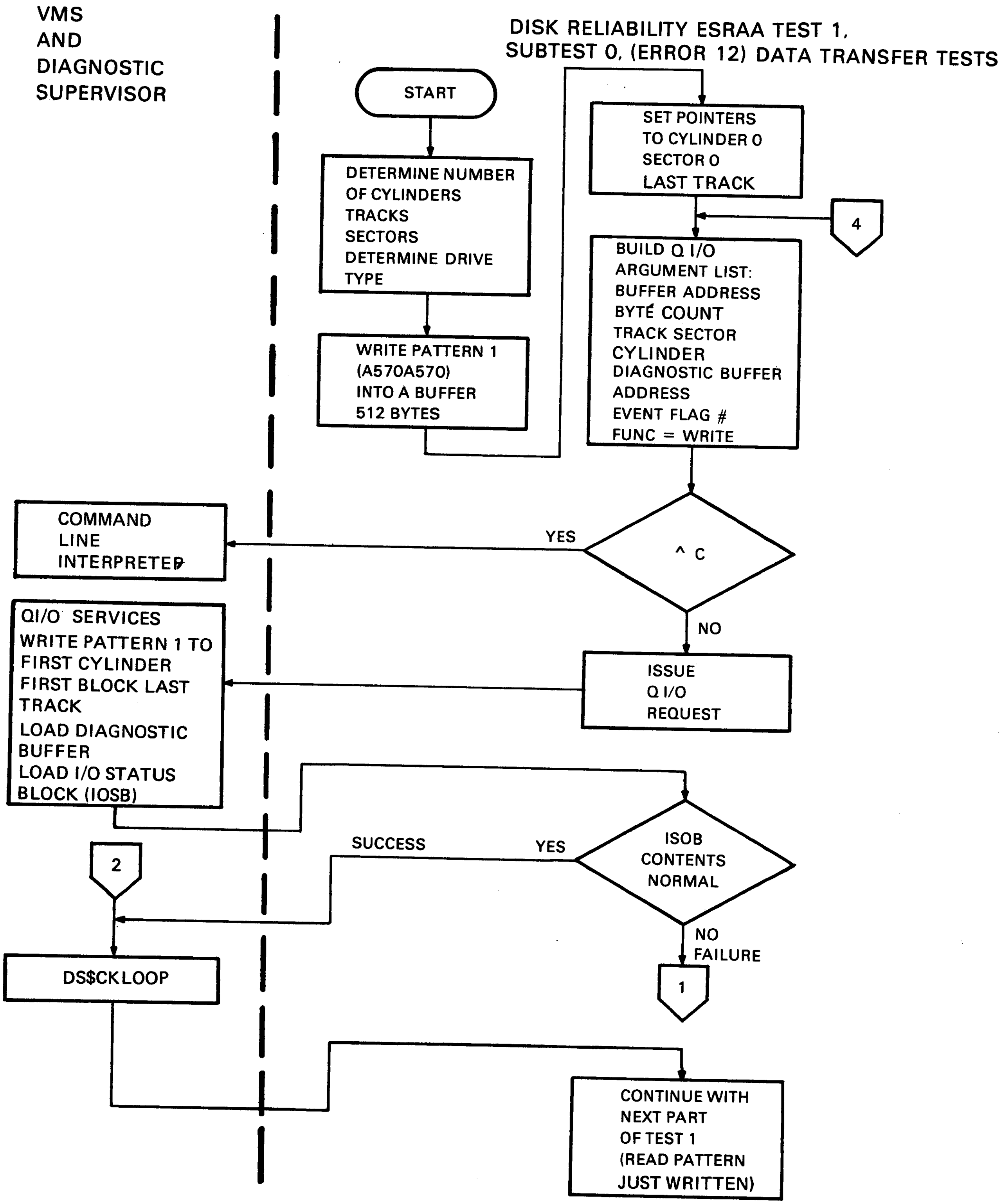

VMS
AND
DIAGNOSTIC
SUPERVISOR
DISK RELIABILITY ESRAA TEST 1,
SUBTEST 0, (ERROR 12) DATA TRANSFER TESTS
START
DETERMINE NUMBER OF CYLINDERS TRACKS SECTORS DETERMINE DRIVE TYPE
WRITE PATTERN 1 (A570A570) INTO A BUFFER 512 BYTES
SET POINTERS TO CYLINDER 0 SECTOR 0 LAST TRACK
4
BUILD Q I/O ARGUMENT LIST: BUFFER ADDRESS BYTE COUNT TRACK SECTOR CYLINDER DIAGNOSTIC BUFFER ADDRESS EVENT FLAG # FUNC = WRITE
COMMAND LINE INTERPRETER
YES
^ C
NO
QI/O SERVICES WRITE PATTERN 1 TO FIRST CYLINDER FIRST BLOCK LAST TRACK LOAD DIAGNOSTIC BUFFER LOAD I/O STATUS BLOCK (IOSB)
ISSUE Q I/O REQUEST
SUCCESS
YES
ISOB CONTENTS NORMAL
NO
FAILURE
2
1
DS$CKLOOP
CONTINUE WITH NEXT PART OF TEST 1 (READ PATTERN JUST WRITTEN)
TK-0582

Figure 5-15 ESRAA Test 1, Subtest Ø, Error 12 Flowchart (Sheet 1 of 3)

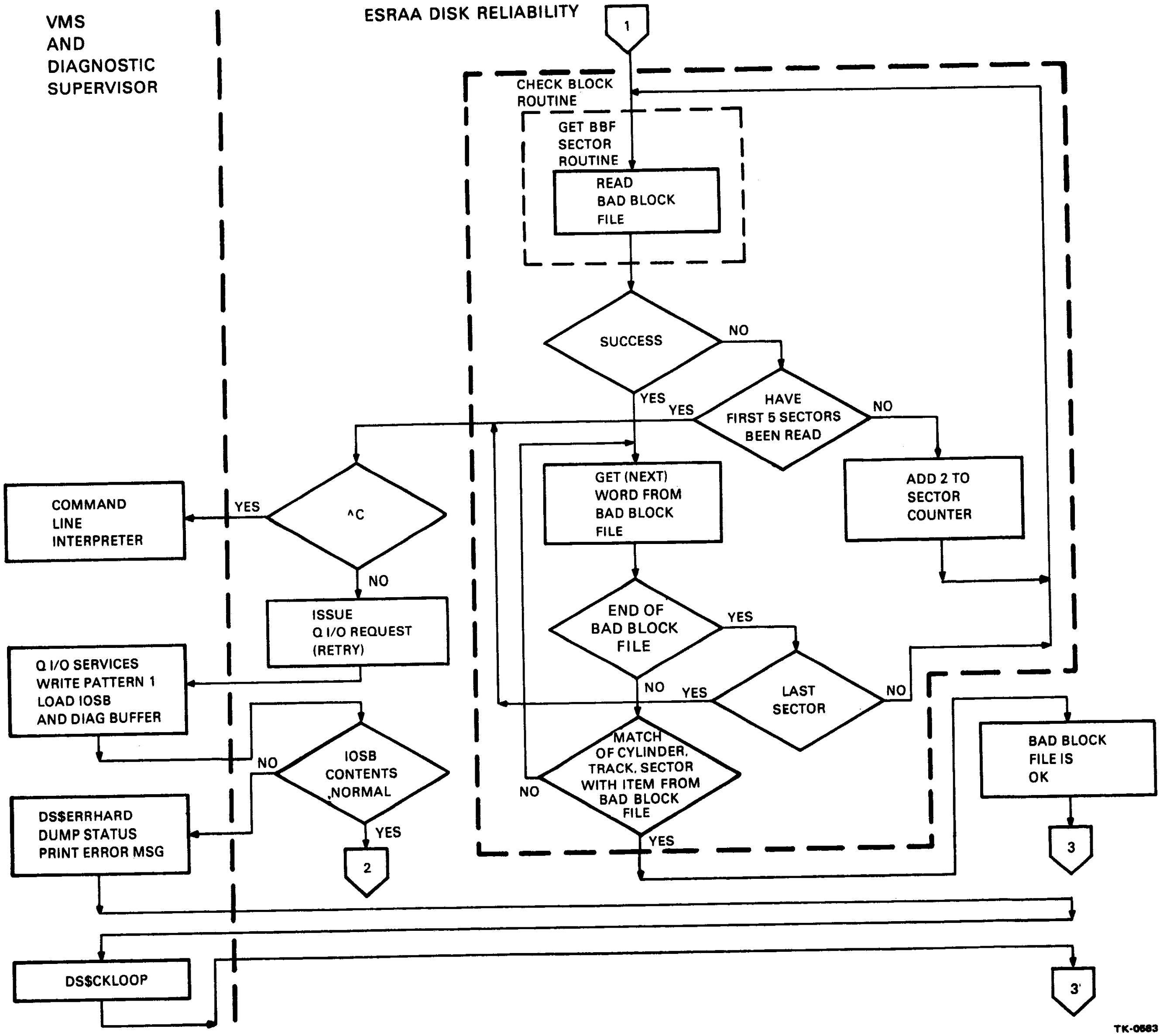

VMS AND DIAGNOSTIC SUPERVISOR
ESRAA DISK RELIABILITY
1
CHECK BLOCK ROUTINE
GET BBF SECTOR ROUTINE
READ BAD BLOCK FILE
SUCCESS
NO
YES
YES
HAVE FIRST 5 SECTORS BEEN READ
NO
GET (NEXT) WORD FROM BAD BLOCK FILE
ADD 2 TO SECTOR COUNTER
^C
YES
COMMAND LINE INTERPRETER
NO
ISSUE Q I/O REQUEST (RETRY)
END OF BAD BLOCK FILE
YES
Q I/O SERVICES WRITE PATTERN 1 LOAD IOSB AND DIAG BUFFER
NO
YES
LAST SECTOR
NO
IOSB CONTENTS NORMAL
NO
MATCH OF CYLINDER, TRACK, SECTOR WITH ITEM FROM BAD BLOCK FILE
NO
BAD BLOCK FILE IS OK
DS$ERRHARD DUMP STATUS PRINT ERROR MSG
YES
2
YES
3
DS$CKLOOP
3'
TK-0683

Figure 5-15 ESRAA Test 1,
Subtest Ø, Error 12 Flowchart
(Sheet 2 of 3)

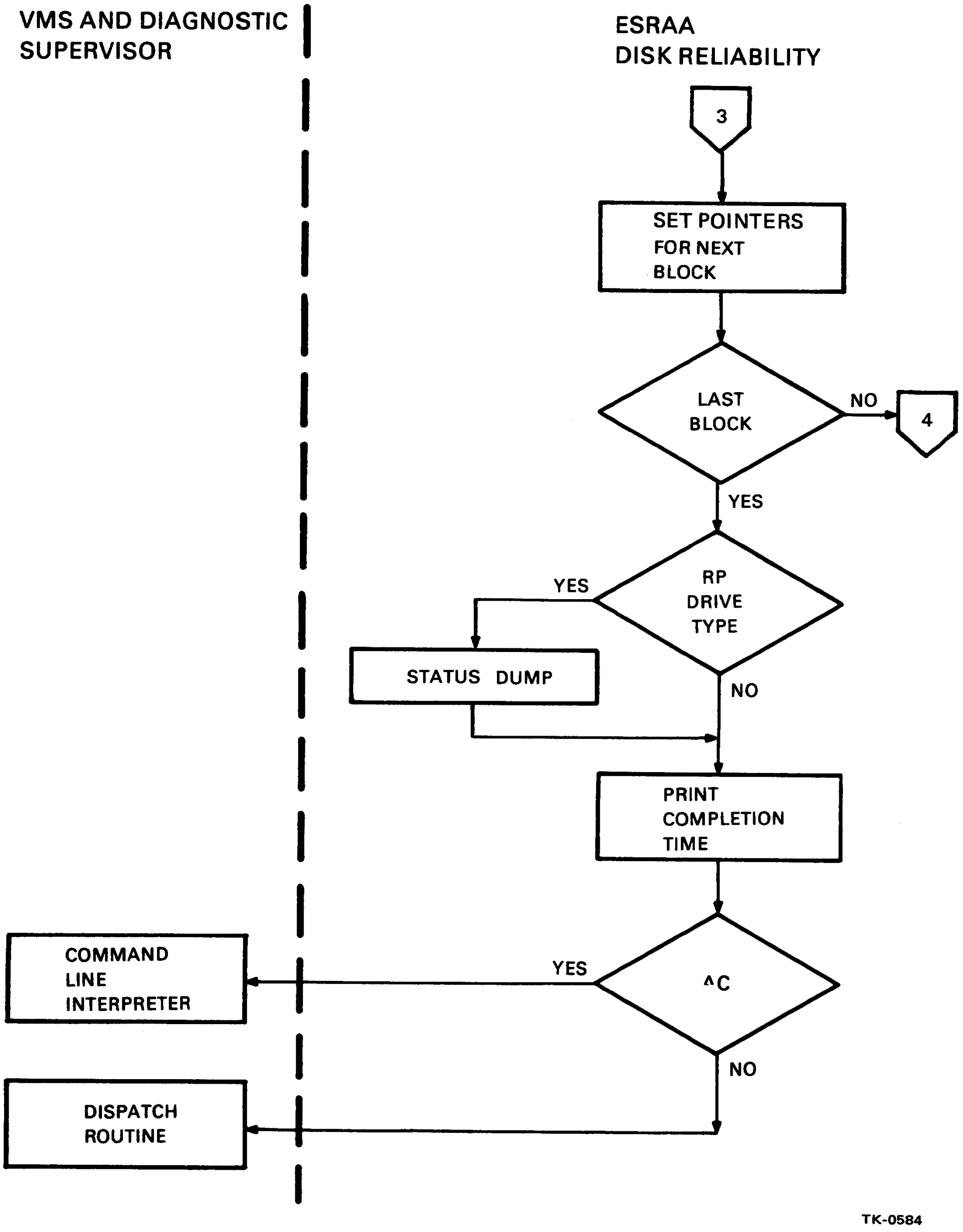


Figure 5-15 ESRAA Test 1, Subtest Ø, Error 12 Flowchart (Sheet 3 of 3)

The first test, when it is called by the dispatch routine in the supervisor, sets up a pointer to the I/O status block and tests various drive commands (drive clear, seek, recalibrate, NOP, offset, and reset). Test 1 then performs an oscillating seek test before beginning the data transfer tests.

In the data transfer tests portion of test 1, the program sets up a write transfer of a data pattern (A57ØA57Ø) to the first block on the disk pack in the drive under test. The data to be written to the disk is loaded into a buffer area in memory. The program then builds an argument list containing the address of the data buffer, the byte count of the data to be transferred, the location of the target block on the disk pack (track, sector, cylinder), the diagnostic buffer address, the event flag number, and the function to be performed. Then, after checking for Control C, the program calls SYS$QIOW.

If the program is being run in the user mode (VMS environment), the call to SYS$QIOW invokes a routine in VMS. If the program is being run in the standalone mode, the SYS$QIOW call invokes a similar routine in the supervisor. SYS$QIOW builds an I/O packet from the parameters passed from the diagnostic program and then (if in VMS) checks the privilege of the calling process (the diagnostic supervisor) through internal data structures. The SYS$QIOW routine then places the packet in a queue for processing by a device driven routine, clears the event flag, and waits for completion of the I/O function (indicated by the setting of the event flag). When the driver completes the I/O function, it examines the controller and drive status registers, formulates a status message that is stored in the I/O packet, and loads the diagnostic buffer with drive and adapter register contents. The I/O packet is then inserted into the I/O post queue and a software interrupt to initiate I/O post processing is requested. The I/O post routine performs final I/O request processing and status posting (IOSB), loads the diagnostic buffer with device and adapter (on MBA or UBA) register contents, and sets the event flag.

With the event flag set, the calling program (ESRAA) resumes control. The diagnostic program then checks the I/O Status Block (IOSB) to determine whether or not the requested function was completed successfully. The IOSB has the format shown in Figure 5-16.

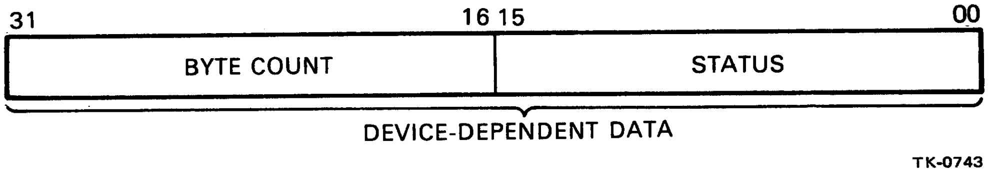
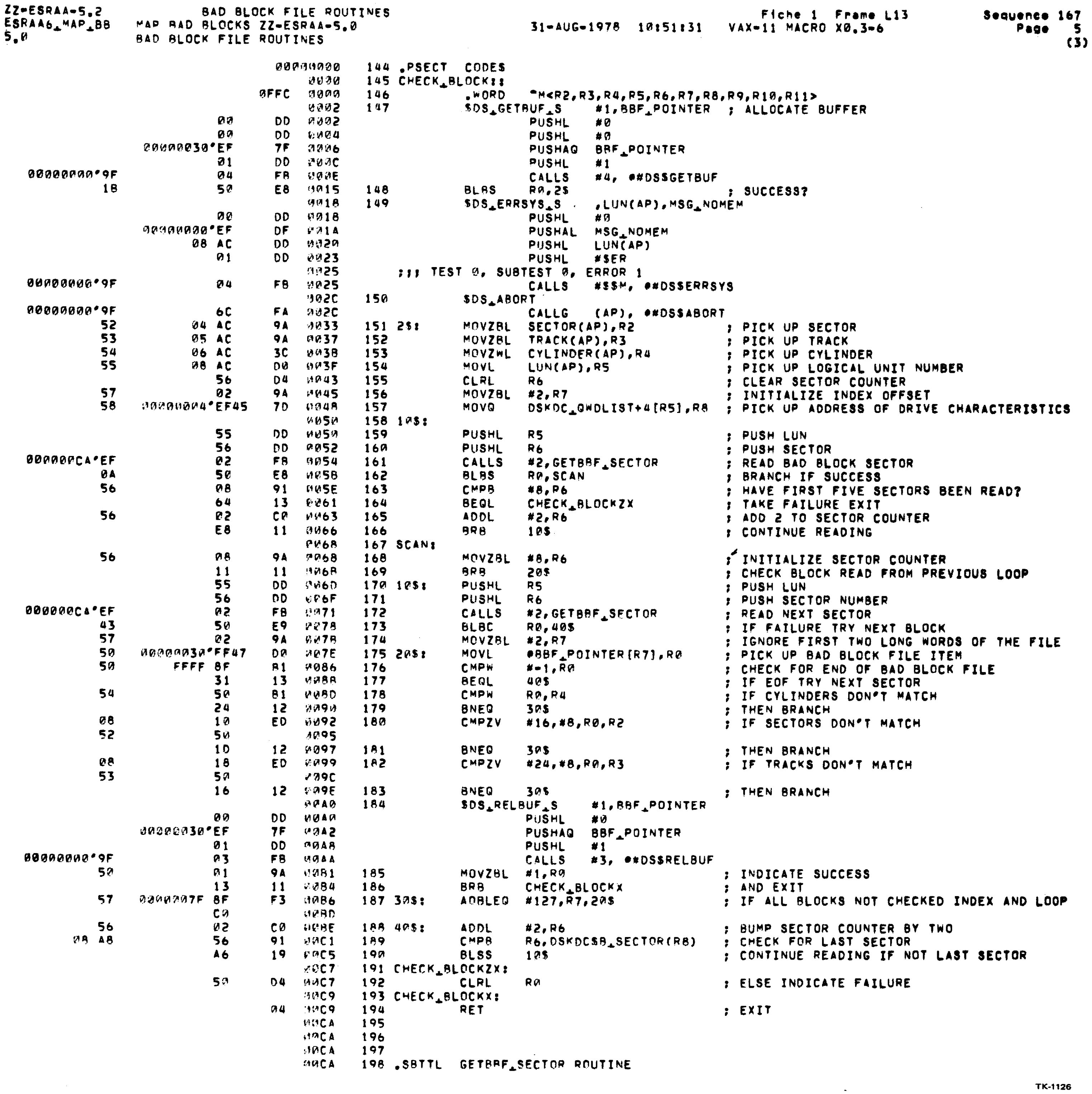


Figure 5-16 I/O Status Block Contents (for disks)

If the low order bit of the first longword of the IOSB is set, indicating success, the program does a branch to the next portion of the test (label 210$), where it tests the loop flag and then reads the data just written to the disk pack.

If the low order bit of the IOSB is not set, the test calls the CHECKBLOCK routine (refer to the code in Figure 5-17), which is located in the load blocks module (module 6) of the disk reliability program (refer to the link map). This routine, in turn, calls the GETBBFSECTOR routine (Figure 5-18) which reads the load block sector on the disk pack. If the load block file cannot be read at all, the routine returns control to test 1, where the queue I/O request is retried. If the load block file has been read successfully, the routine checks through the item in the file to see if the address of the block which cannot be written to is already noted in the load block file. If so, the load block file is OK, and control returns to test 1, which sets up pointer to access the next block on the disk, builds a new queue I/O argument list, and again calls SYS$QIOW, as previously explained, in an attempt to write the pattern into the next block on the disk.

