CRAY Y-MP™, CRAY X-MP EA™, CRAY X-MP™, and CRAY-1® Computer Systems

UNICOS® On-line Diagnostic Maintenance Manual

SMM-1012 C

Cray Research, Inc.

Due to space restrictions, the following abbreviations are used in place of the specific system names:

CX/1	Includes all models of the CRAY X-MP and CRAY-1 computer systems
CEA	Includes all models of the Extended Architecture (EA) series, including the CRAY Y-MP and CRAY X-MP EA computer systems
CRAY-2	Includes all models of the CRAY-2 computer system
CX/CEA	Includes all models of the CRAY X-MP computer systems plus all models of the CRAY Y-MP and CRAY X-MP EA computer systems. It does not include the CRAY-1 computer systems.

Requests for copies of Cray Research, Inc. publications should be sent to the following address:

Cray Research, Inc.
Distribution Center
2360 Pilot Knob Road
Mendota Heights, MN 55120

NEW AND ENHANCED FEATURES

This UNICOS release 5.0 overview describes the new and enhanced features contained in the CRAY Y-MP, CRAY X-MP EA, CRAY X-MP, and CRAY-1 Computer Systems UNICOS On-line Diagnostic Maintenance Manual, CRI publication SMM-1012.

With UNICOS 5.0, there is support for diagnostics that run on CRAY Y-MP and CRAY X-MP EA computer systems, as follows:

- Y-mode (32-bit addressing), available only as indicated in appendix A, On-line Diagnostic Programs
- X-mode (24-bit addressing), unless otherwise indicated

Specific new and enhanced features are as follows:

Feature	Status	Section	Description
cleario	Enhanced	6	Adds support for the Operator Workstation (OWS) and the CRAY Y-MP and CRAY X-MP EA computer systems.
dsdiag	Enhanced	6	Adds support for the OWS and the CRAY Y-MP and CRAY X-MP EA computer systems.
donut	New	5	On-line disk maintenance program
offmon	New	2	Off-line confidence monitor
olcfpt	New	3	Comprehensive floating-point instructions and data test
olcm	New	3	Common memory test
olcrit	Enhanced	3	Adds cluster selection.
oldmon	New	5	Down CPU monitor
olhpa	Enhanced	7	Adds support for DD-40 disk drives, SSD errors, and the CRAY Y-MP and CRAY X-MP EA computer systems.

Feature	Status	Section	Description
olibuf	New	3	Instruction buffer test
olsbt	New	3	On-line semaphore, shared B and shared T register test
runsequence	Enhanced	7	Adds examples of sequence files used for testing and file cleanup. Invokes one less shell.
unitap	New	5	On-line magnetic tape test

RECORD OF REVISION CRAY RESEARCH, INC. PUBLICATION NUMBER SMM-1012

Each time this manual is revised and reprinted, all changes issued against the previous version are incorporated into the new version and the new version is assigned an alphabetic level.

Every page changed by a reprint with revision has the revision level in the lower righthand corner. Changes to part of a page are noted by a change bar in the margin directly opposite the change. A change bar in the margin opposite the page number indicates that the entire page is new. If the manual is rewritten, the revision level changes but the manual does not contain change bars.

Requests for copies of Cray Research, Inc. publications should be directed to the Distribution Center and comments about these publications should be directed to:

CRAY RESEARCH, INC.
1345 Northland Drive
Mendota Heights, Minnesota 55120

Revision	Description
	September 1986 - Original printing. This printing supports the on-line diagnostic tests that run under the Cray operating system UNICOS, release 2.0, on the CRAY X-MP and CRAY-1 computer systems. The on-line diagnostic tests for CRAY-1 computer systems are not available for UNICOS release 2.0. All trademarks are listed in the record of revision.
A	June 1987 - Rewrite. This printing supports the on-line diagnostic tests that run under the Cray operating system UNICOS, release 3.0, on CRAY X-MP and CRAY-1 computer systems.
B	July 1988 - Rewrite. This printing supports the on-line diagnostic tests that run under the Cray operating system UNICOS, release 4.0, on CRAY Y-MP, CRAY X-MP EA, CRAY X-MP, and CRAY-1 computer systems.
C	March 1989 - Rewrite. This printing supports the on-line diagnostic tests that run under the Cray operating system UNICOS, release 5.0, on CRAY Y-MP, CRAY X-MP EA, CRAY X-MP, and CRAY-1 computer systems.

PREFACE

This manual describes the on-line environment for diagnostic tests that run under the Cray operating system UNICOS, release 5.0, on CRAY Y-MP, CRAY X-MP EA, CRAY X-MP, and CRAY-1 computer systems. It is intended for Cray Research, Inc. (CRI) field engineers and analysts. A working knowledge of UNICOS is assumed.

CONVENTIONS

To aid in identifying the various groups of Cray mainframes, this manual uses the naming conventions shown in the Hardware Product Line sheet, which is located at the end of the preface. The Hardware Product Line sheet shows both the chronological evolution of Cray mainframes and the characteristics of each group. The reverse side contains definitions of the terms used on the sheet and throughout this manual.

The conventions for entering the diagnostic commands are as follows:

Convention	Description
bold	**Bold** indicates one of the following: - Diagnostic program - Command option - Man page entry - File name
italic	*Italic* indicates variable or user-supplied information.
O'*x*	The prefix O' indicates that *x* is an octal value.
RETURN	This indicates the RETURN key. You must press the RETURN after entering each keyboard command.
[ ]	Square brackets indicate optional items.
+*option*	A plus sign (+) preceding a command option indicates that the option is enabled.
-*option*	A minus sign (-) preceding a command option indicates that the option is disabled.

Convention	Description
command(1)	This refers to an entry in the UNICOS User Commands Reference Manual, CRI publication SR-2011.
command(1M)	This refers to an entry in the UNICOS Administrator Commands Reference Manual, CRI publication SR-2022.
system call(2)	This refers to an entry in the UNICOS System Calls Reference Manual, CRI publication SR-2012.
entry(4X)	This refers to an entry in the UNICOS File Formats and Special Files Reference Manual, CRI publication SR-2014. The x indicates the section of the manual that contains the entry.

OTHER PUBLICATIONS

CRI off-line diagnostic publications that may be of interest are as follows:

HQ-01004	CRAY-1 Computer Systems Diagnostic Ready Reference Guide
HQ-01005	CRAY X-MP Computer Systems Diagnostic Ready Reference Guide
HQ-01007	I/O Subsystem (IOS) Diagnostic Ready Reference Guide
HM-01010	CRAY X-MP Computer Systems IOS-based Diagnostic Reference Manual

CRI software publications that may be of interest are as follows:

SQ-0083	CRAY Y-MP, CRAY X-MP EA, CRAY X-MP and CRAY-1 CAL Assembler Version 2 Ready Reference
SD-0235	Software Problem Report (SPR) User's Guide
SG-0307	I/O Subsystem (IOS) Administrator's Guide
SG-2005	I/O Subsystem (IOS) Operator's Guide for UNICOS
SR-2011	UNICOS User Commands Reference Manual
SR-2012	Volume 4: UNICOS System Calls Reference Manual
SR-2014	UNICOS File Formats and Special Files Reference Manual
SR-2022	UNICOS Administrator Commands Reference Manual
SN-3030	Operator Workstation (OWS) Guide

CRI hardware publications that may be of interest are as follows:

HR-0030	I/O Subsystem Model B Hardware Reference Manual
HR-0081	I/O Subsystem Model C/D Hardware Reference Manual
CSM0110000	CRAY X-MP/2 System Programmer Reference Manual
CSM-0111-000	CRAY X-MP/1 System Programmer Reference Manual
CSM0112000	CRAY X-MP/4 System Programmer Reference Manual
CSM-0400-000	CRAY Y-MP System Programmer Reference Manual

For additional information, refer to the on-line diagnostic listings.

UNICOS SYSTEM INSTALLATION BULLETIN

Refer to the UNICOS System Installation Bulletin for the following information:

- Build and installation procedures
- Configuration guidelines

Each site receives this bulletin with the UNICOS release package. You can order additional copies from the CRI Distribution Center.

Note that appendix G, Installation Information, describes the procedure for on-line diagnostic re-installation subsequent to system installation.

READER COMMENTS

If you have any comments about the technical accuracy, content, or organization of this manual, please tell us. You can contact us in any of the following ways:

- Call our Technical Publications department at (612) 681-5729 during the hours of 7:30 A.M. to 6:00 P.M. (Central Time).

- Send us electronic mail from a UNICOS or UNIX system, using the following UUCP addresses:

  uunet!cray!publications

  sun!tundra!hall!publications

- Send us electronic mail from a UNICOS or UNIX system, using the following ARPAnet address:

  publications@cray.com

- Send a facsimile of your comments to the attention of "Publications" at FAX number (612) 681-5602.

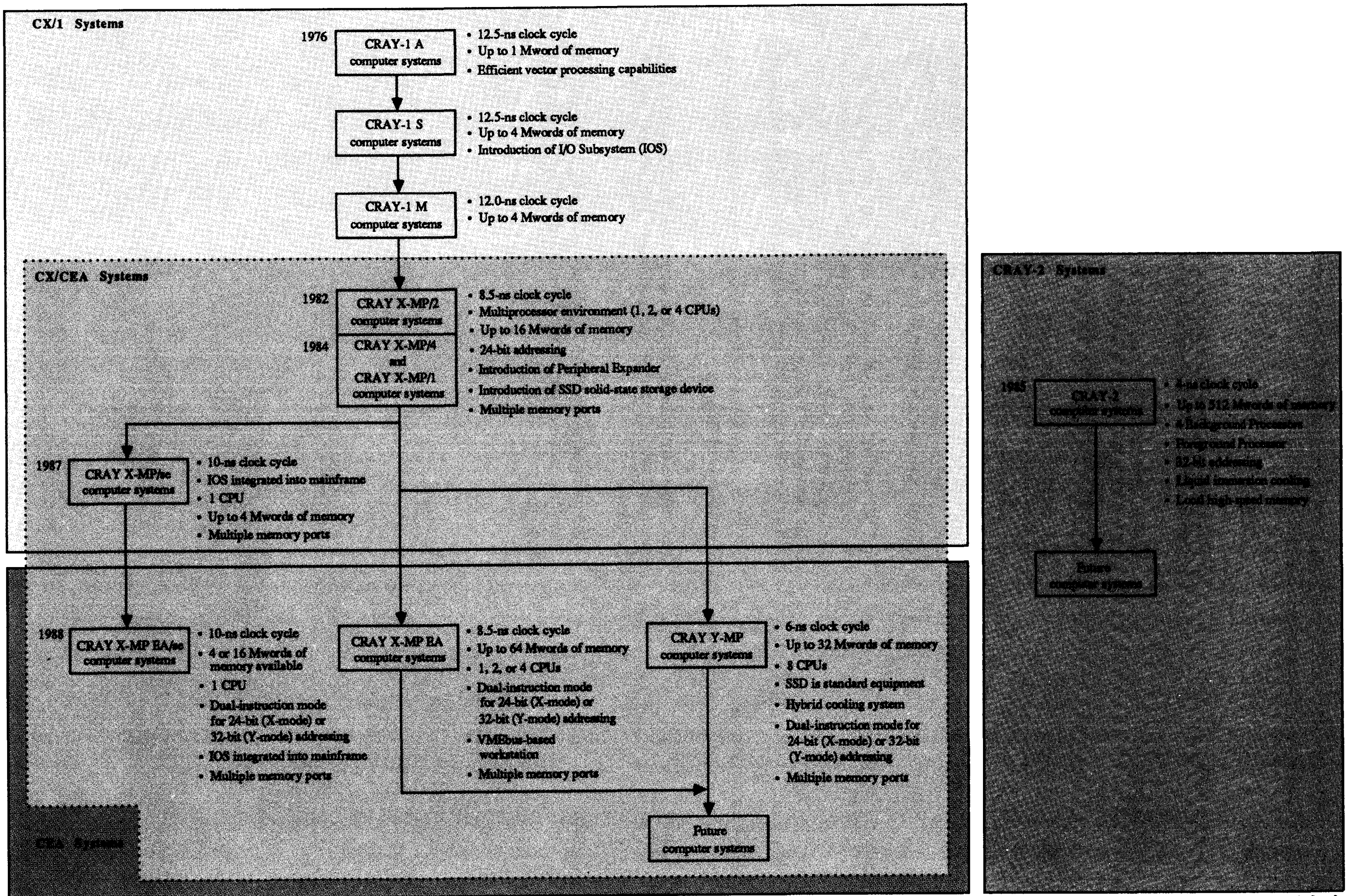
- Use the postage-paid Reader's Comment form at the back of this manual.

- Write to us at the following address:

  Cray Research, Inc.
  Technical Publications Department
  1345 Northland Drive
  Mendota Heights, Minnesota 55120

We value your comments and will respond to them promptly.

Hardware Product Line



CRAY RESEARCH, INC.

The following list defines architecture terms:

Term	Definition
CX/1 systems	This group includes all models of the CRAY X-MP and CRAY-1 computer systems. It is characterized by 24-bit addressing capabilities.
CEA systems	This group includes all models of the Extended Architecture (EA) series, which are the CRAY Y-MP and CRAY X-MP EA computer systems. It is characterized by 32-bit addressing capabilities.
CRAY-2 systems	This group includes all models of the CRAY-2 computer systems. It is characterized by 32-bit addressing capabilities, large common memories, and immersion cooling.
CX/CEA systems	This group designates all models of CRAY X-MP computer systems plus all models of the CRAY Y-MP and CRAY X-MP EA computer systems. It does not include CRAY-1 computer systems.
EAM bit (hardware)	In CX/1 systems, the EAM bit is the Enhanced Addressing Mode bit in the Flag register. When set, it sign-extends certain instructions for memory addressing in 8- and 16-Mword systems. In CEA systems, the EAM bit is the Extended Addressing Mode bit in the Flag register. It is set by the operating system to select either 24- or 32-bit addressing.
EMA feature (software)	In CX/1 systems, EMA is the Extended Memory Addressing feature for 8- or 16-Mword systems.
X-mode	This term refers to the 24-bit addressing mode in CEA systems. The operating systems select this mode with the EAM bit in the Exchange Package.
Y-mode	This term refers to the 32-bit addressing mode in CEA systems. The operating systems select this mode with the EAM bit in the Exchange Package.

CONTENTS

APPENDIX SECTION

FIGURES

FIGURES (continued)

TABLES

TABLES (continued)

INDEX

1. ON-LINE DIAGNOSTIC SYSTEM

This manual describes the on-line test environment for diagnostics that run under the Cray operating system UNICOS on the following computer systems:

- CEA systems
  - Y-mode (32-bit addressing)
  - X-mode (24-bit addressing)
- CX/1 systems

The on-line diagnostic system performs error detection and isolation concurrent with system operation. This type of on-line maintenance provides the following benefits:

- Ensures an enhanced level of continuous system operation
- Prevents possible system software failures and identifies data integrity problems in system output
- Provides the capability for concurrent maintenance
- Reduces mean time to repair (MTTR) by isolating the failing hardware while the system is running
- Reduces off-line preventive maintenance (PM) time required for failure detection, isolation, and repair

1.1 ON-LINE DIAGNOSTIC ENVIRONMENT

The on-line diagnostic system consists of programs that reside in Cray central memory or in Cray mass storage. To run the on-line diagnostic programs in a Cray computer system configuration, UNICOS must be running in at least one Central Processing Unit (CPU).

Throughout this document, the term *operator's station* refers to one of the following devices, as appropriate to your site:

- Peripheral expander
- Operator workstation

1.2 ON-LINE DIAGNOSTIC PROGRAMS

To ensure maximum system reliability, the on-line diagnostic programs do the following:

- Detect, isolate, and report hardware faults
- Gather and analyze system performance data

The on-line diagnostic programs are grouped as follows:

Diagnostic Group	Description
Confidence tests	These tests provide error detection and isolation. To verify system integrity, it is recommended that these tests be run at system startup and at intervals thereafter.
Maintenance tests	These tests provide error detection and isolation. These tests are variants of off-line diagnostic tests.
Down-device programs	The down-device programs provide on-line CPU and peripheral testing while the hardware is removed from normal system operations.
Network test (**olnet**)†	This test detects and isolates faults in the communications link between a Cray mainframe and a front-end computer system.
I/O Subsystem (IOS) deadstart programs	These programs can be run prior to system deadstart to verify the integrity of the IOS hardware. They isolate failures to the functional area, at which point a CRI field engineer must interpret the results.
Utility programs	These are on-line diagnostic tools.

† The **olnet** test is described in the On-line Diagnostic Network Communications Program (OLNET) Maintenance Manual, CRI publication SMM-1016.

2. CONFIDENCE TEST AND MONITOR OVERVIEW

On-line diagnostic confidence tests provide a comprehensive performance check of the system hardware. This test level consists of the following:

- High-level language diagnostic programs
- A set of CAL Version 2 diagnostic programs that direct hardware testing to specific logic areas

This section provides an overview of the following:

- On-line confidence monitor (**olcmon**)
- Program synopsis
- Test execution
- Test termination
- Test examples
- Test messages
- Off-line confidence monitor (**offmon**)

For a brief description of each confidence test, refer to appendix A, On-line Diagnostic Programs. For a list of test execution times, refer to appendix B, Test Execution Times. For additional information on specific confidence tests and their command options, refer to section 3, Confidence Test Descriptions.

2.1 ON-LINE CONFIDENCE MONITOR (olcmon)

The on-line confidence monitor program, **olcmon**, does the following:

- Accepts and interprets command options and arguments
- Sends test results to **stdout** (standard output device) by default or to a file when UNICOS output redirection is indicated on the command line

2.2 PROGRAM SYNOPSIS

The **olcmon** command options are entered with the test command options of each confidence test to be executed. The test-specific command options are described in section 3, Confidence Test Descriptions.

The **olcmon** command options can be entered in any order. If an option is omitted, the program uses the default value.

The following command options provide different methods of specifying the starting seed value (specify only one for each test executed):

- **+/-getseed**
- **getseed** *file*
- **seed** *n* (a test-specific command option described in section 3, Confidence Test Descriptions)

Synopsis:

test [**chkpnt** *mode*] [**cpu** *clist*] [**cputime** *h*:*m*:*s*] [**+/-getseed**]

[**getseed** *file*] [**help**] [**maxerr** *n*] [**maxp** *n*] [**+/-parcel**] [**time** *h*:*m*:*s*]

[**+/-verbose**] [**+xmp**] [**+cray1**]

[*test options*]†

chkpnt *mode*

Indicates whether restart files are to be generated. *mode* is one of the following arguments:

Argument	Description
first	Generates a restart file for the first failure detected (default)
all	Generates a restart file for each failure detected, including failures detected during error isolation
none	Does not generate restart files

The default generates a restart file for the first failure detected.

For additional information, refer to the following: **chkpnt**(1), **restart**(1), **chkpnt**(2), and **restart**(2).

† For additional information on confidence tests and their test-specific command options, refer to section 3, Confidence Test Descriptions.

cpu *clist*

Selects the CPUs to be tested. Enter *clist* in the following format:

x,x,...,x

x can be **a**, **b**, **c**, **d**, **e**, **f**, **g**, or **h**. The first CPU selected is the master CPU. The default is **cpu a**.

If you enter an invalid CPU value in *clist* or a value for a CPU that is currently down, you will receive an error message.

cputime *h*:*m*:*s*

Sets the test execution time in CPU time. The time is specified in hours (*h*), minutes (*m*), and seconds (*s*); minutes and seconds; or just seconds. Use colons as delimiters, as follows: *h*:*m*:*s*.

Generally, actual execution time is within one second of the specified CPU time. If **cputime** is allowed to default, or is set to 0, the test uses the **maxp** value. However, if set to a value other than 0, **cputime** overrides **maxp**.

+/-getseed

Enables (**+getseed**) or disables (**-getseed**) the option that reads the file *test***.seed** to obtain a starting seed. If the test terminates because the maximum pass or error limit is reached, the seed from the last pass is saved in the file *test***.seed**. If there are any problems with reading the seed from this file, the program uses the default seed (0'33). If you select **+getseed**, do not select **seed** *n* (test-specific command option). The default is **-getseed**.

getseed *file*

Gets a starting seed from *file*. *file* can contain a dump from a previous failure or a single seed value. If allowed to default, the program uses the seed value specified by **+getseed** or **seed** *n* (test-specific command option).

help Generates an on-line help display containing a synopsis and a brief description of the command options and arguments. If **help** is entered with a test name, help information is written to **stdout**, and the test terminates.

maxerr *n* Sets the maximum number of errors. *n* is an octal value. The default for *n* is 1.

maxp *n* Sets the maximum number of passes. *n* is an octal value. The default for *n* is O'1000. If **cputime** or **time** is set to a value other than 0, the specified option overrides **maxp**.

+/-parcel

Enables (**+parcel**) or disables (**-parcel**) the option that forces dumped data to parcel format. **+parcel** forces data that would otherwise be in word format (64 bits in octal, with leading 0's) to parcel format (four groups of 16 bits in octal). Parcel format displays two words (8 parcels) per line. Word format displays four words per line. The default is **-parcel**.

time *h*:*m*:*s*

Sets the test execution time in elapsed (wall-clock) time. The time is specified in hours (*h*), minutes (*m*), and seconds (*s*); minutes and seconds; or just seconds. Use colons as delimiters, as follows: *h*:*m*:*s*.

Generally, actual execution time is within one second of the specified elapsed time. If **time** is allowed to default (or is set to 0), the test uses the **maxp** value. However, if specified to a value other than 0, **time** overrides **maxp**.

+/-verbose

Enables (**+verbose**) or disables (**-verbose**) the generation of informational messages. The **+verbose** option causes a line of output to be generated after each pass of the diagnostic. The default is **-verbose**.

+xmp
+cray1 Indicates the test mode for the following computer systems:

Command	Computer System
+xmp	CRAY X-MP
+cray1	CRAY-1

If allowed to default, the monitor determines the machine type during test execution and selects the appropriate test mode. This option can be used to override the default selection. These command options are not applicable to a CEA system.

2.3 TEST EXECUTION

To start a single diagnostic test, enter the following on the command line:

- *test*
- Monitor command options
- Test-specific command options

To run a sequence of diagnostics, use the **runsequence** utility described in section 7, Utility Programs.

Before a test can be started, UNICOS must be running in the CPUs to be tested. The master CPU (the first CPU selected) does the following:

- Generates instructions and data
- Generates expected results
- Compares the test execution buffers of the selected CPUs to the expected results
- Generates and formats error reports
- Controls error isolation

Each CPU, including the master, does the following:

- Loads registers and buffers
- Executes test instructions
- Saves results

2.4 TEST TERMINATION

A test stops under the following conditions:

- The test successfully completes the maximum number of passes (**maxp** *n*).
- The test reaches the specified CPU time (**cputime** *h*:*m*:*s*) or elapsed (wall-clock) time (**time** *h*:*m*:*s*).
- The test detects and isolates the maximum number of errors (**maxerr** *n*). Error reports are automatically sent to **stdout** (standard output device), but they can be redirected to an error file.

- The **help** option is entered with a test name, help information is written to **stdout**, and the test terminates.

- The monitor or test detects an error in a command line entry and writes a message to **stderr** (standard error device). Only the first error detected is reported.

2.5 TEST EXAMPLES

The following example executes **olcsvc** in CPUs c, a, and b, with c as the master.

Example:

```
olcsvc cpu c,a,b
```

The following example executes **olcsvc** in CPUs a and b, with a as the master. The **seed** *x* option provides an octal seed value to start random number generation.

Example:

```
olcsvc seed x cpu a,b
```

In the following example, the **nohup**(1) command allows **olcsvc** to continue executing after you log off the system. The ampersand (&) causes the entire command to execute in the background, so that another prompt is immediately displayed and you can continue to use the system.

Example:

```
nohup olcsvc &
```

The following example shows the test-specific help information that is displayed if **help** is entered with a test name.