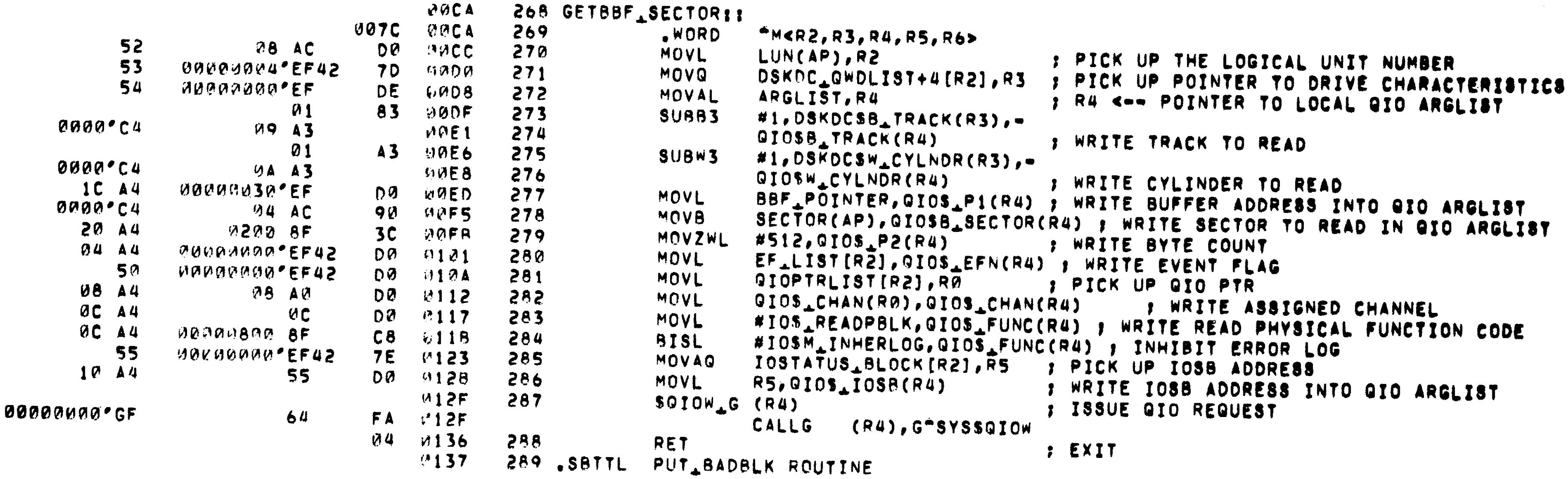
If the failing block address is not listed in the load block file, the CHECKBLOCK routine returns control to test 1 which, in turn, issues a second queue I/O request (after checking for Control C). If the request fails again (IOSB status code = 0), the program calls the DS$ERRHARD routine in the supervisor, which dumps the status block and diagnostic buffer contents and other error information. After detecting a failure of this type, the program checks the loop flag. If the loop flag is set, the program repeats the queue I/O request indefinitely. Otherwise, the program checks each block on the disk pack until it finds one that it can write into successfully, before going on to check the next function (read the block just written, beginning at label 210$).



```
ZZ-ESRAA-5.2          BAD BLOCK FILE ROUTINES                                          Fiche 1  Frame L13         Sequence 167
ESRAA6_MAP_BB     MAP BAD BLOCKS ZZ-ESRAA-5.0                       31-AUG-1978  10:51:31   VAX-11 MACRO X0.3-6           Page   5
5.0               BAD BLOCK FILE ROUTINES                                                                                    (3)

                             00000000   144 .PSECT  CODES
                                 0030   145 CHECK_BLOCK::
                        0FFC     0000   146            .WORD    ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
                                 0002   147            $DS_GETBUF_S     #1,BBF_POINTER  ; ALLOCATE BUFFER
                   00      DD    0002                           PUSHL    #0
                   00      DD    0004                           PUSHL    #0
          00000030'EF      7F    0006                           PUSHAQ   BBF_POINTER
                   01      DD    000C                           PUSHL    #1
 00000000'9F       04      FB    000E                           CALLS    #4, @#DS$GETBUF
          18       50      E8    0015   148        BLBS     R0,2$                          ; SUCCESS?
                                 0018   149        $DS_ERRSYS_S      ,LUN(AP),MSG_NOMEM
                   00      DD    0018                           PUSHL    #0
          00000000'EF      DF    001A                           PUSHAL   MSG_NOMEM
                08 AC      DD    0020                           PUSHL    LUN(AP)
                   01      DD    0023                           PUSHL    #$ER
                                 0025       ;;; TEST 0, SUBTEST 0, ERROR 1
 00000000'9F       04      FB    0025                           CALLS    #$$M, @#DS$ERRSYS
                                 002C   150        $DS_ABORT
 00000000'9F       6C      FA    002C                           CALLG    (AP), @#DS$ABORT
          52    04 AC      9A    0033   151 2$:    MOVZBL   SECTOR(AP),R2              ; PICK UP SECTOR
          53    05 AC      9A    0037   152        MOVZBL   TRACK(AP),R3               ; PICK UP TRACK
          54    06 AC      3C    003B   153        MOVZWL   CYLINDER(AP),R4            ; PICK UP CYLINDER
          55    08 AC      D0    003F   154        MOVL     LUN(AP),R5                 ; PICK UP LOGICAL UNIT NUMBER
                   56      D4    0043   155        CLRL     R6                         ; CLEAR SECTOR COUNTER
          57       02      9A    0045   156        MOVZBL   #2,R7                      ; INITIALIZE INDEX OFFSET
          58  00000004'EF45    7D    0048   157        MOVQ     DSKDC_QWDLIST+4[R5],R8     ; PICK UP ADDRESS OF DRIVE CHARACTERISTICS
                                 0050   158 10$:
                   55      DD    0050   159        PUSHL    R5                         ; PUSH LUN
                   56      DD    0052   160        PUSHL    R6                         ; PUSH SECTOR
 000000CA'EF       02      FB    0054   161        CALLS    #2,GETBBF_SECTOR           ; READ BAD BLOCK SECTOR
          0A       50      E8    0058   162        BLBS     R0,SCAN                    ; BRANCH IF SUCCESS
          56       08      91    005E   163        CMPB     #8,R6                      ; HAVE FIRST FIVE SECTORS BEEN READ?
                   64      13    0061   164        BEQL     CHECK_BLOCKZX              ; TAKE FAILURE EXIT
          56       02      C0    0063   165        ADDL     #2,R6                      ; ADD 2 TO SECTOR COUNTER
                   E8      11    0066   166        BRB      10$                        ; CONTINUE READING
                                 0068   167 SCAN:
          56       08      9A    0068   168        MOVZBL   #8,R6                      ; INITIALIZE SECTOR COUNTER
                   11      11    006B   169        BRB      20$                        ; CHECK BLOCK READ FROM PREVIOUS LOOP
                   55      DD    006D   170 10$:   PUSHL    R5                         ; PUSH LUN
                   56      DD    006F   171        PUSHL    R6                         ; PUSH SECTOR NUMBER
 000000CA'EF       02      FB    0071   172        CALLS    #2,GETBBF_SECTOR           ; READ NEXT SECTOR
          43       50      E9    0078   173        BLBC     R0,40$                     ; IF FAILURE TRY NEXT BLOCK
          57       02      9A    007B   174        MOVZBL   #2,R7                      ; IGNORE FIRST TWO LONG WORDS OF THE FILE
          50  00000030'FF47    D0    007E   175 20$:   MOVL     @BBF_POINTER[R7],R0        ; PICK UP BAD BLOCK FILE ITEM
          50     FFFF 8F   B1    0086   176        CMPW     #-1,R0                     ; CHECK FOR END OF BAD BLOCK FILE
                   31      13    008B   177        BEQL     40$                        ; IF EOF TRY NEXT SECTOR
          54       50      B1    008D   178        CMPW     R0,R4                      ; IF CYLINDERS DON'T MATCH
                   24      12    0090   179        BNEQ     30$                        ; THEN BRANCH
          08       10      ED    0092   180        CMPZV    #16,#8,R0,R2               ; IF SECTORS DON'T MATCH
          52       50            0095
                   1D      12    0097   181        BNEQ     30$                        ; THEN BRANCH
          08       18      ED    0099   182        CMPZV    #24,#8,R0,R3               ; IF TRACKS DON'T MATCH
          53       50            009C
                   16      12    009E   183        BNEQ     30$                        ; THEN BRANCH
                                 00A0   184        $DS_RELBUF_S     #1,BBF_POINTER
                   00      DD    00A0                           PUSHL    #0
          00000030'EF      7F    00A2                           PUSHAQ   BBF_POINTER
                   01      DD    00A8                           PUSHL    #1
 00000000'9F       03      FB    00AA                           CALLS    #3, @#DS$RELBUF
          50       01      9A    00B1   185        MOVZBL   #1,R0                      ; INDICATE SUCCESS
                   13      11    00B4   186        BRB      CHECK_BLOCKX               ; AND EXIT
          57   0000007F 8F     F3    00B6   187 30$:   AOBLEQ   #127,R7,20$                ; IF ALL BLOCKS NOT CHECKED INDEX AND LOOP
                   C0            00BD
          56       02      C0    00BE   188 40$:   ADDL     #2,R6                      ; BUMP SECTOR COUNTER BY TWO
       08 A8       56      91    00C1   189        CMPB     R6,DSKDC$B_SECTOR(R8)      ; CHECK FOR LAST SECTOR
                   A6      19    00C5   190        BLSS     10$                        ; CONTINUE READING IF NOT LAST SECTOR
                                 00C7   191 CHECK_BLOCKZX:
                   50      D4    00C7   192        CLRL     R0                         ; ELSE INDICATE FAILURE
                                 00C9   193 CHECK_BLOCKX:
                           04    00C9   194        RET                                 ; EXIT
                                 00CA   195
                                 00CA   196
                                 00CA   197
                                 00CA   198 .SBTTL  GETBBF_SECTOR ROUTINE
```

TK-1126

Figure 5-17 CHECKBLOCK Routine Code



```
                                     00CA       GETBBF_SECTOR::
                               007C  00CA                   .WORD    ^M<R2,R3,R4,R5,R6>
      52          08 AC          D0  00CC                   MOVL     LUN(AP),R2                 ; PICK UP THE LOGICAL UNIT NUMBER
      53   00000004'EF42         7D  00D0                   MOVQ     DSKDC_QWDLIST+4[R2],R3     ; PICK UP POINTER TO DRIVE CHARACTERISTICS
      54   00000000'EF           DE  00D8                   MOVAL    ARGLIST,R4                 ; R4 <-- POINTER TO LOCAL QIO ARGLIST
                          01     83  00DF                   SUBB3    #1,DSKDC$B_TRACK(R3),-
 0000'C4               09 A3         00E1                            QIO$B_TRACK(R4)            ; WRITE TRACK TO READ
                          01     A3  00E6                   SUBW3    #1,DSKDC$W_CYLNDR(R3),-
 0000'C4               0A A3         00E8                            QIO$W_CYLNDR(R4)           ; WRITE CYLINDER TO READ
   1C A4   00000030'EF           D0  00ED                   MOVL     BBF_POINTER,QIO$_P1(R4)    ; WRITE BUFFER ADDRESS INTO QIO ARGLIST
 0000'C4               04 AC     90  00F5                   MOVB     SECTOR(AP),QIO$B_SECTOR(R4) ; WRITE SECTOR TO READ IN QIO ARGLIST
   20 A4            0200 8F      3C  00FB                   MOVZWL   #512,QIO$_P2(R4)           ; WRITE BYTE COUNT
   04 A4   00000000'EF42         D0  0101                   MOVL     EF_LIST[R2],QIO$_EFN(R4)   ; WRITE EVENT FLAG
      50   00000000'EF42         D0  010A                   MOVL     QIOPTRLIST[R2],R0          ; PICK UP QIO PTR
   08 A4               08 A0     D0  0112                   MOVL     QIO$_CHAN(R0),QIO$_CHAN(R4)      ; WRITE ASSIGNED CHANNEL
   0C A4                  0C     D0  0117                   MOVL     #IO$_READPBLK,QIO$_FUNC(R4) ; WRITE READ PHYSICAL FUNCTION CODE
   0C A4   00000800 8F           C8  011B                   BISL     #IO$M_INHERLOG,QIO$_FUNC(R4) ; INHIBIT ERROR LOG
      55   00000000'EF42         7E  0123                   MOVAQ    IOSTATUS_BLOCK[R2],R5      ; PICK UP IOSB ADDRESS
   10 A4                  55     D0  0128                   MOVL     R5,QIO$_IOSB(R4)           ; WRITE IOSB ADDRESS INTO QIO ARGLIST
                                     012F                   $QIOW_G  (R4)                       ; ISSUE QIO REQUEST
00000000'GF               64     FA  012F                            CALLG   (R4),G^SYS$QIOW
                                 04  0136                   RET                                 ; EXIT
                                     0137       .SBTTL      PUT_BADBLK ROUTINE
```

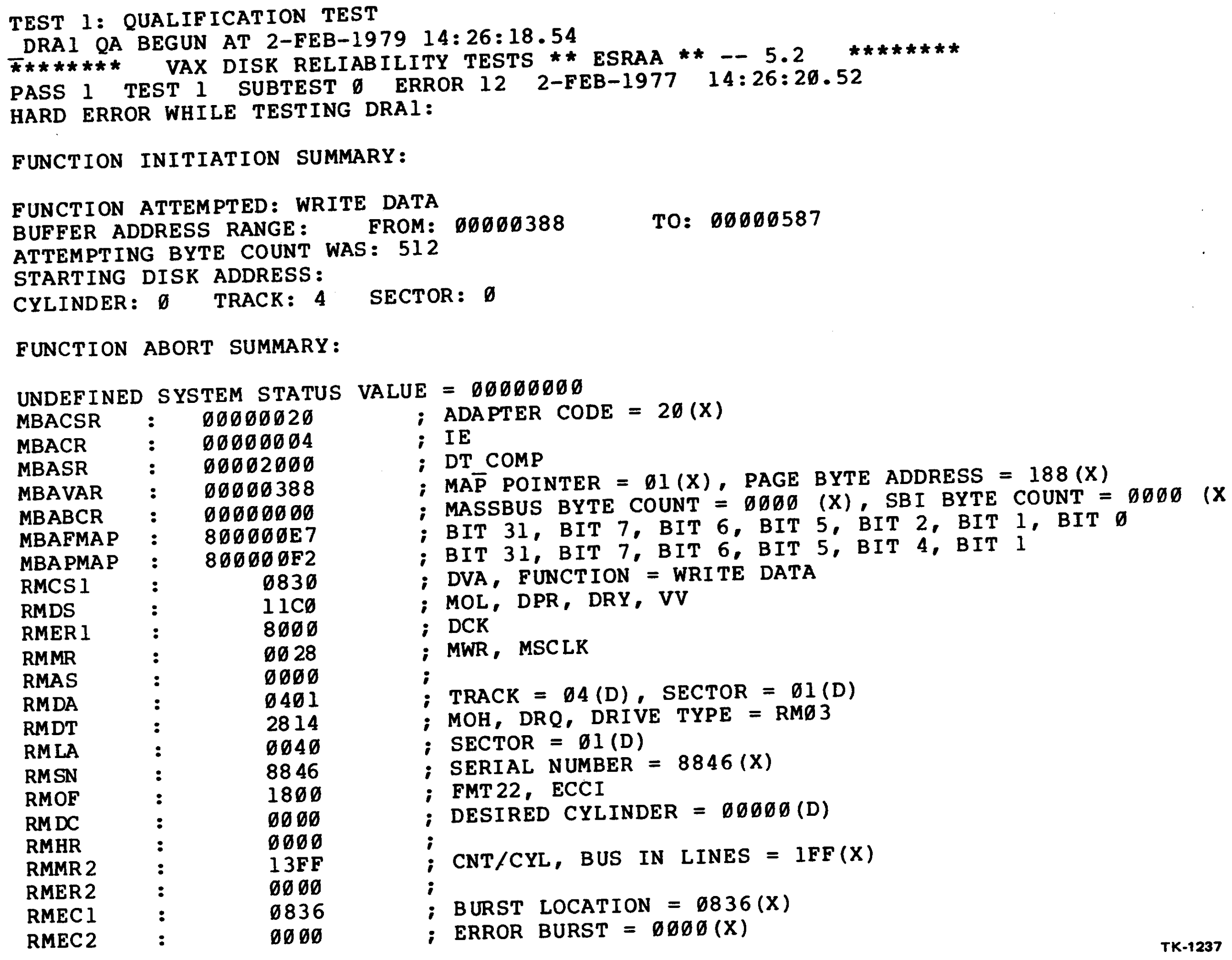
TK-1125

Figure 5-18 GETBBFSECTOR
Routine Code

5.8.2 Disk Reliability Program Sample Error Message

Test 1, Subtest Ø, Error 12 of the disk reliability program identifies bad blocks, on the disk pack under test, that are not entered in the bad block file. The message shown in Figure 5-19 identifies the failing test, subtest, and error numbers. The message also includes a dump of the channel registers (MBA registers in this case) and the RMØ3 registers.

Notice that bits 12 and 15 of the RMER1 register are set, indicating a data check error. The starting cylinder is Ø. The bad block is located in sector 1, on track 4, as shown by the contents of the RMDA registers.



```
TEST 1: QUALIFICATION TEST
_DRA1 QA BEGUN AT 2-FEB-1979 14:26:18.54
********   VAX DISK RELIABILITY TESTS ** ESRAA ** -- 5.2    ********
PASS 1  TEST 1  SUBTEST Ø  ERROR 12  2-FEB-1977  14:26:2Ø.52
HARD ERROR WHILE TESTING DRA1:

FUNCTION INITIATION SUMMARY:

FUNCTION ATTEMPTED: WRITE DATA
BUFFER ADDRESS RANGE:    FROM: ØØØØØ388       TO: ØØØØØ587
ATTEMPTING BYTE COUNT WAS: 512
STARTING DISK ADDRESS:
CYLINDER: Ø   TRACK: 4   SECTOR: Ø

FUNCTION ABORT SUMMARY:

UNDEFINED SYSTEM STATUS VALUE = ØØØØØØØØ
MBACSR    :   ØØØØØØ2Ø       ; ADAPTER CODE = 2Ø(X)
MBACR     :   ØØØØØØØ4       ; IE
MBASR     :   ØØØØ2ØØØ       ; DT_COMP
MBAVAR    :   ØØØØØ388       ; MAP POINTER = Ø1(X), PAGE BYTE ADDRESS = 188(X)
MBABCR    :   ØØØØØØØØ       ; MASSBUS BYTE COUNT = ØØØØ (X), SBI BYTE COUNT = ØØØØ (X
MBAFMAP   :   8ØØØØØE7       ; BIT 31, BIT 7, BIT 6, BIT 5, BIT 2, BIT 1, BIT Ø
MBAPMAP   :   8ØØØØØF2       ; BIT 31, BIT 7, BIT 6, BIT 5, BIT 4, BIT 1
RMCS1     :       Ø83Ø       ; DVA, FUNCTION = WRITE DATA
RMDS      :       11CØ       ; MOL, DPR, DRY, VV
RMER1     :       8ØØØ       ; DCK
RMMR      :       ØØ28       ; MWR, MSCLK
RMAS      :       ØØØØ       ;
RMDA      :       Ø4Ø1       ; TRACK = Ø4(D), SECTOR = Ø1(D)
RMDT      :       2814       ; MOH, DRQ, DRIVE TYPE = RMØ3
RMLA      :       ØØ4Ø       ; SECTOR = Ø1(D)
RMSN      :       8846       ; SERIAL NUMBER = 8846(X)
RMOF      :       18ØØ       ; FMT22, ECCI
RMDC      :       ØØØØ       ; DESIRED CYLINDER = ØØØØØ(D)
RMHR      :       ØØØØ       ;
RMMR2     :       13FF       ; CNT/CYL, BUS IN LINES = 1FF(X)
RMER2     :       ØØØØ       ;
RMEC1     :       Ø836       ; BURST LOCATION = Ø836(X)
RMEC2     :       ØØØØ       ; ERROR BURST = ØØØØ(X)
```

TK-1237

Figure 5-19 ESRAA
Sample Error Listing

CHAPTER 6
CPU CLUSTER EXERCISER PACKAGE

The CPU cluster exerciser package consists of three separate programs (ESKAX, ESKAY, ESKAZ). Two modules, the control module and the common instruction test services module (CITS), are common to all three programs. ESKAX, the first program, is the quick verify portion of the cluster exerciser package. This program includes the compatibility mode entry and exit test, the first part done test, and the SBI exerciser. The second program, ESKAY, contains the timer and clock tests and the native mode instruction set tests. ESKAZ contains the memory management test and the compatibility mode instruction set tests. Figure 6-1 is a map showing the memory allocations of the three programs.

The cluster exerciser diagnostics will handle three classes of errors, providing three corresponding types of error messages: unexpected exceptions or interrupts; test failures; and safe return halts (resulting from fatal errors). The code for the cluster exerciser programs is not as easy to follow as the code for other diagnostic programs. However, the error messages which the programs generate are detailed and, for the most part, self-explanatory. The operator should understand the general structure of each test and the error message formats in order to use all of the facilities provided by the cluster exerciser programs.

6.1 CONTROL MODULE

The control module in the cluster exerciser programs serves as the interface between the programs and the diagnostic supervisor. The module performs the following functions:

^ Program initialization and clean up

^ Execution of all tests twice in one pass

^ Print out of a module summary message at the end of each pass, if errors exist

^ Initialization and reinitialization of pertinent control variables

^ Set up of all vectors for interrupt and exception handling

^ Proper fielding of all exceptions and interrupts (expected and unexpected).

When the control module detects an unexpected interrupt or exception, it prints out an error message as shown in Example 6-1.

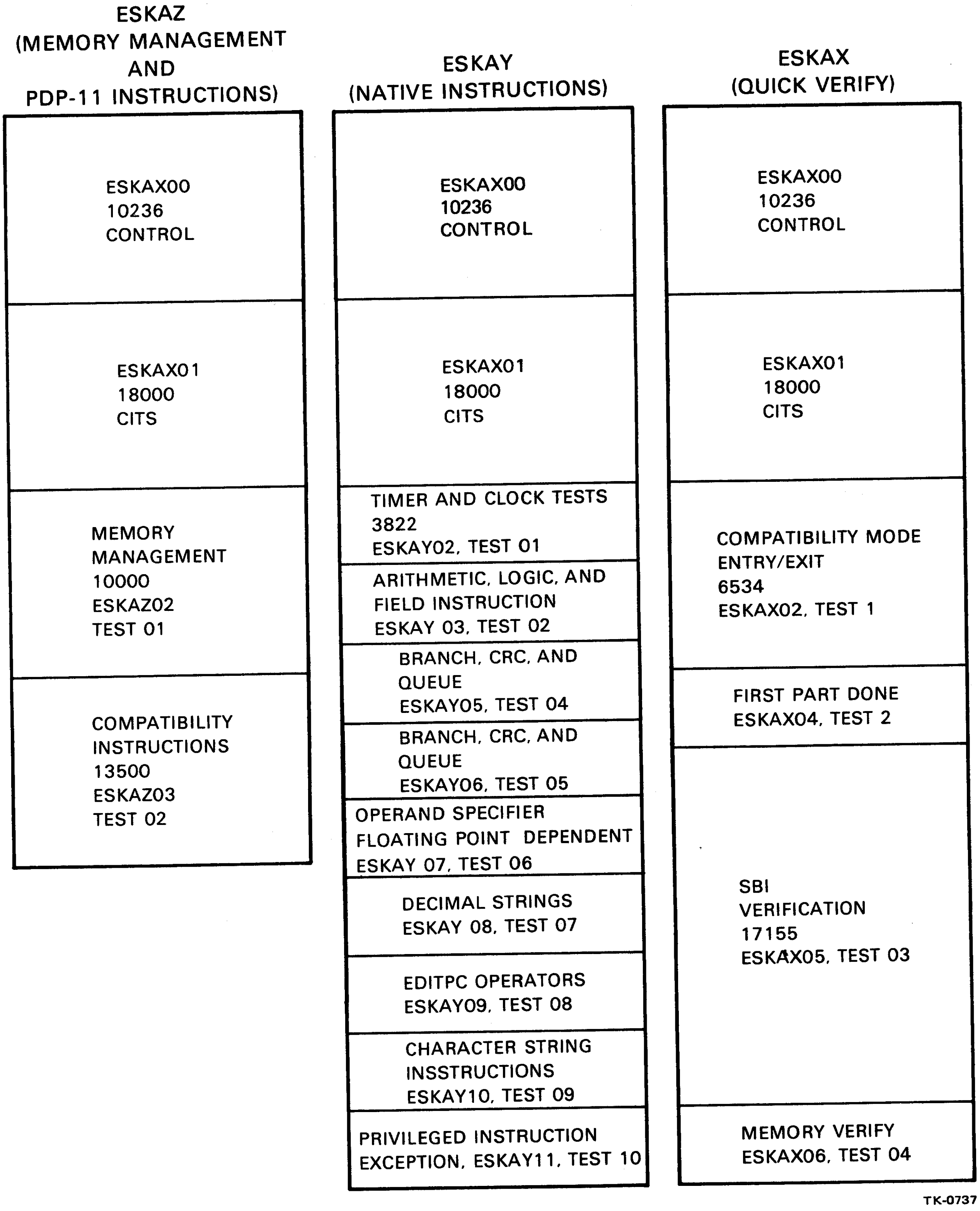


Figure 6-1 CPU Cluster Exerciser Package Memory Allocation

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 6  SUBTEST 4  ERROR 1  . . . . .

HARD ERROR WHILE TESTING CPU: EXCEPTION SERVICE ROUTINE

UNEXPECTED EXCEPTION

ERROR# ØØØØØØØ1

VECTOR# ØØØØØØ3Ø

SUBTYPE# ØØØØØØØ6

PSL 83CØØØØØ

PC ØØØØ11ØD
```

Example 6-1 Unexpected Exception Error Message

Refer to Chapter 2 of the VAX-11 KA78Ø Central Processor Technical Description for a discussion of vectors and subtypes. When the machine check vector is asserted, the exception handler attempts to log out relevant status registers on the stack before pushing two longword parameters (summary and length) on the stack (Table 6-1). In addition, the subtypes for vector 4 (machine check) are listed in Table 6-1.

Table 6-1 Summary Parameter, Length Parameter for Vector 4

Summary Parameter

Subtype	Byte Ø
ØØ	CP Read Timeout/SBI Error Confirmation Fault
Ø2	CP TBUF Parity Error Fault
Ø3	CP Cache Parity Error Fault
Ø5	CP Read Data Substitute Fault
ØA	Instruction Buffer TBUF Parity Error Fault
ØC	Instruction Buffer Read Data Substitute Fault
ØA	Instruction Buffer TBUF Parity Error Fault
ØC	Instruction Buffer Read Data Substitute Fault
ØD	IB Read Timeout/SBI Error Confirmation Fault
ØF	IB Cache Parity Error Fault

Table 6-1 Summary Parameter, Length Parameter for Vector 4 (Cont)

Summary Parameter	
Subtype	Byte Ø
FØ	CP Read Timeout/SBI Error Confirmation Abort
F1	CS Parity Error Abort
F2	CP TBUF Parity Error Abort
F3	CP Cache Parity Error Abort
F5	CP Read Data Substitute Abort
F6	CP (Not Supposed To Be Here) Abort
	Byte 1
	This byte will be a nonzero value if a CP timeout or CP error confirmation interrupt is pending.
	Bytes 2 & 3
	These two bytes must be zero.
Length Parameter	
	Byte Ø
	The number of bytes logged out are exclusive of this parameter.
	Byte 1--3
	These three bytes must be zero.

When an unexpected interrupt or exception occurs, information is pushed on the stack by the exception handler as shown in Table 6-2.

Table 6-2 Information Pushed on the Stack by the Exception Handler

Mnemonic	Meaning	ID Bus Address
SP:	Length Parameter	
	Summary Parameter	
CES	CPU Error & Status	ØC
	Trapped UPC	
	Virtual Address/ Virtual Instruction Buffer Address	
D	Interface Between Data Paths and Memory	Ø8 (Bytes 1 & 3)
TBERØ	Translation Buffer Error Register Ø	12
TBER1	Translation Buffer Error Register 1	13
TIME.ADDR		1A
PARITY	Cache Parity Register	1E
SBI.ERR	SBI Error Register	19
	PC	
	PSL	

Note that information on the stack is not saved by the exception handler. The EIH module must be breakpointed before this data is accessed.

6.2 COMMON INSTRUCTION TEST SERVICES MODULE (CITS)

This module consists of a group of software routines that implement a table-driven test method for a majority of the VAX-11 instruction set. CITS interprets the contents of a specially coded test table and executes tests of VAX-11 instructions. CITS is also used for tests of the first part done function and memory management. A copy of each of these test instructions is coded in register deferred mode (RN). Before the test instruction is executed, the test data is placed somewhere in memory, and the registers are loaded with the addresses of that test data. After the test instruction is executed, the contents of the registers and the contents of the test data area of memory are checked.

There are four main routines in CITS that do the work of executing tests: CITS_DECODE, CITS_SETUP, CITS_EXECUTE, and CITS_CHECK.

6.2.1 CITS_DECODE

This routine decodes one test table entry, in a table of cases, and generates directions for the other three routines to use. These directions are lists of addresses and other variables placed in the parameter blocks of the CITS data area.

6.2.2 CITS_SETUP

CITS_SETUP moves the test data from the common data pool into the operand buffer. The operand buffer is the location of the data referenced during execution of the test instruction. The locations to be used for destination data are filled with a standard background pattern, hexadecimal A5, in each byte. Also, each operand, whether source or destination, is preceded and followed by a longword of the background pattern. CITS_SETUP loads registers RØ--R6 with the operand addresses to be used by the test instruction. The initial PC and PSL calculated by CITS_DECODE are pushed on the stack by CITS_SETUP along with a return address. The return address points to a routine to save the result PSL and registers.

6.2.3 CITS_EXECUTE

CITS_EXECUTE enables the exerciser's exception handler to react properly for the current test. It passes the address of a CITS unexpected exception handler and enables validation of the exception of trace trap being tested, if any. CITS_EXECUTE then executes an REI to start the test. A NOP instruction precedes the REI and can be used for scope sync if the microbreak address is set up correctly from the console. When the test instruction finishes, the test subroutine returns to a result-saving routine. The PSL and registers RØ--R6 are saved in the execution parameter block, as are the contents of the exception handler Interface Data Block (IDB). Also saved is an indication of whether the instruction branched, if it is a branch instruction.

6.2.4 CITS_CHECK

CITS_CHECK checks the results of instruction execution, and also checks the source operands and background pattern. It uses the directions and addresses put into its parameter block by

CITS_DECODE to control checking. It checks branches, the PSL, exceptions (whether an exception happened and at the right PC), registers RØ--R6, and memory data. When checking memory data, CITS_CHECK also checks the longword before and after each operand to make sure that the background pattern has not been disturbed. CITS_CHECK keeps a list of all errors found during one test case. This complete list will be typed out when the test module using CITS makes the $DS_ERRHARD call to the diagnostic supervisor.

6.2.5 CITS_SUBTEST

CITS_SUBTEST is a common subtest control routine that is used by most of the tests that call CITS. It processes a complete test table, calling the preceding four CITS routines in the proper order and calling the supervisor error reporter when necessary.

6.2.6 CITS Error Messages

6.2.6.1 **Message Heading** -- A standard diagnostic supervisor heading is typed (by the supervisor). That is followed by an extended error printout that supplies the test, subtest, and error numbers; the test case number; the op code of the failing instruction; addresses referenced; operand data; etc. Refer to Paragraph 6.4.2 for examples and detailed interpretation.

6.2.6.2 **CITS Subtest Troubleshooting Features -- SCOPE SYNC** -- CITS_EXECUTE executes a NOP instruction just before the REI to the test instruction. Putting the microaddress of 8E into ID Bus register 21 will cause a sync pulse to be generated on the microsequencer board (M8235) each time a NOP is executed.

To loop on a failure with SCOPE SYNC, perform the following steps:

```
>>> D /ID 21 8E

>>> START 1ØØØØ

          (DIAGNOSTIC SUPERVISOR STARTUP)

DS> SET IE1,LOOPD

DS> START /TEST: N        (WHERE 'N' IS FAILING TEST NUMBER)

          ETC.
```

^ **Halt Before the Failing Test Case** -- At the beginning of the test-case executing loop, the case number (CITS_CASE) is always incremented and compared with the content of CASE_HALT. If these are equal, a HALT is executed. This feature allows the operator to stop before execution of a particular case, in order to examine registers, e.g., deposit the hex case number into CASE_HALT. Run the test until the halt is executed. (If needed, use CONTINUE until you get to the right subtest.) Then either set a breakpoint or deposit a byte of zero (a

HALT) in CITS_SYNC and type CONTINUE to get to that HALT. You have now stopped just before the REI to the test instruction of interest. Use the console to set up whatever operation you wish to perform and continue as desired.

Figure 6-2 shows the sequence of events followed by CITS in the execution of ESKAYØ3, test 2, the arithmetic, logic, and field instruction test module.

6.2.6.3 Unexpected Exceptions in CITS -- If an unexpected exception occurs during a test, CITS will print a header containing the case number and the error information from the exerciser exception handler. This printout only occurs while CITS is handling unexpected exceptions, i.e., only during the execution of the five instructions before the test instruction and approximately through the three instructions after it (Example 6-2). If an exception occurs outside of that set of instructions, then the error typeout is not handled by CITS and will not have a case number heading.

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 2  SUBTEST 1  ERROR 25  Ø

HARD ERROR WHILE TESTING KAØ: INSTRUCTION TEST ERROR

? ERROR IN TEST CASE NUMBER: 25


UNEXPECTED EXCEPTION

ERROR# ØØØØØØØ1

VECTOR# ØØØØØØØ4

SUBTYPE# ØØØØØØØØ

PSL ØØ1FØØEØ

PC ØØØØØØØ3
```

Example 6-2 Unexpected Exception in CITS, Error Message

Refer to Example 6-1 (Paragraph 6.1) for an explanation of the unexpected exception error message format.

6.2.6.4 Result Register Errors -- If the contents of any of the registers RØ--R6 are not as expected, CITS prints out initial, expected, and actual values, as shown in Example 6-3.

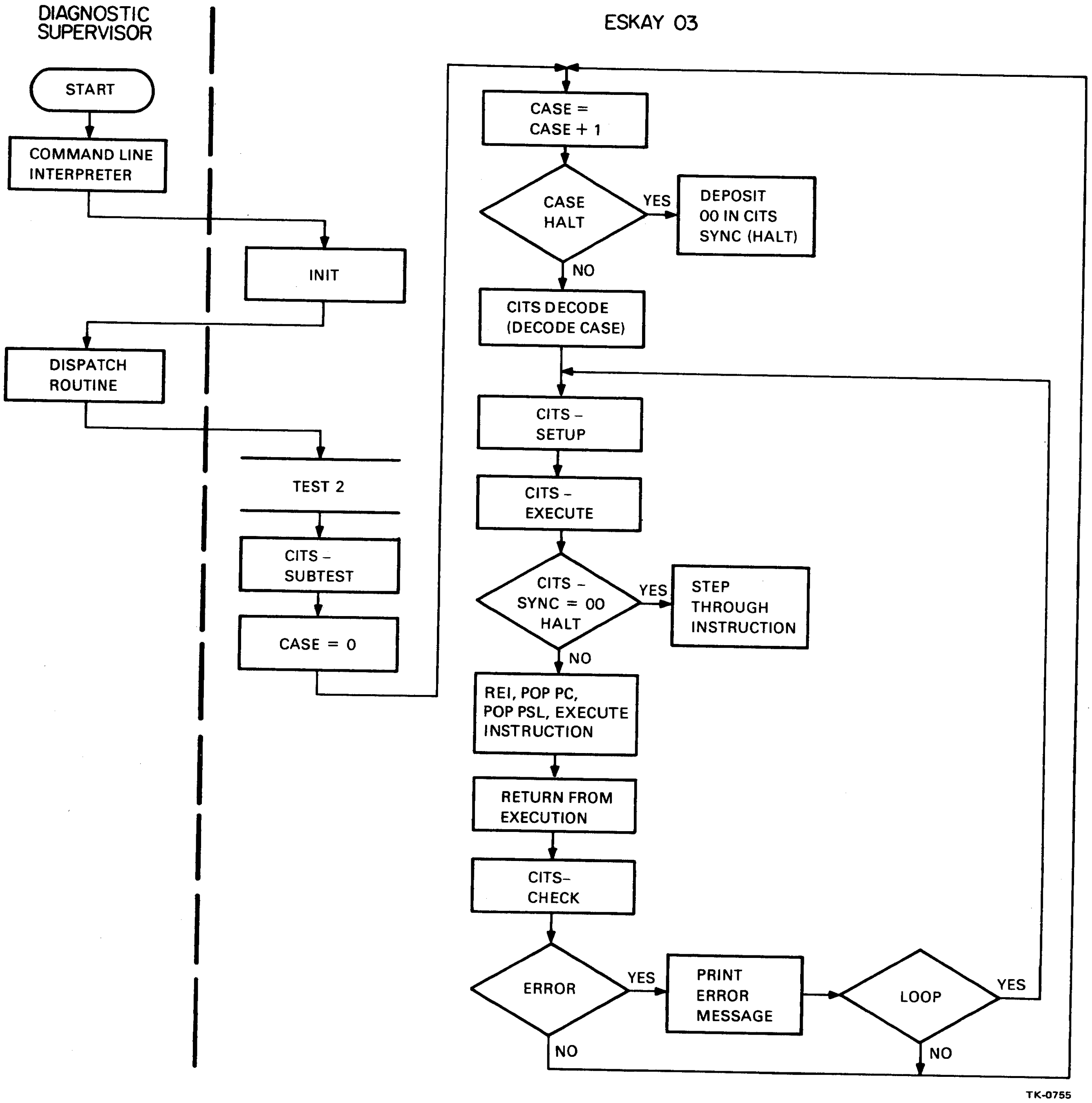


Figure 6-2 Execution of a Test Case in ESKAYØ3

```
********   CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 2  SUBTEST 4  ERROR 17  Ø

HARD ERROR WHILE TESTING KAØ: INSTRUCTION TEST ERROR

? ERROR IN TEST CASE NUMBER: 17

? REGISTER CONTENTS ERROR

         INITIAL    EXPECTED    ACTUAL

RØ       ØØØØ54Ø4   ØØØØ38ØØ    ØØØØ388Ø

R1       ØØØØ5414   ØØØØ8ØØØ    ØØØØ8ØØØ

R2       ØØØØ5418   ØØØØØØØØ    ØØØØØØØØ

R3       ØØØØØØØØ   ØØØØ5476    ØØØØ5476

R4       ØØØØØØØØ   ØØØØØØØØ    ØØØØØØØØ

R5       ØØØØØØØØ   ØØØØØØØØ    ØØØØØØØØ

R6       ØØØØØØØØ   ØØØØØØØØ    ØØØØØØØØ
```

Example 6-3 Result Register Errors

Initial data shows the values loaded into the registers at the start of the instruction.

6.2.6.5 Leading or Trailing Background Errors -- If the longword before or the longword after an operand is changed during execution, CITS reports the error. Leading means the longword before the data (lower address than the data). Hexadecimal A5A5A5A5 is the standard background pattern.

6.2.6.6 Data Errors -- When CITS detects a data error, part of the error typeout is an operand number. That is, a number in the range 1 to 6 corresponding to the left-to-right order of the operands for the instruction. For example, in a MOVL instruction the source longword will be called operand 1 and the destination longword operand 2.

If the incorrect operand is not of a writable or modifiable access type, then the error message includes the statement: read-only operand overwritten.

If the incorrect operand is writable or modifiable, then the error message includes the statement: incorrect result (Example 6-4).

```
********    CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 2  SUBTEST 7  ERROR 1Ø  Ø

HARD ERROR WHILE TESTING KAØ: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 1Ø

? INCORRECT RESULT OPERAND 2

EXPECTED          ACTUAL

CØØØØØØØ          4ØØØØØØØ
```

Example 6-4 CITS Detects a Longword Data Error

Example 6-4 shows incorrect longword data. For word and byte data errors, the format is the same except that a word is typed as four hex digits, and a byte as two hex digits. In a quadword or double-floating word typeout, the lowest addressed longword is the first line of data typed. That is the longword containing the sign and the exponent for the double-floating case. (In the quadword case, the sign is in the longword typed on the second line of data, Example 6-5.)

```
********    CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 2  SUBTEST 1  ERROR 91  Ø

HARD ERROR WHILE TESTING KAØ: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 91

? INCORRECT RESULT, OPERAND 2

EXPECTED          ACTUAL

ØØØØØØØØ          99674ØD6

ØØØØØØØØ          86A2E99E
```

Example 6-5 CITS Detects a Quadword Data Error

Errors in string data (character string, packed decimal string, etc.) are displayed in Example 6-6.

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 2  SUBTEST 1  ERROR 44  Ø

HARD ERROR WHILE TESTING KAØ: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 44

? INCORRECT RESULT, OPERAND 4

EXPECTED        ACTUAL

38              ...A5A5*39*4E39...

Ø BYTES FROM START OF STRING
```

Example 6-6 CITS Detects a String Data Error

Each byte is typed as two hexadecimal digits. The expected data only shows the good value of the byte that did not compare. The actual data shows five bytes of the result string. The beginning of the string is to the left, the end is to the right. The left hand two bytes (four digits) are good result data; the byte between asterisks (*) is the one that failed to compare; and the right two bytes are the start of the rest of the (uncompared) string. The last line tells how far from the beginning of the string the bad byte is.

6.2.6.7 PSL Errors -- Result PSL errors are typically wrong condition codes. The condition codes are the right-hand hex digit of the PSL. The E in the second from right-hand digit indicates that the decimal overflow, floating underflow, and integer overflow traps are enabled (Example 6-7). This condition is always true when the test instruction is being executed.

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 2  SUBTEST 1  ERROR 5Ø  Ø

HARD ERROR WHILE TESTING KAØ: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 5Ø

? RESULT PSL ERROR

EXPECTED        ACTUAL

ØØ1FØØE8        ØØ1FØØE1
```

Example 6-7 PSL Error

6.2.6.8 Branch Errors -- When testing instructions that may branch, failure to branch when expected or a branch taken when not expected produces a message like that in Example 6-8.

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 2  SUBTEST 4  ERROR 1  Ø

HARD ERROR WHILE TESTING KAØ: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 1

? EXPECTED BRANCH DIDN'T HAPPEN
```

Example 6-8 Branch Error

6.2.6.9 Expected Exception or Trace Trap Errors

1. An error message is produced if an expected exception or trace trap fails to occur at all.

2. The PC and PSL of expected exceptions and trace traps are checked. If an error is detected, a message like that in Example 6-9 is typed.

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 2  SUBTEST 4  ERROR 2  Ø

HARD ERROR WHILE TESTING KAØ: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 2

? INCORRECT EXCEPTION PC

EXPECTED          ACTUAL

ØØØØ25F5          ØØØØ2566

? INCORRECT EXCEPTION PSL

EXPECTED          ACTUAL

ØØ1FØØE5          ØØ1FØØE4
```

Example 6-9 Expected Exception Error

In Example 6-9 both the PC and the PSL were incorrect at the time of the exception. In Example 6-1Ø only the PSL was wrong at the time a valid trace trap occurred.