Example:

```
olcsvc help
```

Help display:

```
olcsvc help
olcsvc [chkpnt mode] [cpu clist] [+/-getseed] [getseed file] [help] [maxerr n]
  [maxp n] [+/-parcel] [+/-verbose] [+cray1] [+xmp] [cputime h:m:s]
  [time h:m:s] [disable ilist] [enable ilist] [+/-isolate] [isop n] [numpar n]
  [+/-repeat] [seed n] [+/-sgci] [vl n] [+/-cm] [+/-fpadd] [+/-fpmult]
  [+/-fprecip] [+/-int] [+/-logical] [+/-pop] [+/-shift] [+/-onezero]
  [+/-random] [+/-slide]
 chkpnt mode    - Checkpoint mode:  none, first, or all.  (Default: first)
 cpu clist      - Run in selected CPUs.  (Default: a)
 +/-getseed     - Get/don't get seed from test.seed.  (Default: -getseed)
 getseed file   - Search file for starting seed
 help           - Provides a help display.
 +/-verbose     - Enable/disable info. messages to stdout.  (Default: -verbose)
 maxp n         - Set maximum pass limit to n.  (Default: O'1000)
 maxerr n       - Set maximum error limit to n.  (Default: 1)
 +/-parcel      - Force/don't force dump to parcel format.  (Default: -parcel)
 +cray1/+xmp    - Selects CRAY-1/CRAY X-MP test mode.  (Default: host machine)
 cputime h:m:s  - Set amount of CPU time to execute.
 time h:m:s     - Set amount of wall clock time to execute.
 disable ilist  - Do not run specific instructions.  Ignored if invalid.
 enable ilist   - Run specific instructions.  Ignored if invalid.
 +/-isolate     - Enable/disable isolation.  (Default: +isolate)
 isop n         - Loop during isolation n times to find error.  (Default: O'1000)
 numpar n       - Number of parcels to run in vector buffer.  (Default: O'100)
 +/-repeat      - Repeat/do not repeat first pass.  (Default: -repeat)
 seed n         - Set seed for random number generator to n.  (Default: O'33)
 +/-sgci        - Enable/disable scatter/gather/compressed index testing.
 vl n           - Set VL. 0 <= n <= 100.  If n = 0, VL is random.  (Default: 0)
 +/-cm, +/-fpadd, +/-fpmult, +/-fprecip, +/-int, +/-logical, +/-pop, +/-shift
    - Enable/disable specific instruction groups.  (Default: all instructions)
 +/-onezero, +/-random, +/-slide
    - Enable/disable specific data patterns.  (Default: all data patterns)
```

The following example shows the output that is displayed when **olcsvc** is run with all default values.

Example:

```
    olcsvc
```

Output:

```
olcsvc
olcsvc: started in cpu A on Thu Jan 8 08:55:46 1987
CRAY X-MP MODE
olcsvc reached maximum pass limit with 1000 passes and 0 errors
on Thu Jan 8 08:56:08 1987
```

The following example shows the output that is displayed if **+verbose** is specified and **maxp** reaches 10.

Example:

```
    olcsvc +verbose maxp 10
```

Output:

```
olcsvc +verbose maxp 10
olcsvc: started in cpu A on Thu Jan 8 08:56:43 1987
CRAY X-MP MODE
olcsvc: pass =          1, error =           0  Thu Jan 8 08:56:43 1987
olcsvc: pass =          2, error =           0  Thu Jan 8 08:56:43 1987
olcsvc: pass =          3, error =           0  Thu Jan 8 08:56:43 1987
olcsvc: pass =          4, error =           0  Thu Jan 8 08:56:43 1987
olcsvc: pass =          5, error =           0  Thu Jan 8 08:56:43 1987
olcsvc: pass =          6, error =           0  Thu Jan 8 08:56:43 1987
olcsvc: pass =          7, error =           0  Thu Jan 8 08:56:43 1987
olcsvc: pass =         10, error =           0  Thu Jan 8 08:56:43 1987
olcsvc reached maximum pass limit with 10 passes and 0 errors
on Thu Jan 8 08:56:43 1987
```

2.6 TEST MESSAGES

Each test generates the following types of messages:

- Informative
- Error

These messages are listed in the subsections that follow.

2.6.1 INFORMATIVE MESSAGES

This subsection lists the informative messages, which are sent to **stdout** (standard output device).

test: Cannot open ***test*.seed**. Seed cannot be saved.
The test cannot write ***test*.seed**. Therefore, the ending seed cannot be saved. Check write permissions of the current directory.

test: Cannot write restart file. errno = *n*.
The test cannot write a restart file. Contact your CRI representative.

2.6.2 ERROR MESSAGES

This subsection lists the error messages, which are sent to **stderr** (standard error device).

test: Illegal option *x*.
Option *x* is invalid. Correct and rerun.

test: Illegal argument *x*.
Argument *x* is invalid. Correct and rerun.

test: Illegal CPU selection *x*.
CPU *x* is invalid. Correct and rerun.

test: Maximum of O'*x* items in *option* list.
Too many items are in the argument list for *option*. The maximum number of items allowed in the argument list is O'*x*. Correct and rerun.

test: An error occurred when selecting CPU *x*.
CPU *x* is unavailable. Contact your CRI representative.

test: Cannot allocate memory. Cannot save buffers.
The test cannot allocate memory or save buffers. Regenerate the diagnostic and rerun. If the problem persists, contact your CRI representative.

test: Too many buffers. Cannot save buffers.
The test cannot save buffers. Regenerate the diagnostic and rerun. If the problem persists, contact your CRI representative.

test: Cannot open *file*.
The test cannot open the file name specified by the getseed option. Correct and rerun.

test: Cannot find seed in *file*.
The test cannot find the seed in *file*. Ensure that *file* is valid and rerun.

test: Error selecting cluster x.
Cluster x is unavailable. Contact your CRI representative.

2.7 OFF-LINE CONFIDENCE MONITOR (**offmon**)

The **offmon**† monitor allows the following on-line confidence tests to be executed either in an off-line environment or in a down CPU under the down CPU monitor, **oldmon**:††

- **olcfpt**
- **olcm**
- **olcrit**
- **olcsvc**
- **olibuf**

To execute in these environments, each on-line confidence test is concatenated to **offmon** and assembled (instead of being linked to **olcmon**). To ensure compatibility between the on-line and off-line test environments, the on-line and off-line confidence tests are built from the same source code. The equivalent off-line confidence test names start with the prefix **off** instead of **ol**. For example, the off-line equivalent of **olcrit** is **offcrit**.

To generate the same test conditions in both the on-line and off-line test environments, use the same seed value. Set the seed value for the on-line confidence test (refer to subsection 2.2, Program Synopsis), and use the same value for the off-line test.

For information on executing **offmon**, refer to the diagnostic listing.

---

† The **offmon** monitor is supported on CX/CEA systems only.
†† The **oldmon** monitor is supported on multiple-CPU Cray computer systems only.

3. CONFIDENCE TEST DESCRIPTIONS

This section describes the following on-line confidence tests:

Test	Description
olcfdt	Mass storage device test
olcfpt	Comprehensive floating-point test
olcm	Central memory test
olcrit	Comprehensive random instruction test
olcsvc	Comprehensive scalar and vector comparison test
olibuf	Instruction buffer test
olsbt	Semaphore, shared B and shared T register test

For general information on confidence tests, refer to section 2, Confidence Test and Monitor Overview. For a list of test execution times, refer to appendix B, Test Execution Times.

3.1 olcfdt

The **olcfdt** test is an on-line confidence test for mass storage devices. It creates a user-specified file that is used for all input and output operations during test execution.

To test a specific device, specify the absolute path name to the device. If an absolute path name is not specified, **olcfdt** creates a file on the user's current working directory and tests the device associated with the working directory. Your system file configuration determines which directories and files reside on each device.

The created file is permanent. To delete the file, use the **rm**(1) command.

The test uses the values specified by the record size (**rsz**) and file size (**sz**) options to determine the following:

- Data record size
- Size of the device file to be created
- Number of data records required to fill the file

The default values for the tests and patterns to be run (specified by the **test** and **pat** options, respectively) are designed for optimum functionality. When selecting arguments for these options, be aware that varying degrees of functionality may be achieved.

If a failure occurs, messages are output to **stdout**, provided the program is in control after the failure. However, you can redirect output from **stdout** to a specified file.

3.1.1 TEST SYNOPSIS

The **olcfdt** command options can be entered in any order. If an option is omitted, the program uses the default value.

Synopsis:

olcfdt [**disp** *display*] **dt** *type* [**fn** *file*] [**help**] [**maxp** *n*] [**ntks**]

[**pat** *patterns*] [**rsz** *n*] [**seed** *n*] [**sz** *n*] [**test** *tests*] [**upat** *n*]

disp *display*

Enables or disables the option that generates an error information/history display option. The default is **err** (all error information is displayed). *display* is one of the following:

Value	Description
hst	Displays a history of the current iteration (test pattern and test sections executed)
err	Displays all error information
none	Does not display error information or a history of the current iteration
all	Displays all error information and a history of the current iteration

dt *type*

Device type (required). If the specified device type is not associated with the specified file name, the program overrides the **dt** command option and tests the device type associated with *file*. *type* is one of the following (only one device type can be selected at a time):

Device Type	Description
dd10	DD-10 disk drive
dd19	DD-19 disk drive
dd29	DD-29 disk drive
dd39	DD-39 disk drive

dt *type*
(continued)

Device Type	Description
dd40	DD-40 disk drive
dd49	DD-49 disk drive
bmr	Buffer memory resident storage
ssd	SSD solid-state storage device

fn *file* File name. *file* is the absolute path name to a file. The created file is permanent. When assigning a file, you must know which directory is associated with the selected device type. Consult your CRI analyst to determine the directory associated with a specific device. The default is **workfil** under the current working directory.

help Produces an on-line help display containing a synopsis and brief description of the command options and arguments. If the **help** option is entered with a test name, help information is written to **stdout**, and the test terminates.

maxp *n* Pass count (decimal). On each pass, all selected test patterns and test sections are run. The default for *n* is 512.

ntks File size is in number of tracks. This command option indicates that the argument associated with the **sz** command option is the file size in number of tracks (decimal). If allowed to default, the file size is in data sectors (decimal).

pat *patterns*
Patterns to be run. The default is **all** (all test patterns are run). If the **upat** option is specified, you must either set **pat** to **all** or include **user** in the list of arguments. *patterns* is a comma-separated list of up to nine test pattern arguments. Duplicate entries are allowed. For example:

```
pat zeros,ones
```

patterns can be one of the following:

Argument	Pattern
zeros	All 0's
ones	All 1's
chkbrd	Checkerboard (1252525252525252525252B, 0525252525252525252525B...)

pat *patterns*
(continued)

Argument	Pattern
chkbrdc	Complement of the **chkbrd** pattern
rwi	Record/word index. The record number in the upper 31 bits of the data word, followed by the data word number within the record in the lower 33 bits (hardware numbered bits).
rwic	Complement of the **rwi** pattern
fpn	Random floating-point numbers
rdm	Random numbers
user	User pattern. This is the pattern specified by the **upat** option (**upat** must be specified if this argument is entered).
all	All patterns are run (default). The patterns are processed in the following order: **zeros,ones,chkbrd,chkbrdc, rwi,rwic,fpn,rdm,user** The **user** argument is processed only if the **upat** option is entered. **all** is a stand-alone argument.

rsz *n* Record size in data words. *n* is a decimal record size of 512, or a multiple thereof, up to a maximum value of 4096. The default is 512 words.

seed *n* Random number seed. *n* is an octal value that is less than or equal to 48 bits. The default for *n* is **rdm**, which selects the nearest integer of the product of a random number and the real-time clock.

sz *n* File size (decimal). If **sz** *n* is specified without the **ntks** command option, the file size is in data sectors; if **ntks** is specified, the file size is in number of tracks. The minimum value for *n* is 1. The maximum value for *n* is as follows:

(Track size * number of tracks) - 1

or

Maximum file size allowed by the system

sz *n*
(continued)

The default for *n* is the track size of the device specified by the command option **dt.**

test *tests*

Test sections to be run. The test does a sequential write before executing the selected test sections. The default for *tests* is **all** (all test sections are run).

tests is a comma-separated list of up to three test section entries. The test sections are processed in the order in which they are entered on the command line. Duplicate entries are allowed. For example:

```
rw,rw,rr
```

tests can be one of the following:

Test Section	Description
rr	Random read; performs random reads on the work file. A data compare is performed on each record read. On a miscompare, a message is displayed and the program is aborted.
rw	Random write; performs random writes on the work file. This section automatically performs a sequential read (**sr**) if **sr** is not selected after a random write (**rw**). For example, the following entry runs test sections **rr**, **rw**, and **sr**, respectively: `test rr,rw`
sr	Sequential read; reads the work file sequentially. A data compare is performed on each record read. On a miscompare, a message is displayed and the program is aborted.
all	Runs all test sections (default). This is a stand-alone argument. The tests are run in the following order: **rr,rw,sr**.

upat *n* User pattern. *n* is an octal value that is less than or equal to 64 bits. An error occurs if the **upat** option is not specified when **user** is entered in the argument list for the **pat** option. The default is no user pattern.

3.1.2 TEST EXAMPLES

This subsection contains **olcfdt** execution examples.

The following example runs **olcfdt** using default command options to test a DD-29 disk drive. It is assumed that the current user directory is associated with the DD-29 disk drive to be tested.

Example:

```
olcfdt dt dd29 rsz 512
```

Output:

```
olcfdt dt dd29 rsz 512
olcfdt submitted on Wed Mar 11 15:38:30 1987

odt06 - Test completed.
```

The following example runs **olcfdt** using user-specified command options to test a DD-29 disk drive. It is assumed that the specified file name, */w/xxx/yyy*, is associated with the DD-29 disk drive to be tested.

Example:

```
olcfdt fn /w/xxx/yyy dt dd29 sz 36 rsz 512 test all pat all
  upat 0707070707070707070707 seed 7070707070707070 maxp 10
  disp none
```

Output:

```
olcfdt fn /w/xxx/yyy dt dd29 sz 36 rsz 512 test all pat all
upat 0707070707070707070707 seed 7070707070707070 maxp 10 disp none
olcfdt submitted on Wed Mar 11 16:26:20 1987

odt06 - Test completed.
```

The following example runs **olcfdt** using default options and the checkerboard data pattern to test a DD-29 disk drive. The test displays the data compare error output by default.

The test output indicates that a data compare error was detected at word 99 of record 9. The test displays expected, actual, and difference data for the following words:

- Ten words on either side of the failing word
- Last word of the preceding record
- First word of the next record

If there are less than 10 words preceding or following the word that failed, more words are displayed from one side than another to make up the difference. In the following example, data information is displayed for words 89 through 109 of record 9, word 1024 of record 8, and word 1 of record 10.

Example:

```
olcfdt dt dd29 pat chkbrd rsz 1024
```

Output:

```
olcfdt dt dd29 pat chkbrd rsz 1024
olcfdt submitted on Wed Mar 11 13:14:19 1987

odt14 - Data compare error.

                    *****  DATA COMPARE ERROR  *****

    FILENAME              : workfil
    FILE SIZE             : 18
    DEVICE TYPE           : dd29
    CURRENT DATA PATTERN  : chkbrd
    CURRENT TEST          : sr
    ITERATION COUNT       :                    512
    NUMBER OF PASSES      :                    100
    RECORD SIZE           :                   1024
    NUMBER OF RECORDS     :                     13
    FAILING RECORD NUMBER :                      9
    FAILING WORD NUMBER   :                     99
    USER PATTERN          : 0000000000000000000000
    RANDOM NUMBER SEED    : 0000003427130120254365

WORD        EXPECTED                 ACTUAL                  DIFFERENCE

  89  1252525252525252525252  1252525252525252525252  0000000000000000000000
  90  0525252525252525252525  0525252525252525252525  0000000000000000000000
  91  1252525252525252525252  1252525252525252525252  0000000000000000000000
  92  0525252525252525252525  0525252525252525252525  0000000000000000000000
  93  1252525252525252525252  1252525252525252525252  0000000000000000000000
  94  0525252525252525252525  0525252525252525252525  0000000000000000000000
  95  1252525252525252525252  1252525252525252525252  0000000000000000000000
  96  0525252525252525252525  0525252525252525252525  0000000000000000000000
  97  1252525252525252525252  1252525252525252525252  0000000000000000000000
  98  0525252525252525252525  0525252525252525252525  0000000000000000000000
  99  1252525252525252525252  1777777777777777777777  0525252525252525252525
 100  0525252525252525252525  0525252525252525252525  0000000000000000000000
 101  1252525252525252525252  1252525252525252525252  0000000000000000000000
 102  0525252525252525252525  0525252525252525252525  0000000000000000000000
```

Output (continued):

WORD	EXPECTED	ACTUAL	DIFFERENCE
103	1252525252525252525252	1252525252525252525252	0000000000000000000000
104	1252525252525252525252	1252525252525252525252	0000000000000000000000
105	0525252525252525252525	0525252525252525252525	0000000000000000000000
106	1252525252525252525252	1252525252525252525252	0000000000000000000000
107	0525252525252525252525	0525252525252525252525	0000000000000000000000
108	1252525252525252525252	1252525252525252525252	0000000000000000000000
109	0525252525252525252525	0525252525252525252525	0000000000000000000000

***** LAST WORDS OF PREVIOUS RECORD *****

WORD	EXPECTED	ACTUAL
1024	0525252525252525252525	0525252525252525252525

***** FIRST WORDS OF NEXT RECORD *****

WORD	EXPECTED	ACTUAL
1	1252525252525252525252	1252525252525252525252

The following example runs **olcfdt** with user-specified command options to test a DD-29 disk drive. Test output is sent to */a/b/ccc*.

Example:

```
olcfdt fn /w/xxx/yyy dt dd29 sz 36 rsz 4096 test all pat rdm
  seed 7070707070707070 > /a/b/ccc
```

3.1.3 TEST MESSAGES

The **olcfdt** test produces the following types of messages:

- Informative
- Error

These messages are listed in the subsections that follow.

3.1.3.1 Informative messages

This subsection lists the informative messages, which are sent to **stdout** (standard output device).

odt06 - Test completed.

odt16 - iteration pattern tests
odt16 - *iteration* *pattern* *tests*
This message is generated if the **disp** command option is set to display the history of the current iteration. On each iteration through the test, the selected device is tested with one of the selected patterns in all of the selected test sections. The following information is displayed:

iteration	Current iteration
pattern	Current test pattern (64-bit octal word)
tests	Test sections being run

3.1.3.2 Error messages

This subsection lists the error messages, which are sent to **stderr** (standard error device).

odt01 - Option *x* is invalid.
Enter a valid option and rerun.

odt02 - Argument *x* is invalid.
Enter a valid argument and rerun.

odt03 - Too many items in value list *l*.
Reenter argument list and rerun.

odt04 - Required option *x* is not present.
Enter option *x* and rerun.

odt15 - Argument is missing.
An option requiring an argument was entered alone. Reenter the option with an argument and rerun the test.

The following error messages are sent to **stdout**:

odt05 - Specified record size exceeds
odt05 - the maximum limit of 4096.
Reenter the **rsz** option and rerun.

odt07 - Cannot open file.
Contact your CRI representative.

odt08 - Cannot close file.
Contact your CRI representative.

odt09 - Cannot seek file.
Contact your CRI representative.

odt10 - Cannot read file.
Contact your CRI representative.

odt11 - Cannot write file.
Contact your CRI representative.

odt12 - User pattern option (upat) must be specified
odt12 - when pattern option (pat) is 'user'.
Enter the **upat** option and rerun.

odt13 - Pattern option (pat) must be 'user' or 'all'
odt13 - when the user pattern option (upat) is specified.
Enter the **pat** option and rerun.

odt14 - Data compare error.
Examine the error output to identify the point at which the failure occurred.

3.2 olcfpt

The **olcfpt** test is an on-line comprehensive floating-point test. It generates floating-point instructions and data to detect data-sensitive failures in the floating-point functional units. The generated instructions are simulated and then executed. The simulation and execution results are compared, and any differences are reported. This process continues until the maximum pass, error, or time limit is reached. If an error is detected, the diagnostic attempts to isolate the failing data.

3.2.1 TEST SYNOPSIS

The **olcfpt** command options can be entered in any order. If an option is omitted, the program uses the default value. The test synopsis lists the **olcfpt** command options and arguments in the following order:

1. Monitor options
2. Test-specific options
3. Data pattern options
4. Instruction options

Synopsis:

olcfpt [**chkpnt** *mode*] [**cpu** *clist*] [**cputime** *h*:*m*:*s*] [+/-**getseed**]

[**getseed** *file*] [**help**] [**maxerr** *n*] [**maxp** *n*] [+/-**parcel**] [**time** *h*:*m*:*s*]

[+/-**verbose**] [+**xmp**] [+**cray1**]†

[**disable** *ilist*] [**enable** *ilist*] [+/-**isolate**] [**isop** *n*]

[**numins** *n*] [+/-**repeat**] [**seed** *n*] [**vl** *n*] [+/-**vload**]

[+/-**fpbits**] [+/-**fprand**] [+/-**random**]

[+/-**fpadd**] [+/-**fpmult**] [+/-**fprecip**] [+/-**scalar**] [+/-**vector**]

---

† The monitor command options are described in section 2, Confidence Test and Monitor Overview.

disable *ilist*

Deselects specific instructions. Enter *ilist* in the following format:

n,n,...,n

n is the octal value in the *gh* field of the specific instruction. The **disable** *ilist* option overrides the **enable** *ilist* option and any selected (+) or deselected (-) instruction options.

enable *ilist*

Selects specific instructions. Enter *ilist* in the following format:

n,n,...,n

n is the octal value in the *gh* field of the specific instruction. The **enable** *ilist* option overrides any selected (+) or deselected (-) instruction options. When the test is run with default values for the +/- instruction options, and the **enable** *ilist* option is selected, only the instructions specified by the **enable** *ilist* option are run.

+/-isolate

Enables (**+isolate**) or disables (**-isolate**) the error isolation option. The default is **+isolate.**

isop *n*

Sets the isolation pass limit to *n* (octal). During isolation, the diagnostic repeatedly executes the suspected failing sequence. If the sequence fails, the loop terminates and the diagnostic attempts to isolate the sequence further. If the sequence does not fail, the loop terminates after *n* passes, and **olcfpt** assumes that the error is not in the tested sequence. The default for *n* is O'1000.

numins *n*

Sets the number of instructions to be generated. *n* can be any octal value within the range 1 through O'20. The default for *n* is O'20.

+/-repeat

Enables (**+repeat**) or disables (**-repeat**) the option that repeats the first pass until the diagnostic terminates. **+repeat** is useful for recreating an error. It is normally used with one of the following options: **seed** *n*, **+getseed**, or **getseed** *file*. The default is **-repeat** (the program generates new instructions and data after each pass).

seed *n* Sets the random seed to *n*. *n* can be any 64-bit octal value. If *n* is 0, the test reads the real-time clock and uses the value for the initial seed. The default for *n* is O'33. If **seed** *n* is selected, do not select **+getseed** or **getseed** *file*.

vl *n* Sets the vector length to *n*. *n* can be any octal value in the range 0 through O'100. If **vl** is set to 0, a random **vl** value is used to initialize the test. The default for *n* is 100.

+/-vload Selects (**+vload**) or deselects (**-vload**) vector instructions for the instruction buffer and, in the case of **-vload**, does not allow you to load (write) or save (read) the vector registers. **-vload** overrides vector instructions selected by **+vector** and **enable** *ilist*. The default is **+vload**.

+/-fpbits, **+/-fprand**, **+/-random**
Selects (+) or deselects (-) specific data patterns. If allowed to default, all of the data patterns are run. If the **vl** option is 0 or not specified, the vector length register is initialized with 6-bits of random data. The data patterns are as follows:

Option	Data Pattern
fpbits	Random number of consecutive 1-bits in the coefficient. Exponent data depends on the floating-point instruction. For example: 0370000000000007740000 1574777740003777777777 0217600000000000030000 0237740000000000100000
fprand	Random bit generation in the coefficient. Exponent data depends on the floating-point instruction. For example: 0224055214537525453301 1327217472141363076211
random	Random bit generation in a word. For example: 1023122123232122777127 0003423100233344322177 1640034356453221213532 1123235467543221322120 1304322300332105534311

+/-fpadd, +/-fpmult, +/-fprecip, +/-scalar, +/-vector
Selects (+) or deselects (-) specific instruction groups for the following options:

Option	Instruction Type
fpadd	Floating-point addition
fpmult	Floating-point multiply
fprecip	Floating-point reciprocal
scalar	Scalar instruction (destination)
vector	Vector instruction (destination)

If allowed to default, all instruction groups are run. The groups are as follows:

Option	Instruction Group
fpadd	062, 063 170 through 173
fpmult	064 through 067 160 through 167
fprecip	070, 174
scalar	062, 063 064 through 067 070
vector	160 through 167 170 through 174

3.2.2 TEST EXECUTION

The **olcfpt** execution sequence is as follows:

1. Test initialization
2. Random floating-point instruction and data generation
3. Random floating-point instruction buffer simulation
4. Random floating-point instruction buffer execution
5. Comparison of simulation and execution results
6. Error isolation

Steps 2 through 5 occur on each pass through the test loop. Step 6 occurs only on error.

3.2.2.1 Test initialization

At test initialization, the selected instructions are processed in the following order:

1. All instructions are initially enabled unless either of the following occurs (in which case no instructions are initially enabled):
   - An instruction group is selected (+*option*)
   - An **enable** option is entered and there are no deselected (-*option*) instruction group entries
2. Selected groups are processed, enabling instructions in the selected groups.
3. Deselected groups are processed, disabling instructions in the deselected groups.
4. Individually selected instructions are processed (all instructions specified by the **enable** option).
5. Individually deselected instructions are processed (all instructions specified by the **disable** option).
6. Vector instructions disabled by **-vload** are processed.
7. If no instructions are selected, an error message is displayed and the test is terminated.

3.2.2.2 Random floating-point instruction and data generation

These routines build and generate the floating-point instruction buffer and initial data. Instructions for the buffer are randomly selected from a list of enabled floating-point instructions.

If the *i*, *j*, or *k* field is represented by an *x* in the Cray Assembly Language (CAL), a 0 is used for the field (for additional information, refer to the CRAY Y-MP, CRAY X-MP EA, CRAY X-MP and CRAY-1 CAL Assembler Version 2 Ready Reference, CRI publication SQ-0083).

3.2.2.3 Random floating-point instruction buffer simulation

After the instructions and data are generated, the floating-point instruction buffer is simulated by the master CPU only. The **save** monitor routine saves the results.

Each instruction type has a unique simulation routine. The simulation routines use machine resources differently from the instruction being simulated. For example, the scalar add, pop, leading zero, and logical functional units are used to simulate the floating-point add functional unit.

3.2.2.4 Random floating-point instruction buffer execution

After the instructions are simulated, all of the selected CPUs execute the floating-point instruction buffer. Before the instructions can be executed, the program loads the following:

- Scalar registers
- Vector registers
- Vector length register

Then an unconditional jump to the floating-point instruction buffer is executed. At the end of the floating-point instruction buffer is an unconditional jump to a routine that unloads the contents of all the registers. The **save** monitor routine saves the results.

3.2.2.5 Comparison of simulation and execution results

After the instructions are executed in all of the selected CPUs, the **compare** monitor routine compares the results, and one of the following actions occurs:

- If the results match, the test proceeds with the next data pattern. After all of the selected data patterns are run, the pass count is incremented.

- If the results do not match, the test dumps all of the data related to the suspected failure and, if the isolation option is enabled (**+isolate**), attempts to isolate the failure.

3.2.2.6 Error isolation

If an error is detected and the isolation option is enabled (**+isolate**), the test attempts to identify and isolate the failing instruction by executing the instructions in the floating-point instruction buffer, one at a time.

For scalar instructions, error isolation occurs as follows:

1. The *j* operand is set to 0. If no error is detected, the operand is restored.

2. The *k* operand is set to 0. If an error is not detected, the operand is restored.

3. Each bit of the *j* operand is set to 0 (one at a time). If no error is detected, the bit is restored.

4. Each bit of the *k* operand is set to 0 (one at a time). If no error is detected, the bit is restored.

For vector instructions, error isolation occurs as follows:

1. Each element of the *j* operand is set to 0 (one at a time). If no error is detected, the element is restored.

2. Each element of the *k* operand is set to 0 (one at a time). If no error is detected, the element is restored.

3. Each bit of the *j* operand is set to 0 (one at a time, for all elements). If no error is detected, the bit is restored.

4. Each bit of the *k* operand is set to 0 (one at a time, for all elements). If no error is detected, the bit is restored.

When the isolation process terminates, the output dump contains the following:

- Floating-point instruction buffer
- Data used when the failure occurred
- Simulated execution results
- Actual execution results (if different from the simulated results)
- An exclusive OR of the simulated and actual execution results

If the failure is very intermittent, the isolation process may terminate without detecting an error, and then the output dump does not contain any actual execution results (differences). In this case, increase the value of **isop** *n*, enable the **+repeat** option, select the failing CPU, and use the failing seed to rerun the test.

The program may report an error resulting from a failure in either the simulated or actual execution. To determine if the error is the result of an actual execution failure, start **olcfpt** in a different CPU and select the suspected failing CPU. For example, the following entry starts **olcfpt** in CPU c:

```
olcfpt cpu c
```

If **olcfpt** fails, and the simulated execution is suspect, rerun **olcfpt** using a different master CPU and the failing seed, as follows:

olcfpt cpu a,c +repeat seed *n*

If **olcfpt** fails in CPU c, the failure is in the actual execution of the floating-point instruction buffer. If **olcfpt** does not fail, the error is either in the simulated execution results from CPU c or it is very intermittent.

3.2.3 TEST TERMINATION

For information on test termination, refer to section 2, Confidence Test and Monitor Overview.

3.2.4 TEST EXAMPLES

This subsection contains **olcfpt** execution examples.

The following example runs **olcfpt** for O'10000000 passes. Output is redirected to **olcfpt.log**. The **nohup**(1) command allows the program to continue executing after you log off the system. You can later log on to check the test's progress. The ampersand (**&**) causes the entire command to execute in the background, so that another prompt is immediately displayed and you can continue to use the system.

```
nohup olcfpt maxp 10000000 >olcfpt.log &
```

The following example runs **olcfpt** with selected command options and shell facilities. The test runs for O'1000000 passes in CPU a with all default instructions. The job runs as a background process, and the output is sent to **olcfpt.log**.

```
olcfpt maxp 1000000 cpu a >olcfpt.log &
```

The following example shows a procedure for determining how frequently an error occurs. The test is rerun with the **+repeat** option, so that the first pass is run repeatedly until the test terminates. The test uses the seed value from the output at the time of the initial error. Error isolation is disabled. The output is filtered to **olcfpt.log**.

```
olcfpt +repeat -isolate maxerr 100 maxp 100 cpu d seed
  1436651016713554002511 | tail >olcfpt.log &
```

The following example runs **olcfpt** with floating-point multiply instructions, and instructions 70 and 174.

```
olcfpt +fpmult enable 70,174 >olcfpt.log &
```

The following example runs **olcfpt** with all of the floating-point vector instructions except instructions 166 and 167.

```
olcfpt +vector disable 166,167 >olcfpt.log &
```

The following example runs **olcfpt** with all of the instructions except floating-point multiply.

```
olcfpt -fpmult >olcfpt.log &
```

The following example shows the output displayed when **olcfpt** is run with all default values.

```
olcfpt
```

Output:

```
olcfpt
olcfpt started in cpu A on Tue Aug 25 15:32:16 1987
olcfpt reached maximum pass limit with 1000 passes and 0 errors
on Tue Aug 25 15:32:32 1987
```

The following example runs **olcfpt** with the **+verbose** option enabled so that a line of output is generated after each pass.

```
olcfpt +verbose
```

Output:

```
olcfpt +verbose
olcfpt started in cpu A on Tue Aug 25 11:42:47 1987
olcfpt: pass = 1, error = 0  Tue Aug 25 11:42:47 1987
olcfpt: pass = 2, error = 0  Tue Aug 25 11:42:47 1987
olcfpt: pass = 3, error = 0  Tue Aug 25 11:42:47 1987
    .
    .
    .
olcfpt: pass = 1000, error = 0  Tue Aug 25 11:43:03 1987
olcfpt reached maximum pass limit with 1000 passes and 0 errors
on Tue Aug 25 11:43:03 1987
```

The following example runs **olcfpt** in CPU c only.

```
olcfpt cpu c
```

Output:

```
olcfpt cpu c
olcfpt started in cpu C on Tue Aug 25 11:44:51 1987
olcfpt reached maximum pass limit with 1000 passes and 0 errors
on Tue Aug 25 11:45:07 1987
```

The following example runs **olcfpt** in CPUs a and b, with a as the master. On each pass, **olcfpt** tests a sequence of instructions, using **fpbits** data for the initial register values.

```
    olcfpt +fpbits cpu a,b
```

Output on an error:

```
olcfpt +fpbits cpu a,b
olcfpt started in cpus A, B with master cpu A on Wed Oct 26 10:38:22 1988
CRAY X-MP mode
olcfpt:  restart file written to A34408-olcfpt
name            <        1640> =  'olcfpt   '
rev             <        1641> =  '5.0      '
date            <        1642> =  '10/21/88'
pass            <        1643> =  11
error           <        1644> =  1
seed            <        1645> =  1260350316637024772740
failpat         <        3254> =  'fpbits   '
isop            <        1656> =  1000

random floating-point instruction buffer

                                                  ibuff
```

(*the floating-point instruction buffer is displayed*)

```
   6040a        165431                            V4          V3*RV1
   6040b        063556                            S5          S5-FS6
   6040c        062607                            S6          S0+FS7
   6040d        062031                            S0          S3+FS1
   6041a        066742                            S7          S4+RS2
   6041b        163360                            V3          V6*HV0
   6041c        163125                            V1          V2*HV5
   6041d        174670                            V6          /HV7
   6042a        006000 016400                     J           3500a


initial scalar register data
inits0          <       12740> =  0200777600017777777777
inits1          <       12741> =  1200174777777777777777
inits2          <       12742> =  0201747777740037777777
inits3          <       12743> =  1200070000000000000100
inits4          <       12744> =  0201767777400000000007
inits5          <       12745> =  0277760000000000037777
inits6          <       12746> =  0277607777777777777617
inits7          <       12747> =  1200750077777777777776

initial vector length register
initvl          <       12750> =  0000000000000000000100

initial vector register data
```

(*vector register data is displayed*)

Output (continued):

```
simulated floating-point instruction buffer results
The expected data shown below has the following format:

  name       + index     <offset>  = data ...

 name:     The name of the data dumped on this line.
 index:    The index into the data starting at name.  Optional, default: 0.
 offset:   The offset into the data buffer.
 data:     The actual data dumped.

*** Expected Results ***   cpu A (master)

Source data buffer at 13640 in Memory
   Memory address in source data buffer = <offset> + 13640 (source data buffer)



simulated scalar register data results
s0                  <   1100>  =  1200174777777777777777
s1                  <   1101>  =  1200174777777777777777
s2                  <   1102>  =  0201747777740037777777
s3                  <   1103>  =  1200070000000000000100
s4                  <   1104>  =  0201767777400000000007
s5                  <   1105>  =  1277607777777777777617
s6                  <   1106>  =  1200677777777777777600
s7                  <   1107>  =  0000000000000000000000

simulated vector length register data results
vl                  <   1110>  =  0000000000000000000100

simulated vector register data results
(vector register data is displayed)

Differences are the results from actual execution of the floating-point
instruction buffer that differ from the master (simulated or
actual) execution.

s0-s7   = scalar register data results
vl      = vector length register data result
v0-v7   = vector register data results
The difference data shown below has the following format:

  name       + index     <offset>  = data ...
                                     data differences . . . .
```

Output (continued):

```
name:     The name of the data dumped on this line.
index:    The index into the data starting at name.  Optional, default: 0.
offset:   The offset into the data buffer.
data:     The actual data dumped.
          The differences are marked with an asterisk (*) preceding
          the data word.

data differences: The bits in difference between the actual results and
the expected results.

*** Differences ***   cpu A (master)
Source data buffer at 15640 in Memory copied to save buffer at 112573 in Memory
   Memory address in source data buffer = <offset> + 15640 (source data buffer)
   Memory address in save data buffer = <offset> + 112573 (save data buffer)

actual floating-point buffer execution results

*** Differences ***   cpu B
Source data buffer at 15640 in Memory copied to save buffer at 113705 in Memory
   Memory address in source data buffer = <offset> + 15640 (source data buffer)
   Memory address in save data buffer = <offset> + 113705 (save data buffer)

actual floating-point buffer execution results

s5                <    1105> =  *1277607777776000000000
                                 0000000000001777777617

Beginning error isolation
Error isolation complete

name            <       1640> =  'olcfpt   '
rev             <       1641> =  '5.0      '
date            <       1642> =  '10/21/88'
pass            <       1643> =  11
error           <       1644> =  1
seed            <       1645> =  1260350316637024772740
failpat         <       3254> =  'fpbits   '
isop            <       1656> =  1000

isolation: random floating-point instruction buffer

                                      ibuff
   6040b     063556                        S5          S5-FS6
   6040c     006000 016400                 J           3500a
```

Output (continued):

```
isolation: initial scalar register data
inits0          <      12740> =  0200777600017777777777
inits1          <      12741> =  1200174777777777777777
inits2          <      12742> =  0201747777740037777777
inits3          <      12743> =  1200070000000000000100
inits4          <      12744> =  0201767777400000000007
inits5          <      12745> =  0200000000000000002000
inits6          <      12746> =  0000000000000000000000
inits7          <      12747> =  1200750077777777777776
```

(From this point on, the dump is similar to the previously listed portion of the dump that displayed the unisolated error information.)

```
The first address (FADD) of the diagnostic is 1640a
olcfpt reached maximum error limit with 11 passes and 1 errors
on Wed Oct 26 10:40:37 1988
```

3.2.5 TEST MESSAGES

The **olcfpt** test produces the following types of messages:

- Informative
- Error

These messages are described in the subsections that follow.

3.2.5.1 Informative messages

If no error occurs, **olcfpt** produces two messages, one at start-up time and another at test termination. If the **+verbose** option is enabled, a message is sent to **stdout** (standard output device) after each pass through the test loop.

On an error, the test provides information such as the following:

- Pass and error counts
- Seed at the beginning of the pass on which the error occurred
- Contents of the instruction buffer
- Initial data

- Data results from the simulated instruction execution in the master CPU

- Differences between the simulated execution results from the master CPU and the actual execution results from all of the selected CPUs

3.2.5.2 Error messages

One of the following error messages is sent to **stderr** (standard error device) if an invalid command option is entered:

olcfpt: selins: No executable instructions selected.
Correct and rerun.

olcfpt: selins: Vector length must be in the range of 0 through 100.
Correct the **vl** option and rerun.

olcfpt: No data patterns(s) selected.
All data patterns are deselected. Correct and rerun.

One of the following error messages is sent to **stderr** if **olcfpt** detects an unexpected error. Select a different master CPU and rerun the test. If the problem persists, contact your CRI representative.

olcfpt: simulate: (software error) The *gh* field is greater than 177.

olcfpt: simulate: (software error) The instruction does not have a sim*xxx* routine.

olcfpt: generate: (software error) The instruction does not have a gen*xxx* routine.

3.3 olcm

The **olcm** test is an on-line central memory test. It tests central memory and the paths for the S, T, B, and V registers by using unique algorithms that perform an ascending and descending READ/TEST/WRITE loop of central memory, one word at a time with scalars and one block (100_8) at a time with the T, B, and V registers. **olcm** also has a random-data section and a section to create memory conflicts. **olcm** runs on CX/CEA and CX/1 systems.

3.3.1 TEST SYNOPSIS

The **olcm** command options can be entered in any order. If an option is omitted, the program uses the default value. The test synopsis lists the **olcm** command options and arguments in the following order:

1. Monitor options
2. Test-specific options

Synopsis:

olcm [**chkpnt** *mode*] [**cpu** *clist*] [**cputime** *h*:*m*:*s*] [**+/-getseed**]

[**getseed** *file*] [**help**] [**maxerr** *n*] [**maxp** *n*] [**+/-parcel**] [**time** *h*:*m*:*s*]

[**+/-verbose**] [**+xmp**] [**+cray1**]†

[**section** *slist*] [**seed** *n*] [**words** *n*]

section *slist*

Selects the test sections to be executed. *slist* is entered in the following format:

n,*n*,...,*n*

n can be any of the following test sections, entered in any order (if allowed to default, all test sections are executed):

Section	Description
1	Central memory storage and scalar path test
2	Central memory storage and T register path test

† The monitor command options are described in section 2, Confidence Test and Monitor Overview.

section *slist* (continued)

Section	Description
3	Central memory storage and B register path test
4	Central memory storage and vector register path test using only the first vector logical unit
5	Central memory storage and vector register path test using both vector logical units
6	Central memory random data test
7	Central memory conflict test

seed *n* Sets the random seed to *n*. *n* can be any 64-bit octal value. If *n* is 0, the test reads the real-time clock and uses the value for the initial seed. The default for *n* is O'33. If **seed** *n* is selected, do not select **+getseed** or **getseed** *file*.

words *n* Indicates the number of words to be tested in central memory. *n* is a value in the range O'100 through O'4,000,000. All values are rounded down to the nearest O'100 words. For example, O'150 is rounded down to O'100; O'1000 remains unchanged. The default for *n* is O'3,000.

3.3.2 TEST EXECUTION

The **olcm** execution sequence is as follows:

1. Test initialization
2. Test section execution
3. Comparison of expected and actual data within each test section
4. Error report

Steps 2 and 3 occur on each pass through the test loop. Step 4 occurs only on error.

3.3.2.1 Test initialization

At test initialization, the test information is processed as follows:

1. The number of words to be tested in central memory is validated (**words** *n*).

2. Selected test sections are validated (**section** *slist*).

3. The random seed is validated (**seed** *n*).

3.3.2.2 Test section execution

The subsections that follow describe the **olcm** test sections.

Test section 1 - This section tests central memory storage and the scalar paths.

The following algorithm is used to perform an ascending and descending read/test/write loop of central memory (one word at a time):

1. Write a 64-bit address pattern to all memory locations in the test buffer.

2. Load the scalar register with the pattern from the address register.

3. Verify data integrity by comparing the memory location written with the 64-bit address pattern to the scalar register. Generate a dump on a data miscompare.

4. Write the 64-bit address pattern to the previously tested memory location.

5. Increment location if ascending, or decrement if descending.

6. Repeat steps 2 through 5 until all locations are written.

Test sections 2 and 3 - These sections test the T and B register paths, respectively, and central memory storage.

The algorithm used in test section 1 is used in these test sections to perform an ascending and descending read/test/write loop of central memory. However, in test sections 2 and 3, the algorithm differs as follows:

- Data transfers are done in 64-word blocks (rather than one word at a time).

- Data transfers use ascending memory addresses only (the descending loops contain descending data blocks with ascending addresses).

Test section 4 - This section tests central memory storage and the vector register paths, using only the first vector logical unit.

The algorithm used in test section 1 is used in this test section to perform an ascending and descending test of central memory storage and the vector register paths. However, in test section 4, the algorithm differs as follows:

- Data transfers are done in 64-word blocks (rather than one word at a time).
- Data transfers use negative indexing in the descending test subsections, so that the 64-bit pattern is stored in the vector registers in reverse order of the way the pattern is stored in test sections 2 and 3.

Test section 5 - This section tests central memory storage and the vector register paths, using both vector logical units.

In section 5, the following occurs:

- Vector loads are doubled to force the use of more than one central memory port.
- Vector comparisons are doubled to force the use of both logical units.
- The 64-bit pattern is generated with vector recursion. (In a vector instruction, *vector recursion* results when V*i* and V*j* or V*i* and V*k* refer to the same vector register).

The algorithm used in test section 1 is used in this test section to perform an ascending and descending test of central memory storage and the vector register paths. However, in test section 5, the algorithm differs as follows:

- Data transfers are done in 64-word blocks (rather than one word at a time).
- Data transfers use negative indexing in the descending test subsections, so that the 64-bit pattern is stored in the vector registers in reverse order of the way the pattern is stored in test sections 2 and 3.

Test section 6 - This section tests central memory by generating random data in the subroutine RANCOM. The test does the following in subsection 1:

1. Loads random data (64 bits) into V1 (all 100 elements).
2. Writes V1 to the central memory area under test (the same block of 100 random words is written consecutively, so that each 100th word is the same).

3. The central memory area under test is read into V2.

4. V1 and V2 are compared in V0.

The test does the following in subsection 2:

1. Loads random data (64 bits) into T00 through T77.

2. Writes T00 through T77 to the central memory area under test (the same block of 100 random words is written consecutively, so that each 100th word is the same).

3. The central memory area under test is read into S2.

4. The T registers are loaded into S1, one word at a time.

5. S1 and S2 are compared in S0.

The test does the following in subsection 3:

1. Loads random data (32 bits) into B02 through B77. (B00 and B01 are skipped because they are used for return jumps.)

2. Writes B02 through B77 to the central memory area under test (the same block of 100 random words is written consecutively, so that each 100th word is the same).

3. The central memory area under test is read into A2.

4. The B registers are loaded into A1, one word at a time.

5. A1 and AS are compared in A0.

Test section 7 - This section tests central memory by generating conflicts in the vector reads. The conflicts are generated as follows:

1. Do a vector read from the first memory buffer location to V2, using an increment of 0.

2. Increment the memory location by O'40.

3. Initiate a fetch.

4. Do a vector read from the memory location (from step 2) to V3, using an increment of 0.

5. Compare V2 and V3 to V4.

6. Increment the memory location (from step 1) by O'1000, and write V4 to the new memory location, using an increment of 1.

7. Check for error.

8. Increment the memory location (from step 1) by 1.

9. Repeat steps 1 through 8 until all memory locations are read.

The two vector reads to locations 40-words apart generate section and subsection conflicts. A fetch issued between the two reads generates conflicts in port D.

3.3.2.3 Comparison of expected and actual data

After each test section is executed, the actual results are compared to the expected results. If the results match, the test continues. If the results do not match, the test dumps all of the data related to the suspected failure. After all of the selected sections are run, the pass count is incremented.

3.3.2.4 Error report

If an error is detected, the test dumps all of the data related to the suspected failure. The output dump contains the following:

- Diagnostic Information Blocks (DIBs)
- Section and subsection under test
- Number of central memory words being tested
- Expected results
- Actual results
- Differences
- Address of the code at the time the error was detected
- Buffer address of the data at the time the error was detected

3.3.3 TEST TERMINATION

There are several monitor options that can cause a test to terminate. Refer to the information on test termination in section 2, Confidence Test and Monitor Overview.

3.3.4 TEST EXAMPLES

This subsection contains **olcm** execution examples.

The following example executes **olcm** for a maximum of O'500 passes, testing O'100,000 words of central memory.

```
olcm maxp 500 words 100000
```

The following example executes **olcm** for a maximum of O'1500 passes, with test sections 1 and 5 enabled.

```
olcm maxp 1500 section 1,5
```

The following example executes **olcm** for a maximum of O'1000 passes (default), using an initial seed value of 12345, with test sections 1, 2, 3, 6, and 7 enabled.

```
olcm seed 12345 section 6,3,2,1,7
```

The following example runs **olcm** for O'1000 passes (default), with test sections 1, 2, 3, and 4 enabled. Output is redirected to **olcm.log**. The **nohup**(1) command allows the program to continue executing after you log off the system. You can later log on to check the test's progress. The ampersand (**&**) causes the entire command to execute in the background, so that another prompt is immediately displayed and you can continue to use the system.

```
nohup olcm section 1,2,3,4 >olcm.log &
```

The following example shows the output displayed when **olcm** is run with all default values.

```
olcm
```

Output:

```
olcm
olcm started in cpu A on Mon Jul 18 11:14:10 1988
CRAY Y-MP MODE
olcm reached maximum pass limit with 1000 passes and 0 errors
on Mon Jul 18 11:14:42 1988
```

The following example executes **olcm** for a maximum of O'1000 passes (default), testing O'150 words of central memory.

```
olcm words 150
```

Output:

```
olcm words 150
olcm started in cpu A on Fri Jul 15 15:30:12 1988
CRAY Y-MP MODE
The value for words was rounded down to the nearest 100 octal words
olcm reached maximum pass limit with 1000 passes and 0 errors
on Fri Jul 15 15:30:47 1988
```

The following example executes **olcm** for 0 passes (terminated on error), testing O'1234 words of central memory.

```
    olcm words 1234
```

Output (on error):

```
olcm words 1234
olcm started in cpu A on Mon Jul 18 14:58:37 1988
CRAY Y-MP MODE
 The value for words was rounded down to the nearest 100 octal words
olcm:  restart file written to A5392-olcm
name            <        550> =  'olcm    '
rev             <        551> =  '5.0     '
date            <        552> =  '07/18/88'
pass            <        553> =  0
error           <        554> =  1
seed            <        555> =  33
failsec         <        556> =  1
words           <        603> =  1200
subs            <       2753> =  2
lower           <       2755> =  13270
upper           <       2756> =  14470
$dif            <       2765> =  0000000000000000000004
$exp            <       2763> =  0000000000000000004467
$act            <       2764> =  0000000000000000004463
$elem           <       2766> =  0000000000000000000000
$vm             <       2767> =  0000000000000000000000
Error Address of the executing code
errcode         <       2760> =  0000000000000000004577
Error Address of the data area
errdata         <       2761> =  0000000000000000014467
A registers at the time of error
savea           <       4340> =  0000000000000000000000  ...
savea    + 0004 <       4344> =  0000000000000001100333  ...

S registers at the time of error
saves           <       4350> =  0000000000000000001234  ...
saves    + 0004 <       4354> =  0000000000000000000000  ...

B registers (sections 3 and 6 only)
$actb           <       3640> =  0000000000000000000000  ...
$actb    + 0004 <       3644> =  0000000000000000000000  ...
$actb    + 0010 <       3650> =  0000000000000000000000  ...
    .
    .
    .
$actb    + 0074 <       3734> =  0000000000000000000000  ...
```

Output (continued):

```
T registers (sections 2 and 6 only)
$actt           <       3740> =  0000000000000006720344  ...
$actt    + 0004 <       3744> =  0000000015033227672440  ...
$actt    + 0010 <       3750> =  0000356647785921190300  ...
     .
     .
     .
$actt    + 0074 <       4034> =  3987564008722334539870  ...

V0 - Difference (section 6 only)
$difv0          <       4040> =  0000000000000000000000  ...
$difv0   + 0004 <       4044> =  0000000000000000000000  ...
$difv0   + 0010 <       4050> =  0000000000000000000000  ...
     .
     .
     .
$difv0   + 0074 <       4034> =  0000000000000000000000  ...

V1 - Expected (section 6 only)
$expv1          <       4140> =  0000000000000000000000  ...
$expv1   + 0004 <       4144> =  0000000000000000000000  ...
$expv1   + 0010 <       4150> =  0000000000000000000000  ...
     .
     .
     .
$expv1   + 0074 <       4234> =  0000000000000000000000  ...

V2 - Actual (section 6 only)
$actv2          <       4240> =  0000000000000000000000  ...
$actv2   + 0004 <       4244> =  0000000000000000000000  ...
$actv2   + 0010 <       4250> =  0000000000000000000000  ...
     .
     .
     .
$actv2   + 0074 <       4334> =  0000000000000000000000  ...

The first address (FADD) of the diagnostic is 550a

olcm reached maximum pass limit with 0 passes and 1 errors
on Mon Jul 18 14:58:37 1988
```

3.3.5 TEST MESSAGES

The **olcm** test produces the following types of messages:

- Informative
- Error

These messages are described in the subsections that follow.