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 2  SUBTEST 2  ERROR 127  Ø

HARD ERROR WHILE TESTING KAØ: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 127

? INCORRECT TRACE TRAP PSL

EXPECTED          ACTUAL

ØØ1FØØFØ          ØØ1FØØF8
```

Example 6-1Ø Trace Trap Error

6.2.6.1Ø Extended Printout -- If the extended error printout flag is enabled, the following additional data will be typed out on error detection (Example 6-11).

```
[1.]    INITIAL CONDITIONS:

        PC ØØØØ4873  PSL ØØ1FØØFF

        OP CODE -- 74 WITH REGISTER INDIRECT OPERANDS

[2.]    INITIAL REGISTERS RØ-R6:

        RØ     ØØØØACØ4   R1     ØØØØAC14  R2     ØØØ

        R3     ØØØØAC2D   R4     ØØØØAC39  R5     ØØØ

        R6     ØØØØØØØØ

[3.]    SOURCE OPERAND DATA:

        OPERAND 1

        FFFE4FFF

        FFFFFFFF

        OPERAND 2

        FF

        OPERAND 3

        ØØØØ4Ø8Ø

        ØØØØØØØØ
```

Example 6-11 Extended Printout

Notes for Example 6-11.

1. This is the first line of the extended typeout. PC is the location of the test instruction, which can be examined if the user wants to see the hex code.

   PSL is the value of the PSL before the instruction is executed.

   OP CODE is the hex value of the instruction, which in the example is 74 = EMODD.

   REGISTER INDIRECT OPERANDS means that RØ has the address of operand 1; R1 has the address of operand 2; etc.

2. The INITIAL REGISTERS typeout tells where the operands of the test instruction were in memory when the instruction was executed.

3. These are the actual contents of the addresses pointed to by the registers listed above (2).

   All source (read or modifiable) operands are typed.

Formats:

Byte	XX
Word	XXXX
Longword	XXXXXXXX
Quadword	XXXXXXXX -- Low Address Longword XXXXXXXX -- High Address Longword
Strings	XX, XX, XX, XX,...,XX Left side of printout is lowest address byte. Long strings are printed 16 bytes per line and are continued for as many lines as needed.

In the preceding example we have the following operands. (Refer to the VAX-11/78Ø Architecture Handbook or the instruction card for further help.)

Operand 1	MULR	(RØ)	Double	FFFFFFFF FFFE4FFF
Operand 2	MULRX	(R1)	Byte	FF
Operand 3	MULD	(R2)	Double	ØØØØØØØØ ØØØØ4Ø8Ø

The 4th and 5th Operands are Destinations:

Operand 4	INT	(R3)	Long
Operand 5	FRACT	(R4)	Double

6.2.7 How To NO-OP a Test Case

If it is necessary to bypass a test case while waiting for a hardware ECO or a microcode ECO, refer to Example 6-12.

```
             2C9   946         ;CASE 1Ø5
             2C9   947         ;SUBD2 INSTRUCTION
             2C9   948         ;
             2C9   949         ;
             2C9   95Ø         ;OPERANDS
             2C9   951         ;SUB: Ø
             2C9   952         ;DIF: 1.Ø              EXP-DIF:          1.Ø
             2C9   953         ;CONDITION CODES       INITIAL: 1111     EXPECTED:      ØØØØ
             2C9   954
             2C9   955         TB I_SUBD2, CC_NZVC, CC_Ø, D8_D12, D8_D2, DP_D2
Ø8FØ9D62     2C9                     .BYTE I_SUBD2, <-C<I_SUBD2>>, <16*<CC_NZVC&15>+CC_Ø&15>>
      Ø8     2CC                     .BYTE DP_D12
      Ø1     2CD                     .BYTE DP_D2
      Ø1     2CE                     .BYTE DP_D2
```

Example 6-12 Case 1Ø5 SUBD2 Instruction

Load ESKAX.EXE
Look up the base address of the PSECT <. BLANK .> in the link map of this program for the module that has the data for the test case to be No-Oped.

Set the console base register to that value (i.e., SE R: VALUE).

Find the TB line of the right test case and examine it to make sure you are in the right place (Example 6-12).

Examination of 2C9 location (E 2C9) should give Ø8FØ9D62.

Count the number of single bytes following the line that has three bytes. That would be 3 for this example.

Deposit a new longword, at the address just examined, made up of the count from the preceding step followed by Ø3FC.

In Example 6-12, D 2C9 3Ø3FC, where 2C9 is the address just examined.

Set the relocation register back to zero when finished (i.e., SE R: Ø).

6.3 ESKAX DESCRIPTION

6.3.1 Compatibility Mode Entry/Exit Module (ESKAXØ2, Test Ø1)

This module tests the conditions generated when the central processor enters and leaves the compatibility mode. The following conditions and functions are tested.

ESKAX Test 1, Subtest 1 -- This subtest performs illegal entries in compatibility mode expecting and checking for reserved operand faults. The bit settings in the PSL that will cause reserved operand faults, on an attempt to enter compatibility mode, are shown in Table 6-3.

Table 6-3 Reserved Operand Faults and PSL Bit Settings on Compatibility Mode Entry

PSL Bit/s	Condition
DV<7>	Nonzero
FU<6>	Nonzero
IV<5>	Nonzero
IPL<2Ø:16>	Nonzero
CUR MOD<25:24>	Not = 3
PRV MOD<23:22>	Not = 3
IS<26>	Nonzero
FPD<27>	Nonzero

The conditions in Table 6-3 are tested one at a time.

The following two examples are typical of ESKAX test 1, subtest 1, error messages.

```
********   CPU CLUSTER EXERCISER -- 9.0    ********

PASS 1  TEST 1  SUBTEST 1  ERROR 2  19-JUN-1977  21:25:41.22

HARD ERROR WHILE TESTING CPU:  EXCEPTION PC FROM CM ILLEGAL ENTRY
                               INCORRECT

VECTOR   TYPE CODE   EXPECTED PC   ACTUAL PC   PSL ENTRY  MNEMONIC

18       NONE        00007D74      00007D76    83C00080   DV
```

Example 6-13 ESKAX Test 1, Subtest 1, Error 2

```
********   CPU CLUSTER EXERCISER -- 9.0    ********

PASS 1  TEST 1  SUBTEST 1  ERROR 2  19-JUN-1977  21:25:53.04

HARD ERROR WHILE TESTING CPU:  EXCEPTION PC FROM CM ILLEGAL ENTRY
                               INCORRECT

VECTOR   TYPE CODE   EXPECTED PC   ACTUAL PC   PSL ENTRY  MNEMONIC

18       NON         00007D74      00007D54    83C00040   FU
```

Example 6-14 ESKAX Test 1, Subtest 1, Error 2

Interpretation of Example 6-13.

1. 18 is the reserved operand fault vector expected.
2. There is no type code pushed on the stack.
3. The state of the PSL to cause the fault was 83C00080.
4. DV is the PSL bit that was nonzero (Table 6-3).
5. EXPECTED and ACTUAL PCs are self-explanatory.

ESKAX Test 1, Subtest 2 -- Compatibility mode trap instructions upon a valid entry into compatibility mode (Table 6-4).

Table 6-4 Compatibility Mode Trap Instructions

Op code	Mnemonic
000003	BPI
000004	IOT
104000	EMT+0
104400	TRAP+0

-- Compatibility mode reserved instructions upon a valid entry into compatibility mode (Table 6-5).

Table 6-5 Compatibility Mode Reserved Instructions

Op code	Mnemonic
ØØØØØØ	HALT
ØØØØØ1	WAIT
ØØØØØ5	RESET
ØØØ23Ø	SPL
ØØ64ØØ	MARK
Ø75ØØØ	FADD
Ø75Ø1Ø	FSUB
Ø75Ø2Ø	FMUL
Ø75Ø3Ø	FDIV
17ØØØØ	FP11

Typical Error Messages

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 2  ERROR 3  19-JUN-1977  21:29:3Ø.4Ø

HARD ERROR WHILE TESTING CPU: EXCEPTION PSL FROM COMPATIBILITY

MODE TRAP INCORRECT

VECTOR   TYPE CODE   EXPECTED PSL   ACTUAL PSL   TRAP    MNEMONIC

3Ø       1           83CØØØØØ       83CØØØØ2     ØØØ3    BPT
```

Example 6-15 ESKAX Test 1, Subtest 2, Error 3

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 2  ERROR 3  19-JUN-1977  21:29:42.21

HARD ERROR WHILE TESTING CPU:  EXCEPTION PSL FROM COMPATIBILITY

MODE TRAP INCORRECT

VECTOR    TYPE CODE    EXPECTED PSL    ACTUAL PSL    TRAP     MNEMONIC

3Ø        2            83CØØØØØ        83CØØØØ2      ØØØ4     IOT
```

Example 6-16 ESKAX Test 1, Subtest 2, Error 3

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 2  ERROR 3  19-JUN-1977  21:29:54.Ø2

HARD ERROR WHILE TESTING CPU:  EXCEPTION PSL FROM COMPATIBILITY

MODE TRAP INCORRECT

VECTOR    TYPE CODE    EXPECTED PSL    ACTUAL SPL    TRAP     MNEMONIC

3Ø        3            83CØØØØØ        83CØØØØ2      88ØØ     EMT
```

Example 6-17 ESKAX Test 1, Subtest 2, Error 3

Interpretation of Example 6-16

1. 3Ø is the compatibility mode TRAP vector expected.

2. A type code of 2 is pushed on the stack.

3. Referencing Chapter 6 of the system reference manual would show that a typecode of 2 indicates an IOT fault.

4. IOT is shown as well as the hex equivalent of the octal code (Table 6-4).

5. EXPECTED and ACTUAL PSLs are self-explanatory.

Subtest 3 -- This subtest tests the T-bit trap by having the T-bit (PSL<4>) set upon entry into compatibility mode:

a. for an instruction that does not trap
b. for an instruction that does trap.

NOTE

Both a and b cases are serviced in **NATIVE** mode.

Typical Error Message

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 3  ERROR 4  19-JUN-1977  21:32:Ø6.8Ø

HARD ERROR WHILE TESTING CPU:  A T-BIT TRAP NOT TAKEN

EXPECTED EXC    VECTOR    TYPE CODE    MNEMONIC

ØBCØ            3Ø        NONE         TST RØ
```

Example 6-18 ESKAX Test 1, Subtest 3, Error 4

Interpretation of Example 6-18 (this printout is for Case B):

1. This instruction, which was to execute and then take a T-bit trap, was 'TST RØ' with TRACE PENDING prior to its execution (PSL<TP>).
2. The hex equivalent of the octal code for 'TST RØ' is BCØ.
3. 3Ø is the vector expected to field the T-bit trap.
4. No type code is pushed on the stack.

Subtest 4 -- This subtest performs an RTT/RTI instruction with the T-bit set in the PSW image on the stack, which is to be popped from the stack by the RTT/RTI.

Typical Error Messages

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 4  ERROR 3  19-JUN-1977  21:33:43.6Ø

HARD ERROR WHILE TESTING CPU:  PC FROM RTT TRACE TRAP

INCORRECT

VECTOR   TYPE CODE   EXPECTED PC   ACTUAL PC   TRAP     MNEMONIC

3Ø       NONE        ØØØØ85Ø8      ØØØØ84Ø8    ØØØ6     RTT
```

Example 6-19 ESKAX Test 1, Subtest 4, Error 3

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 4  ERROR 3  19-JUN-1977  21:33:54.7Ø

HARD ERROR WHILE TESTING CPU:  PC FROM RTI TRACE TRAP

INCORRECT

VECTOR   TYPE CODE   EXPECTED PC   ACTUAL PC   TRAP      MNEMONIC

3Ø       NONE        ØØØØ85Ø8      ØØØØ84Ø8    ØØØ2      RTI
```

Example 6-2Ø ESKAX Test 1, Subtest 4, Error 3

Interpretation of Example 6-19

1. 3Ø is the vector expected to field the T-bit trap.
2. No type code is pushed on the stack.
3. The RTT instruction was under test.
4. The hex equivalent of the octal code for RTT is 6.
5. EXPECTED and ACTUAL PCs are self-explanatory.

Subtest 5 -- This subtest performs checking of Odd Address errors while in compatibility mode. This is accomplished by executing a PDP-11 MOV instruction with unaligned SRC and DST operands.

Typical Error Messages

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 5  ERROR 4  19-JUN-1977  21:35:17.19

HARD ERROR WHILE TESTING CPU:  ODD ADDRESS TRAP CAUSED

UNALIGNED SOURCE CONTENTS CHANGE

VECTOR   TYPE CODE   EXPECTED VAL  ACTUAL VAL  TRAP      MNEMONIC

3Ø       6           ØØØA          ØØØE        17DF      UNALIGNED
```

Example 6-21 ESKAX Test 1, Subtest 5, Error 4

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 5  ERROR 4  19-JUN-1977  21:35:29.58

HARD ERROR WHILE TESTING CPU:  ODD ADDRESS TRAP CAUSED

UNALIGNED SOURCE CONTENTS CHANGE

VECTOR   TYPE CODE   EXPECTED VAL  ACTUAL VAL  TRAP      MNEMONIC

3Ø       6           ØØØA          ØØØE        17DF      UNALIGNED
                                                         DST
```

Example 6-22 ESKAX Test 1, Subtest 5, Error 4

Interpretation of Example 6-22

1. 3Ø is the compatibility mode TRAP vector expected.

2. A type code of 6 is pushed on the stack.

3. The position of the destination address on a boundary caused the failure.

4. The hex equivalent of the octal code for the instruction under test is 17DF (this translates to Ø13737 in PDP-11 code).

5. EXPECTED and ACTUAL VALUES are self-explanatory.

> **NOTE**
>
> **On an Odd Address trap neither SRC nor DST initial values should change, since the instruction should not go to completion.**

Subtest 6 -- This subtest performs checking of illegal instructions with a register destination, i.e.,

```
JMP R4 or
JSR R4, R5
```

Typical Error Messages

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 6  ERROR 3  19-JUN-1977  21:36:32.Ø3

HARD ERROR WHILE TESTING CPU:  PSL FROM ILLEGAL INSTRUCTION

TRAP INCORRECT

VECTOR    TYPE CODE   EXPECTED PSL  ACTUAL PSL  TRAP       MNEMONIC

3Ø        5           83CØØØØØ      83DØØØØØ    ØØ44       JMP R4
```

Example 6-23 ESKAX Test 1, Subtest 6, Error 3

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 6  ERROR 3  19-JUN-1977  21:36:43.64

HARD ERROR WHILE TESTING CPU:  PSL FROM ILLEGAL INSTRUCTION

TRAP INCORRECT

VECTOR    TYPE CODE   EXPECTED PSL  ACTUAL PSL  TRAP   MNEMONIC

3Ø        5           83CØØØØØ      83ØØØØØØ    Ø9Ø5   JSR (R5 DST)
```

Example 6-24 ESKAX Test 1, Subtest 6, Error 3

Interpretation of **Example** 6-24

1. 3Ø is the vector expected to field the TRAP.

2. A type code of 5 is pushed on the stack.

3. The instruction that failed was the JSR R4, R5.

4. The hex equivalent for the octal code of this instruction is Ø9Ø5.

5. EXPECTED and ACTUAL PSLs are self-explanatory.

ESKAXØ2 (test 1) is executed in the user mode for test purposes. However, the module is serviced in the kernel mode, and control is returned to the kernel mode on completion of the module.

The operator should note that testing of the T-bit operation with servicing done in the compatibility mode has not been covered.

6.3.2 First Part Done Test (ESKAXØ4, Test 2)

First Part Done (FPD) is the name of bit 27 in the PSL. It provides a facility for interrupting certain potentially long executing instructions during processing and resuming them later. Only a few instructions are interruptable in this sense. Most instructions acknowledge interrupts before their execution, or acknowledge them in mid-operation by backing up to the beginning and pretending that they have not yet started. A few instructions, however, are potentially so lengthy that this is not feasible. These are the character and decimal string instructions, POLYF, POLYD, and CRC. Each of these instructions writes a control block into the general registers. Should an interrupt be requested during processing, the current state of the operation (i.e., what it is doing and how far it has gotten) can be saved in this control block to be retrieved after the interrupt is processed. The instruction then sets FPD in the PSL, and acknowledges the interrupt. Upon return from the interrupt, the FPD bit is set in the PSL, so that rather than restarting, the instruction restores its state from the point at which it was interrupted.

6.3.2.1 Possible First Part Done Failures -- The microcode implementing the FPD capability must be able to correctly save and restore state anywhere it does a memory reference (which may cause a fault) and anywhere it checks for interrupts. The state of the operation in some cases is complex, and it is possible that the microcode does not save or restore everything correctly. If the instruction is later resumed, the state of the machine e.g., in the form of contents of general registers, may well have been changed by the instructions executed in the interim, and will thus be incorrect. This will cause unpredictable results, most likely in the form of incorrect data written, wrong lengths and wrong condition codes, and will be easy to detect.

The instructions may also fail by saving state (perhaps correctly) and failing to set FPD. This would normally appear when modified registers are used as arguments in the restarting of the instruction. This condition will be detected in the test by checking in the interrupt routine to make certain that if FPD is still clear, the original arguments are unchanged.

6.3.2.2 First Part Done Test Procedures -- The interval timer is used to generate interrupts during the testing of each instruction, in order to check the microcode and the taking of interrupts. Although each instruction is interrupted constantly during execution, it is eventually run to completion.

After having been tested with interrupts, the instruction's ability to handle page faults is tested. An instruction may have up to six operands; twelve pages are set up to hold them, allowing each operand to be placed near an independent page boundary. When the instruction begins execution, each page is invalid. As it attempts to access its operands, the instruction is repeatedly faulted. Each fault validates the page referenced, so that the instruction progresses, but this alone does not ensure that it is tested fully. As each page is referenced (and faulted) the first time, all the other pages holding operands are made invalid. This process tests all the cases. Since each page has a first reference only once, the test instruction manages to finish.

As an example, consider the testing of an instruction with two string operands and one non-string operand in which the instruction accesses the non-string operand first, and then processes the strings (e.g., CMP3). First, the non-string operand is referenced, faulting the page containing it. Upon restart, the instruction fetches the non-string operand without problem, and begins processing the strings. Since each operand is located just before a page boundary, the strings will cross the boundaries. As the instruction progresses, it will attempt to reference the first page of the first operand, the first page of the second operand, the second page of the first operand, and the second page of the second operand. Because faulting in a page for the first time signals the test to invalidate all the other pages, however, the string of references and validations proceeds as shown in Table 6-6.

Table 6-6 Page Faulting with First Part Done

Page 1	Page 2	Page 3	Page 4	
I	I	I	I	;All the pages start invalid.
FAULT	I	I	I	;The first page is faulted
V	I	I	I	;and is made valid.
V	I	FAULT	I	;The first page of operand 2 is
I	I	V	I	;faulted in, and the rest out.
FAULT	I	V	I	;Page 1 is refaulted, and page
V	I	V	I	;3 is left valid.
V	FAULT	V	I	;String 1 processing reaches
I	V	I	I	;page 2, all others faulted.
I	V	FAULT	I	;Page 1 is not needed now, but
I	V	V	I	;page 3 still is needed.
I	V	V	FAULT	;String 2 reaches its second ;page
I	I	I	V	;faulting page 4 for first ;time.
I	FAULT	I	V	;Page 2 is still needed, and
I	V	I	V	;is faulted back in.
I	V	I	V	;The instruction is completed.

The first part done test uses the CITS routines to help set up, execute, and check the results of instruction tests. The instructions to test, and the data with which to test them, are stored in a table. The table entries are of variable length, and they begin as shown in Table 6-7.

Table 6-7 **First Part Done Test Table Entries**

.BYTE	Ø	;the op code of the test instruction
.BYTE	Ø	;the op code's complement
.BYTE	Ø	;initial condition codes
.BYTE	Ø	;resultant condition codes
.BYTE	Ø	;operand specifiers
	.	
	.	
	.	

The faulting section uses the Interface Data Block (IDB) to communicate with the exception and interrupt handler. The format of the IDB is shown in Table 6-8.

Table 6-8 **First Part Done IDB Format**

T-Bit Count	Exception and Subtype	State Bits
PSL of exception		
PC of exception		
PSL of latest T-bit trap		
PC of latest T-bit trap		
User service routine address		
Number of arguments (zero)		

The service routine address points to the code that implements the faulting algorithm. The exception type and subtype are loaded with the values for translation-not-valid faults.

This test also interfaces with the CITS routine through a Test Control Block (TCB). The TCB format is shown in Table 6-9.

Table 6-9 **First Part Done TCB General Format**

Current Test Table Address			
Unused	Exception	Subtype	T-bit trap
Operand 1 address, or Ø			
Operand 2 address, or Ø			
Operand 3 address, or Ø			
Operand 4 address, or Ø			
Operand 5 address, or Ø			
Operand 6 address, or Ø			

The current test table address points into the table of test instructions.

The TCB passed to CITS_DECODE and to CITS_REDECODE is shown in Table 6-10.

Table 6-10 First Part Done TCB Passed to CITS_DECODE

```
TCB:

TCB_INST_ADDR:      .LONG   0    ;current test table address
TCB_T_BIT:          .BYTE   0    ;trace trap expected flag
TCB_SUBTYPE:        .BYTE   0    ;exception subtype
TCB_EXCEPTION:      .BYTE   0    ;expected exception vector
                    .BYTE   0    ;unused
TCB_OPERANDS:       .BLKL   6    ;optional operand addresses
```

Typical Error Message

```
********   CPU CLUSTER EXERCISER -- 9.0    ********

PASS 1  TEST 2  SUBTEST 0  ERROR 212  19-JUN-1977  21:41:22.03

HARD ERROR WHILE TESTING CPU:   An unexpected type of fault
occurred.

Fault code      Referenced address    PC          PSL

00000000        0001F7F8              00003DD0    000000EB

Table number    Test case

1               1

TCB's address   Test table address    Current entry address

00008F9C        00000A7C              00000A7C
```

Example 6-25 ESKAX Test 2, Subtest 0, Error 212

Interpretation of Example 6-25
The printout is representative of the first part done test, which interfaces to the CITS portion of the program for its data pool as follows.

1. This test interfaces with CITS through the TCB whose format is shown in Table 6-10.

   In this example, the first address of the TCB is 8F9C.

   The current test table address (which is the first longword of the TCB) is given as A7C.

2. The starting address of the test table within CITS is A7C.

3. CITS contains a number of tables; each table contains a number of cases (or distinct pieces of data) where:

   Table 1 represents BASE FP instructions
   Table 2 represents DECIMAL instructions
   Table 3 represents EDIT PC instructions
   Table 4 represents FPA instructions

   A summary of the information presented so far on the First Part Done Test follows.

   a. We are using Table 1 from CITS for our data.

   b. The starting address of this table is A7C.

   c. We are using DATA CASE 1.

   d. The address for DATA CASE 1 is A7C.

   e. The address of the CONTROL BLOCK guiding this test execution is 8F9C (whose format is shown above in Table 6-1Ø).

   f. Examination of the next n locations starting with the CURRENT ENTRY ADDRESS (in this case A7C) will give information concerning the instruction under test as follows:

      1st byte is the op code of the test instruction.
      2nd byte is the op code's complement.
      3rd byte is the INITIAL condition code (N,Z,V,C).
      4th byte is the RESULTANT condition code (N,Z,V,C).

      The next n bytes represent operand specifiers.
      The number of operand specifiers depends on the instruction under test.

4. The starting address of the area where the test instruction is placed (residing) while undergoing test is the PC of 3DDØ.

A Second Error Message

```
********   CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 2  SUBTEST Ø  ERROR 213  19-JUN-1977  21:41:22.Ø3

HARD ERROR WHILE TESTING CPU:  Page fault on non-test instruction.

Fault Code      Reference address    PC          PSL

ØØØØØØØØ        ØØØ1FFF8             ØØØØ3DDØ    ØØØØØØEB

Table number    Test case

1               1

TCB's address    Test table address    Current entry address

ØØØØ8F9C         ØØØØØA7C              ØØØØØA7C
```

Example 6-26 ESKAX Test 2, Subtest Ø, Error 213

Interpretation of Example 6-26

The REFERENCE ADDRESS of 1FFF8 represents the address which caused the FAULT CODE of Ø.

The breakdown of the FAULT CODE is as follows:

Bit Position	Meaning
Ø	Ø = translation not valid 1 = access control violation
1	1 = fault occurred during virtual reference to the PTE of the stored process virtual address
2	Ø = read access 1 = write or modify access

The interrupts portion of the test begins by setting up the test instruction and data, using the CITS routines named CITS_DECODE and CITS_SETUP.

CITS_SETUP returns with the PC and initial PSL of the test instruction on the stack. The test saves a copy of the test instruction's PC and general registers, so that its progress may be observed. Then the test initializes the interval timer service routine.

It then sets up a timer interrupt and executes an REI to the test instruction, which is interrupted immediately.

Since the state of the instruction is contained in the registers, if they are unchanged since the previous interrupt, the instruction has not progressed. This condition results from interrupting too soon. In this case, the interval timer is increased, and the test instruction is resumed.

In the other case, when the general registers have changed, the instruction has progressed.

Next, a divide-packed instruction is executed in an attempt to modify the state of any internal registers that might be used by the instruction under test. The timer is then set up for the new wait time, started, and the test instruction is resumed.

When the test instruction has been completed, the results are checked and any errors are reported.

Once interrupts and faults have been tried, the next entry in the test instruction table is selected, and the testing of that instruction begins.

There are four classes of errors that may occur.

Class 1 -- Unexpected exceptions or interrupts.

Class 2 -- Exception or interrupt identifier reports, which simply state

ERROR IN TEST CASE NN.
These occur when an exception or interrupt occurs during the testing of an instruction, and they are immediately followed by the exception report. They exist solely to inform the operator of the test case in which the exception occurred.

Class 3 -- Instruction test errors describe incorrect results from instruction testing. The instructions tested are a subset of those tested in ESKAYØ5_TESTØ4 and ESKAYØ6_TESTØ5, so that the instruction test errors are identical between those tests and this test, ESKAXØ4_TESTØ2. This data is in module ESKAXØ3_FPD_DATA.

Class 4 -- These errors are first part done specific (Example 6-27).

They have error numbers 2ØØ through 2Ø9. Each reports error specific information, the table number, and the test case number. The interpretation of table numbers is as follows:

Table Number	**Meaning**
1	Floating-point test table
2	Decimal string test table
3	EDITPC test table
4	Floating-point test table (Executed with FPA enabled if an FPA exists)

The test case number indexes into the appropriate table to indicate a single test.

```
*********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 2  SUBTEST Ø  ERROR 2Ø7  . . . . .

HARD ERROR WHILE TESTING CPU:   EXPECTED  TIMER  INTERRUPTS  DIDN'T
                                OCCUR.

VALUE PASSES

-3Ø


TABLE NUMBER   TEST CASE

1              115


TCB'S ADDRESS   TEST TABLE ADDRESS   CURRENT ENTRY ADDRESS

18EØ            9F6                  9F6
```

Example 6-27 ESKAX Test 2, Subtest Ø, Error 2Ø7

6.3.3 SBI Verification Module (ESKAXØ5, Test 3)

The SBI verification test is designed to exercise and partially diagnose faults on the SBI nexus connected to it. Error reports will differentiate between faults on the SBI proper and faults caused by a nexus. The error printouts will serve as guides to selection of the appropriate repair level diagnostic to further isolate the problem.

With the exception of the interactive mode setup subtest, errors will be reproducible via looping. For interactive mode, due to the asynchronous operation of the exerciser, only errors introduced by interrupts from the interval timer are guaranteed to be reproducible.

Note that failing devices are deselected from further testing at the point of failure. This means that if an MBA or UBA fails in a test before MBE or UBE checkout, the MBEs or UBEs attached are not checked for the failing MBA or UBA.

The SBI verification test is composed of the following parts.

^ SBI checkout -- Verifies configuration register of each nexus that can be accessed.

^ UBA checkout -- Verifies that each selected UBA on the SBI can sustain data transfers without incurring errors and that interrupts occur at the proper BR level.

^ MBA checkout -- Verifies that each selected MBA on the SBI can sustain data transfers without incurring errors and that interrupts occur at the proper BR level.

^ SBI interaction -- Verifies that all UBAs are capable of block data transfers in a controlled sequential mode of operation.

^ UBE checkout -- Verifies that all existing UBEs are capable of sustaining data transfers and interrupting on completion without errors.

^ MBE checkout -- Verifies that all existing MBEs for selected MBAs are capable of sustaining data transfers and interrupting on completion without errors.

6.3.3.1 SBI Checkout Subtest -- The SBI checkout subtest will perform reads and writes to the configuration register of each nexus on the SBI as defined by the hardware P-Table. This subtest will set up the Hardware Interrupt Request Table (HIRT), which will contain an entry for each UBA and/or MBA responding to a read of its configuration/status register. This table will be used by all subtests within the SBI verification test. A nexus that does not respond will not be entered into the HIRT and, therefore, will not be used in the following subtests. No response from a nexus is treated as an error.

The SBI checkout subtest uses the CPU silo comparator register to check the validity of the commands and responses from the receiving nexus on the SBI.

The primary purpose of this subtest is to provide the field user with a detailed check of the SBI. It will isolate faults in such a manner that the error information printed will aid the user in the selection of the proper diagnostic, which may then be run to further isolate the fault.

^ Silo Compare Servicing -- The silo compare service routine will read back the SBI silo and compare the contents with the arguments supplied in the IDB (interface data block). Null cycles between command issue and read reply are checked for continuity of function, i.e., TR lines not continually asserted. No checking will be made for the number of null cycles.

On completion of the silo read back, the fault bit in the CPU SBI status register will be checked for clear. The error flag will be set and the appropriate information will be placed on the error stack if it is set. The fault bit will be reset if set. The interrupt on silo compare bit will be cleared and the SBI silo compare register will be cleared.

A return is then made to the point of invocation of the interrupt causing this routine to be executed.

6.3.3.2 UBA Checkout Subtest -- This subtest will only be run for UBAs that exist in the HIRT and have been qualified by the SBI checkout subtest.

Each UBA will be set up to operate in a wraparound mode so that access from the SBI to Unibus memory space will be mapped into SBI memory space.

This subtest will check the following UBA capabilities.

1. DDP and BDP1 data paths are operational.
2. Interrupts can be initiated by the adapter and result in the correct vector being accessed.
3. The map registers can be accessed and used correctly.
4. Purging operates correctly.
5. A read to nonexistent Unibus memory space causes the correct error sequence and interrupt.

The subtest will autosize the Unibus memory and set the map register disable portion of the Unibus Adapter Control Register (UACR) for use by other subtests within the SBI verification test.

Faults detected within this subtest will cause the UBA under test to be disqualified from further use by any other subtest within the SBI verification test.

UBA Interrupt Servicing -- Interrupts generated by the Unibus Adapter are serviced by this routine.

The routine will compare the configuration register and the Unibus Adapter Status Register (UASR) with arguments supplied in the Service Data Block (SDB). If there are any differences, they will be pushed on the error stack and the error flag will be set. For an invalid map register type interrupt, the failed mapped entry register will be compared with the SDB entry. Also, for a Unibus SSYN timeout, the failed Unibus address register will be compared with the SDB entry.

The AEIL (Additional Exception or Interrupt Longword) is used as the transfer vehicle to indicate to the interrupted program the IPL level at which the interrupt occurred.

6.3.3.3 MBA Checkout Subtest - This subtest is run only on MBAs that exist in the HIRT and have been qualified by the SBI checkout subtest.

Each MBA is set up to operate in maintenance mode.

This subtest checks the following MBA capabilities.

1. Initialization clears registers and does not cause interrupts.

2. DT_BUSY can be set and causes no interrupts.

3. PGE can be set and causes an interrupt.

4. Read and write transfers operate correctly; on completion of read data transfer, DONE is set and causes an interrupt.

Faults detected within this subtest cause the MBA under test to be disqualified from further use by any other subtest within the SBI verification test.

MBA Interrupt Servicing -- Interrupts generated by the Massbus adapter are serviced by this routine.

This routine compares the status register with an argument supplied in the SDB.

If there are any differences, the SDB + 2 will be set to indicate error and the error information will be put into the appropriate slots in the Master Control Space (MCS).

6.3.3.4 SBI Interaction Subtest -- After the UBAs are set up for wraparound operation, the following data transfer types are initiated.

1. read word
2. write byte
3. write word
4. modify byte
5. modify word

6.3.3.5 UBE Checkout Subtest -- This subtest determines the number and location of Unibus exercisers for each Unibus adapter and checks each as it is found. If no faults are detected, the UBE is entered in the HIRT and the qualify bit will be set.

Only qualified UBAs are used during this subtest. If none exists, the subtest will be skipped.

UBAs are set up with two map registers pointing to SBI memory space. One map uses the Direct Data Path (DPØ) and the other uses buffered Data Path One (DP1). All interrupts are enabled.

Autosizing is used to determine the location of a Unibus exerciser.

Each exerciser is checked for the following two capabilities:

1. ability to execute DATO, DATI functions,
2. ability to interrupt at BR levels 4 through 7 following a function.

If any of the above conditions is not met, the UBE is not entered in the HIRT.

UBE Interrupt Servicing -- The contents of the BRRVR and UBA base address are passed into the test from the master exception and interrupt handler. In addition, the routine performs the following four functions.

1. If bit 31 is set in the BRRVR value read, then call the UBA service routine.
2. Derive the UBE address from the vector supplied in the low word of the BRRVR value.
3. Examine bit 15 (error bit) of CR1. If the bit is set, push the error type information and contents of the control registers, CR1 and CR2, on the error stack and set the error flag. Clear the error bit.
4. Return.

6.3.3.6 MBE Checkout Subtest -- This subtest determines if an MBE is present for each MBA that has been previously qualified.

Each exerciser will be checked for the following:

1. read transfers
2. write transfers.

Additionally, the MBA is checked for whether

1. Attention can be set which causes an interrupt.
2. Massbus exception can be set which causes an interrupt.

If any of the above conditions is not met, the MBE is not entered in the HIRT.

Typical Error Messages for the SBI Verification Module:

```
********   CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 3  SUBTEST 2  ERROR 1Ø  19-JUN-1977  21:53:25.Ø6

HARD ERROR WHILE TESTING UBA: INVALIDATED MAP REGISTER ACCESS

ERROR: DESTINATION OVERWRITTEN

ADD ACCESS    NEXUS ADD    MR ADD     FUNC      EXP DATA   ACT DATA

2Ø1ØØ9F8      6ØØØ6ØØØ     6ØØØ681Ø   WRITE     25255252   24255252
```

Example 6-28 ESKAX Test 3, Subtest 2, Error 1Ø

```
********   CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 3  SUBTEST 3  ERROR 4  19-JUN-1977  21:57:Ø3.12

HARD ERROR WHILE TESTING MBAØ:  MBA WRITE


ERROR: RESULT

ADD ACCESS    NEXUS ADD    EXP DATA   ACT DATA

6ØØ1Ø4Ø       6ØØ1ØØØØ     ØØØØ2ØØØ   ØØØØ24ØØ
```

Example 6-29 ESKAX Test 3, Subtest 3, Error 4

```
********   CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 3  SUBTEST 3  ERROR 4  19-JUN-1977  21:57:13.Ø7

HARD ERROR WHILE TESTING MBA1: MBA WRITE


ERROR: RESULT

ADD ACCESS    NEXUS ADD    EXP DATA   ACT DATA

6ØØ124ØØ      6ØØ12ØØØ     ØØØØ2ØØØ   ØØØØ3ØØØ
```

Example 6-3Ø ESKAX Test 3, Subtest 3, Error 4

Interpretation of Example 6-28

These printouts are typical of ESKAX Test 2, where:

1. The nexus address is 6ØØØ6ØØØ. A nexus is defined as a physical connection to the SBI. In this case the nexus is the UBA.

2. Since the SBI deals in 3Ø-bit addresses, 18-bit Unibus addresses must be translated to 3Ø-bit SBI addresses. This function is performed by the Unibus adapter through one of the 496 UBA memory map registers, as shown in Table 6-11.

Table 6-11 Unibus Adapter Map Register Address Offsets

Unibus Memory Page	Offset from the UBA Base Address
Ø	8ØØ
1	8Ø4
2	8Ø8
3	8ØC
4	81Ø
.	.
.	.
.	.
495	FBC
.	FCØ
.	.
.	. Reserved
.	FFC

In the example given, the MR ADDRESS is 6ØØØ681Ø. The underlined portion of the address (using Table 6-11) tells us that we are working with the map register for Unibus memory, page 4.

One Unibus memory page equals 512 bytes.

3. The function performed was a write.

4. Each UBA has an associated Unibus address space with a physical starting address as follows:

UBA Number	Physical Starting Address
Ø	2Ø1ØØØØØ
1	2Ø14ØØØØ
2	2Ø18ØØØØ
3	2Ø1CØØØØ

From Example 6-28 the ADDRESS ACCESSED is 2Ø1ØØ9F8, indicating UBA Ø under test.

5. EXPECTED and ACTUAL DATA are self-explanatory.

6.3.3.7 Memory Verify (ESKAXØ6_TESTØ4) -- Not yet implemented.

6.4 ESKAY

6.4.1 Interval Timer and Day Clock Verification Module (ESKAYØ2_TESTØ1)

This module tests the interval timer and the day clock. The interval timer is used extensively throughout the cluster exerciser package during interactive operation.

6.4.1.1 Interval Timer Functions

Subtest 1 -- The interrupt enable bit in the control status register can be set and cleared.

Typical Error Message

```
********    CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1   TEST 1   SUBTEST 1   ERROR 2   18-JUN-1977   Ø6:38:Ø5.1Ø

HARD ERROR WHILE TESTING CPU: INTERRUPT ENABLE BIT CAN'T BE
CLEARED
```

Example 6-31 ESKAY Test 1, Subtest 1, Error 2

Subtest 2 -- This subtest checks that the transfer bit (bit Ø4) in the control status register can be set, thus activiating a transfer of the contents of the next interval register to the current interval register.

A check that the transfer bit is read as Ø is also performed.

Typical Error Messages

```
********    CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1   TEST 1   SUBTEST 2   ERROR 1   18-JUN-1977   6:39:58.61

HARD ERROR WHILE TESTING CPU: XFER BIT STUCK AT 1
```

Example 6-32 ESKAY Test 1, Subtest 2, Error 1

```
********    CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1   TEST 1   SUBTEST 2   ERROR 2   18-JUN-1977   Ø6:39:58.61

HARD ERROR WHILE TESTING CPU: XFER FROM NEXT INTERVAL TO INTERVAL
COUNT INCORRECT
```

Example 6-33 ESKAY Test 1, Subtest 2, Error 2

Subtest 3 -- The single clock bit (bit 5) in the control status register can be set, thus causing the current interval register to advance by one.

The test also checks that the single clock bit is read as zero.

```
********    CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 1  SUBTEST 3  ERROR 2  18-JUN-1977  Ø6:43:14.82

HARD ERROR WHILE TESTING CPU: SINGLE CLOCK BIT NOT FUNCTIONING
PROPERLY
```

Example 6-34 ESKAY Test 1, Subtest 3, Error 2

Subtest 4 -- This test floats a one through a field of zeros in the current interval register. The medium of transfer is the read/write unit comprised of the current interval register and the next interval register, respectively. Since the next interval register is write-only, only the current interval register is checked at the end of the transfer. If a failure is detected in the current interval register, it is possible that the failure originated in the next interval register.

Typical Error Message

```
********    CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 1  SUBTEST 4  ERROR 1  18-JUN-1977  Ø6:44:31.89

HARD ERROR WHILE TESTING CPU: ADJACENT PIN STICKING IN INTERVAL
COUNT REGISTER


EXPECTED RESULT    RECEIVED RESULT    ENTRY VALUE

ØØØØØØØ4           ØØØØØØØ6           ØØØØØØØ4
```

Example 6-35 ESKAY Test 1, Subtest 4, Error 1

Interpretation

1. The ENTRY VALUE of ØØØØØØØ4 represents the value loaded into the next interval register (hex 19).

2. The EXPECTED RESULT of ØØØØØØØ4 represents what the content of the current interval register (hex 1A) should be after the transfer is complete.

3. The RECEIVED RESULT is self-explanatory.

Subtest 5 -- This subtest checks the carry bits of the current interval register. This is accomplished by preloading the next interval register with the value to force the carry, transferring this to the next interval register, and then single-clocking to force the carry expected.

Typical Error Message

```
********    CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1   TEST 1   SUBTEST 5   ERROR 1   18-JUN-1977   Ø6:45:5Ø.65

HARD ERROR WHILE TESTING CPU: INTERVAL TIMER COUNTING NOT
PROCEEDING PROPERLY


EXPECTED RESULT    RECEIVED RESULT    ENTRY VALUE

ØØØØØØØ2           ØØØØØØØ1           ØØØØØØØ1
```

Example 6-36 ESKAY Test 1, Subtest 5, Error 1

```
********    CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1   TEST 1   SUBTEST 5   ERROR 1   18-JUN-1977   Ø6:45:5Ø.65

HARD ERROR WHILE TESTING CPU: INTERVAL TIMER COUNTING NOT
PROCEEDING PROPERLY


EXPECTED RESULT    RECEIVED RESULT    ENTRY VALUE

ØØØØØØØ4           ØØØØØØØ3           ØØØØØØØ3
```

Example 6-37 ESKAY Test 1, Subtest 5, Error 1

Interpretation of Example 6-36

1. The ENTRY VALUE of ØØØØØØØ1 represents the value loaded into the next interval register (hex 19).

2. The EXPECTED RESULT of ØØØØØØØ2 represents what the content of the current interval register (hex 1A) should be after the transfer is complete and the single clock bit has been ticked once.

3. The RECEIVED RESULT is self-explanatory.

Subtest 6 -- This subtest checks that the error bit in the control status register will set in the case of a current interval register overflow occurrence before a previous interrupt has been serviced.

The error messages are self-explanatory.

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 6  ERROR 1  18-JUN-1977  Ø6:53:13.11

HARD  ERROR  WHILE  TESTING  CPU:  INTERRUPT  REQUEST  NOT  SET  ON
OVERFLOW
```

Example 6-38   ESKAY Test 1, Subtest 6, Error 1

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 6  ERROR 2  18-JUN-1977  Ø6:53:13.11

HARD ERROR WHILE TESTING CPU: ERR NOT SET FROM UNSERVICED OVERFLOW
```

Example 6-39   ESKAY Test 1, Subtest 6, Error 2

Subtest 7 -- This subtest checks the run bit of the control status register with the interrupt enable bit not set (i.e., a check of no interrupt capability).

Typical Error Message

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 7  ERROR 3  18-JUN-1977  Ø6:53:13.11

HARD ERROR WHILE TESTING CPU: ERR BIT SET -- SHOULD NOT BE
```

Example 6-4Ø   ESKAY Test 1, Subtest 7, Error 3

Subtest 8 -- This subtest checks the run bit of the control status register with the interrupt enable bit set, a check of interrupt capability. A check is also made to verify that the interrupt is enabled at IPL 24 (hex 18).