3.3.5.1 Informative messages

If no error occurs, **olcm** produces two messages, one at start-up time and another at test termination. If the **+verbose** option is enabled, a message is sent to **stdout** (standard output device) after each pass through the test loop.

If the value for **words** *n* is rounded down to the nearest O'100 words, the following informative message is displayed:

> The value for words was rounded down to the nearest 100 octal words.

If the value for **seed** *n* is set to 0, the following informative message is displayed:

> Seed selected was 0, so the test read RTC to initial seed.

3.3.5.2 Error messages

One of the following error messages is sent to **stderr** (standard error device) if an invalid command option is entered:

> Invalid section selected. Valid sections are: 1, 2, 3, 4, 5, 6, and 7.
> Rerun **olcm** using a valid value for **section** *slist*.

> Number of words selected is too small (minimum is 100 octal).
> Rerun **olcm** using a valid value for **words** *n*.

> Number of words selected is too large (maximum is 4,000,000 octal).
> Rerun **olcm** using a valid value for **words** *n*.

> System could not allocate words; words selected may be too large.
> Rerun **olcm** using a smaller value for **words** *n*.

3.3.5.3 Error output definitions

The following are definitions of the output that is dumped on error. Refer to section 3.3.4, Test Examples, for an example of error output.

Output	Description
failsec	Test section that was executing when the error occurred
words	Size of the central memory buffer being tested
subs	Subsection of the test section
lower	Address of the beginning of the buffer defined by words
upper	Address of the end of the buffer defined by words
errcode	Address where the test code was executing
errdata	Address within the central memory buffer that was being tested at the time the error occurred

3.4 olcrit

The **olcrit** test is an on-line comprehensive random instruction test. It randomly generates instructions and data to detect instruction-sensitive and data-sensitive sequence failures. The generated instructions are simulated and then executed. The simulation and execution results are compared, and any differences are reported. If an error is detected, the diagnostic attempts to isolate the failing instruction sequence. The test generates, simulates, executes, and compares new instructions and data until the maximum pass, error, or time limit is reached.

The **olcrit** test runs under the confidence monitor program, **olcmon**. The **olcmon** monitor compares the test simulation and execution results. For additional information on **olcmon**, refer to section 2, Confidence Test and Monitor Overview.

3.4.1 TEST SYNOPSIS

The **olcrit** command options can be entered in any order. If an option is omitted, the program uses the default value. The test synopsis lists the **olcrit** command options and arguments in the following order:

1. Monitor options
2. Test-specific options
3. Data pattern options
4. Instruction options

Synopsis:

olcrit [**chkpnt** *mode*] [**cpu** *clist*] [**cputime** *h*:*m*:*s*] [**+/-getseed**]

[**getseed** *file*] [**help**] [**maxerr** *n*] [**maxp** *n*] [**+/-parcel**] [**time** *h*:*m*:*s*]

[**+/-verbose**] [**+xmp**] [**+cray1**]†

[**+/-cluster**] [**cluster** *n*] [**disable** *ilist*] [**enable** *ilist*]

[**+/-isolate**] [**isop** *n*] [**numins** *n*] [**+/-repeat**] [**seed** *n*] [**vl** *n*]

[**+/-vload**]

[**+/-bits**] [**+/-onezero**] [**+/-random**]

[**+/-address**] [**+/-ci**] [**+/-cm**] [**+/-ema**] [**+/-fpadd**] [**+/-fpmult**]

[**+/-fprecip**] [**+/-int**] [**+/-jump**] [**+/-logical**] [**+/-pop**] [**+/-scalar**]

[**+/-shift**] [**+/-shr**] [**+/-vector**]

+/-cluster
Enables (**+cluster**) or disables (**-cluster**) cluster selection. This option is recommended only for sites that run multitasking jobs. If a site runs multitasking jobs and **olcrit** detects a failure in the shared registers, the only way to determine which cluster was used is to enable the **+cluster** option. However, selecting a specific cluster with the **cluster** *n* option does not ensure that **olcrit** will be able to access that cluster immediately. The UNICOS scheduler must wait for that cluster to become available. The default is **-cluster**.

† The monitor command options are described in section 2, Confidence Test and Monitor Overview.

cluster *n*

Selects a specific cluster. *n* can be any one of the following cluster numbers associated with the indicated mainframe (cluster number 1 is reserved for the operating system):

Mainframe	Cluster Numbers
CRAY Y-MP/8	2, 3, 4, 5, 6, 7, 10, 11
CRAY Y-MP/4	2, 3, 4, 5
CRAY X-MP/4	2, 3, 4, 5
CRAY X-MP/2	2, 3
CRAY X-MP/1	2, 3

If **cluster** *n* is selected, the **+cluster** option must also be selected. The default for *n* is a random cluster number.

disable *ilist*

Deselects specific instructions. Enter *ilist* in the following format:

n,*n*,...,*n*

n is the octal value in the *gh* or *ghijk* field of the specific instruction. If the *gh* field does not specify a unique instruction, the *ijk* field can be used to deselect a specific instruction. For example, the following instructions all have the same *gh* field:

030*j*0, 036*jk*, 037*jk*

To deselect the preceding instructions, you must specify the *ghijk* field, as follows:

```
disable 03000,03600,03700
```

The **disable** *ilist* option overrides the **enable** *ilist* option and any selected (+) or deselected (-) instruction options.

enable *ilist*

Selects specific instructions. Enter *ilist* in the following format:

n,*n*,...,*n*

enable *ilist*
(continued)

n is the octal value in the *gh* or *ghijk* field of the specific instruction. If the *gh* field does not specify a unique instruction, the *ijk* field can be used to select a specific instruction. For example, the following instructions all have the same *gh* field:

0030*j*0, 0036*jk*, 0037*jk*

To select the preceding instructions, you must specify the *ghijk* field, as follows:

```
enable 003000,003600,003700
```

The **enable** *ilist* option overrides any selected (+) or deselected (-) instruction options. When the test is run with default values for the +/- instruction options, and the **enable** *ilist* option is selected, only the instructions specified by the **enable** *ilist* option are run.

When using the **enable** option to select any of the following instructions, **numins** *n* should be greater than 1 or the selected instructions will not be placed in the instruction buffer:

34 through 37
56, 57, 76, 77
100 through 130
150 through 153
176, 177

All of these instructions use an A register for operations such as an index or a shift count. Before each of the selected instructions is executed, the test executes an A register load instruction. As a result, if **numins** is set to 1, there is no buffer space remaining for the instruction using the A register.

+/-isolate

Enables (**+isolate**) or disables (**-isolate**) the error isolation option. The default is **+isolate.**

isop *n* Sets the isolation pass limit to *n* (octal). During isolation, the diagnostic repeatedly executes the suspected failing sequence. If the sequence fails, the loop terminates and the diagnostic attempts to isolate the sequence further. If the sequence does not fail, the loop terminates after *n* passes, and **olcrit** assumes that the error is not in the tested sequence. The default for *n* is O'1000.

numins *n* Sets the number of instructions to be generated. *n* can be any octal value within the range 1 through O'2000. The default for *n* is O'200.

+/-repeat

Enables (**+repeat**) or disables (**-repeat**) the option that repeats the first pass until the diagnostic terminates. **+repeat** is useful for recreating an error. It is normally used with one of the following options: **seed** *n*, **+getseed**, **getseed** *file*, or **+cluster** together with **cluster** *n*. The default is **-repeat** (the program generates new instructions and data after each pass).

seed *n* Sets the random seed to *n*. *n* can be any 64-bit octal value. If *n* is 0, the test reads the real-time clock and uses the value for the initial seed. The default for *n* is O'33. If **seed** *n* is selected, do not select **+getseed** or **getseed** *file*.

vl *n* Sets the vector length to *n*. *n* can be any octal value within the range 0 through O'100. The default for *n* is 0.

If **vl** is set to 0, a random **vl** value is used to initialize the test and the value may change during the execution of the random instruction buffer.

If the **vl** value is within the range 1 through O'100, instruction 00200*k* is disabled. The **vl** value is initialized to *n* and remains set to *n* during the execution of the random instruction buffer. However, if instruction 00200*k* is selected by the **enable** option, the **vl** value is initialized to *n* and may change each time a 00200*k* instruction is executed in the random instruction buffer.

+/-vload Selects (**+vload**) or deselects (**-vload**) vector instructions for the instruction buffer and, in the case of **-vload**, does not allow you to load (write) or save (read) the vector registers. **-vload** overrides vector instructions selected by **+vector** and **enable** *ilist*. The default is **+vload**.

+/-bits, +/-onezero, +/-random

Selects (+) or deselects (-) specific data patterns. If allowed to default, all of the data patterns are run. The selected data patterns are used for the initial register and memory values. However, the vector length (VL) register is always initialized with 6-bits of random data. The data patterns are as follows:

+/-bits, +/-onezero, +/-random
(continued)

Option	Data Pattern
bits	Random number of consecutive 1-bits in a word. For example: 0000017777777776000000 1777000000000000000377 1777777777777777777777 0000000000000000000000 0000000000100000000000
onezero	Random selection of all 1's or all 0's in a word. For example: 1777777777777777777777 0000000000000000000000
random	Random bit generation in a word. For example: 1023122123232122777127 0003423100233344322177 1640034356453221213532 1123235467543221322120 1304322300332105534311

+/-address, +/-ci, +/-cm, +/-ema, +/-fpadd, +/-fpmult, +/-fprecip, +/-int, +/-jump, +/-logical, +/-pop, +/-scalar, +/-shift, +/-shr, +/-vector

Selects (+) or deselects (-) specific instruction groups for the following options:

Option	Instruction Type
address	Address register
ci	Compressed index
cm	Central memory
ema	Extended memory addressing
fpadd	Floating-point addition
fpmult	Floating-point multiply
fprecip	Floating-point reciprocal
int	Integer
jump	Jump
logical	Logical
pop	Population/parity count
scalar	Scalar register
shift	Shift
shr	Shared register
vector	Vector register

+/-address, +/-ci, +/-cm, +/-ema, +/-fpadd, +/-fpmult, +/-fprecip, +/-int, +/-jump, +/-logical, +/-pop, +/-scalar, +/-shift, +/-shr, +/-vector (continued)

The instruction groups are as follows:

Option	CX/CEA Instructions	CRAY-1 Instructions
address	001000, 00200*k* 002200, 002300 002500 through 002700 01*hijkm* 010*ijkm* through 013*ijkm* 020 through 022 023*i*01 024, 025 026*ij*7, 027*ij*7 030 through 032 034, 035 10*hijkm*, 11*hijkm*	001000, 00200*k* 010*ijkm* through 013*ijkm* 020 through 022 023*i*01 024, 025 030 through 032 034, 035 10*hijkm*, 11*hijkm*
ci	175*ij*4, 175*ij*5 175*ij*6, 175*ij*7	None
cm	10*h* through 13*h* 34 through 37 176*i*00, 1770*j*0	10*h* through 13*h* 34 through 37 176*i*00, 1770*j*0
ema†	01h*ijkm*	None
fpadd	062, 063 170 through 173	062, 063 170 through 173
fpmult	064 through 067 160 through 167	064 through 067 160 through 167
fprecip	070, 174*ij*0	070, 174*ij*0
int	030 through 032 060 through 061 154 through 157	030 through 032 060 through 061 154 through 157

† Extended memory instructions are not available on CEA systems in Y-mode.

+/-address, +/-ci, +/-cm, +/-ema, +/-fpadd, +/-fpmult, +/-fprecip, +/-int, +/-jump, +/-logical, +/-pop, +/-scalar, +/-shift, +/-shr, +/-vector (continued)

Option	CX/CEA Instructions	CRAY-1 Instructions
jump	005, 006, 007 010 through 017	005, 006, 007 010 through 017
logical	042 through 051 140 through 147 175	042 through 051 140 through 147 175
pop	026*ij*0, 026*ij*1 027*ij*0 174*ij*1, 174*ij*2	026*ij*0, 026*ij*1 027*ijx* 174*ij*1, 174*ij*2
scalar	0036*jk*, 0037*jk* 014*jkm* through 017*jkm* 023*ij*0 026*ij*0, 026*ij*1 027*ij*0 036 through 071 072*i*02, 072*ij*3 073*i*02, 073*ij*3 074, 075 12*hijkm*, 13*hijkm*	014*jkm* through 017*jkm* 023*ij*0 026*ij*0, 026*ij*1 027*ij*0 036 through 071 074, 075 12*hijkm*, 13*hijkm*
shift	052 through 057 150 through 153	052 through 057 150 through 153
shr	0036*jk*, 0037*jk* 026*ij*7, 027*ij*7 072*i*02, 072*ij*3 073*i*02, 073*ij*3	None
vector	0030*j*0, 073*i*00 076, 077 140 through 177	003, 073, 076, 077 140 through 177

The diagnostic does not currently execute the following instructions in the random instruction buffer: 0, 002400, 0034*jk*, 4, 33, 072*i*00, 073*ij*1, 176*i*0*k*, 176*i*1*k*, 1770*jk*, 1771*jk*.

+/-address, +/-ci, +/-cm, +/-ema, +/-fpadd, +/-fpmult, +/-fprecip, +/-int, +/-jump, +/-logical, +/-pop, +/-scalar, +/-shift, +/-shr, +/-vector (continued)

If allowed to default on a CEA system in Y-mode, all instruction groups are selected with the following exceptions:

- If the cluster number assigned to the job is 0, the shared register (**shr**) instruction group is deselected.

- The extended memory addressing (**ema**) instruction group is deselected.

If allowed to default on a CRAY X-MP computer system, all instruction groups are selected with the following exception: if extended memory addressing (**ema**) or compressed index (**ci**) hardware is not present in the system, the **ema** and **ci** instruction groups are deselected, respectively.

If allowed to default on a CRAY-1 computer system, all instruction groups are selected except **ema**, **ci**, and **shr**. However, the vector population count and parity (**pop**) instruction group is selected only if **pop** hardware is present in the system.

3.4.2 TEST EXECUTION

The **olcrit** execution sequence is as follows:

1. Test initialization and hardware configuration detection
2. Random instruction and data generation
3. Random instruction buffer simulation
4. Random instruction buffer execution
5. Comparison of simulation and execution results
6. Error isolation

Hardware configuration detection occurs only at test initiation. Steps 2 through 5 occur on each pass through the test loop. Step 6 occurs only on error.

3.4.2.1 Test initialization and hardware configuration detection

At test initialization, instructions are processed in the following order:

1. All instructions are initially enabled unless either of the following occurs (in which case no instructions are initially enabled):
   - An instruction group is selected (+*option*)
   - An **enable** option is entered and there are no deselected (-*option*) instruction group entries
2. Selected groups are processed, enabling instructions in the selected groups.
3. Deselected groups are processed, disabling instructions in the deselected groups.
4. If the **vl** option is set to a value within the range 1 through O'100, instruction 00200*k* is deselected.
5. Individually selected instructions are processed (all instructions specified by the **enable** option).
6. Individually deselected instructions are processed (all instructions specified by the **disable** option).
7. Any vector instructions disabled by -**vload** are processed.
8. If no instructions are selected, an error message is displayed and the test is terminated.

The hardware configuration detection routine determines which of the following computer systems is configured:

- CRAY X-MP computer system
- CRAY-1 computer system

Then the hardware configuration detection routine adjusts testing accordingly, by determining the following:

Mainframe	Hardware Configuration Detection Routine
CEA (Y-mode)	Determines whether cluster 0 is in use
CRAY X-MP	Determines whether the system contains extended memory addressing and/or compressed indexing hardware, and whether cluster 0 is in use

Mainframe	Hardware Configuration Detection Routine
CRAY-1	Determines whether the system contains a vector population count functional unit

After determining the hardware characteristics, the routine writes a message to **stdout** to indicate the type of system detected, and disables all instructions that are not available because of hardware constraints.

Instruction generation is dependent on the hardware configuration detected, as follows (you can use **+/-ci**, **+/-ema**, **+/-pop**, or **+/-shr** to override this default instruction generation process):

Mainframe — **Instructions Generated**

CEA (Y-mode): All instructions except extended memory addressing instructions are generated

CRAY X-MP: All instructions are generated with the following exception: compressed indexing and extended memory instructions are generated only if present in the hardware.

CRAY-1: All instructions are generated except the following:

- A load VL instruction (00200*k*)
- Scatter/gather/compressed indexing instructions
- Extended memory instructions
- Shared register instructions
- Vector pop/parity instructions are generated only if the hardware contains a vector population count functional unit.

3.4.2.2 Random instruction and data generation

These routines build and generate the random instruction buffer and initial data. Instructions for the buffer are randomly selected from a list of enabled instructions. The values of the *i*, *j*, and *k* fields are randomly selected when appropriate.

3.4.2.3 Random instruction buffer simulation

After the instructions and data are generated, the random instruction buffer is simulated by the master CPU only. The **save** monitor routine saves the results.

Each instruction type has a unique simulation routine. The simulation routines use machine resources differently from the instruction being simulated. For example, the address multiply functional unit may be simulated with the floating-point multiply functional unit.

3.4.2.4 Random instruction buffer execution

After the instructions are simulated, all of the selected CPUs execute the random instruction buffer code. Before the instructions can be executed, the program loads the following:

- Vector registers
- Vector length register
- Vector mask register
- Address registers
- B registers
- T registers
- Semaphore registers
- Shared T registers
- Shared B registers
- Scalar registers
- Central memory

Then an unconditional jump to the random instruction buffer is executed. At the end of the random instruction buffer is an unconditional jump to a routine that unloads the contents of the registers and central memory. The **save** monitor routine saves the results.

3.4.2.5 Comparison of simulation and execution results

After the instructions are executed in all of the selected CPUs, the **compare** monitor routine compares the results, and one of the following actions occurs:

- If the results match, the test proceeds with the next data pattern. After all of the selected data patterns are run, the pass count is incremented.

- If the results do not match, the test dumps all of the data related to the suspected failure and, if the isolation option is enabled (**+isolate**), attempts to isolate the failure.

3.4.2.6 Error isolation

If an error is detected and the isolation option is enabled (**+isolate**), the test attempts to reduce the random instruction buffer to the minimum number of failing instructions. The isolation process consists of two parts.

In the first part of the isolation process, the instruction buffer is shortened from the end, one instruction at a time. The isolation routine initially tests the number of instructions to be generated minus one (**numins** *n*-1). The routine executes until the specified number of passes is reached (**isop** *n*) or an error is detected. If an error is detected, the number of instructions tested is decremented by one, and testing continues for **isop** *n* passes. This process continues until no errors are detected or there are no remaining instructions to be tested.

If there are no remaining instructions to be tested and the test detects an error resulting from loading and unloading the registers, the test generates an output dump and the isolation process terminates.

In the second part of the isolation process, the last instruction removed is tested by itself for **isop** *n* passes. If an error is not detected, the last instruction removed and the instruction preceding it in the random instruction buffer are tested for **isop** *n* passes. Until the program detects an error or reaches the beginning of the instruction buffer, one more preceding instruction is added to the test sequence on each iteration of the isolation process.

When the isolation process terminates, the output dump contains the following:

- Isolated instruction buffer
- Data used when the failure occurred
- Simulated execution results
- Actual execution results (if different from the simulated results)
- An exclusive OR of the simulated and actual execution results

If the failure is very intermittent, the second part of the isolation process may terminate without detecting an error, and then the output dump will not contain any actual execution results (differences). In this case, increase the value of **isop** *n*, enable the **+repeat** option, select the failing CPU, and use the failing seed to rerun the test.

The program may report an error resulting from a failure in either the simulated or actual execution. To determine if the error is the result of an actual execution failure, start **olcrit** in a different CPU and select the suspected failing CPU. For example, the following entry starts **olcrit** in CPU c:

```
olcrit cpu c
```

If **olcrit** fails, and the simulated execution is suspect, rerun **olcrit** using a different master CPU and the failing seed, as follows:

```
olcrit cpu a,c +repeat seed n
```

If **olcrit** fails in CPU c, the failure is in the actual execution of the random instruction buffer. If **olcrit** does not fail, the error is either in the simulated execution results from CPU c or it is very intermittent.

3.4.3 TEST TERMINATION

For information on test termination, refer to section 2, Confidence Test and Monitor Overview.

3.4.4 TEST EXAMPLES

This subsection contains **olcrit** execution examples.

The following example runs **olcrit** for O'10000000 passes. Output is redirected to **crit.log**. The **nohup**(1) command allows the program to continue executing after you log off the system. You can later log on to check the test's progress. The ampersand (&) causes the entire command to execute in the background, so that another prompt is immediately displayed and you can continue to use the system.

```
nohup olcrit maxp 10000000 >crit.log &
```

The following example runs **olcrit** with selected command options and shell facilities. The test runs for O'1000000 passes in CPU b with all default instructions. The job runs as a background process, and output is sent to **crit.log**.

```
olcrit maxp 1000000 cpu b >crit.log &
```

The following example shows a procedure for determining how frequently an error occurs. The test is rerun with the **+repeat** option, so that the first pass is run repeatedly until the test terminates. The test uses the seed value from the output at the time of the initial error. Error isolation is disabled. The output is filtered to **crit.log**

```
olcrit +repeat -isolate maxerr 100 maxp 100 cpu d seed
  1436651016713554002511 | tail >crit.log &
```

The following example runs **olcrit** with floating-point and vector instructions.

```
olcrit +fpadd +fpmult +fprecip +vector >crit.log &
```

The following example runs **olcrit** with all of the vector instructions except instructions 146 and 147.

```
olcrit +vector disable 146,147 >crit.log &
```

The following example runs **olcrit** with instructions 026*ij*0, 026*ij*1, 026*ij*7, 031, and 072*i*02.

```
olcrit enable 26,31,072002 &
```

The following example runs **olcrit** with all of the default instructions except floating-point add and multiply.

```
olcrit -fpadd -fpmult >crit.log &
```

The following example shows the output displayed when **olcrit** is run with all default values.

```
olcrit
```

Output:

```
olcrit
olcrit started in cpu A on Tue Aug 25 11:32:08 1987
CRAY X-MP MODE
olcrit reached maximum pass limit with 1000 passes and 0 errors
on Tue Aug 25 11:32:18 1987
```

The following example runs **olcrit** with the **+verbose** option enabled so that a line of output is generated after each pass.

```
olcrit +verbose
```

Output:

```
olcrit +verbose
olcrit started in cpu A on Tue Aug 25 11:42:47 1987
CRAY X-MP MODE
olcrit: pass = 1, error = 0  Tue Aug 25 11:42:47 1987
olcrit: pass = 2, error = 0  Tue Aug 25 11:42:47 1987
olcrit: pass = 3, error = 0  Tue Aug 25 11:42:47 1987
    .
    .
    .
olcrit: pass = 1000, error = 0  Tue Aug 25 11:42:57 1987
olcrit reached maximum pass limit with 1000 passes and 0 errors
on Tue Aug 25 11:42:57 1987
```

The following example runs **olcrit** for 10 seconds (wall-clock time) in CPU c only.

```
olcrit cpu c time 10
```

Output:

```
olcrit cpu c time 10
olcrit started in cpu C on Tue Aug 25 11:44:51 1987
CRAY X-MP MODE
olcrit reached maximum time limit with 1016 passes and 0 errors
on Tue Aug 25 11:45:01 1987
```

The following example runs **olcrit** in CPUs a and b, with b as the master. On each pass, **olcrit** tests a sequence of 15 instructions, using random data for the initial register and memory values.

```
olcrit numins 15 +random cpu b,a
```

Output on an error:

```
olcrit numins 15 +random cpu b,a
olcrit started in cpus A, B with master cpu B on Tue Mar 1 12:40:37 1988
olcrit:  restart file written to B67350-olcrit
CRAY X-MP MODE
name           <       2100> =  'olcrit  '
rev            <       2101> =  '4.0     '
date           <       2102> =  '03/01/88'
pass           <       2103> =  31
error          <       2104> =  1
seed           <       2105> =  1114623621420641250446
failpat        <       4027> =  'random  '
isop           <       2116> =  1000
numins         <       2107> =  15
```

Output (continued):

random instruction buffer

```
                                         ibuff
  10100a    144744                           V7         S4 V4
  10100b    061032                           S0         S3-S2
  10100c    012000 042400                    JAP        10500a
  10101a    020000 026211                    A0         00026211
  10101c    077406                           V4,A6      S0
  10101d    144107                           V1         S0 V7
  10102a    030367                           A3         A6+A7
  10102b    002700                           CMR
  10102c    037705                           0,A0       T05,A7
  10102d    067045                           S0         S4*IS5
  10103a    020600 000172                    A6         00000172
  10103c    007000 042410                    R          10502a
  10104a    021033 031327                    A0         #06631327
  10104c    006000 021200                    J          4240a
```

jump buffer (used by the random instruction buffer)

```
                                         jbuff
  10500a    001000                           PASS
  10500b    110000 026400                    26400,0    A0
  10500d    001000                           PASS
  10501a    006000 040404                    J          10101a
  10501c    000000                           ERR
  10501d    000000                           ERR
  10502a    024100                           A1         B00
  10502b    110100 026401                    26401,0    A1
  10502d    001000                           PASS
  10503a    005000                           J          B00
  10503b    000000                           ERR
  10503c    000000                           ERR
  10503d    000000                           ERR
```

```
initial address register data
inita0          <       21600> = 0000000000000016317572
inita1          <       21601> = 0000000000000017662707
inita2          <       21602> = 0000000000000066352041
inita3          <       21603> = 0000000000000066313277
inita4          <       21604> = 0000000000000014173556
inita5          <       21605> = 0000000000000027243236
inita6          <       21606> = 0000000000000055114565
inita7          <       21607> = 0000000000000006421710
```

Output (continued):

```
initial scalar register data
inits0        <     21610> = 0570435766134171410070
inits1        <     21611> = 0657045641432164307775
inits2        <     21612> = 0362774051154520352750
inits3        <     21613> = 1427136526115123426026
inits4        <     21614> = 1510553624661224560223
inits5        <     21615> = 1734474576202245120017
inits6        <     21616> = 1460472150234237442222
inits7        <     21617> = 1214375337067423156017
```

initial vector length and mask register data

(*vector length and mask register data is displayed*)

initial central memory data

(*central memory data is displayed*)

initial jump data (octal ones pattern)

(*jump data is displayed*)

initial vector register data

(*vector register data is displayed*)

initial shared B register data

(*shared B register data is displayed*)

initial shared T register data

(*shared T register data is displayed*)

initial semaphore register data

(*semaphore register data is displayed*)

initial B register data

(*B register data is displayed*)

initial T register data

(*T register data is displayed*)

simulated random instruction buffer results

The expected data shown below has the following format:

The expected data shown below has the following format:

```
  name       + index     <offset>  = data ...
```

name:	The name of the data dumped on this line.
index:	The index into the data starting at name. Optional, default: 0.
offset:	The offset into the data buffer.
data:	The actual data dumped.

Output (continued):

```
*** Expected Results ***   cpu B (master)

Source data buffer at 22100 in Memory
  Memory address in source data buffer = <offset> + 22100 (source data buffer)


simulated address register data results
a0                   <  2500>  =  0000000000000071146450
a1                   <  2501>  =  0000000000000000040420
a2                   <  2502>  =  0000000000000066352041
a3                   <  2503>  =  0000000000000055114565
a4                   <  2504>  =  0000000000000014173556
a5                   <  2505>  =  0000000000000027243236
a6                   <  2506>  =  0000000000000000000172
a7                   <  2507>  =  0000000000000006421710

simulated scalar register data results
s0                   <  2510>  =  0600005600346143005524
s1                   <  2511>  =  0657045641432164307775
s2                   <  2512>  =  0362774051154520352750
s3                   <  2513>  =  1427136526115123426026
s4                   <  2514>  =  1510553624661224560223
s5                   <  2515>  =  1734474576202245120017
s6                   <  2516>  =  1460472150234237442222
s7                   <  2517>  =  1214375337067423156017
```

simulated vector length and mask register data results
(*vector length and mask register data is displayed*)

simulated central memory data results
(*central memory data is displayed*)

simulated jump data results
(*jump data is displayed*)

simulated vector register data results
(*vector register data is displayed*)

simulated shared B register data results
(*shared B register data is displayed*)

simulated shared T register data results
(*shared T register data is displayed*)

simulated semaphore register data results
(*semaphore register data is displayed*)

simulated B register data results
(*B register data is displayed*)

Output (continued):

```
simulated T register data results
```
(T register data is displayed)

Differences are the results from actual execution of the random instruction buffer that differ from the master (simulated or actual) execution.

```
a0-a7    = address register data results
s0-s7    = scalar register data results
vl       = vector length register data results
vm       = vector mask register data results
cm       = central memory data results
jmp      = jump buffer data results
v0-v7    = vector register data results
sb       = sb0-sb7 register data results
st       = st0-st7 register data results
sm       = semaphore register data result
br       = b00-b77 register data results
tr       = t00-t77 register data results
```

The difference data shown below has the following format:

```
  name        + index     <offset>  = data ...
                                      data differences ....
```

```
name:    The name of the data dumped on this line.
index:   The index into the data starting at name.  Optional, default: 0.
offset:  The offset into the data buffer.
data:    The actual data dumped.
         The differences are marked with an asterisk (*) preceding the data word.
data differences:  The bits that differ between the actual results and
                   the expected results.
```

```
*** Differences ***   cpu B (master)

Source data buffer at 25100 in Memory copied to save buffer at 106362 in Memory
   Memory address in source data buffer = <offset> + 25100 (source data buffer)
   Memory address in save data buffer   = <offset> + 106362 (save data buffer)


actual random buffer execution results

a3                 <    2503>  = *0000000000000063536475
                                  0000000000000036422110


*** Differences ***   cpu A
```

Output (continued):

```
Source data buffer at 25100 in Memory copied to save buffer at 106362 in Memory
   Memory address in source data buffer = <offset> + 25100 (source data buffer)
   Memory address in save data buffer   = <offset> + 106362 (save data buffer)

actual random buffer execution results

a3                 <   2503> = *00000000000000063536475
                                 00000000000000036422110


Beginning error isolation
Error isolation complete

name            <       2100> =  'olcrit  '
rev             <       2101> =  '4.0     '
date            <       2102> =  '03/01/88'
pass            <       2103> =  31
error           <       2104> =  1
seed            <       2105> =  1114623621420641250446
failpat         <       4027> =  'random  '
isop            <       2116> =  1000
numins          <       2107> =  15

isolation: random instruction buffer

                                           ibuff
  10102a     030367                             A3        A6+A7
  10102b     006000 021200                      J         4240a


jump buffer (may be used by the isolated random instruction buffer)

                                           jbuff
  10500a     001000                             PASS
  10500b     110000 026400                      26400,0   A0
  10500d     001000                             PASS
  10501a     006000 040404                      J         10101a
  10501c     000000                             ERR
  10501d     000000                             ERR
  10502a     024100                             A1        B00
  10502b     110100 026401                      26401,0   A1
  10502d     001000                             PASS
  10503a     005000                             J         B00
  10503b     000000                             ERR
  10503c     000000                             ERR
  10503d     000000                             ERR
```

Output (continued):

```
isolation: initial address register data
inita0          <      21600> =  0000000000000000026211
inita1          <      21601> =  0000000000000017662707
inita2          <      21602> =  0000000000000066352041
inita3          <      21603> =  0000000000000066313277
inita4          <      21604> =  0000000000000014173556
inita5          <      21605> =  0000000000000027243236
inita6          <      21606> =  0000000000000055114565
inita7          <      21607> =  0000000000000006421710

isolation: initial scalar register data
inits0          <      21610> =  1044142454740403053056
inits1          <      21611> =  0657045641432164307775
inits2          <      21612> =  0362774051154520352750
inits3          <      21613> =  1427136526115123426026
inits4          <      21614> =  1510553624661224560223
inits5          <      21615> =  1734474576202245120017
inits6          <      21616> =  1460472150234237442222
inits7          <      21617> =  1214375337067423156017
```

(From this point on, the dump is similar to the previously listed portion of the dump that displayed the unisolated error information.)

```
The first address (FADD) of the diagnostic is 2100a
olcrit reached maximum error limit with 31 passes and 1 errors
at Tue Mar 1 12:40:59 1988
```

3.4.5 TEST MESSAGES

The **olcrit** test produces the following types of messages:

- Test mode
- Informative
- Error

These messages are listed in the subsections that follow.

3.4.5.1 Test mode messages

During test execution, one of the following informational messages is displayed to indicate the test mode:

CRAY Y-MP MODE
: Indicates that the mainframe is a CEA system in Y-mode.

CRAY Y-MP MODE, shared register testing disabled
Indicates that the mainframe is a CEA system in Y-mode, and that shared register instruction testing is disabled because cluster 0 is in use. If this message is inconsistent with your hardware configuration, it normally indicates an instruction failure. To determine where the failure occurred, rerun **olcrit** with the **+shr** command option. Contact your CRI representative for additional assistance.

CRAY X-MP MODE
Indicates that the mainframe is a CRAY X-MP computer system.

CRAY X-MP MODE, shared register testing disabled
Indicates that the mainframe is a CRAY X-MP computer system, and that shared register instruction testing is disabled because cluster 0 is in use. If this message is inconsistent with your hardware configuration, it normally indicates an instruction failure. To determine where the failure occurred, rerun **olcrit** with the **+shr** command option. Contact your CRI representative for additional assistance.

CRAY X-MP MODE, compressed index testing disabled
Indicates that the mainframe is a CRAY X-MP computer system without compressed indexing hardware. If this message is inconsistent with your hardware configuration, it normally indicates an instruction failure. To determine where the failure occurred, rerun **olcrit** with the **+ci** command option. Contact your CRI representative for additional assistance.

CRAY X-MP MODE, extended memory testing disabled
Indicates that the mainframe is a CRAY X-MP computer system without extended memory instruction hardware. If this message is inconsistent with your hardware configuration, it normally indicates an instruction failure. To determine where the failure occurred, rerun **olcrit** with the **+ema** command option. Contact your CRI representative for additional assistance.

CRAY-1 MODE
Indicates that the mainframe is a CRAY-1 computer system.

CRAY-1 MODE, vector pop/parity testing disabled
Indicates that the mainframe is a CRAY-1 computer system without vector population count/parity instruction hardware. If this message is inconsistent with your hardware configuration, it normally indicates an instruction failure. To determine where the failure occurred, rerun **olcrit** with the **+pop** command option. Contact your CRI representative for additional assistance.

3.4.5.2 Informative messages

If the **+verbose** option is enabled, a message is sent to **stdout** (standard output device) after each pass through the test loop.

On an error, the test provides information such as the following:

- Pass and error counts
- Seed at the beginning of the pass on which the error occurred
- Contents of the instruction buffer
- Initial data
- Data results from the simulated instruction execution in the master CPU
- Differences between the simulated execution results from the master CPU and the actual execution results from all of the selected CPUs

In addition, the following informative messages may be displayed:

The *ijk* field is invalid; the instruction was not selected/deselected.
  The *ijk* field specified with the *gh* field for **enable** *ilist* or **disable** *ilist* is invalid. Correct and rerun.

The *ijk* field is not needed to select/deselect the instruction.
  The *ijk* field specified with the *gh* field for **enable** *ilist* or **disable** *ilist* is not required. However, the specified instruction was selected or deselected.

3.4.5.3 Error messages

One of the following error messages is sent to **stderr** (standard error device) if an invalid command option is entered:

olcrit: pattern: No data pattern(s) selected.
  All data patterns are deselected. Correct and rerun.

olcrit: selins: No executable instructions selected.
  All instructions are deselected. Correct and rerun.

olcrit: selins: Vector length must be in the range 0 through 100.
  Vector length is not in the range 0 through 100. Correct the **vl** option and rerun.

One of the following error messages is sent to **stderr** if **olcrit** detects an unexpected error. Select a different master CPU and rerun the test. If the problem persists, contact your CRI representative.

olcrit: simulate: (software error) The instruction does not have a sim*xxx* routine.

olcrit: generate: (software error) The instruction does not have a gen*xxx* routine.

olcrit: simulate: (software error) The *gh* field is greater than 177.

3.5. olcsvc

The **olcsvc** test provides comprehensive testing of the vector registers, functional units, and paths, and limited testing of the scalar registers, functional units, and paths. All address registers, address functional units, and related paths are assumed to be operating correctly.

The **olcsvc** test generates a random sequence of vector instructions, followed by a sequence of scalar instructions. The scalar and vector instructions perform identical functions. The two sets of instructions are executed with random data, and the results are compared. Any differences are reported, and the test attempts to isolate the error. If no differences are detected, the test generates new instructions and data, and repeats the process.

The **olcsvc** test runs under the confidence monitor program, **olcmon**. The **olcmon** monitor compares the scalar and vector execution results. For additional information on **olcmon**, refer to section 2, Confidence Test and Monitor Overview.

3.5.1 TEST SYNOPSIS

The **olcsvc** command options can be entered in any order. If an option is omitted, the program uses the default value. The test synopsis lists the **olcsvc** command options and arguments in the following order:

1. Monitor options
2. Test-specific options
3. Data pattern options
4. Instruction options

Synopsis:

olcsvc [**chkpnt** *mode*] [**cpu** *clist*] [**cputime** *h*:*m*:*s*] [**+/-getseed**]
[**getseed** *file*] [**help**] [**maxerr** *n*] [**maxp** *n*] [**+/-parcel**] [**time** *h*:*m*:*s*]
[**+/-verbose**] [**+xmp**] [**+cray1**]†

[**disable** *ilist*] [**enable** *ilist*] [**+/-isolate**] [**isop** *n*]
[**numpar** *n*] [**+/-repeat**] [**seed** *n*] [**+/-sgci**] [**vl** *n*]

[**+/-onezero**] [**+/-random**] [**+/-slide**]

[**+/-cm**] [**+/-fpadd**] [**+/-fpmult**] [**+/-fprecip**] [**+/-int**] [**+/-logical**]
[**+/-pop**] [**+/-shift**]

disable *ilist*
Deselects specific instructions. Enter *ilist* in the following format:

n,*n*,...,*n*

n is the octal value in the *gh* field of the specific vector instructions. Only vector instructions are valid; all other instructions are ignored. The **disable** *ilist* option overrides the **enable** *ilist* option and any selected (+) or deselected (-) instruction options.

enable *ilist*
Selects specific instructions. Enter *ilist* in the following format:

n,*n*,...,*n*

n is the octal value in the *gh* field of the specific vector instructions. Only vector instructions are valid; all other instructions are ignored. If you do not enter **enable** *ilist*, all vector instructions are run. The **enable** *ilist* option overrides any selected (+) or deselected (-) instruction options. When the test is run with default values for the +/- instruction options, and the **enable** *ilist* option is selected, only the instructions specified by the **enable** *ilist* option are run.

† The monitor command options are described in section 2, Confidence Test and Monitor Overview.

+/-isolate
Enables (**+isolate**) or disables (**-isolate**) the error isolation option. The default is **+isolate.**

isop *n* Sets the isolation pass limit to *n* (octal). During isolation, the diagnostic repeatedly executes the suspected failing sequence. If the sequence fails, the loop terminates and the diagnostic attempts to isolate the shortened sequence further. If the sequence does not fail, the loop terminates after *n* passes, and **olcsvc** assumes that the error is not in the tested sequence. The default for *n* is O'1000.

numpar *n* Sets the minimum number of parcels of vector instructions to be generated on each pass. The actual number of parcels generated can be greater than *n* on any given pass. *n* can be any octal value in the range 1 through O'200. The default for *n* is O'100.

+/-repeat
Enables (**+repeat**) or disables (**-repeat**) the option that repeats the first pass until the diagnostic terminates. **+repeat** is useful for recreating an error. It is normally used with one of the following options: **seed** *n*, **+getseed**, or **getseed** *file*. The default is **-repeat** (the program generates new instructions and data after each pass).

seed *n* Sets the random seed to *n*. *n* can be any 64-bit octal value. If *n* is 0, the test reads the real-time clock and uses the value for the initial seed. The default for *n* is O'33. If **seed** *n* is selected, do not select **+getseed** or **getseed** *file*.

+/-sgci Enables (**+sgci**) or disables (**-sgci**) testing of the scatter/gather/compressed index hardware. When enabled, testing occurs even if the hardware configuration detection routine indicates that the hardware is not present in the system. However, if this option is enabled and the hardware is not present in the system, you will receive a dump indicating that the hardware has failed. When allowed to default, the test determines the type of hardware configuration and sets the default value accordingly.

vl *n* Sets the vector length to *n*. *n* can be any octal value within the range 0 through O'100. The default for *n* is 0.

If **vl** is set to 0, a random **vl** value is used to initialize the test and the value may change during the execution of the random instruction buffer.

vl *n* (continued)

If the **vl** value is within the range 1 through O'100, instruction 00200*k* is disabled. The **vl** value is initialized to *n* and remains set to *n* during the execution of the random instruction buffer. However, if instruction 00200*k* is selected by the **enable** option, the **vl** value is initialized to *n* and may change each time a 00200*k* instruction is executed in the random instruction buffer.

+/-onezero, +/-random, +/-slide

Selects (+) or deselects (-) specific data patterns. Except when the **vl** value is initialized to a value within the range 1 through O'100, random data is used for the vector length register. The default is **+onezero +random +slide**. The data patterns are as follows:

Option	Data Pattern
onezero	Random selection of all 1's or all 0's in a word. For example: 1777777777777777777777 0000000000000000000000
random	Random bit generation in a word. For example: 1023122123232122777127 0003423100233344322177 1640034356453221213532 1123235467543221322120 1304322300332105534311
slide	Random number of consecutive 1's (0's) that slide in either direction through a field of 0's (1's). Consecutive words contain the sliding pattern. For example: 0777777777777777777777 0377777777777777777777 0177777777777777777777 1077777777777777777777 1437777777777777777777 . . . 1777777777777777777770 1777777777777777777774 1777777777777777777776 1777777777777777777777

+/-onezero, +/-random, +/-slide
(continued)

Option	Data Pattern
slide	(Example continued):

```
0000000000000000000001
0000000000000000000003
0000000000000000000007
0000000000000000000017
0000000000000000000036
          .
          .
          .
0740000000000000000000
1700000000000000000000
1600000000000000000000
1400000000000000000000
1000000000000000000000
0000000000000000000000
```

+/-cm, +/-fpadd, +/-fpmult, +/-fprecip, +/-int, +/-logical, +/-pop, +/-shift

Selects (+) or deselects (-) specific instruction groups for the following options:

Option	Instruction Type
cm	Central memory
fpadd	Floating-point addition
fpmult	Floating-point multiply
fprecip	Floating-point reciprocal
int	Integer
logical	Logical
pop	Population/parity count
shift	Shift

If allowed to default, all instruction groups are run. The groups are as follows:

Option	Instruction Group
cm	176, 177
fpadd	170 through 173
fpmult	160 through 167†
fprecip	174*ij*0
int	154 through 157
logical	003, 073, 140 through 147, 175
pop	174*ij*1, 174*ij*2
shift	150 through 153

† Instruction 166 is not generated on a CEA system.

3.5.2 TEST EXECUTION

The **olcsvc** execution sequence is as follows:

1. Test initialization and hardware configuration detection
2. Random instruction and data generation
3. Instruction buffer execution
4. Comparison of execution results
5. Error isolation

Hardware configuration detection occurs only at test initiation. Steps 2 through 4 occur on each pass through the test loop. Step 5 occurs only on error.

3.5.2.1 Test initialization and hardware configuration detection

At test initialization, instructions are processed in the following order:

1. All instructions are initially enabled unless either of the following occurs (in which case no instructions are initially enabled):

   - An instruction group is selected (+*option*)

   - An **enable** option is entered and there are no deselected (-*option*) instruction group entries

2. Selected groups are processed, enabling instructions in the selected groups.

3. Deselected groups are processed, disabling instructions in the deselected groups.

4. If the **vl** option is set to a value within the range 1 through O'100, instruction 00200*k* is deselected.

5. Individually selected instructions are processed (all instructions specified by the **enable** option).

6. Individually deselected instructions are processed (all instructions specified by the **disable** option).

7. If no instructions are selected, an error message is displayed and the test is terminated.

The hardware configuration detection routine determines which of the following computer systems is configured:

- CRAY X-MP computer system

- CRAY-1 computer system

Then the hardware configuration detection routine adjusts testing accordingly, by determining the following:

Mainframe	Hardware Configuration Detection Routine
CRAY X-MP	Determines whether the system contains scatter/gather/compressed indexing hardware
CRAY-1	Determines whether the system contains a vector population count functional unit

After determining the hardware characteristics, the routine writes a message to **stdout** to indicate the type of system detected.

Instruction generation is dependent on the hardware configuration detected, as follows (you can use the **+/-sgci** option to override this default instruction generation process):

Mainframe	Instructions Generated
CEA	All instructions are generated except instruction 166, which is the 32-bit vector integer multiply instruction
CRAY X-MP	All instructions are generated with one condition: scatter/gather/compressed indexing instructions are generated only if present in the hardware.
CRAY-1	All instructions are generated except the following:

- A load VL instruction (00200*k*)
- Scatter/gather/compressed indexing instructions
- Any instructions that would cause vector recursion. (In a vector instruction, *vector recursion* results when V*i* and V*j* or V*i* and V*k* refer to the same vector register).
- Vector pop/parity instructions are generated only if the hardware contains a vector population count functional unit.

3.5.2.2 Random instruction and data generation

These routines build the random vector instruction buffer. As each vector instruction is generated, the sequence of scalar instructions that simulates the vector instructions is generated in the scalar instruction buffer.

The following information applies to the sequence of scalar instructions that is generated for each vector instruction:

- S*a*, S*b*, S*c*, and S*d* are randomly selected S registers. A*m*, A*n*, A*p*, and A*q* are randomly selected A registers. The test uses unique A registers and S registers for each sequence, but not A0 or S0. The registers are not selected based on the *ijk* fields of the vector instruction. Therefore, the same vector instruction does not always generate the same sequence of scalar instructions. The registers used in the scalar sequence will vary.

- The labels v*i*reg, v*j*reg, v*k*reg, s*i*reg, s*j*reg, s*k*reg, and vmreg are central memory locations containing the simulated vector registers, scalar registers, and vector mask register, respectively. The actual address depends on the *i*, *j*, and *k* fields of the actual vector instruction.

- For vector instructions that require A registers to contain certain values (memory and shift instructions), constant loads of the A registers are generated immediately preceding the actual vector instruction in the vector instruction buffer.

- These sequences are altered for certain vector instructions if the *i*, *j*, and *k* fields of the vector instruction refer to the same vector register. For instructions 141, 143, 145, 155, 157, 161, 163, 165, 167, 171, and 173, if the *j* field is equal to the *k* field of the instruction, the read from v*k*reg in the scalar instruction sequence is not generated because it is the same as the read from v*j*reg; this results in faster execution of the scalar instruction sequence.

  The following applies only to CRAY-1 computer systems:

  - For instructions 141, 143, 145, 147 through 153, 155, 157, 161, 163, 165, 167, 171, and 173, the *i* field never equals the *j* field.

  - For instructions 140 through 147, and 154 through 174, the *i* field never equals the *k* field.

- The shift instructions normally produce a shift value in the range 0 through O'77 for a single shift and 0 through O'177 for a double shift, and only occasionally use a random value for the shift amount.

- For instructions 176*i*0*k* and 1770*jk* (read/write vector to central memory), the central memory address is a random address within the first O'400 words of cmbuff. The stride is a random value with its upper limit based on the random address and the current vector length. Therefore, a large stride can be used if the vector length is small.

- For instructions 176*i*1*k* and 1771*jk* (gather and scatter), the program sets up a vector register containing a specific range of values by forcing a sequence of instructions before instruction 176*i*1*k* or 1771*jk* is generated. The forced instructions consist of a load of an S register with a 9-bit mask from the right (042*i*67), followed by a 140 instruction (the logical product of a scalar register with a vector register to a vector register). The resulting vector register is then used as the V*k* register in a 176*i*1*k* or 1771*jk* instruction. This forces the values into the range 0 through O'777, and it reduces the randomness of the instruction sequence generated. The test tracks the vector registers that can be used for a gather/scatter instruction. If the V*k* register is within the range 0 through O'777 when a 176*i*1*k* or 1771*jk* instruction is generated, the set-up sequence is not generated.