Typical Error Message

```
********   CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 1  SUBTEST 8  ERROR 2  18-JUN-1977  Ø6:55:56.48

HARD ERROR WHILE TESTING CPU: INTERRUPT OCCURRED AT OTHER THAN IPL
24


IPL WAS 18
```

Example 6-41 ESKAY Test 1, Subtest 8, Error 2

Interpretation of Example 6-41

1. The IPL WAS would indicate at what IPL level the interrupt did occur (if other than IPL 24).

6.4.1.2 Day Clock Function

Subtest 9 -- This subtest checks the ability of the time of day register to advance from a known state, given 2Ø ms to do so.

Typical Error Message

```
********   CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 1  SUBTEST 9  ERROR 1  18-JUN-1977  Ø6:58:26.79

HARD ERROR WHILE TESTING CPU: TIME OF DAY CLOCK NOT INCREMENTING
```

Example 6-42 ESKAY Test 1, Subtest 9, Error 1

Subtest 1Ø -- This subtest checks the ability of the time of day register to accept a back-to-back loading of two different and unique values.

Typical Error Message

```
********   CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 1  SUBTEST 1Ø  ERROR 2  18-JUN-1977  Ø6:59:43.78

HARD ERROR WHILE TESTING CPU: DOUBLE LOADING OF TIME OF DAY NOT
CORRECT

EXPECTED RESULT   RECEIVED RESULT   1ST LOAD   2ND LOAD

AAAAAAAC          AAAAAAAA          55555555   AAAAAAAA
```

Example 6-43 ESKAY Test 1, Subtest 1Ø, Error 2

Subtest 11 -- This subtest checks for any stuck-at-zero bits in the time of day register.

Typical Error Message

```
********   CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 1  SUBTEST 11  ERROR 1  18-JUN-1977  Ø7:ØØ:3Ø.34

HARD ERROR WHILE TESTING CPU: ADJACENT PIN STICKING IN TIME OF DAY
REGISTER

EXPECTED RESULT   RECEIVED RESULT   ENTRY VALUE

FFFFFFFE          FFFFFFFF          FFFFFFFD
```

Example 6-44 ESKAY Test 1, Subtest 11, Error 1

Subtest 12 -- This subtest checks the Carry bits of the time of day register. This is accomplished by preloading the time of day register with the value to force the Carry, and then expecting a Carry bit in approximately 14--15 ms.

```
********   CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 1  SUBTEST 12  ERROR 2  18-JUN-1977  Ø7:Ø5:Ø2.22

HARD ERROR WHILE TESTING CPU:  TIME OF DAY COUNTING NOT PROCEEDING
                               PROPERLY

EXPECTED RESULT   RECEIVED RESULT   ENTRY VALUE

ØØØØØØØ2          ØØØØØØØ1          ØØØØØØØ1
```

Example 6-45 ESKAY Test 1, Subtest 12, Error 2

Interpretation of Example 6-45

1. The ENTRY VALUE of ØØØØØØØ1 is what is initially loaded into the time of day register.
2. The EXPECTED RESULT of ØØØØØØØ2 is the final value expected to be in the time of day register approximately 14 ms after the initial load.
3. The RECEIVED RESULT is self-explanatory.

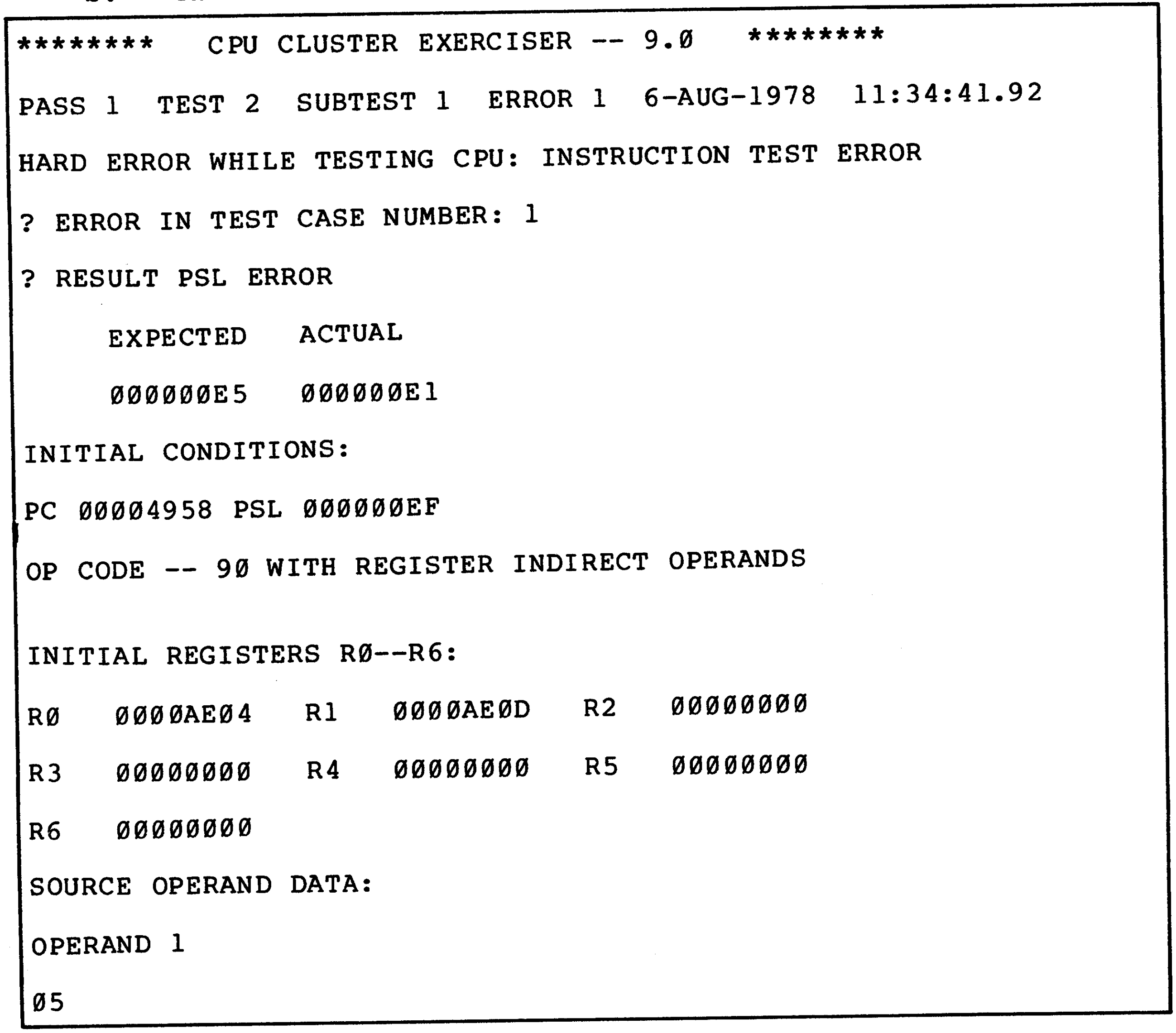
In addition, the test checks for stuck-at-zero bits in the time of day register.

6.4.2 Arithmetic, Logic, and Field Instruction Test Module (ESKAYØ3, Test Ø2)

This module tests the integer arithmetic, logical, and field instruction microcode and associated hardware. CITS performs all of the functional control, building expected data patterns, executing the instructions to be tested, and checking the results.

The following two printouts are typical of error reports coming from this test.

a. One shows a result PSL error.
b. The second shows incorrect operand result contents.



```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 2  SUBTEST 1  ERROR 1  6-AUG-1978  11:34:41.92

HARD ERROR WHILE TESTING CPU: INSTRUCTION TEST ERROR

? ERROR IN TEST CASE NUMBER: 1

? RESULT PSL ERROR

     EXPECTED   ACTUAL

     ØØØØØØE5   ØØØØØØE1

INITIAL CONDITIONS:

PC ØØØØ4958 PSL ØØØØØØEF

OP CODE -- 9Ø WITH REGISTER INDIRECT OPERANDS


INITIAL REGISTERS RØ--R6:

RØ    ØØØØAEØ4    R1    ØØØØAEØD    R2    ØØØØØØØØ

R3    ØØØØØØØØ    R4    ØØØØØØØØ    R5    ØØØØØØØØ

R6    ØØØØØØØØ

SOURCE OPERAND DATA:

OPERAND 1

Ø5
```

Example 6-46 ESKAY Test 2, Subtest 2, Error 1

```
********   CPU CLUSTER EXERCISER -- 9.Ø    ********

PASS 1  TEST 2  SUBTEST 1  ERROR 31  6-AUG-1978  11:35:27.25

HARD ERROR WHILE TESTING CPU: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 31

? INCORRECT RESULT, OPERAND 2

     EXPECTED   ACTUAL

     ØØØØ43ØØ   ØØØØ43Ø4

     ØØØØØØØØ   ØØØØØØØØ


INITIAL CONDITIONS:

PC ØØØØ46FA PSL ØØØØØØEF

OP CODE -- 6C WITH REGISTER INDIRECT OPERANDS

INITIAL CONDITIONS:

RØ   ØØØØAEØ4   R1   ØØØØAEØD   R2   ØØØØØØØØ

R3   ØØØØØØØØ   R4   ØØØØØØØØ   R5   ØØØØØØØØ

R6   ØØØØØØØØ


SOURCE OPERAND DATA:

OPERAND 1

21
```

Example 6-47 ESKAY Test 2, Subtest 1, Error 31

Interpretation of Example 6-47

1. The op code 6C defines the instruction under test as CVTBD (you can know this by simply looking up the given op code on a coding card).

2. The general format of this instruction (again from looking at the code card) is as follows:

   op code scr.rx, dst.wy

   The statement WITH REGISTER INDIRECT OPERANDS indicates that the form of the instruction being tested is CVTBD (RØ), (R1).

   NOTE

   All instruction testing is set up so that the first operand always uses RØ, second operand always uses R1, third operand always uses R2, etc.

3. The initial conditions PC and PSL should be self-explanatory.

4. The TEST CASE NUMBER of 31 shows nothing more than how far into the current test table we are, i.e., 3Ø instruction types were tested up to this point with no errors.

   For all intents and purposes, you can ignore this number.

5. The error indication of INCORRECT RESULT, OPERAND 2 states that the final contents of the destination operand were wrong. OPERAND 2 is shown above as (R1).

6. The SOURCE OPERAND DATA of 21 is self-explanatory.

7. The INITIAL REGISTERS RØ--R6 specify the addresses in memory in use for the instruction. In this case, CVTBC (AEØ4), (AEØD).

   NOTE

   R2 through R6 contain Øs since the CVTBD instruction uses only two operands.

8. Finally, the EXPECTED value of 43ØØ and the ACTUAL value of 43Ø4.

   If you examine the content of AEØD (/W) it should contain 43Ø4.

Example 6-48 is another form of printout similar to the preceding two examples with a twist. An unexpected exception occurred during the testing of an instruction.

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 2  SUBTEST 3  ERROR 1  6-AUG-1978  11:34:42.57

HARD ERROR WHILE TESTING CPU: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 1

     UNEXPECTED EXCEPTION

ERROR# ØØØØØØØ1

VECTOR# ØØØØØØ34

SUBTYPE# ØØØØØØØ1

PSL ØØØØØØEB

PC ØØØØ4933


INITIAL CONDITIONS:

PC ØØØØ493Ø PSL ØØØØØØEØ

OP CODE -- 8E WITH REGISTER INDIRECT OPERANDS


INITIAL REGISTERS RØ--R6:

RØ   ØØØØAEØ4   R1   ØØØØAEØD   R2   ØØØØØØØØ

R3   ØØØØØØØØ   R4   ØØØØØØØØ   R5   ØØØØØØØØ

R6   ØØØØØØØØ

SOURCE OPERAND DATA:

OPERAND 1

8Ø
```

Example 6-48 ESKAY Test 2, Subtest 3, Error 1

Interpretation of Example 6-48

1. The unexpected exception occurred through VECTOR 34 (Paragraph 2.7 of the KA78Ø Central Processor Technical Description lists VECTOR 34 as the ARITHMETIC TRAP vector).

2. The SUBTYPE of 1 informs you that the condition was INTEGER OVERFLOW (Paragraph 2.7 of the KA78Ø Central Processor Technical Description).

3. The PC of 4933 and PSL of EB are those existing at the time of the exception occurrence.

4. The ERROR 1 is nothing more than a repetition of the ERROR 1 printout of the header report.

5. The rest of the printout is as outlined for the two printouts preceding (i.e., the same breakdown applies).

6.4.3 Branch, CRC, and Queue Test Module (ESKAYØ4, Test Ø3)

Not yet implemented.

6.4.4 Floating-Point Instructions Test Module (ESKAYØ5, Test 4; ESKAYØ6, Test 5)

Tests 4 and 5 check both the basic floating-point instructions and the accelerated floating-point instructions. Arithmetic and reserved operand exceptions pertaining to floating-point instructions are also tested. Since the FPA takes part in the execution of MULL2 and MULL3, the tests also check these instructions. The floating-point accelerator is turned off for test 4 and on for test 5.

Typical Error Messages for Test 4

```
********   CPU CLUSTER EXERCISER (ZZ-ESKAY) -- 9.Ø   ********

PASS 1  TEST 4  SUBTEST 1  ERROR 2  2Ø-FEB-1978  11:26:ØØ.ØØ

HARD ERROR WHILE TESTING CPU: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 2

? RESULT PSL ERROR

     EXPECTED   ACTUAL

     ØØ1FØØE3   ØØ1FØØE1


INITIAL CONDITIONS:

PC ØØØØ442Ø PSL ØØ1FØØEF

OP CODE -- 4F WITH REGISTER INDIRECT OPERANDS


INITIAL REGISTERS RØ--R6:

RØ   ØØØØACØ4   R1   ØØØØAC1Ø   R2   ØØØØAC1C

R3   ØØØØØØØØ   R4   ØØØØØØØØ   R5   ØØØØØØØØ

R6   ØØØØØØØØ


SOURCE OPERAND DATA:


OPERAND 1

ØØØØ4Ø8Ø

OPERAND 2

ØØØØ4Ø8Ø

OPERAND 3

ØØØØØØØØ
```

Example 6-49 ESKAY Test 4, Subtest 1, Error 2

```
********   CPU CLUSTER EXERCISER (ZZ-ESKAY) -- 9.Ø   ********

PASS 1  TEST 4  SUBTEST 1  ERROR 7  2Ø-FEB-1978  11:26:ØØ.ØØ

HARD ERROR WHILE TESTING CPU:  INSTRUCTION TEST ERROR

? ERROR IN TEST CASE NUMBER: 7

? RESULT PSL ERROR

     EXPECTED   ACTUAL

     ØØ1FØØE5   ØØ1FØØE4

INITIAL CONDITIONS:

PC ØØØØ4695 PSL ØØ1FØØEB

OP CODE -- 71 WITH REGISTER INDIRECT OPERANDS


INITIAL REGISTERS RØ--R6:

RØ   ØØØØACØ4   R1   ØØØØAC14   R2   ØØØØØØØØ

R3   ØØØØØØØØ   R4   ØØØØØØØØ   R5   ØØØØØØØØ

R6   ØØØØØØØØ


SOURCE OPERAND DATA:


OPERAND 1

ØØØØ4Ø8Ø

ØØØØØØØØ

OPERAND 2

ØØØØ4Ø8Ø

ØØØØØØØØ
```

Example 6-5Ø ESKAY Test 4, Subtest 1, Error 7

```
********   CPU CLUSTER EXERCISER (ZZ-ESKAY) -- 9.Ø   ********

PASS 1  TEST 4  SUBTEST 1  ERROR 24  2Ø-FEB-1978  11:26:ØØ.ØØ

HARD ERROR WHILE TESTING CPU: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 24

? RESULT PSL ERROR

     EXPECTED   ACTUAL

     ØØ1FØØE8   ØØ1FØØEØ


INITIAL CONDITIONS:

PC ØØØØ4873 PSL ØØ1FØØEF

OP CODE -- WITH REGISTER INDIRECT OPERANDS

INITIAL REGISTERS RØ--R6

RØ   ØØØØACØ4   R1   ØØØØAC14   R2   ØØØØAC1D

R3   ØØØØAC2D   R4   ØØØØAC39   R5   ØØØØØØØØ

R6   ØØØØØØØØ


SOURCE OPERAND DATA:

OPERAND 1

FFFE4FFF

FFFFFFFF

OPERAND 2

FF

OPERAND 3

ØØØØ4Ø8Ø

ØØØØØØØØ
```

Example 6-51 ESKAY Test 4, Subtest 1, Error 24

A lengthy detailed description of this type of error report has been supplied in Paragraph 6.4.2. Using that description as a reference, interpretations of the preceding three error reports follow.

Interpretation of Example 6-49

The instruction being tested is

```
ACBF (RØ), (R1), (R2), displacement

BREAKING DOWN FURTHER--
ACBF (ACØ4), (AC1Ø), (AC1C), displacement

BREAKING DOWN FURTHER--
ACBF 4Ø8Ø, 4Ø8Ø, Ø, displacement
      ↓     ↓     ↓
   limit  addend  index
```

Interpretation of Example 6-5Ø

The instruction being tested is

```
CMPD (RØ), (R1)

BREAKING DOWN FURTHER--
CMPD (ACØ4, (AC14)

BREAKING DOWN FURTHER--
CMPD 4Ø8Ø, 4Ø8Ø
      ↓     ↓
   source  destination
```

Interpretation of Example 6-51

Going through a similar analysis

```
EMODD (RØ), (R1), (R2), integer, fraction

EMODD   FFFE4FFF            ØØØØ4Ø8Ø

        FFFFFFFF,   FF,     ØØØØØØØØ, integer, fraction
            ↓       └──┐        ↓
        floating-      │    multiplicand
        point multiplier│
                       ↓
              floating-point
              multiplier extension
```

Typical Error Messages for Test 5

```
********   CPU CLUSTER EXERCISER (ZZ-ESKAY) -- 9.Ø   ********

PASS 1  TEST 5  SUBTEST 2  ERROR 7  2Ø-FEB-1978  11:26:ØØ.ØØ

HARD ERROR WHILE TESTING CPU: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 7

? INCORRECT TRACE TRAP PSL

     EXPECTED   ACTUAL

     ØØ1FØØF5   ØØ1FØØF4


INITIAL CONDITIONS:

PC ØØØØ4695 PSL ØØ1FØØFB

OP CODE -- 71 WITH REGISTER INDIRECT OPERANDS

INITIAL REGISTERS RØ--R6:

RØ   ØØØØACØ4   R1   ØØØØAC14   R2   ØØØØØØØØ

R3   ØØØØØØØØ   R4   ØØØØØØØØ   R5   ØØØØØØØØ

R6   ØØØØØØØØ


SOURCE OPERAND DATA:

OPERAND 1

ØØØØ4Ø8Ø

ØØØØØØØØ

OPERAND 2

ØØØØ4Ø8Ø

ØØØØØØØØ
```

Example 4-52 ESKAY Test 5, Subtest 2, Error 7

```
********   CPU CLUSTER EXERCISER (ZZ-ESKAY) -- 9.Ø    ********
PASS 1   TEST 5   SUBTEST 8   ERROR 1ØØ   2Ø-FEB-1978   11:26:ØØ.ØØ
HARD ERROR WHILE TESTING CPU: FLOATING NORMALIZE SUBTEST

? ERROR IN TEST CASE NUMBER: 113
     EXPECTED   ACTUAL
     FFF849FF   ØØØØ3F8Ø

TEST AT PC: ØØØØ9A87  ADDF3 RØ, R2, R4
     RØ  ØØØØ4Ø8Ø
     R2  ØØØØCØ4Ø
```

Example 6-53 ESKAY Test 5, Subtest 8, Error 1ØØ

```
********   CPU CLUSTER EXERCISER (ZZ-ESKAY) -- 9.Ø    ********
PASS 1   TEST 5   SUBTEST 8   ERROR 1Ø1   2Ø-FEB-1978   11:26:ØØ.ØØ
HARD ERROR WHILE TESTING CPU: FLOATING NORMALIZE SUBTEST

? ERROR IN TEST CASE NUMBER: 161
     EXPECTED   ACTUAL
     ØØØØ4Ø4Ø   ØØØØ4ØØØ
     ØØØØØØØØ   ØØØØØØØØ
TEST AT PC: ØØØØ9B11  ADD3 RØ, R2, R4
     R1  ØØØØCØØØ
     R2  FFFF4D7F
     R3  ØØØØEØØØ
```

Example 6-54 ESKAY Test 5, Subtest 8, Error 1Ø1

The interpretation of Example 6-52 is similar to that already given for Example 6-47.

Examples 6-53 and 6-54 are for the FLOATING NORMALIZE SUBTEST and differ from the standard CITS printouts as follows:

1. Both printouts give the instructions under test and their operands, i.e.,

   ```
   ADDF3  RØ, R2, R4
   ADD3  RØ, R2, R4
   ```

2. The operand data is listed directly under the TEST AT PC statements.

3. The EXPECTED and ACTUAL data in both cases reference the contents of R4 (R4, by definition, specifies the destination operand).

6.4.5 Operand Specifier Dependent Floating-Point Test (ESKAYØ7, Test 6)

Not yet implemented.

6.4.6 Decimal Strings Module (ESKAXØ8, Test 7)

This module tests the microcode and hardware used for decimal string execution.

Interpretation of Example 6-55

The error printouts coming from this test are designed like those of test 5. An overall interpretation has already been described in the test 2 writeup.

Analysis should show the instruction under test to be