  The following conditions indicate that a vector register is within the range 0 through O'777:

  - The register was set up for a previous gather/scatter instruction.
  - The register received the results from a 174*ij*1 or 174*ij*2 instruction (pop/parity).
  - The register received the results from a 140 instruction, and the V*k* field of the instruction was set up for scatter/gather.
  - The register received the results from a 141 instruction, and either the V*j* or V*k* field of the instruction was set up for scatter/gather.
  - The register received the results from a 143, 145, or 147 instruction, and the V*j* and V*k* fields of the instruction were set up for scatter/gather.
  - The register received the results from a 151 instruction (single shift right), and the shift value was greater than 55 (decimal).
  - The register received the results from a 153 instruction (double shift right), and the shift value was greater than 119 (decimal).

The scalar instruction sequence that is generated for each vector instruction follows.

Scalar instructions are not generated for vector instruction 00200*k*. However, during the vector instruction sequence, the VL value to be used in scalar instruction sequences is loaded into an A register and, subsequently, the VL register is loaded from the A register.

The scalar instruction sequence for vector instruction 0030*j*0 is as follows:

```
          Sb        sjreg,        ; Read Sj value
          vmreg,    Sb            ; Store Resulting VM
```

The scalar instruction sequence for vector instruction 073*i*00 is as follows:

```
          Sa        vmreg,        ; Read simulated VM reg.
          sireg,    Sa
```

The scalar instruction sequence for vector instruction 076 is as follows:

```
          Ap        element       ; Random element number
          Sa        vjreg,Ap      ; Read element from Vj
          sireg,    Sa            ; Store into Si
```

The scalar instruction sequence for vector instruction 077 is as follows:

```
          Ap        element       ; Random element number
          Sa        sjreg,        ; Read Sj
          vireg,Ap  Sa            ; Store into Vi
```

The scalar instruction sequence for vector instructions 140, 142, 144, 154, 156, 160, 162, 164, 166,† 170, and 172 is as follows:

```
          Am        vl            ; Current simulated VL
          An        0             ; Index
          Sb        sjreg,        ; Get S register value
 loop     Sc        vkreg,An      ; Get next vector element
          Sa        SbopSc        ; Perform operation
          vireg,An  Sa            ; Store result
          An        An+1          ; Update index
          A0        Am-An         ; Test for end
          jan       loop          ; Loop until index = VL
```

op can be one of the following:

&, !, , +, -, *f, *h, *r, *i, +f, -f

† Instruction 166 is not generated on a CEA system in Y-mode.

The scalar instruction sequence for vector instructions 141, 143, 145, 155, 157, 161, 163, 165, 167, 171, and 173 is as follows:

```
        Am         vl           ; Current simulated VL
        An         0            ; Index
loop    Sb         vjreg,An     ; Get next vector
        Sc         vkreg,An     ;     elements
        Sa         SbopSc       ; Perform operation
        vireg,An   Sa           ; Store result
        An         An+1         ; Update index
        A0         Am-An        ; Test for end
        jan        loop         ; Loop until index = VL
```

op can be one of the following:

&, !, , +, -, *f, *h, *r, *i, +f, -f

The scalar instruction sequence for vector instruction 146 is as follows:

```
        Am         vl           ; Current simulated VL
        An         0            ; Index
        Sd         vmreg,       ; Get simulated VM reg.
loop    S0         Sd           ; VM to S0 for testing
        jsp        skip1        ; Decide on result
        Sa         sjreg,       ; Read Sj register
        j          skip2        ; Skip vector read
skip1   Sa         vkreg,An     ; Read vector element
skip2   vireg,An   Sa           ; Write result element
        Sd         Sd<1         ; Shift VM value
        An         An+1         ; Update index
        A0         Am-An        ; Test for end
        jan        loop         ; Loop until index = VL
```

The scalar instruction sequence for vector instruction 147 is as follows:

```
        Am         vl           ; Current simulated VL
        An         0            ; Index
        Sd         vmreg,       ; Get simulated VM reg.
loop    S0         Sd           ; VM to S0 for testing
        jsp        skip1        ; Decide on result
        Sa         vjreg,An     ; Read Vj element
        j          skip2        ; Skip vector read
skip1   Sa         vkreg,An     ; Read Vk element
skip2   vireg,An   Sa           ; Write result element
        Sd         Sd<1         ; Shift VM value
        An         An+1         ; Update index
        A0         Am-An        ; Test for end
        jan        loop         ; Loop until index = VL
```

The scalar instruction sequence for vector instructions 150 and 151 is as follows:

```
        Ap       shift        ; Amount to shift
        Am       vl           ; Current simulated VL
        A5       0            ; Index
loop    Sa       vjreg,An     ; Get Vj element
        Sa       SaopAp       ; Do the shift
        vireg,An Sa           ; Store result
        An       An+1         ; Update index
        A0       Am-An        ; Test for end
        jan      loop         ; Loop until index = VL
```

op can be < (left shift) or > (right shift).

The scalar instruction sequence for vector instruction 152 is as follows:

```
        Ap       shift        ; Amount of shift
        Am       vl           ; Current simulated VL
        An       0            ; Index
        Sa       vjreg,An     ; Get element 0
loop    An       An+1         ; Update index
        A0       Am-An        ; Test for end
        Sb       0            ; 0 fill at end
        jaz      skip         ; Skip read at end
        Sb       vjreg,An     ; Get Vj element
skip    Sa       Sa,Sb<Ap     ; Do the shift
        vireg-1,An Sa         ; Store result
        Sa       Sb           ; Copy Sb to Sa
        jan      loop         ; Loop until index = VL
```

The scalar instruction sequence for vector instruction 153 is as follows:

```
        Ap       shift        ; Amount of shift
        Am       vl           ; Current simulated VL
        An       0            ; Index
        Sb       0            ; Zero fill the shift
loop    Sa       vjreg,An     ; Get Vj element
        Sd       Sa           ; Copy Sa into Sd
        Sa       Sb,Sa>Ap     ; Do the shift
        vireg,An Sa           ; Store the result
        Sb       Sd           ; Copy Sd into Sb
        An       An+1         ; Update index
        A0       Am-An        ; Test for end
        jan      loop         ; Loop until index = VL
```

The scalar instruction sequence for vector instruction 174*ij*0 is as follows:

```
            Am        vl            ; Current simulated VL
            An        0             ; Index
loop        Sb        vjreg,An      ; Get Vj element
            Sa        /hS1          ; Perform operation
            vireg,An  Sa            ; Store result
            An        An+1          ; Update index
            A0        Am-An         ; Test for end
            jan       loop          ; Loop until index = VL
```

The scalar instruction sequence for vector instructions 174*ij*1 and 174*ij*2 is as follows:

```
            Am        vl            ; Current simulated VL
            An        0             ; Index
loop        Sb        vjreg,An      ; Get Vj element
            Ap        opS1          ; Perform operation
            vireg,An  Ap            ; Store result
            An        An+1          ; Update index
            A0        Am-An         ; Test for end
            jan       loop          ; Loop until index = VL
```

op can be P or Q

The scalar instruction sequence for vector instructions 175*ij*0 through 175*ij*3 is as follows:

```
            Am        vl            ; Current simulated VL
            An        0             ; Index
            Sc        SB            ; Mask of current element
            Sa        0             ; Build VM in this register
loop        S0        vjreg,A5      ; Get next element
            jump      skip          ; Set VM bit?
            Sa        Sa!Sc         ; Yes - Set bit in VM
skip        Sc        Sc>1          ; Shift for next element
            An        An+1          ; Update index
            A0        Am-An         ; Test for end
            jan       loop          ; Loop until index = VL
            vmreg,    Sa            ; Store resulting VM
```

The *jump* value is determined by the vector instruction, as follows:

Vector Instruction	Jump Value
175*ij*0	jsn
175*ij*1	jsz
175*ij*2	jsm
175*ij*3	jsp

The scalar instruction sequence for vector instructions 175*ij*4 through 175*ij*7 is as follows:

```
         Am       vl          ; Current simulated VL
         An       0           ; Index
         Sc       SB          ; Mask of current element
         Sa       0           ; Build VM in this register
         Ap       0           ; Compressed index pointer
loop     S0       vjreg,An    ; Get next element
         jump     skip        ; Set VM bit?
         Sa       Sa!Sc       ; Yes - set bit in VM
         vireg,Ap An          ; Store index in Vi
         Ap       Ap+1        ; Update compressed index
skip     Sc       Sc>1        ; Shift for next element
         An       An+1        ; Update index
         A0       Am-An       ; Test for end
         jan      loop        ; Loop until index = VL
         vmreg,   Sa          ; Store resulting VM
```

The *jump* value is determined by the vector instruction, as follows:

Vector Instruction	Jump Value
175*ij*4	jsn
175*ij*5	jsz
175*ij*6	jsm
175*ij*7	jsp

The scalar instruction sequence for vector instruction 176*i*0*k* is as follows:

```
         Ap       cmaddress   ; CM address in cmbuff
         Aq       stride      ; Random stride value
         Am       vl          ; Current simulated VL
         An       0           ; Index
loop     Sa       ,Ap         ; Read from cmbuff
         vireg,An Sa          ; Store element of vector
         Ap       Ap+Aq       ; Increment address by stride
         An       An+1        ; Update index
         A0       Am-An       ; Test for end
         jan      loop        ; Loop until index = VL
```

The scalar instruction sequence for vector instruction 176*i*1*k* is as follows:

```
        Ap        cmbuff        ; Address of cmbuff
        Am        vl            ; Current simulated VL
        An        0             ; Index
loop    Aq        vkreg,An      ; Get element of vector
        Aq        Aq+Ap         ; Calculate address
        Sa        ,Aq           ; Get word from memory
        vireg,An  Sa            ; Store vector element
        An        An+1          ; Update index
        A0        Am-An         ; Test for end
        jan       loop          ; Loop until index = VL
```

The scalar instruction sequence for vector instruction 177*ij*0 is as follows:

```
        Ap        cmaddress     ; CM address in cmbuff
        Aq        stride        ; Random stride value
        Am        vl            ; Current simulated VL
        An        0             ; Index
loop    Sb        vjreg,An      ; Get element of vector
        ,Ap       Sb            ; Write to cmbuff
        Ap        Ap+Aq         ; Increment address by stride
        An        An+1          ; Update index
        A0        Am-An         ; Test for end
        jan       loop          ; Loop until index = VL
```

The scalar instruction sequence for vector instruction 177*ij*1 is as follows:

```
        Ap        cmbuff        ; Address of cmbuff
        Am        vl            ; Current simulated VL
        An        0             ; Index
loop    Aq        vkreg,An      ; Get element of vector
        Aq        Aq+Ap         ; Calculate address
        Sb        vjreg,An      ; Get vector element
        ,Aq       Sb            ; Write word to memory
        An        An+1          ; Update index
        A0        Am-An         ; Test for end
        jan       loop          ; Loop until index = VL
```

3.5.2.3 Instruction buffer execution

After the instructions and data are generated, the scalar and vector instruction buffers are executed first in the master CPU, and then in each of the other selected CPUs. Immediately following the execution of an instruction buffer, the **save** monitor routine is called to save the execution results.

3.5.2.4 Comparison of execution results

After the scalar and vector instruction buffers are executed in all of the selected CPUs, the **compare** monitor routine compares the results, and one of the following actions occurs:

- If the results match, the test proceeds with the next pass.
- If the results do not match, the test dumps all of the data related to the suspected failure and, if the isolation option is enabled (**+isolate**), attempts to isolate the failure by reducing the number of instructions in the execution buffers in which the failure is occurring. Refer to the test output to determine which CPU has failed.

3.5.2.5 Error isolation

If an error is detected and the isolation option is enabled (**+isolate**), the test attempts to reduce the random vector instruction buffer to the minimum number of failing instructions. If an instruction sequence is removed from the vector instruction buffer, the corresponding scalar instruction sequence is removed from the scalar instruction buffer. If a vector instruction requires that a set of registers be used together to perform a specific function, such as the address registers for memory references, the set of instructions is considered to be a single instruction sequence.

The isolation process consists of two parts. During the first part, the vector instruction buffer is shortened from the end, one instruction sequence at a time. The isolation routine initially tests the number of instruction sequences generated minus one. The routine executes until the specified number of passes is reached (**isop** *n*) or an error is detected. If an error is detected, the number of instruction sequences tested is decremented by one, and testing continues for **isop** *n* passes. This process continues until no errors are detected or until there are no remaining instructions to be tested.

If there are no remaining instructions to be tested and the test detects an error resulting from loading and unloading the registers, the test generates an output dump and the isolation process terminates.

During the second part of the isolation process, the last instruction sequence removed is tested by itself for **isop** *n* passes. If no error is detected, the preceding instruction sequence is loaded into the random vector instruction buffer and tested for **isop** *n* passes. Until the program detects an error or reaches the beginning of the instruction buffer, one more preceding instruction is added to the test sequence on each iteration of the isolation process.

When the isolation process terminates, the output dump contains the following:

- Isolated vector and scalar instruction buffers
- Data used when the failure occurred
- Scalar execution results from the master CPU
- Vector execution differences from the master CPU
- Scalar and vector execution differences from other CPUs

If the failure occurs intermittently, the second part of the isolation process may terminate without detecting an error, and execution difference results do not appear in the output dump. In this case, increase the value of **isop** *n*, enable the **+repeat** option, select the failing CPU, and use the failing seed to rerun the test.

All of the selected CPUs execute the scalar and vector instruction buffers. Therefore, if the program reports an error resulting from a failure in either the scalar or vector execution, the differences results should indicate where the failure occurred. For example, if the scalar and vector results indicate differences in all of the selected CPUs, the scalar instruction buffer in the master CPU is suspect. In this case, use the failing seed to rerun **olcsvc** in a different master CPU.

3.5.3 TEST TERMINATION

For information on test termination, refer to section 2, Confidence Test and Monitor Overview.

3.5.4 TEST EXAMPLES

This subsection contains **olcsvc** execution examples.

The following example runs **olcsvc** for O'10000000 passes in CPU b. Output is redirected to **olcsvc.log**. The **nohup**(1) command allows the program to continue executing after you log off the system. You can later log on to check the test's progress. The ampersand (&) causes the entire command to execute in the background, so that another prompt is immediately displayed and you can continue to use the system.

```
nohup olcsvc maxp 10000000 cpu b >olcsvc.log &
```

The following example shows a procedure for determining how frequently an error occurs. The test is rerun with the **+repeat** option, so that the first pass is run repeatedly until the test terminates. The test uses the seed value from the output sent to **fail.log** at the time of the initial error. Error isolation is disabled. The output is filtered to **olcsvc.log**.

```
olcsvc +repeat -isolate maxerr 100 maxp 100 cpu d getseed fail.log |
  tail >olcsvc.log &
```

The following example runs **olcsvc** with floating-point multiply and central memory instructions, and instructions 140 through 143. The test uses a constant vector length of O'100.

```
olcsvc +fpmult +cm enable 140,141,142,143 vl 100 >olcsvc.log &
```

The following example runs **olcsvc** with all of the vector logical instructions except instructions 146 and 147.

```
olcsvc +logical disable 146,147 >olcsvc.log &
```

The following example runs **olcsvc** with all of the instructions except floating-point multiply.

```
olcsvc -fpmult >olcsvc.log &
```

The following example shows the output displayed when **olcsvc** is run with all default values.

```
olcsvc
```

Output:

```
olcsvc
olcsvc started in cpu A on Tue Aug 25 13:42:07 1987
CRAY X-MP MODE
olcsvc reached maximum pass limit with 1000 passes and 0 errors
on Tue Aug 25 13:42:15 1987
```

The following example runs **olcsvc** with the **+verbose** option enabled so that a line of output is generated after each pass.

```
olcsvc +verbose
```

Output:

```
olcsvc +verbose
olcsvc started in cpu A on Tue Aug 25 11:42:47 1987
CRAY X-MP MODE
olcsvc: pass = 1, error = 0  Tue Aug 25 11:42:47 1987
olcsvc: pass = 2, error = 0  Tue Aug 25 11:42:47 1987
olcsvc: pass = 3, error = 0  Tue Aug 25 11:42:47 1987
               .
               .
               .
olcsvc: pass = 1000, error = 0  Tue Aug 25 11:42:55 1987
olcsvc reached maximum pass limit with 1000 passes and 0 errors
on Tue Aug 25 11:42:55 1987
```

The following example runs **olcsvc** for 10 seconds (CPU time) in CPU c only.

```
    olcsvc cpu c cputime 10
```

Output:

```
olcsvc cpu c cputime 10
olcsvc started in cpu C on Tue Aug 25 11:44:51 1987
CRAY X-MP MODE
olcsvc reached maximum cputime limit with 1510 passes and 0 errors
on Tue Aug 25 11:45:06 1987
```

The following example runs **olcsvc** in CPUs a and c, with a as the master. On each pass, the test generates 20 parcels of vector instructions.

```
    olcsvc cpu a,c numpar 20
```

Output on an error:

```
olcsvc cpu a,c numpar 20
olcsvc started in cpus A, C with master cpu A  on Mon Feb 9 17:19:19 1987
CRAY X-MP MODE
olcsvc:  restart file written to A11524-olcsvc
name            <       11760> =  'olcsvc  '
rev             <       11761> =  '4.0     '
date            <       11762> =  '02/09/87'
pass            <       11763> =  4
error           <       11764> =  1
seed            <       11765> =  37507312636362015466
vl              <       11770> =  0
numpar          <       12016> =  20
isop            <       14527> =  1000
failpat         <       12475> =  'slide '
```

Output (continued):

random vector instruction buffer

```
                                          vbuff
15456a     175073              VM         V7,M
15456b     160464              V4         S6*FV4
15456c     143005              V0         V0!V5
15456d     153060              V0         V6,V6>A0
15457a     020600 000072       A6         00000072
15457c     150156              V1         V5<A6
15457d     163334              V3         V3*HV4
15460a     162334              V3         S3*HV4
15460b     147015              V0         V1!V5&VM
15460c     163607              V6         V0*HV7
15460d     165604              V6         V0*RV4
15461a     141752              V7         V5&V2
15461b     162716              V7         S1*HV6
15461c     141227              V2         V2&V7
15461d     172347              V3         S4-FV7
15462a     006000 057120       J          13624a
```

scalar instruction buffer

```
                                          sbuff
15521a     020600 000002       A6         00000002
15521c     022200              A2         00
15521d     051200              S2         S0!S0
15522a     043600              S6         >00
15522b     122000 024457       S0         24457,A2
15522d     016000 066516       JSP        15523c
15523b     051662              S6         S6!S2
15523c     055277              S2         S2>100-77
15523d     030220              A2         A2+A0
15524a     031062              A0         A6-A2
15524b     011000 066511       JAN        15522b
15524d     130600 023546       23546,0    S6
15525b     020600 000002       A6         00000002
15525d     022500              A5         00
15526a     120200 023555       S2         23555,0
15526c     125300 024157       S3         24157,A5
15527a     064423              S4         S2*FS3
15527b     135400 024157       24157,A5   S4
15527d     030550              A5         A5+A0
15530a     031065              A0         A6-A5
15530b     011000 066532       JAN        15526c
```

(*scalar instructions simulating all of the vector instructions are displayed*)

initial vector length and mask register data

```
initvl        <       21533> =  0000000000000000000002
initvm        <       21534> =  1600000000000000000000
```

Output (continued):

```
initial scalar register data
inits0          <       21535> =  1700000000000000000000
inits1          <       21536> =  1740000000000000000000
inits2          <       21537> =  1760000000000000000000
inits3          <       21540> =  1770000000000000000000
inits4          <       21541> =  1774000000000000000000
inits5          <       21542> =  1776000000000000000000
inits6          <       21543> =  1777000000000000000000
inits7          <       21544> =  1777400000000000000000
```

initial vector register data
(*vector register data is displayed*)

initial Central Memory data
(*central memory data is displayed*)

```
scalar instruction buffer execution results

The expected data shown below has the following format:

  name       + index     <offset>  = data ...

name:     The name of the data dumped on this line.
index:    The index into the data starting at name.  Optional, default: 0.
offset:   The offset into the data buffer.
data:     The actual data dumped.

*** Expected Results ***   cpu A (master)

Source data buffer at 16300 in Memory copied to save buffer at 73613 in Memory
   Memory address in source data buffer = <offset> + 16300 (source data buffer)
   Memory address in save data buffer   = <offset> + 73613 (save data buffer)

Scalar Buffer Execution Results

scalar buffer execution: vector length and mask register data results
vlreg          <    2010> =  0000000000000000000002
vmreg          <    2011> =  0000000000000000000000

scalar buffer execution: scalar register data results
s0reg          <    2000> =  1700000000000000000000
s1reg          <    2001> =  1740000000000000000000
s2reg          <    2002> =  1760000000000000000000
s3reg          <    2003> =  1770000000000000000000
s4reg          <    2004> =  1774000000000000000000
s5reg          <    2005> =  1776000000000000000000
s6reg          <    2006> =  1777000000000000000000
s7reg          <    2007> =  1777400000000000000000
```

Output (continued):

```
scalar buffer execution: vector register data results
```
(*vector register data is displayed*)

```
scalar buffer execution: central memory data results
```
(*central memory data is displayed*)

```
The following data shows the differences between executing
the scalar buffer in the master CPU and executing the
vector buffer and scalar buffer in any remaining CPUs.

vlreg       = vector length register results
vmreg       = vector mask register results
s0reg-s7reg = scalar register data results
v0reg-v7reg = vector register data results
cmbuff      = central memory data results

The difference data shown below has the following format:

  name       + index     <offset>  = data ...
                                     data differences ....

name:    The name of the data dumped on this line.
index:   The index into the data starting at name.  Optional, default: 0.
offset:  The offset into the data buffer.
data:    The actual data dumped.
         The differences are marked with an asterisk (*) preceding the data word.
data differences:  The bits that differ between the actual results and
                   the expected results.

*** Differences ***    cpu A (master)

Source data buffer at 16300 in Memory copied to save buffer at 75626 in Memory
   Memory address in source data buffer = <offset> + 16300 (source data buffer)
   Memory address in save data buffer   = <offset> + 75626 (save data buffer)

Vector Buffer Execution Results



*** Differences ***    cpu C

Source data buffer at 16300 in Memory copied to save buffer at 77641 in Memory
   Memory address in source data buffer = <offset> + 16300 (source data buffer)
   Memory address in save data buffer   = <offset> + 77641 (save data buffer)

Scalar Buffer Execution Results
```

Output (continued):

```
*** Differences ***    cpu A (master)

Source data buffer at 16300 in Memory copied to save buffer at 101654 in Memory
   Memory address in source data buffer = <offset> + 16300 (source data buffer)
   Memory address in save data buffer   = <offset> + 101654 (save data buffer)

Vector Buffer Execution Results
v0reg          <       23557> = *1773777777777777777000   ...
                                 0004000000000000000000   ...

Beginning error isolation
Error isolation complete

name           <       11760> =  'olcsvc  '
rev            <       11761> =  '4.0     '
date           <       11762> =  '02/09/87'
pass           <       11763> =  4
error          <       11764> =  1
seed           <       11765> =  3750731263636201546 6
vl             <       11770> =  0
numpar         <       12016> =  20
isop           <       14527> =  1000
failpat        <       12475> =  'slide   '

isolated random vector instruction buffer

                                                vbuff
15460a      162334              V3          S3*HV4
15460b      147015              V0          V1!V5&VM
15460c      006000 057120       J           13624a
```

(From this point on, the dump is similar to the previously listed portion of the dump that displayed the unisolated error information.)

```
The first address (FADD) of the diagnostic is 11760a
olcsvc reached maximum error limit with 4 passes and 1 errors
on Mon Feb 9 17:23:52 1987
```

3.5.5 TEST MESSAGES

The **olcsvc** test produces the following types of messages:

- Test mode
- Informative

These messages are listed in the subsections that follow.

3.5.5.1 Test mode messages

During test execution, one of the following messages is displayed to indicate the test mode:

CRAY Y-MP MODE
: Indicates that the mainframe is a CEA system.

CRAY X-MP MODE
: Indicates that the mainframe is a CRAY X-MP computer system.

CRAY X-MP MODE: scatter/gather/compressed index testing disabled
: Indicates that the mainframe is a CRAY X-MP computer system without scatter/gather/compressed indexing hardware. If this message is inconsistent with your hardware configuration, it normally indicates an instruction failure. To determine where the failure occurred, rerun **olcsvc** with the **+sgci** command option. Contact your CRI representative for additional assistance.

CRAY-1 MODE
: Indicates that the mainframe is a CRAY-1 computer system.

CRAY-1 MODE: vector pop/parity testing disabled
: Indicates that the mainframe is a CRAY-1 computer system without vector population count/parity hardware. If this message is inconsistent with your hardware configuration, it normally indicates an instruction failure. To determine where the failure occurred, rerun **olcsvc** with the **+pop** command option. Contact your CRI representative for additional assistance.

3.5.5.2 Informative messages

If the **+verbose** option is enabled, a message is sent to **stdout** (standard output device) after each pass through the test loop.

On an error, the test provides information such as the following:

- Pass and error counts
- Seed at the beginning of the pass on which the error occurred
- Contents of the vector instruction buffer
- Contents of the scalar instruction buffer
- Initial data
- Data results from the scalar instruction execution in the master CPU
- Differences in the scalar execution results from the master CPU, the scalar execution results from the remaining selected CPUs, and the vector execution results from all of the selected CPUs

3.6 olibuf

The **olibuf** test is an on-line instruction buffer test. To detect data-sensitive failures, the program generates test buffers and runs data patterns through the instruction buffer. To detect branching failures, the program generates test buffers containing in-stack and out-of-stack jumps, compares expected jump addresses to actual jump addresses, and reports any differences. The test continues until the maximum pass, error, or time limit is reached.

3.6.1 TEST SYNOPSIS

The **olibuf** command options can be entered in any order. If an option is omitted, the program uses the default value. The test synopsis lists the **olibuf** command options and arguments in the following order:

1. Monitor options
2. Test-specific options
3. Data pattern options

Synopsis:

olibuf [**chkpnt** *mode*] [**cpu** *clist*] [**cputime** *h*:*m*:*s*] [**+/-getseed**]

[**getseed** *file*] [**help**] [**maxerr** *n*] [**maxp** *n*] [**+/-parcel**] [**time** *h*:*m*:*s*]

[**+/-verbose**] [**+xmp**] [**+cray1**]†

[**+/-repeat**] [**seed** *n*] [**section** *slist*]

[**+/-onezero**] [**+/-random**] [**+/-solid**]

+/-repeat
Enables (**+repeat**) or disables (**-repeat**) the option that repeats the first pass until the diagnostic terminates. **+repeat** is useful for recreating an error. It is normally used with one of the following options: **seed** *n*, **+getseed**, or **getseed** *file*. The default is **-repeat** (the program generates new instructions and data after each pass).

---

† The monitor command options are described in section 2, Confidence Test and Monitor Overview.

seed *n* Sets the random seed to *n*. *n* can be any 64-bit octal value. If *n* is 0, the test reads the real-time clock and uses the value for the initial seed. The default for *n* is 0'33. If **seed** *n* is selected, do not select **+getseed** or **getseed** *file*.

section *slist*

Selects the test sections to be executed. *slist* is entered in the following format:

n,n,...,n

n can be one of the following test sections (if allowed to default, all test sections are executed):

Section	Description
1	Executes a 16-bit pattern through parcel 0 of all words in the instruction buffer
2	Executes a 16-bit pattern through parcel 1 of all words in the instruction buffer
3	Executes a 16-bit pattern through parcel 2 of all words in the instruction buffer
4	Executes a 16-bit pattern through parcel 3 of all words in the instruction buffer
5	Executes random in-stack and out-of-stack jumps in the instruction buffer

+/-onezero, +/-random, +/-solid

Selects (+) or deselects (-) specific data patterns. If allowed to default, all of the data patterns are run. The data patterns are as follows:

Option	Data Pattern
onezero	On each pass, random patterns of all 1's or all 0's are run through the test area. For example: 177777 000000

+/-onezero, +/-random, +/-solid
(continued)

Option	Data Pattern
random	On each pass, random bit patterns are run through the test area. For example:

```
102314
000347
164002
112323
130431
```

solid	On each pass, a random pattern of either all 1's or all 0's is run through the test area with one complement pattern. The location of the complement pattern is randomly selected. For example:

Pass 1

```
177777
177777
  .
  .
  .
000000 (complement)
  .
  .
  .
177777
177777
```

Pass 2

```
000000
  .
  .
  .
177777 (complement)
  .
  .
  .
000000
```

+/-onezero, +/-random, +/-solid
(continued)

Option	Data Pattern
solid	(continued):

Pass 3

000000
.
.
.
177777 (complement)
000000

Pass 4

177777 (complement)
000000
.
.
.
177777

3.6.2 TEST EXECUTION

The **olibuf** execution sequence is as follows:

1. Test initialization
2. Test buffer generation
3. Test buffer execution
4. Comparison of expected and actual data
5. Error report

Steps 2 through 4 occur on each pass through the test loop. Step 5 occurs only on error.

3.6.2.1 Test initialization

At test initialization, the selected sections and patterns are processed in the following order:

1. All sections and patterns are initially enabled.

2. Selected sections are processed.

3. Deselected patterns are processed. If all patterns are deselected, an error message is displayed and the test is terminated.

3.6.2.2 CRAY X-MP computer system test buffer generation

The generation routine builds and generates the test buffers. A test buffer is generated for each section selected. Test sections 1 through 4 use the following instructions to execute a pattern through the instruction buffer:

001000	PASS		Pass
020*ijkm*	A*i*	*exp*	Transmit *exp*=*jkm* to A*i*
11*hi*000	,A*h*	A*i*	Store (A*i*) to (A*h*)
030*ijk*	A*i*	A*j*+A*k*	Integer sum of (A*j*) and (A*k*) to A*i*
0050*jk*	J	B*jk*	Jump to (B*jk*)

Test section 5 uses the following instructions to execute random in-stack and out-of-stack jumps in the instruction buffer:

020*ijkm*	A*i*	*exp*	Transmit *exp*=*jkm* to A*i*
11*hi*000	,A*h*	A*i*	Store (A*i*) to (A*h*)
030*ijk*	A*i*	A*j*+A*k*	Integer sum of (A*j*) and (A*k*) to A*i*
006*ijkm*	J	*exp*	Jump to *exp*
0050*jk*	J	B*jk*	Jump to (B*jk*)

The following example shows a sample test buffer for section 1. The parcel 0 instructions and data patterns are used to test first the odd and then the even words. When the test buffer is executed, each data pattern (*nnnnnn*) is loaded into parcel 0 of each instruction buffer word.

Example:

Address	Opcode	CAL Mnemonics		Instruction Buffer Word
5340a	001000	PASS		
5340b	001000	PASS		
5340c	001000	PASS		
5340d	020100 *nnnnnn*	A1	00*nnnnnn*	001
5341b	112100 000000	0,A2	A1	
5341d	030223	A2	A2+A3	
5342a	001000	PASS		
5342b	001000	PASS		
5342c	001000	PASS		
5342d	020100 *nnnnnn*	A1	00*nnnnnn*	003
5343b	112100 000000	0,A2	A1	
5343d	030223	A2	A2+A3	
5344a	001000	PASS		
5344b	001000	PASS		
5344c	001000	PASS		
.				
.				
.				
5536d	020100 *nnnnnn*	A1	00*nnnnnn*	177

Example (continued):

Address	Opcode		CAL Mnemonics		Instruction Buffer Word
5537b	112100	000000	0,A2	A1	
5537d	030223		A2	A2+A3	
5540a	001000		PASS		
5540b	001000		PASS		
5540c	001000		PASS		
5540d	001000		PASS		
5541a	001000		PASS		
5541b	001000		PASS		
5541c	001000		PASS		
5541d	020100	*nnnnnn*	A1	00*nnnnnn*	002
5542b	112100	000000	0,A2	A1	
5542d	030223		A2	A2+A3	
5543a	001000		PASS		
5543b	001000		PASS		
5543c	001000		PASS		
.					
.					
.					
5735d	020100	*nnnnnn*	A1	00*nnnnnn*	176
5736b	112100	000000	0,A2	A1	
5736d	030223		A2	A2+A3	
5737a	001000		PASS		
5737b	001000		PASS		
5737c	001000		PASS		
5737d	020100	*nnnnnn*	A1	00*nnnnnn*	000
5740b	112100	000000	0,A2	A1	
5740d	030223		A2	A2+A3	
5741a	005000		J	B00	

The following example shows a sample test buffer for section 5.

Example:

Absolute Address	CAL Mnemonics		Jump Address
testbuff :	ERR	000	
testbuff+02:	A1	0000000001	
testbuff+06:	0,A2	A1	
testbuff+12:	A2	A2+A3	
testbuff+14:	J	00000026660	testbuff+214a
testbuff+20:	ERR	000	
testbuff+22:	ERR	000	
testbuff+24:	ERR	000	
testbuff+26:	ERR	000	
testbuff+30:	ERR	000	
testbuff+32:	ERR	000	

Example (continued):

Absolute Address	CAL Mnemonics		Jump Address
testbuff+34:	ERR	000	
testbuff+36:	ERR	000	
testbuff+40:	A1	0000000020	
testbuff+44:	0,A2	A1	
testbuff+50:	A2	A2+A3	
testbuff+52:	J	00000026201	testbuff+100b
testbuff+56:	ERR	000	
testbuff+60:	ERR	000	
testbuff+62:	ERR	000	
testbuff+64:	ERR	000	
testbuff+66:	A1	0000000033	
testbuff+72:	0,A2	A1	
testbuff+76:	A2	A2+A3	
testbuff+100:	J	00000026507	testbuff+161d
.			
.			
.			
testbuff+2340:	ERR	000	
testbuff+2342:	ERR	000	
testbuff+2344:	A1	0000001162	
testbuff+2350:	0,A2	A1	
testbuff+2354:	A2	A2+A3	
testbuff+2356:	J	B00	Return jump
testbuff+2360:	ERR	000	
.			
.			
.			
testbuff+2370:	ERR	000	
testbuff+2372:	ERR	000	
testbuff+2374:	A1	0000001176	
testbuff+2400:	0,A2	A1	
testbuff+2404:	A2	A2+A3	
testbuff+2406:	J	00000026634	testbuff+207a
testbuff+2412:	ERR	000	
testbuff+2414:	ERR	000	
testbuff+2416:	ERR	000	

3.6.2.3 CRAY Y-MP computer system test buffer generation

The generation routine builds and generates the test buffers. A test buffer is generated for each section selected. Test sections 1 through 4 use the following instructions to execute a pattern through the instruction buffer:

0010000	PASS		Pass
020*i*00*mn*	A*i*	*exp*	Transmit *nm* to A*i*
11*hi*00 00	,A*h*	A*i*	Store (A*i*) to (A*h*)
030*ijk*	A*i*	A*j*+A*k*	Integer sum of (A*j*) and (A*k*) to A*i*
0050*jk*	J	B*jk*	Jump to (B*jk*)

Test section 5 uses the following instructions to execute random in-stack and out-of-stack jumps in the instruction buffer:

0010000	PASS		Pass
020*i*00*mn*	A*i*	*exp*	Transmit *nm* to A*i*
11*hi*00 00	,A*h*	A*i*	Store (A*i*) to (A*h*)
030*ijk*	A*i*	A*j*+A*k*	Integer sum of (A*j*) and (A*k*) to A*i*
006*ijkm*	J	*exp*	Jump to *exp*
0050*jk*	J	B*jk*	Jump to (B*jk*)

The following example shows a sample test buffer for section 1. The parcel 0 instructions and data patterns are used to test first the odd and then the even words. When the test buffer is executed, each data pattern (*nnnnnn*) is loaded into parcel 0 of each instruction buffer word.

Example:

Address	Opcode	CAL Mnemonics		Instruction Buffer Word
15740a	001000	PASS		
15740b	001000	PASS		
15740c	001000	PASS		
15740d	020100 nnnnnn 000000	A1	00000nnnnnn	001
15741c	112100 000000 000000	0,A2	A1	
15742b	030223	A2	A2+A3	
15742c	001000	PASS		
15742d	020100 nnnnnn 000000	A1	00000nnnnnn	003
15743c	112100 000000 000000	0,A2	A1	
15744b	030223	A2	A2+A3	
15744c	001000	PASS		
15744d	020100 nnnnnn 000000	A1	00000nnnnnn	005

Example (continued):

Address	Opcode	CAL Mnemonics		Instruction Buffer Word
15745c	112100 000000 000000	0,A2	A1	
15746b	030223	A2	A2+A3	
.				
.				
.				
16136d	020100 nnnnnn 000000	A1	00000nnnnnn	177
16137c	112100 000000 000000	0,A2	A1	
16140b	030223	A2	A2+A3	
16140C	001000	PASS		
16140d	001000	PASS		
16141a	001000	PASS		
16141b	001000	PASS		
16141c	001000	PASS		
16141d	020100 nnnnnn 000000	A1	00000nnnnnn	002
16142c	112100 000000 000000	0,A2	A1	
16143b	030223	A2	A2+A3	
16143c	001000	PASS		
16143d	020100 nnnnnn 000000	A1	00000nnnnnn	004
16144c	112100 000000 000000	0,A2	A1	
16145b	030223	A2	A2+A3	
.				
.				
.				
16335d	020100 nnnnnn 000000	A1	00000nnnnnn	176
16336c	112100 000000 000000	0,A2	A1	
16337b	030223	A2	A2+A3	
16337c	001000	PASS		
16337d	020100 nnnnnn 000000	A1	00000nnnnnn	000
16340c	112100 000000 000000	0,A2	A1	
16341b	030223	A2	A2+A3	
16341c	005000	J	B00	

The following example shows a sample test buffer for section 5.

Example:

```
Absolute Address      CAL Mnemonics

15740a:               A1       00000000000
15740d:               0,A2     A1
15741c:               A2       A2+A3
15741d:               J        0016061b
15742b:               ERR
15742c:               ERR
15742d:               ERR
15743a:               ERR
15743b:               ERR
15743c:               ERR
15743d:               ERR
15744a:               A1       00000000020
15744d:               0,A2     A1
15745c:               A2       A2+A3
15745d:               J        0016040b
15746b:               ERR
15746c:               ERR
15746d:               A1       00000000033
15747c:               0,A2     A1
15750b:               A2       A2+A3
15750c:               J        0016152d
15751a:               ERR
15751b:               ERR
15751c:               ERR
15751d:               ERR
15752a:               ERR
15752b:               ERR
15752c:               ERR
15752d:               ERR
15753a:               ERR
15753b:               ERR
15753c:               ERR
15753d:               ERR
15754a:               ERR
15754b:               ERR
15754c:               ERR
15754d:               ERR
15755a:               ERR
15755b:               ERR
15755c:               A1       00000000066
15756b:               0,A2     A1
15757a:               A2       A2+A3
15757b:               J        0016012c
      .
      .
      .
```

Example (continued):

Absolute Address	CAL Mnemonics		
16034b:	ERR		
16034c:	ERR		
16034d:	ERR		
16035a:	ERR		
16035b:	A1	00000000365	
16036a:	0,A2	A1	
16036d:	A2	A2+A3	
16037a:	J	B00	(Return Jump)
16037b:	ERR		
16037c:	ERR		
16037d:	ERR		
16040a:	ERR		
16040b:	A1	00000000401	
16041a:	0,A2	A1	
16041d:	A2	A2+A3	
16042a:	J	0015746d	
16042c:	ERR		
16042d:	ERR		
.			
.			
.			
16166c:	ERR		
16166d:	A1	00000001133	
16167c:	0,A2	A1	
16170b:	A2	A2+A3	
16170c:	J	0015744a	
16171a:	ERR		
16171b:	ERR		
16171c:	ERR		
16171d:	ERR		
16172a:	ERR		
16172b:	ERR		
16172c:	ERR		
16172d:	ERR		
16173a:	ERR		
16173b:	ERR		
16173c:	ERR		
16173d:	ERR		
16174a:	ERR		
16174b:	ERR		
16174c	A1	00000001162	
16175b:	0,A2	A1	
16176a:	A2	A2+A3	
16176b:	J	0015775c	
16176d:	ERR		
16177a:	ERR		

3.6.2.4 Test buffer execution

After the test buffers are generated, the execution routine jumps to the buffer and executes the test buffer code in all of the selected CPUs. The **save** monitor routine saves the results. If a jump fails and an error exit occurs (section 5 only), no results are saved.

3.6.2.5 Comparison of expected and actual data

After the instructions are executed in all of the selected CPUs, the **compare** monitor routine compares the results. The actual results are compared to the expected results. If the results match, the test continues.

After all of the selected sections and data patterns are run, the pass count is incremented. If the results do not match, the test dumps all of the data related to the suspected failure.

3.6.2.6 Error report

If an error is detected, the test dumps all of the data related to the suspected failure. The output dump contains the following:

- Diagnostic Information Block
- Test buffer data at the time of the failure
- Expected results
- Differences

3.6.3 ERROR ISOLATION TO THE FAILING BIT

An error report is generated for each section in which an error occurs. By examining a dump for any one of the test sections 1 through 4, you can isolate the error to the failing bit.

3.6.3.1 <u>CX/1 system error isolation</u>

Use the following procedure to isolate an error to the failing bit (perform all arithmetic operations in octal):

1. For a CRAY X-MP computer system, use the index to determine the failing word as follows:

Index	Failing Word
O'177	0
index < O'100	(*index* x 2) + 1
index >= O'100	(*index* - O'77) x 2

   For a CRAY-1 computer system, use the index to determine the failing word as follows:

Index	Failing Word
O'77	0
index < O'40	(*index* x 2) + 1
index >= O'40	(*index* - O'37) x 2

2. Examine the failing word to isolate the error to the failing bit.

The following example for a CRAY X-MP computer system shows a dump that was generated after test section 1 detected an error. By examining the dump, you can isolate the error to the failing bit, as follows (perform all arithmetic operations in octal):

1. Use the index (O'100) to determine the failing word as follows:

   (*index* - O'77) x 2 = *failing word*

   (O'100 - O'77) x 2 = 2

2. By examining the failing word, you can see that bit 2^5 is dropped.

Example:

```
olibuf started in cpu A on Mon May 23 15:53:40 1988
olibuf: running
olibuf: restart file written to A33641-olibuf
name          <      1340> = 'olibuf  '
rev           <      1341> = '1.0     '
date          <      1342> = '05/17/88'
pass          <      1343> =  0
error         <      1344> =  1
seed          <      1345> =  33
failsec       <      1422> =  1
failpat       <      2156> =  'random  '
```

Example (continued):

Section 1 - test buffer tests parcel 0

```
                                        buff
   5340a     001000                         PASS
   5340b     001000                         PASS
   5340c     001000                         PASS
   5340d     020100 000033                  A1        00000033
   5341b     112100 000000                  0,A2      A1
   5341d     030223                         A2        A2+A3
   5342a     001000                         PASS
   5342b     001000                         PASS
   5342c     001000                         PASS
     .
     .
     .
   5541a     001000                         PASS
   5541b     001000                         PASS
   5541c     001000                         PASS
   5541d     020100 120304                  A1        00120304
   5542b     112100 000000                  0,A2      A1
   5542d     030223                         A2        A2+A3
   5543a     001000                         PASS
   5543b     001000                         PASS
   5543c     001000                         PASS
   5543d     020100 164114                  A1        00164114
   5544b     112100 000000                  0,A2      A1
   5544d     030223                         A2        A2+A3
   5545a     001000                         PASS
   5545b     001000                         PASS
   5545c     001000                         PASS
     .
     .
     .
```

Expected results

```
data        <  0> = 000000 000000 000000 000033  000000 000000 000000 000505
data + 0002 <  2> = 000000 000000 000000 016667  000000 000000 000000 010021
data + 0004 <  4> = 000000 000000 000000 130653  000000 000000 000000 042425
  .
  .
  .
data + 0174 <174> = 000000 000000 000000 147000  000000 000000 000000 141014
data + 0176 <176> = 000000 000000 000000 073260  000000 000000 000000 042520
```

Difference(s) between exp and act results

```
data+ 0100 <200> = *000000 000000 000000 120304* 000000 000000 000000 164114
                    000000 000000 000000 000040  000000 000000 000000 000000
```

3.6.3.2 CRAY Y-MP computer system error isolation

Use the following procedure to isolate an error to the failing bit (perform all arithmetic operations in octal):

1. Use the index to determine the failing word as follows:

Index	Failing Word
O'177	0
index < O'100	(*index* x 2) + 1
index >= O'100	(*index* - O'77) x 2

2. Examine the failing word to isolate the error to the failing bit.

The following example for a CRAY Y-MP computer system shows a dump that was generated after test section 1 detected an error. By examining the dump, you can isolate the error to the failing bit, as follows (perform all arithmetic operations in octal):

1. Use the index (O'132) to determine the failing word as follows:

   (*index* - O'77) x 2 = *failing word*

   (O'132 - O'77) x 2 = 66

2. By examining the failing word, you can see that bit 2^3 is dropped.

Example:

```
olibuf started in cpu A on Thu Aug 25 15:14:33 1988
olibuf: restart file written to A62851-olibuf
name            <      10740> = 'olibuf  '
rev             <      10741> = '1.0     '
date            <      10742> = '08/19/88'
pass            <      10743> =  0
error           <      10744> =  1
seed            <      10745> =  33
failsec         <      11022> =  1
failpat         <      11616> =  'random  '
```

Example (continued):

Section 1 - test buffer tests parcel 0

```
                                          buff
  15740a     001000                           PASS
  15740b     001000                           PASS
  15740c     001000                           PASS
  15740d     020100 000033 000000             A1        00000000033
  15741c     112100 000000 000000             0,A2      A1
  15742b     030223                           A2        A2+A3
  15742c     001000                           PASS
  15742d     020100 000505 000000             A1        00000000505
  15743c     112100 000000 000000             0,A2      A1
  15744b     030223                           A2        A2+A3
  15744c     001000                           PASS
  15744d     020100 016667 000000             A1        00000016667
  15745c     112100 000000 000000             0,A2      A1
  15746b     030223                           A2        A2+A3
     .
     .
     .
  16223d     020100 063732 000000             A1        00000063732
  16224c     112100 000000 000000             0,A2      A1
  16225b     030223                           A2        A2+A3
  16225c     001000                           PASS
  16225d     020100 165420 000000             A1        00000165420
  16226c     112100 000000 000000             0,A2      A1
  16227b     030223                           A2        A2+A3
  16227c     001000                           PASS
  16227d     020100 152151 000000             A1        00000152151
  16230c     112100 000000 000000             0,A2      A1
  16231b     030223                           A2        A2+A3
     .
     .
     .
```

Expected results

```
data        <  0> = 000000 000000 000000 000033  000000 000000 000000 000505
data + 0002 <  2> = 000000 000000 000000 016667  000000 000000 000000 010021
data + 0004 <  4> = 000000 000000 000000 130653  000000 000000 000000 042425
  .
  .
  .
data + 0174 <174> = 000000 000000 000000 147000  000000 000000 000000 141014
data + 0176 <176> = 000000 000000 000000 073260  000000 000000 000000 042520
```

The difference data shown below has the following format:

```
  name       + index     <offset>  = data ...
                                     data differences ....
```

name: The name of the data dumped on this line.
index: The index into the data starting at name. Optional, default: 0.
offset: The offset into the data buffer.
data: The actual data dumped.
The differences are marked with an asterisk (*) preceding the data word.
data differences: The bits that differ between the actual results and the expected results.

```
*** Differences ***

Source data buffer at 14740 in Memory copied to save buffer at 103362 in
  Memory
   Memory address in source data buffer = <offset> + 14740 (source data
     buffer)
   Memory address in save data buffer   = <offset> + 103362 (save data buffer)

Difference(s) between exp and act results

data + 0132 <264> = *000000 000000 000000 165420* 000000 000000 000000 152151
                     000000 000000 000000 000010  000000 000000 000000 000000
```

3.6.4 TEST TERMINATION

If a jump fails in section 5, an error exit occurs.

There are several monitor options that can cause a test to terminate. Refer to the information on test termination in section 2, Confidence Test and Monitor Overview.

3.6.5 TEST EXAMPLES

This subsection contains **olibuf** execution examples.