```
ADDP6  (RØ),    (R1),    (R2),    (R3),    (R4),    (R5)
         |        ↓        |        ↓        |        ↓
         |     add1addr    |     add2addr    |     sumaddr
         ↓                 ↓                 ↓
      add1len           add2len           sumlen
```

with ADDRESSES REFERENCED shown under INITIAL REGISTERS RØ--R6 and OPERAND DATA as indicated.

6.4.7 EDITPC Operators Module (ESKAYØ9, Test 8)

This module tests the EDITPC microcode and associated hardware.

Typical Error Message

```
********   CPU CLUSTER EXERCISER (ZZ-ESKAY) -- 9.Ø   ********

PASS 1  TEST 7  SUBTEST 2  ERROR 26  2Ø-FEB-1978  11:26:ØØ.ØØ

HARD ERROR WHILE TESTING CPU: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 26

? INCORRECT TRACE TRAP PSL

     EXPECTED   ACTUAL

     ØØ1FØØF8   ØØ1FØØFØ


INITIAL CONDITIONS:

PC ØØØØ4585 PSL ØØ1FØØFF

OP CODE -- 21 WITH REGISTER INDIRECT OPERANDS


INITIAL REGISTERS RØ--R6

RØ   ØØØØACØ4   R1   ØØØØACØE   R2   ØØØØAC1B

R3   ØØØØAC25   R4   ØØØØAC35   R5   ØØØØAC3F

R6   ØØØØØØØØ


SOURCE OPERAND DATA:

OPERAND 1

ØØØ9

OPERAND 2

12, 34, 56, 78, 9C

OPERAND 3

ØØØE

OPERAND 4

ØØ, ØØ, ØØ, 77, 77, 77, 77, 7E

OPERAND 5

ØØ1F
```

Example 6-55 ESKAY Test 7, Subtest 2, Error 26

Typical Error Message

```
********   CPU CLUSTER EXERCISER (ZZ-ESKAY) -- 9.Ø   ********

PASS 1  TEST 8  SUBTEST 1  ERROR 48  2Ø-FEB-1978  11:26:ØØ.ØØ

HARD ERROR WHILE TESTING CPU: INSTRUCTION TEST ERROR


? ERROR IN TEST CASE NUMBER: 48

? RESULT PSL ERROR

     EXPECTED   ACTUAL

     ØØ1FØØE5   ØØ1FØØE4


INITIAL CONDITIONS:

PC ØØØØ485A PSL ØØ1FØØEB

OP CODE -- 38 WITH REGISTER INDIRECT OPERANDS


INITIAL REGISTERS RØ--R6:

RØ   ØØØØACØ4   R1   ØØØØACØE   R2   ØØØØAC1E

R3   ØØØØAC2F   R4   ØØØØØØØØ   R5   ØØØØØØØØ

R6   ØØØØØØØØ

SOURCE OPERAND DATA:

OPERAND 1

ØØØØF

OPERAND 2

ØØ, ØØ, ØØ, ØØ, ØØ, ØØ, ØØ, ØD

OPERAND 3

4Ø, 4Ø, 43, 25, Ø4, 9F, 46, 1Ø, ØØ
```

Example 6-56 ESKAY Test 8, Subtest 1, Error 48

Interpretation of Example 6-56
The error printouts coming from this test are designed like those described in Paragraph 6.4.2.

Therefore, analysis should show the instruction under test to be

```
EDITPC   (RØ),    (R1),    (R2),    (R3)
           ↓        │        ↓        │
         srclen     │     pattern     │
                    ↓                 ↓
                 srcaddr           dstaddr
```

with ADDRESSES REFERENCED shown under INITIAL REGISTERS RØ--R6 and OPERAND DATA as indicated.

6.4.8 Character String Instructions Test Module (ESKAY1Ø, Test 9)

Not yet implemented.

6.4.9 Privileged Instruction Exception Test (ESKAY11, Test 1Ø)

Not yet implemented.

6.5 ESKAZ DESCRIPTION

6.5.1 Memory Management Test Module (ESKAZØ3, Test 1)

The object of this test is to test memory management on a VAX-11/78Ø CPU. Memory management is that part of the CPU which checks protection on memory references, performs virtual to physical address translation, monitors updates to pages of memory (with the modify bit of the page table entry), and resolves unaligned data references. These functions are tested by making many different kinds of references to see that they work. Working is defined as: reading or writing the correct data, leaving the contents of adjacent addresses unaffected, setting the M bit on the first write to a page, and faulting if required. Upon detecting a failure, the test issues an error report containing the failure symptom (e.g., unexpected fault, wrong condition codes) and the circumstances surrounding the failure (instruction and address under test, expected and received data, etc.).

The test is organized in six subsections, each testing some area of memory management functionality.

1. Valid read and write -- The intent is to quickly verify that the basic functions work. Longword aligned reads and writes are performed to each address space (PØ, P1, and System). This process performs initial checks of reading and writing, physical address translation in each address space, translation buffer loading, and setting of the modify bit.

2. Length register boundary checks - References are made just before and just beyond each of the length boundaries to verify the length boundary checks.

3. Page Table Entry (PTE) combinations -- This subsection changes privilege modes to kernel, exec, super, and user. It makes references to pages mapped with each access code, and with the PTEs both valid and invalid, to verify the access privilege checks.

4. Size with PTE combinations -- The size of the access is varied from byte to quadword and tried with PTE combinations.

5. Page boundary checks -- The size of the access and the position of the access with respect to a page boundary are varied.

6. IB references with PTE combinations -- This subsection attempts to make instruction buffer references while varying the protection of the referenced page.

6.5.1.1 Memory Management Test General Flow

Initialization -- Three buffer areas are requested from the supervisor, one each from PØ, Pl, and system spaces. A control block (BVAS) is loaded with their addresses, and with the addresses of three other buffer areas, which are on the last page in each space.

Execution of Subsections -- A loop selects and executes each of the subsections, as follows.

1. Select an SCC -- The entry in the subsection description table associated with the current subsection is selected. It includes a pointer to a Setup Command Chain (SCC). There is an SCC for each subsection. The execution section includes six nested loops, varying access size, address space, and operand alignment. The SCC contains start and end limits for these loops. For instance, the SCC for the fourth loop, page boundary checks, specifies varying access size from byte to quadword, varying the offset from a page boundary from 8 bytes before through 1 byte after the page boundary, and varying address space from PØ to system.

2. Create defaults -- Defaults are provided for any variables not specified in the SCC.

3. Execute subsection -- A procedure is called that will make test references, varying each reference variable specified in the SCC across the range.

Clean up -- At the end of the test, all buffers are returned and control returns to the dispatch routine in the supervisor.

6.5.1.2 Memory Management Test, Subsection Flow

Loop start -- All reference parameters that will vary are loaded with initial values specified in the SCC.

Execute -- The test reference described by the current state of all the reference variables is made.

Increment -- The next value of the most rapidly varying parameter is loaded. If its range has been covered, it is set to its initial value and the next variable is changed.

Loop -- If the slowest varying reference parameter has completed its range, the subsection is complete. Otherwise, the next reference is made.

6.5.1.3 Test Reference Execution

Initialize -- The control blocks for this section (MRDB and TCB) are set up.

Decode -- A CITS (CITS_DECODE) is called to decode the test instruction.

Simulate -- The test reference is simulated, and the expected results are loaded into the MRDB.

Setup -- Another CITS routine (CITS_SETUP) is called to initialize the data areas, general register, and stack for the test instruction.

Map -- The address of the test reference is mapped according to the variables controlling page validity and accessibility.

Probe -- A probe is made to the test address in order to verify the mapping, and the results are compared with the simulated results.

Execute -- The test reference is made.

Remap -- The test address mapping is reset to allow all access, and the result maps are copied and checked.

Data Check -- CITS_CHECK is called to check the results of the test instruction.

Loop -- The flow from setup is repeated for various translation buffer states.

Return.

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 1  ERROR 2Ø212  28-MAY-1978  Ø8:31:Ø1.89

HARD ERROR WHILE TESTING CPU: LENGTH REGISTER BOUNDARY


ERROR OCCURRED DURING: ACCESS OR ACCESS CHECK

ERROR: PAGE TABLE ENTRY WAS MODIFIED


TESTED     PSL    MODE    ACCESS    ACCESS

ADDRESS    PRV    CUR     TYPE      SIZE

8ØØØ1FF8   K      K       R         L
        PTE        PTE        PROTEC-   V-BIT   M-BIT   ACCESS   ALLOWED
                              TION

        ADDRESS    VALUE      CODE      STATE   STATE   K   E    S  U

SPTE1:  ØØØ1F43C   ØØØØØØ98   ØØØ3      VAL     CLR     R   NO   NO NO

SYS BASE   SYS LENGTH

REG  REG

ØØØ1F4ØØ   ØØØØØØ22


EXP DATA   ACT DATA   PTE MODIFIED

ØØØØØØ98   ØØØØØØ98   SPTE1:
```

Example 6-57 ESKAZ Test 1, Subtest 1, Error 2Ø212

```
********   CPU CLUSTER EXERCISER -- 9.Ø   ********

PASS 1  TEST 1  SUBTEST 1  ERROR 2Ø213  28-MAY-1978  Ø8:5Ø:58.91

HARD ERROR WHILE TESTING CPU: LENGTH REGISTER BOUNDARY

ERROR OCCURRED DURING: ACCESS OR ACCESS CHECK

ERROR: MODIFY BIT ERROR

TESTED      PSL   MODE   ACCESS   ACCESS   TESTED   OPERAND

ADDRESS     PRV   CUR    TYPE     SIZE     INSTR    NO.

6Ø13ØØ1F8   K     K      W        L        MOV      Ø2

        PTE        PTE        PROTEC-   V-BIT  M-BIT  ACCESS  ALLOWED
                              TION

        ADDRESS    VALUE      CODE      STATE  STATE  K  E    S  U

SPTE1:  ØØØ1F48Ø   ØØØØØØ94   ØØØ2      VAL    SET    RW NO   NO NO


T-BUFF

STATE

MISS


PPTE1: ØØØ1F2ØØ  ØØØØØØ94     ØØØ2      VAL    SET    RW NO   NO NO

 HIT

SYS BASE   SYS LENGTH

REG        REG

ØØØ1F4ØØ   ØØØØØØ22

P1 BASE    P1 LENGTH

REG        REG

7FCØ1AØØ   ØØ1ØØØØF

EXPECTED   PTE

STATE

SET        SPTE1:
```

Example 6-58 ESKAZ Test 1, Subtest 1, Error 2Ø213

Interpretation of Example 6-57

1. First, a discussion of ERROR 20212

   All error numbers consist of 3 bytes with a breakdown as follows:

   a. The left byte (2) defines the location within the test where the error was encountered

      where,

      Ø = Subtest

      1--6 = Subsection defined prior to the preceding examples (Paragraph 6.5.1).

   b. The middle byte (Ø2) indicates the action being taken by the test at the time of error

      where,

      Ø = SETUP
      1 = PROBE/PROBE CHECK
      2 = ACCESS/ACCESS CHECK
      3 = ACCESS DATA CHECK
      4 = FINAL SETUP

   c. The right byte (12) is indicative of the error itself (it is used by the test software to determine what gets printed at error report time).

So, the ERROR number tells us that we were in the LENGTH REGISTER BOUNDARY subsection (2) performing an ACCESS/ACCESS CHECK (Ø2), when we got a message saying PAGE TABLE ENTRY WAS MODIFIED (12).

2. The ADDRESS UNDER TEST was 8ØØØ1FF8.

3. The PREVIOUS and CURRENT MODES in the PSL at the test time are shown as K K

   where,

   K = Kernel
   E = Executive
   S = Supervisor
   U = User

4. The ACCESS occurring at the time of the error is shown as R

   where,

   R = Read
   W = Write
   M = Modify

5. The SIZE of the access is shown as L

   where,

   L = Longword
   B = Byte
   W = Word
   Q = Quadword

6. A discussion of the line labeled SPTE1 follows.

   In the given example we have only one line of information, but depending on the set of circumstances there can be more than one line (as shown in Example 6-58).

   The lines and combinations that can appear are as follows:

   SPTE1: Page 1 system page table
   SPTE2: Page 2 system page table
   PPTE1: Page 1 processor page table
   PPTE2: Page 2 processor page table

   where,

   Page 1 = page number of the address of the lowest byte of the reference address as determined by either the Base Virtual Address (BVA) or, if the position is negative, by the BVA + position.

   Page 2 = the next page.

   NOTE

   A reference may be either entirely within PAGE 1 or PAGE 2, or it may cross over.

   In our example, the PAGE 1 system page table is being ACCESSED.

7. The PHYSICAL ADDRESS of the PTE is 1F43C.

> **NOTE**
> **The PTE is the medium of translation of all virtual addresses to physical addresses.**

8. The PTE VALUE represents the contents of the PTE or 98.

   The PTE content comprises 4 fields.

   a. Page Frame Number (PFN) -- Bits <2Ø:ØØ>

      This is the upper 21 bits of the physical address of the base of the page.

   b. Modify bit -- Bit <26>

   c. Protection -- Bits <3Ø:27>

   d. Valid -- Bit <31>

9. The PROTECTION CODE for the page accessed was 3.

1Ø. Chapter 5 of the <u>VAX-11 System Reference Manual</u> gives an analysis of a protection code meaning.

   To ease the strain of searching through the <u>VAX-11 System Reference Manual</u>, the protection code breakdown is shown under ACCESS ALLOWED as R, NO, NO, NO.

   This states that the page being accessed can be READ in kernel mode, and cannot be accessed in any other mode.

> **NOTE**
> **A W under this column would indicate that the page can be written in a given mode.**

11. The state of the valid bit (V-BIT) is VAL

   where,

   VAX = 1 (valid)
   INV = Ø (invalid)

12. The state of the modify bit (M-BIT) is CLR

   where,

   CLR = Ø (no modify)
   SET = 1 (modify)

13. The content of the SYSTEM BASE REGISTER was 1F4ØØ and the content of the SYSTEM LENGTH REGISTER was 22.

    These 13 items represent the SETUP portion of the error report (i.e., what were all the initial conditions, or states, at the time of the error).

    The SYS BASE REG and SYS LGTH REG printouts always occur as parts of the SETUP. As a function of the TESTED ADDRESS value, one or two other printouts will occur additionally as follows:

    a. If bit 31 is clear and bit 3Ø is set, the message includes

```
P1 BASE   P1 LENGTH

REG       REG
```

    b. If bit 31 is clear and bit 3Ø is clear, the message includes

```
PØ BASE   PØ LENGTH

REG       REG
```

14. The ERROR portion of the printout shows further proof of the ERROR: PAGE TABLE ENTRY WAS MODIFIED statement by showing the EXPECTED and ACTUAL DATA and the PTE MODIFIED.

Example 6-58 is similar to Example 6-57 except that it shows additional information which reflects circumstances at the time of the error.

The new items in Example 6-58 are:

a. TESTED INSTR is a MOV
b. The OPERAND NO. in question is Ø2 (i.e., the DST).
c. The translation buffer state (TB-STATE) is listed

   where,

   HIT = 1
   MISS = Ø.

6.6 COMPATIBILITY MODE INSTRUCTION TEST (ESKAZØ3, TEST 2)

6.6.1 Instructions Tested

Most of the instructions provided by the compatibility mode hardware are exercised using various data patterns and address modes (Figure 6-3). These instructions are listed in Table 6-12.

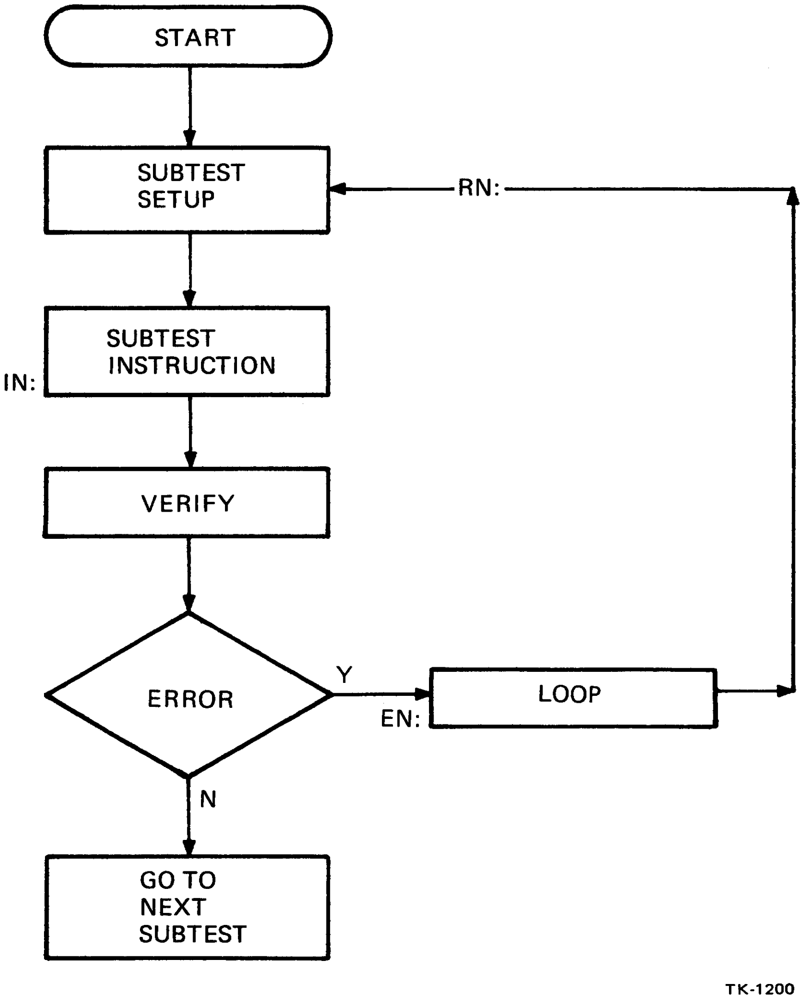


Figure 6-3 Compatibility Mode Instruction Module Subtest Structure

Table 6-12 Compatibility Mode Instructions Provided by Compatibility Mode Hardware and Exercised by ESKAZ Test 2

Op Code (8)	Mnemomic	Op Code (8)	Mnemonic
.Ø55DD	ADC(B)	ØØØ1DD	JMP
.Ø6SSDD	ADD	ØØ4RDD	JSR
.Ø63DD	ASL(B)	.1SSDD	MOV(B)
.Ø62DD	ASR(B)	.Ø54DD	NEG(B)
.4SSDD	BIC(B)	.Ø61DD	ROL(B)
.5SSDD	BIS(B)	.Ø6ØDD	ROR(B)
.3SSDD	BIT(B)	.ØØ2ØR	RTS
4ØØ-3777	BRANCHES(*)	ØØØØØ6	RTT
1ØØØØØ-3777	BRANCHES(**)	.Ø56DD	SBC(B)
.Ø5ØDD	CLR(B)	Ø77RNN	SOB
.2SSDD	CMP(B)	16SSDD	SUB
.Ø51DD	COM(B)	ØØØ3DD	SWAB
24Ø-277	CND CODES(***)	ØØ67DD	SXT
.Ø53DD	DEC(B)	.Ø57DD	TST(B)
.Ø52DD	INC(B)	Ø74RSS	XOR

where,

(*)=BR, BNE, BEQ, BGE, BLT, BGT, BLE
(**)=BPL, BMI, BVC, BCC, BCS, BHI, BLOS, BHIS, BLO
(***)=CLC, CLV, CLZ, CLN, CCC, SEC, SEV, SEZ, SEN, SCC

The instructions provided by the compatibility mode hardware that have not yet been included in this test are listed in Table 6-13.

Table 6-13 Compatibility Mode Instructions Not Yet Tested

Op Code(8)	Mnemonic
Ø72RSS	ASH
Ø73RSS	ASHC
Ø71RSS	DIV
1Ø65SS	MFPD
ØØ65SS	MFPI
1Ø66DD	MTPD
ØØ66DD	MTPI
Ø7ØRSS	MUL

NOTES

1. The test instruction is always tagged IN.

2. The error is always tagged EN.

3. The return point for looping is always tagged RN.

4. If more than one verification is made in a single subtest, the entries to subsequent checks are tagged AN, BN, etc.

5. If more than one error is included, subsequent errors are tagged E1N, E2N, E3N, etc.

The RTI instruction provided by compatibility mode hardware is not tested in this module. The compatibility mode entry/exit module (ESKAXØ2, Test Ø1) tests this instruction thoroughly.

6.6.2 Compatibility Mode Test Error Message Format

The following header is printed when an error is detected.

```
(PC)    (PSW)    (SP)    (R1)    (R2)    (R3)    (R4)
```

Interpretation of Compatibility Mode Test Error Message Format

(PC) Indicates the content of the program counter at the time of the error call. This is normally an address that is used to locate the error call statement in the failing subtest.

(PSW) Indicates the content of the processor status word at the time of the error call.

(SP) Indicates the content of the stack pointer (R6) at the time of the error.

NOTE

The error call will push the stack twice.

(R1) Indicates a mnemonic of the instruction under test e.g., MOVB, ASL . . . et al.

(R2) For single- and double-operand instructions, R2 normally contains the destination address.

(R3) For single- and double-operand instructions, R3 contains what the result (destination operand) actually was after the test instruction was executed.

(R4) For single- and double-operand instructions, R4 contains what the result (destination operand) should have been (S/B).

In some cases, the error information may deviate from that previously described but the program annotation for those subtests will describe the meaning of those entries that have been redefined.

The error call statement is encoded to print only the information relative to the particular function being tested. Interpretation of the error calls is shown below.

ERROR	Print all 7 columns
ERROR1	Print only column 1
ERROR2	Print columns 1, 2
ERROR3	Print columns 1, 2, 3
ERROR4	Print columns 1, 2, 3, 4
ERROR5	Print columns 1, 2, 3, 4, 5
ERROR6	Print columns 1, 2, 3, 4, 5, 6

6.6.3 Sample Error Message Explanation

```
********   CPU CLUSTER . . . . . . . . . . . . .      ********

PASS 1  TEST XX  SUBTEST 2Ø8 . . . . . . .

HARD ERROR WHILE TESTING CPU: COMPATIBILITY MODE . . . . .

(PC)        (PSW)       (SP)        (R1)  (R2)        (R3)        (R4)

ØØØØ8DDØ    ØØØØØØØ4    ØØØØA5AA    MOV   ØØØØØ4ØØ    ØØØØØØØØ    ØØØØFFFF
```

Example 6-59 ESKAZ Compatibility Mode Test Error

Interpretation of Example 6-59

8DDØ Represents the PC of the error call in the listing.

4 Represents the content of the PSW prior to the error call.

A5AA Represents the last position of the PDP-11 mode stack pointer (R6).

MOV Is a clue that the MOV instruction failed under test.

4ØØ Represents the address used by the destination mode portion of the MOV instruction.

ØØØØ Represents the actual content of the destination after instruction execution.

FFFF Represents what the content of the destination should have been after the MOV instruction was executed.

The listing is laid out with a subtitle printed at the top of each page. The operator can look through the program listing for subtest 2Ø8. The subtest description of 2Ø8 shows that a MOV instruction is tested with source mode 2 and destination mode 3.

6.6.4 Compatibility Mode Instruction Module Assumptions

Four compatibility mode trap instructions are used to control the execution of this test, as follows.

1. SUBTYPE Ø (SPL) Used as program end indicator.
2. SUBTYPE 2 (IOT) Used as next subtest indicator.

NOTE

Appears as SCOPE statement in listing.

3. SUBTYPE 3 (EMT) Used as error report indicator.

NOTE

Appears as ERROR + XX statement in listing.

4. SUBTYPE 4 (TRAP) Used as PSW reference indicator.

NOTE

Appears as TRAP + XX statement in listing.

It is assumed that the test performing the exercising of compatibility mode entry/exit conditions has been executed prior to this test, in which event, the compatibility mode trap instructions have been checked out.

APPENDIX A
GLOSSARY OF DIAGNOSTICS SOFTWARE TERMS

absolute (ABS) -- A program section (psect) attribute. An absolute psect contains only symbol definitions. It does not contribute binary code to the image. Therefore, it must have a zero-length memory allocation. The converse is relocatable (REL).

access mode -- Any of the four processor access modes in which software executes. Processor access modes are, in order, from most to least privileged and protected: kernel (mode Ø), executive (mode 1), supervisor (mode 2), and user (mode 3).

When the processor is in kernel mode, the executing software has complete control of, and responsibility for, the system. When the processor is in any other mode, the processor is inhibited from executing privileged instructions. The processor status longword contains the current access mode field. The operating system uses access modes to define protection levels for software executing in the context of a process. For example, the executive runs in kernel and executive modes and is most protected. The command interpreter is less protected and runs in supervisor mode. The debugger runs in user mode and is no more protected than normal user programs.

access type -- The way in which the processor accesses instruction operands. Access types are: read, write, modify, address, and branch.

alignment -- The address boundary at which a program section is based.

allocate a device -- To reserve a particular device unit for exclusive use. A user process can allocate a device only when that device is not allocated by any other process.

allocation -- The number of bytes of memory contributed by a program section to a particular module.

alphanumeric character -- An upper or lower case letter (A--Z, a--z), a dollar sign ($), an underscore (_), or a decimal digit (Ø--9).

ancillary control process (ACP) -- A process that acts as an interface between user software and an I/O driver. An ACP provides functions supplemental to those performed in the driver, such as file and directory management.

argument -- An independent value within a command statement that specifies where, or on what, the command will operate (e.g., address, data).

argument pointer -- General register 12 (R12). By convention, AP contains the address of the base of the argument list for procedures initiated using the CALL instructions.

assign a channel -- To establish the necessary software linkage between a user process and a device unit before a user process can transfer any data to or from that device. A user process requests the system to assign a channel and the system returns a channel number.

assembler -- A program that translates source language code, whose operations correspond directly to machine op codes, into object language code.

asynchronous system trap (AST) -- A software-simulated interrupt to a user-defined service routine. ASTs enable a user process to be notified asynchronously, with respect to its execution, of the occurrence of a specific event. If a user process has defined an AST routine for an event, the system interrupts the process and executes the AST routine when that event occurs. When the AST routine exits, the system resumes the process at the point where it was interrupted.

attributes -- Various characteristics that can be assigned by the programmer to each psect in a module (e.g., ABS).

base register -- A general register used to contain the address of the entry in a list, table, array, or other data structure.

block -- 1. The smallest addressable unit of data that the specified device can transfer in an I/O operation (512 contiguous bytes for most disk devices). 2. An arbitrary number of contiguous bytes used to store logically related status, control, or other processing information (i.e., process control block).

breakpoint -- In diagnostics, an address assigned through the diagnostic supervisor. When the PC equals the value of the breakpoint, control returns to the diagnostic supervisor.

boot (bootstrap) -- A program that loads another (usually larger) program into memory from a peripheral device.

buffer -- A temporary data storage area.

call frame -- A standard data structure built on the stack during a procedure call, starting from the location addressed by the FP to lower addresses, and popped off during a return from procedure (also called stack frame).

channel -- A logical path connecting a user process to a physical device unit. A user process requests the operating system to assign a channel to a device so that the process can transfer data to or from that device.

command file -- A file containing command strings.

command interpreter -- Procedure-based code to receive, syntax check, and parse commands typed by the user at a terminal or submitted in a command file.

command parameter -- The positional operand of a command delimited by spaces, such as a file specification, option, or constant.

command string -- A line, or a set of continued lines, normally terminated by typing the carriage return key containing a command, and optionally, information modifying the command. A complete command string consists of a command; its qualifiers, if any; its parameter (file specifications, for example), if any; and their qualifiers, if any.

concatenate (CON) -- A program section attribute. If a psect is concatenated, all psects of the same name yet from different modules are to be assigned contiguous addresses in the virtual address space. Each module can specify an independent alignment. The linker performs the necessary padding of zero bytes between contributions. The base alignment of the resulting concatenated psects is according to the greatest alignment granularity of all the contributions to the psect. For example, if the greatest alignment granularity of all contributors is a page, the psect is page-aligned; although, some contributors may be byte-aligned, others word-aligned, etc.

condition -- An exception condition detected and declared by software.

condition codes -- Four bits in the processor status word that indicate the results of the previously executed instruction.

condition handler -- A procedure that a process wants the system to execute when an exception condition occurs. When an exception condition does occur, the operating system searches for a condition handler. When it finds the condition handler, the operating system initiates the handler immediately. The condition handler may perform some act to change the situation that caused the exception condition and then continue execution of the process that incurred the exception condition. Condition handlers execute in the context of the process at the access mode of the code that incurred the exception condition.

context switching -- Interrupting the activity in progress and switching to another activity. Context switching occurs as one process after another is scheduled for execution. The operating system saves the interrupted process's hardware context in its hardware PCB using the save process context instruction, loads another process's hardware PCB into the hardware context using the load process context instruction, and schedules that process for execution.

cylinder -- The tracks at the same radius on all recording surfaces of a disk pack.

default -- Assumed value supplied when a command qualifier does not specifically override the normal command function; also, fields in a file specification that the system fills in when the specification is not complete.

default disk -- The system disk to which the system writes all files that the operator creates, by default. The default is used whenever a file specification in a command does not explicitly name a device.

delimiter -- A character or symbol used to separate or limit items within a command or data string. However, the delimiter is not a member of the string.

device -- The general name for any physical terminus or link connected to the processor that is capable of receiving, storing, or transmitting data. Card readers, line printers, and terminals are examples of record-oriented devices. Magnetic tape devices and disk devices are examples of mass storage devices. Terminal line interfaces and interprocessor links are examples of communications devices.

device interrupt -- An interrupt received on interrupt priority levels 16 through 23. Device interrupts can be requested only by devices, controllers, and memories.

device name -- The field in a file specification that identifies the device unit on which a file is stored. Device names also include the mnemonics that identify an I/O peripheral device in a data transfer request. A device name consists of a mnemonic followed by a controller identification letter (if applicable), followed by a unit number (if applicable). A colon (:) separates it from following fields.

direct I/O -- A mode of access to peripheral devices in which the program addresses the device registers directly, without relying on support from the operating system drivers.

drive -- The electro-mechanical unit of a mass storage device system on which a recording medium (disk cartridge, disk pack, or magnetic tape reel) is mounted.

driver -- The set of system code that handles physical I/O to a device.

entry mask -- A word (1) whose bits represent the registers to be saved or restored on a subroutine or procedure call using the call and return instructions, and (2) which includes trap enable bits.

entry point -- A location that can be specified as the object of a call. It contains an entry mask and exception enables known as the entry point mask.

event -- A change in process status or an indication of the occurrence of some activity that concerns an individual process or cooperating processes. An incident reported to the scheduler that affects a process's ability to execute. Events can be synchronous with the process's execution (a wait request, or they can be asynchronous (I/O completion). Some examples of events: swapping, wake request, page fault.

event flag -- A bit in an event flag cluster that can be set or cleared to indicate the occurrence of the event associated with that flag. Event flags are used to synchronize activities in a process or among many processes.

exception -- An event detected by the hardware (other than an interrupt or jump, branch, case, or call instruction) that changes the normal flow of instruction execution. An exception is always caused by the execution of an instruction or set of instructions, while an interrupt is caused by an activity in the system independent of the current instruction. There are three types of hardware exceptions: traps, faults, and aborts. Examples are: attempts to execute a privileged or reserved instruction; trace traps; compatibility mode faults; breakpoint instruction execution; and arithmetic traps such as overflow, underflow, and divide-by-zero.

exception condition -- A hardware- or software-detected event (other than an interrupt or jump, branch, case, or call instruction) that changes the normal flow of instruction execution.

exception dispatcher -- An operating system procedure that searches for a condition handler when an exception condition occurs. If no exception handler is found for an exception or condition, the image that incurred the exception is terminated.

executable (EXE) -- A program section attribute. The psect contains only instructions. This attribute provides the capability to separate instructions from read-only and read/write data. The linker uses this attribute in gathering psects and in the verification of the transfer address that must be present in an executable psect.

executable image -- An image that is capable of being run in a process. When run, an executable image is read from a file for execution in a process.

executive -- The generic name for the collection of procedures included in the operating system software that provides the basic control and monitor functions of the operating system.

file -- A logically related collection of data treated as a physical entity that occupies one or more blocks on a volume such as disk or magnetic tape. A file can be referenced by a name assigned by the user. A file normally consists of one or more logical records.

file specification -- A unique name for a file on a mass storage medium.

frame pointer -- General register 13 (R13). By convention, FP contains the base address of the most recent call frame on the stack.

global symbol -- A symbol defined in a module that is potentially available for reference by another module. The linker resolves (matches references with definitions) global symbols. Contrast with local symbol.

granularity -- The alignment of a contribution to a psect on a boundary. The alignment granularity may be byte, word, quadword, or page.

home block -- A block in the index file that contains the volume identification, such as volume label and protection.

image -- An image consists of procedures and data that have been bound together by the linker. There are three types of images: executable, sharable, and system.

index file -- The file on a FILES-11 volume that contains the access information for all files on the volume and enables the operating system to identify and access the volume.

interrupt -- An event (other than an exception or branch, jump, case, or call instruction) that changes the normal flow of instruction execution. Interrupts are generally external to the process executing when the interrupt occurs.

interrupt stack -- The system-wide stack used when executing in an interrupt service context. At any time, the processor is either in a process context executing in user, supervisor, executive, or kernel mode; or in system-wide interrupt service context operating with kernel privileges, as indicated by the interrupt stack and current mode bits in the PSL. The interrupt stack is not context-switched.

I/O function code -- A 6-bit value specified in a queue I/O request system service that describes the particular I/O operation to be performed (e.g., read, write, rewind).

library file -- A direct access file containing one or more modules of the same module type.

linked commands -- A group of independent commands connected together (linked) so as to form a single executable list of commands. Once initiated, the entire linked command list may be executed without further operator intervention.

linker -- A program that reads one or more object modules created by language processors and produces an executable image file, a sharable image file, or a system image file.

linking -- The resolution of external references between object modules used to create an image; the acquisition of referenced library routines, service entry points, and data for the image; and the assignment of virtual addresses to components of an image.

link map -- A link map shows the virtual memory allocation of the total program image. The link map is found in a program listing in the program section allocation synopsis.

literal -- An operand which is used immediately, without being translated to some other value. An operand which specifies itself.

literal argument -- An independent value within a command statement that specifies itself.

local symbol -- A symbol that is meaningful only to the module that defines it. Symbols not identified to a language processor as global symbols are considered to be local symbols. A language processor resolves (matches references with definitions) local symbols. They are known to the linker and cannot be made available to another object module. They can, however, be passed through the linker to the symbolic debugger. Contrast with global symbol.

logical block -- A block on a mass storage device identified by using the volume-relative address rather than the physical (device-oriented) address or the virtual (file-relative) address. The blocks that comprise the volume are labeled sequentially starting with logical block Ø.

macro -- A statement that requests a language processor to generate a predefined set of instructions.

memory management -- The system functions that include the hardware's page mapping and protection and the operating system's image activator and pager.

module -- A part of a program assembled as a unit. Modular programming allows the development of large programs in which separate parts share data and routines.

mount a volume -- To logically associate a volume with the physical unit on which it is loaded (an activity accomplished by system software at the request of an operator). Or, to load or place a magnetic tape or disk pack on a drive and place the drive on-line (an activity accomplished by a system operator).

object module -- The binary output of a language processor such as the assembler or a compiler, which is used as input to the linker.

operand -- a value (address or data) that is operated on, or with, by an instruction.

overlay (OVR) -- A program section attribute. If a psect is overlayed, all contributions to the psect have the same base address. The length of the psect is the size of the largest contribution. All contributions to an overlayed psect must have the same alignment.

page -- A set of 512 contiguous byte locations used as the unit of memory mapping and protection. Also, the data between the beginning of a file and a page marker, between two markers, or between a marker and the end of a file.

page frame number (PFN) -- The address of the first byte of a page in physical memory. The high-order 21 bits of the physical address of the base of a page make up the PFN.

page table entry (PTE) -- The data structure that identifies the location and status of a page of virtual address space. When a virtual page is in memory, the PTE contains the page frame number needed to map the virtual page to a physical page. When it is not in memory, the PTE contains the information needed to locate the page on secondary storage (disk).

parameter -- A parameter is the object of a command. It can be a file specification, a keyword option, or a symbol value passed to a command procedure. In diagnostics, parameters are usually operator-supplied answers to questions asked by a program concerning devices to be tested.

parameter switch -- A command qualifier. In diagnostics, it is preceded by a slash (/).

parser -- A procedure that breaks down the components of a command into structural forms.

physical address -- The address used by hardware to identify a location in physical memory or on directly addressable secondary storage devices such as disks. A physical memory address consists of a page frame number and the number of a byte within the page. A physical disk block address consists of a cylinder or track and sector number.

physical block -- A block on a mass storage device referred to by its physical (device-oriented) address rather than a logical (volume-relative) or virtual (file-relative) address.

position independent code (PIC) -- A program section attribute. The contents of the psect do not depend on a specific location in virtual memory. The converse is nonposition independent code (NOPIC).

priority -- The rank assigned to an activity that determines its level of service. For example, when several jobs contend for system resources, the job with the highest priority receives service first.

program section -- A portion of a module. The assembler creates a number of program sections (psect) within a module, according to directives by the program developer. In addition, any code that precedes the first defined program section is placed in the BLANK program section by the assembler.

Through program sectioning the program developer controls the virtual memory allocation of a program. Any program attributes established by the program section directive are passed on to the linker. Thus, program sections can be declared as read only, nonexecutable, etc. See the VAX-11 MACRO Language Reference Manual for an explanation of the various program section attribute functions.

In the diagnostic programs, each test is given a separate program section.

prompt -- A program's typed out response to and/or request for operator action.

qualifier -- A portion of a command string that modifies a command verb or command parameter by selecting one of several options. A qualifier, if present, follows the command verb or parameter to which it applies and is in the format: /qualifier:option. For example, in the command string "PRINT <filename> /COPIES:3", the COPIES qualifier indicates that the user wants three copies of a given file printed.

queue -- A list of commands or jobs waiting to be processed.

queue I/O -- A mode of access to peripheral devices in which a program calls on driver routines provided by the VMS operating system or the diagnostic supervisor to transfer data.

radix -- The base of the number system currently in use.

readable (RD) -- A program section attribute. The contents of the psect can be read at the execute time. The converse is nonreadable (NORD).

record -- A collection of adjacent items of data treated as a unit. A logical record can be of any length whose significance is determined by the programmer. A physical record is a device-dependent collection of contiguous bytes such as a block on a disk, or a collection of bytes sent to or received from a record-oriented device.

relocatable (REL) -- A program section attribute. The psect must be assigned a base address by the linker. This psect can contain code and/or data.

script file -- A line-oriented ASCII file that contains a list of commands.

section -- A group of tests in a diagnostic program that may be selected by the operator.

sector -- A portion of a track on the surface of a disk. On a VAX-11 system, each track on a disk is normally divided into 22 sectors.

semantics -- The interpretation of and relation between commands or command symbols.

sharable image -- An image that has all of its internal references resolved, but which must be linked with an object module(s) to produce an executable image. A sharable image cannot be executed. A sharable image file can be used to contain a library of routines. A sharable image can be installed as a global section by the system manager.

stack -- An area of memory set aside for temporary storage, or for procedure and interrupt service linkages. A stack uses the last-in, first-out concept. As items are added to (pushed on) the stack, the stack pointer decrements. As items are retrieved from (popped off) the stack, the stack pointer increments.

stack frame -- A standard data structure built on the stack during a procedure call, starting from the location addressed by the FP to lower addresses, and popped off during a return from procedure. Also called call frame.

stack pointer -- General register 14 (R14). SP contains the address of the top (lowest address) of the processor-defined stack. Reference to SP will access one of the five possible stack pointers: kernel, executive, supervisor, user, or interrupt, depending on the value in the current mode and interrupt stack bits in the Processor Status Longword (PSL).

standalone mode -- A diagnostic program environment in which the program and the diagnostic supervisor run without the VMS operating system. The operator must use the console terminal when running diagnostics in the standalone mode, and no other users have access to the system.

symbolic argument -- An argument within a command that refers to another value.

syntax -- The rules governing a command language structure. The way in which command symbols are ordered to form meaningful statements.

syntactic unit -- An item contained within a command statement (e.g., an argument, a qualifier).

system image -- The image that is read into memory from secondary storage when the system is started up.

test -- A unit of a diagnostic program that checks a specific function or portion of the hardware.

time stamp -- A statement of the time of day at which a specific event occurred.

track -- A collection of blocks at a single radius on one recording surface of a disk.

trap -- An exception condition that occurs at the end of the instruction that caused the exception. The PC saved on the stack is the address of the next instruction that would normally have been executed. All software can enable and disable some of the trap conditions with a single instruction.

unit record device -- A device such as a card reader or line printer.

unwind the call stack -- To remove call frames from the stack by tracing back through nested procedure calls using the current content of the FP register and FP register content stored on the stack for each call frame.

UUT (unit under test) -- The device or portion of the computer hardware being tested by a diagnostic program.

virtual block number -- A number used to identify a block on a mass storage device. The number is a file-relative address rather than a logical (volume-oriented) or physical (device-oriented) address. The first block in a file is always virtual block number one.

writable (WRT) -- A program section attribute. The content of the psect can be modified at execute time. The converse is nonwritable (NOWRT).

VAX-11/780 DIAGNOSTIC SYSTEM
TECHNICAL DESCRIPTION
EK-DS780-TD-001

Reader's Comments

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our publications.

What is your general reaction to this manual? In your judgment is it complete, accurate, well organized, well written, etc.? Is it easy to use? ______________________________

____

____

____

What features are most useful? ______________________________

____

____

____

What faults or errors have you found in the manual? ______________________________

____

____

____

Does this manual satisfy the need you think it was intended to satisfy? ______________

Does it satisfy *your* needs? ______________ Why? ______________

____

____

____

□ Please send me the current copy of the *Technical Documentation Catalog*, which contains information on the remainder of DIGITAL's technical documentation.

Name ______________	Street ______________
Title ______________	City ______________
Company ______________	State/Country ______________
Department ______________	Zip ______________

Additional copies of this document are available from:

Digital Equipment Corporation
444 Whitney Street
Northboro, Ma 01532
Attention: Communications Services (NR2/M15)
Customer Services Section

Order No. EK-DS780-TD-001

— — — — — — — — — — — Fold Here — — — — — — — — — — — —

— — — — — — — — Do Not Tear - Fold Here and Staple — — — — — — — —

Digital Equipment Corporation
Communications Development and Publishing
1925 Andover Street
Tewksbury, Massachusetts 01876