The following example runs **olibuf** with selected command options and shell facilities. The test runs for O'1000000 passes in CPU b with all default instructions. The job runs as a background process, and the output is sent to **olibuf.log**.

```
olibuf maxp 1000000 cpu b >olibuf.log
```

The following example runs **olibuf** with section 1 selected.

```
olibuf section 1
```

The following example runs **olibuf** for O'10000000 passes. Output is redirected to **olibuf.log**. The **nohup**(1) command allows the program to continue executing after you log off the system. You can later log on to check the test's progress. The ampersand (&) causes the entire command to execute in the background, so that another prompt is immediately displayed and you can continue to use the system.

```
nohup olibuf maxp 10000000 >olibuf.log &
```

The following example shows the output displayed when **olibuf** is run with all default values.

```
olibuf
```

Output:

```
olibuf
olibuf started in cpu A on Fri Aug 28 11:14:10 1987
olibuf reached maximum pass limit with 1000 passes and 0 errors
on Fri Aug 28 11:14:14 1987
```

The following example runs **olibuf** with the **+verbose** option enabled so that a line of output is generated after each pass.

```
olibuf +verbose
```

Output:

```
olibuf +verbose
olibuf started in cpu A on Fri Aug 28 11:14:14 1987
olibuf: pass = 1, error = 0  Fri Aug 28 11:14:14 1987
olibuf: pass = 2, error = 0  Fri Aug 28 11:14:14 1987
olibuf: pass = 3, error = 0  Fri Aug 28 11:14:14 1987
    .
    .
    .
olibuf: pass = 1000, error = 0  Fri Aug 28 11:14:14 1987
olibuf reached maximum pass limit with 1000 passes and 0 errors
on Fri Aug 28 11:14:14 1987
```

The following example runs **olibuf** in CPU c only.

```
    olibuf cpu c
```

Output:

```
olibuf cpu c
olibuf started in cpu C on Fri Aug 28 11:14:14 1987
olibuf reached maximum pass limit with 1000 passes and 0 errors
on Fri Aug 28 11:14:14 1987
```

The following example runs **olibuf** in CPUs a and b, with a as the master.

```
    olibuf cpu a,b
```

```
olibuf cpu a,b
olibuf started in cpus A, B with master cpu A on Fri Aug 28 11:14:14 1987
olibuf reached maximum pass limit with 1000 passes and 0 errors
on Fri Aug 28 11:14:14 1987
```

The following example runs **olibuf** with the **+verbose** option enabled. The output is generated after an error is detected.

```
    olibuf +verbose
```

Output:

```
olibuf +verbose
olibuf started in cpu A on Fri Aug 28 11:14:14 1987
olibuf:  restart file written to A7465-olibuf
name            <      14420> =  'olibuf  '
rev             <      14421> =  '1.0     '
date            <      14422> =  '08/27/87'
pass            <      14423> =  0
error           <      14424> =  1
seed            <      14425> =  52301500217376
failsec         <      23221> =  1
failpat         <      15174> =  'solid   '

Generated test buffer tests parcel 0

                                   buff
```

(the test buffer that was executing when the error was detected is dumped in parcel and ASCII format)

```
Section 1 - parcel 0 test
The expected data shown below has the following format:

  name        + index     <offset>  = data ...

name:      The name of the data dumped on this line.
index:     The index into the data starting at name.  Optional, default: 0.
offset:    The offset into the data buffer.
data:      The actual data dumped.


*** Expected Results ***   cpu A (master)

Source data buffer at 6427 in Memory copied to save buffer at 70201 in Memory
   Memory address in source data buffer = <offset> + 6427 (source data buffer)
   Memory address in save data buffer   = <offset> + 70201 (save data buffer)


*** Expected Results ***
```

(*the expected data is dumped in parcel format*)

```
The difference data shown below has the following format:

  name        + index     <offset>  = data ...
                                      data differences ....

name:     The name of the data dumped on this line.
index:    The index into the data starting at name.  Optional, default: 0.
offset:   The offset into the data buffer.
data:     The actual data dumped.
          The differences are marked with an asterisk (*) preceding the data word.
data differences:  The bits that differ between the actual results and
                   the expected results.

*** Differences ***   cpu A (master)
```

Example (continued):

```
Source data buffer at 6427 in Memory copied to save buffer at 71204 in Memory
   Memory address in source data buffer = <offset> + 6427 (source data buffer)
   Memory address in save data buffer   = <offset> + 71204 (save data buffer)

*** Differences ***
```

(The differences are displayed. Differences are the results of the actual execution of the test buffer that differ from the expected results.)

```
The first address (FADD) of the diagnostic is 14420a
olibuf reached maximum error limit with 0 passes and 1 errors
on Fri Aug 28 11:14:23 1987
```

3.6.6 TEST MESSAGES

The **olibuf** test produces the following types of messages:

- Informative
- Error

These messages are described in the subsections that follow.

3.6.6.1 Informative messages

If no error occurs, **olibuf** produces two messages, one at start-up time and another at test termination. If the **+verbose** option is enabled, a message is sent to **stdout** (standard output device) after each pass through the test loop.

On an error, the test provides information such as the following:

- Pass and error counts
- Seed at the beginning of the pass on which the error occurred
- Failing word and parcel
- Test buffer data used when the error occurred
- Expected results
- Actual results
- Differences between the expected results from the master CPU and the actual execution results from all of the selected CPUs

3.6.6.2 Error messages

One of the following error messages is sent to **stderr** (standard error device) if an invalid command option is entered:

```
olibuf:  initpat:  No data patterns selected
   Select one or more data patterns and rerun.

olibuf:  bldtbl:  Invalid section selected.  Valid sections are:  1-5.
   Select one or more valid test sections and rerun.
```

3.7 olsbt

The **olsbt** test is an on-line semaphore, shared B and shared T register test for CX/CEA systems. It tests the following components:

- Shared B registers
- Shared T registers
- Semaphores
- Clusters

The **olsbt** test generates a random sequence of shared register instructions and data to detect inter-CPU communication failures. The generated instructions are simulated and then executed. If no differences are detected, the test generates new instructions and data, and repeats the process until the maximum pass, error, or time limit is reached for the selected cluster number.

The **olsbt** test runs under the confidence monitor program, **olcmon.** The **olcmon** monitor compares the actual and simulated results. For additional information on **olcmon**, refer to section 2 of this manual, Confidence Test and Monitor Overview.

For additional information on inter-CPU communication, refer to the following manuals (as appropriate to your system configuration):

Publication	Title
CSM0110000	CRAY X-MP/2 System Programmer Reference Manual
CSM-0111-000	CRAY X-MP/1 System Programmer Reference Manual
CSM0112000	CRAY X-MP/4 System Programmer Reference Manual
CSM-0400-000	CRAY Y-MP System Programmer Hardware Reference Manual

3.7.1 TEST SYNOPSIS

The **olsbt** command options can be entered in any order. If an option is omitted, the program uses the default value. The test synopsis lists the **olsbt** command options and arguments in the following order:

1. Monitor options
2. Test-specific options
3. Data pattern options
4. Instruction options

Synopsis:

olsbt [**chkpnt** *mode*] [**cpu** *clist*] [**cputime** *h*:*m*:*s*] [+/-**getseed**]

[**getseed** *file*] [**help**] [**maxerr** *n*] [**maxp** *n*] [+/-**parcel**] [**time** *h*:*m*:*s*]

[+/-**verbose**]†

[**cluster** *n*] [**numins** *n*] [+/-**repeat**] [**seed** *n*]

[+/-**bits**] [+/-**onezero**] [+/-**random**]

cluster *n*

Selects specific cluster. *n* can be any one of the following cluster numbers associated with the indicated mainframe (cluster number 1 is reserved for the operating system):

Mainframe	Cluster Numbers
CRAY Y-MP/8	2, 3, 4, 5, 6, 7, 10, 11
CRAY Y-MP/4	2, 3, 4, 5
CRAY X-MP/4	2, 3, 4, 5
CRAY X-MP/2	2, 3
CRAY X-MP/1	2, 3

The default for *n* is a random cluster number. The cluster number does not change during test execution. **cluster** *n* must be used to recreate a failure.

numins *n* Sets the number of instructions to be generated. *n* can be any value within the range 1 through O'20. The default for *n* is O'20.

+/-**repeat**

Enables (+**repeat**) or disables (-**repeat**) the option that repeats the first pass until the diagnostic terminates. +**repeat** is useful for recreating an error. It is normally used with **cluster** *n* and one of the following options: **seed** *n*, +**getseed**, or **getseed** *file*. The default is -**repeat** (the program generates new instructions and data after each pass).

† The monitor command options are described in section 2, Confidence Test and Monitor Overview.

seed *n* Sets the random seed to *n*. *n* can be any 64-bit octal value. If *n* is 0, the test reads the real-time clock and uses the value for the initial seed. The default for *n* is O'33. If **seed** *n* is selected, do not select **+getseed** or **getseed** *file*.

+/-bits, +/-onezero, +/-random
Selects (+) or deselects (-) specific data patterns. The default selects all of the patterns. The data patterns are as follows:

Option	Data Pattern
bits	Random number of consecutive 1-bits in a word. For example: 0000017777777776000000 1777000000000000000377 1777777777777777777777 0000000000000000000000 0000000000100000000000
onezero	Random selection of all 1's or all 0's in a word. For example: 1777777777777777777777 0000000000000000000000
random	Random bit generation in a word. For example: 1023122123232122777127 0003423100233344322177 1640034356453221213532 1123235467543221344120 1304322300332105534311

3.7.2 TEST EXECUTION

The **olsbt** test should be executed with the maximum number of CPUs available on the system. This allows the requested cluster number to become available more quickly, since one process will be started in each CPU.

The **olsbt** test execution sequence is as follows:

1. Test initialization and hardware configuration detection
2. Random instruction and data generation
3. Random instruction buffer simulation
4. Random instruction buffer execution
5. Comparison of simulation and execution results
6. Error isolation

Steps 2 through 5 occur on each pass through the test loop. Step 6 occurs only on error.

3.7.2.1 Test initialization and hardware configuration detection

At test initialization, all instructions are enabled. The hardware configuration detection routine identifies the number of available clusters. If the cluster specified by the command option **cluster** *n* is not available, the program overrides **cluster** *n* and uses a random cluster.

3.7.2.2 Random instruction and data generation

These routines build and generate the random instruction buffers and initial data. Instructions for the buffers are randomly selected from a list of instructions. The values of the *i*, *j*, and *k* fields are randomly selected when appropriate.

If four CPUs are selected, four random instruction buffers are created; one for each CPU. If only one CPU is selected, two random instruction buffers are created and both are executed in the selected CPU. Each instruction buffer contains instructions that enable it to write to the shared registers. Only one buffer can write to the shared registers at a time. The buffer that can write to the shared registers is rotated through the selected CPUs, starting with the selected master CPU. The other buffers can read from the shared registers if the master is not writing to that particular shared register. Before another buffer can begin writing to the shared registers, all buffers must be syncronized.

A sample of the instruction buffers for four CPUs is as follows:

```
        ibuff0
003416      SM16      1,TS
003404      SM04      1,TS
003401      SM01      1,TS
003603      SM03      0
003627      SM27      0
003434      SM34      1,TS
003730      SM30      1
003726      SM26      1
003702      SM02      1
026227      A2        SB2
003634      SM34      0
003635      SM35      0
003405      SM05      1,TS
003605      SM05      0
003617      SM17      0
003413      SM13      1,TS
003613      SM13      0
003410      SM10      1,TS
003415      SM15      1,TS
003406      SM06      1,TS
003636      SM36      0
005000      J         B00


        ibuff1
003616      SM16      0
003403      SM03      1,TS
072473      S4        ST7
072333      S3        ST3
026607      A6        SB0
003603      SM03      0
003431      SM31      1,TS
003425      SM25      1,TS
003427      SM27      1,TS
003634      SM34      0
003620      SM20      0
026427      A4        SB2
003623      SM23      0
003405      SM05      1,TS
003605      SM05      0
003600      SM00      0
003413      SM13      1,TS
003613      SM13      0
003610      SM10      0
003436      SM36      1,TS
003636      SM36      0
005000      J         B00
```

Example (continued):

```
        ibuff2
003604      SM04      0
003403      SM03      1,TS
003603      SM03      0
003631      SM31      0
003434      SM34      1,TS
003634      SM34      0
003433      SM33      1,TS
003435      SM35      1,TS
003423      SM23      1,TS
026267      A2        SB6
072663      S6        ST6
073343      ST4       S3
003605      SM05      0
072213      S2        ST1
026647      A6        SB4
003621      SM21      0
003413      SM13      1,TS
003613      SM13      0
003615      SM15      0
003436      SM36      1,TS
003636      SM36      0
005000      J         B00


        ibuff3
003601      SM01      0
003403      SM03      1,TS
003603      SM03      0
026067      A0        SB6
026367      A3        SB6
026767      A7        SB6
003614      SM14      0
003625      SM25      0
003434      SM34      1,TS
003634      SM34      0
003633      SM33      0
03405       SM05      1,TS
003605      SM05      0
003417      SM17      1,TS
003400      SM00      1,TS
003421      SM21      1,TS
003613      SM13      0
003606      SM06      0
003436      SM36      1,TS
003636      SM36      0
05000       J         B00
```

3.7.2.3 Random instruction buffer simulation

After the instructions and data are generated, the master CPU simulates the random instruction buffers. The **save** monitor routine saves the results.

Each instruction type has a unique simulation routine. The simulation routines do not use any of the shared register hardware.

3.7.2.4 Random instruction buffer execution

After the instructions are simulated, all of the selected CPUs execute their own instruction buffer in the selected cluster. The master CPU uses the system call **cpu**(4D) to select the cluster.

The **olsbt** test allows you to test inter-CPU control and communication by synchronizing code execution among selected CPUs. The first CPU selected is the master CPU, which generates and simulates all instruction buffers for all selected CPUs.

The following characteristics apply to instruction buffer execution:

- The master CPU creates and schedules processes using the following system calls:

System Call	Description
tfork(2)	Creates a multitasking process for each selected CPU
cpselect(2)	Schedules the processes in the CPUs

- Only one buffer can write to the shared B and shared T registers in the specified cluster at a time.

- The master CPU loads the shared registers with the generated data before starting the other CPUs. The master CPU then waits for all CPUs to execute their buffers before unloading the shared registers.

- All semaphores used in the test and set instructions in the instruction buffers are initially set.

Before the instructions can be executed, the master CPU loads the following:

- Shared B registers
- Shared T registers
- Semaphore register
- Address registers for the master CPU
- Scalar registers for the master CPU

The other CPUs load the following:

- Address registers
- Scalar registers

Then an unconditional jump to the random instruction buffer is executed in each CPU. At the end of the random instruction buffer is a jump to B0. Each CPU unloads the contents of its address and scalar registers. The master CPU waits until all CPUs have executed and then unloads the contents of the shared registers. The **save** monitor routine saves the results.

3.7.2.5 Comparison of simulation and execution results

After the instructions execute in all of the selected CPUs, the **compare** monitor routine compares the results, and one of the following actions occurs:

- If the results match, the test proceeds with the next data pattern. After all of the selected data patterns are run, the pass count is incremented.

- If the results do not match, the test dumps all of the data related to the suspected failure.

If a deadlock interrupt was received, a core dump is produced and the test terminates.

3.7.2.6 Error isolation

The output dump contains the following:

- Data used when the failure occurred
- Simulated execution results
- Actual execution results (if different from the simulated results)
- Exclusive OR of the simulated and actual execution results

The program may report an error resulting from a failure in either the simulated or actual execution. To determine if the error is the result of an actual execution failure, start **olsbt** in a different CPU and select the suspected failing CPU. For example, the following entry starts **olsbt** in CPU c:

```
olsbt cpu c
```

If **olsbt** fails, and the simulated execution is suspect, rerun **olsbt** using a different master CPU, the failing seed, and the failing cluster, as follows:

olsbt cpu a,c +repeat seed *n* cluster *n*

If **olsbt** fails in CPU c, the failure is in the actual execution of the random instruction buffer. If **olsbt** does not fail, the error is either in the simulated execution results from CPU c or it is very intermittent.

3.7.3 TEST TERMINATION

For information on test termination, refer to section 2.4, Test Termination.

3.7.4 TEST EXAMPLES

This subsection contains **olsbt** execution examples.

The following example runs **olsbt** with all defaults. **olsbt** executes in CPU a. The output is displayed at the operator console.

```
olsbt
```

The following example runs **olsbt** in CPUs a, b, c, and d. The output is displayed at the operator console.

```
olsbt cpu a,b,c,d
```

The following example runs **olsbt** for O'10000000 passes. By default, **olsbt** executes in CPU a. Output is redirected to **sbt.log**. The **nohup**(1) command allows the program to continue executing after you log off the system. You can later log on to check the test's progress. The ampersand (&) causes the entire command to execute in the background, so that another prompt is immediately displayed and you can continue to use the system.

```
nohup olsbt maxp 10000000 >sbt.log &
```

The following example runs **olsbt** with selected command options and shell facilities. **olsbt** runs for O'1000000 passes in CPUs a and b. The job runs as a background process, and output is sent to **sbt.log**.

```
olsbt maxp 1000000 cpu a,b >sbt.log &
```

The following example shows a procedure for determining how frequently an error occurs. **olsbt** is rerun with the **+repeat** option, so that the first pass is run repeatedly until the test terminates. The test uses the seed value and the failing cluster number from the output at the time of the initial error. Error isolation is disabled and **olsbt** executes in CPUs a, b, c, and d. The job runs as a background process, and output is sent to **sbt.log**.

```
olsbt +repeat -isolate maxerr 100 maxp 100 cpu a,b,c,d seed
  14366510167135540025ll cluster 4 >sbt.log &
```

The following example shows the ouput displayed when **olsbt** is run with all default values.

```
olsbt
```

Output:

```
olsbt
olsbt started in cpu A on Wed Dec 14 15:18:56 1988
CRAY Y-MP MODE

olsbt reached maximum pass limit with 1000 passes and 0 errors
on  Wed Dec 14 15:20:23 1988
```

The following example runs **olsbt** in four CPUs with the **+verbose** option enabled so that a line of output is generated after each pass.

```
olsbt cpu a,b,c,d +verbose
```

Output:

```
olsbt cpu a,b,c,d +verbose
olsbt started in cpus A, B, C, D with master cpu A on Wed Dec 14 15:19:08 1988
CRAY Y-MP MODE

olsbt: pass =          1, error =          0  Wed Dec 14 15:19:26 1988
olsbt: pass =          2, error =          0  Wed Dec 14 15:19:26 1988
olsbt: pass =          3, error =          0  Wed Dec 14 15:19:26 1988
    .
    .
    .
```

Output (continued):

```
olsbt: pass =       1000, error =          0  Wed Dec 14 15:21:23 1988
olsbt reached maximum pass limit with 1000 passes and 0 errors
on  Wed Dec 14 15:21:23 1988
```

The following example runs **olsbt** in CPUs a, b, c, d with CPU a as the master.

```
    olsbt cpu a,b,c,d
```

Output on an error:

```
olsbt cpu a,b,c,d
olsbt started in cpus A, B, C, D with master cpu A on Wed Dec  7 14:27:00 1988
CRAY Y-MP MODE

olsbt: restart file written to A35411-olsbt
name            <        200> =  'olsbt   '
rev             <        201> =  '5.0     '
date            <        202> =  '12/07/88'
pass            <        203> =  4
error           <        204> =  1
seed            <        205> =  1033360000000000000000
failpat         <       1774> =  'bits    '
failcln         <        220> =  2
numins          <        206> =  20

TASK  0 random instruction buffer executed in CPU A

                                 ibuff0
   4200a     003416                          SM16      1,TS
   4200b     003404                          SM04      1,TS
   4200c     003401                          SM01      1,TS
   4200d     003603                          SM03      0
   4201a     003627                          SM27      0
   4201b     003434                          SM34      1,TS
   4201c     003730                          SM30      1
   4201d     003726                          SM26      1
   4202a     003702                          SM02      1
   4202b     026227                          A2        SB2
   4202c     003634                          SM34      0
   4202d     003635                          SM35      0
   4203a     003405                          SM05      1,TS
   4203b     003605                          SM05      0
   4203c     003617                          SM17      0
   4203d     003413                          SM13      1,TS
   4204a     003613                          SM13      0
   4204b     003410                          SM10      1,TS
   4204c     003415                          SM15      1,TS
   4204d     003406                          SM06      1,TS
   4205a     003636                          SM36      0
   4205b     005000                          J         B00
```

Output (continued):

TASK 1 random instruction buffer executed in CPU B

```
                                 ibuff1
  4240a   003616                          SM16    0
  4240b   003403                          SM03    1,TS
  4240c   072473                          S4      ST7
  4240d   072333                          S3      ST3
  4241a   026607                          A6      SB0
  4241b   003603                          SM03    0
  4241c   003431                          SM31    1,TS
  4241d   003425                          SM25    1,TS
  4242a   003427                          SM27    1,TS
  4242b   003634                          SM34    0
  4242c   003620                          SM20    0
  4242d   026427                          A4      SB2
  4243a   003623                          SM23    0
  4243b   003405                          SM05    1,TS
  4243c   003605                          SM05    0
  4243d   003600                          SM00    0
  4244a   003413                          SM13    1,TS
  4244b   003613                          SM13    0
  4244c   003610                          SM10    0
  4244d   003436                          SM36    1,TS
  4245a   003636                          SM36    0
  4245b   005000                          J       B00
```

TASK 2 random instruction buffer executed in CPU C

```
                                 ibuff2
  4300a   003604                          SM04    0
  4300b   003403                          SM03    1,TS
  4300c   003603                          SM03    0
  4300d   003631                          SM31    0
  4301a   003434                          SM34    1,TS
  4301b   003634                          SM34    0
  4301c   003433                          SM33    1,TS
  4301d   003435                          SM35    1,TS
  4302a   003423                          SM23    1,TS
  4302b   026267                          A2      SB6
  4302c   072663                          S6      ST6
  4302d   073343                          ST4     S3
  4303a   003605                          SM05    0
  4303b   072213                          S2      ST1
  4303c   026647                          A6      SB4
  4303d   003621                          SM21    0
  4304a   003413                          SM13    1,TS
  4304b   003613                          SM13    0
  4304c   003615                          SM15    0
  4304d   003436                          SM36    1,TS
  4305a   003636                          SM36    0
  4305b   005000                          J       B00
```

Output (continued):

TASK 3 random instruction buffer executed in CPU D

```
                                          ibuff3
  4340a     003601                                    SM01     0
  4340b     003403                                    SM03     1,TS
  4340c     003603                                    SM03     0
  4340d     026067                                    A0       SB6
  4341a     026367                                    A3       SB6
  4341b     026767                                    A7       SB6
  4341c     003614                                    SM14     0
  4341d     003625                                    SM25     0
  4342a     003434                                    SM34     1,TS
  4342b     003634                                    SM34     0
  4342c     003633                                    SM33     0
  4342d     003405                                    SM05     1,TS
  4343a     003605                                    SM05     0
  4343b     003417                                    SM17     1,TS
  4343c     003400                                    SM00     1,TS
  4343d     003421                                    SM21     1,TS
  4344a     003613                                    SM13     0
  4344b     003606                                    SM06     0
  4344c     003436                                    SM36     1,TS
  4344d     003636                                    SM36     0
  4345a     005000                                    J        B00


initial address register data for TASK 0
initar0          <       5210> =  0000000000020000000000   . . .
initar0   + 0004 <       5214> =  0000000000000000000000   . . .


initial scalar register data for TASK 0
initsr0          <       5200> =  0377777777776000000000   . . .
initsr0   + 0004 <       5204> =  0000000000000000000000   . . .


initial address register data for TASK 1
```
(*address register data is displayed for task 1*)

```
initial scalar register data for TASK 1
```
(*scalar register data is displayed for task 1*)

```
initial address register data for TASK 2
```
(*address register data is displayed for task 2*)

Output (continued):

initial scalar register data for TASK 2
(*scalar register data is displayed for task 2*)

initial address register data for TASK 3
(*address register data is displayed for task 3*)

initial scalar register data for TASK 3
(*scalar register data is displayed for task 3*)

```
initial shared B register data
initsb          <        5300> =  0000000000000000000000   . . .
initsb   + 0004 <        5304> =  0000000000000177777777   . . .


initial shared T register data
initst          <        5310> =  0000000000000777760000   . . .
initst   + 0004 <        5314> =  1777740000000001777777   . . .


initial semaphore register data
initsm          <        5320> =  1577777777700000000000

simulated random instruction buffer results
The expected data shown below has the following format:

   name      + index     <offset>  = data ...

 name:              The name of the data dumped on this line.
 index:             The index into the data starting at name.  Optional, default: 0.
 offset:            The offset into the data buffer.
 data:              The actual data dumped.

*** Expected Results ***   cpu A (master)

Source data buffer at 6200 in Memory
   Memory address in source data buffer = <offset> + 6200 (source data buffer)


simulated address register data results for TASK 0
actar0             <      10>  =  0000000000020000000000   . . .
actar0   + 0004    <      14>  =  0000000000000000000000   . . .
```

Output (continued):

```
simulated scalar register data results for TASK 0
actsr0            <      0> =  0377777777776000000000   . . .
actsr0   + 0004   <      4> =  0000000000000000000000   . . .
```

simulated address register data results for TASK 1
(*address register data is displayed for task 1*)

simulated scalar register data results for TASK 1
(*scalar register data is displayed for task 1*)

simulated address register data results for TASK 2
(*address register data is displayed for task 2*)

simulated scalar register data results for TASK 2
(*scalar register data is displayed for task 2*)

simulated address register data results for TASK 3
(*address register data is displayed for task 3*)

simulated scalar register data results for TASK 3
(*scalar register data is displayed for task 3*)

```
simulated shared B register data results
actsb             <    100> =  0000000000000000000000   . . .
actsb    + 0004   <    104> =  0000000000000177777777   . . .

simulated shared T register data results
actst             <    110> =  0000000000000777760000   . . .
actst    + 0004   <    114> =  1777777777777777777777   . . .

simulated semaphore register data results
actsm             <    120> =  1657473777200000000000
```

Output (continued):

Differences are the results from actual execution of the random instruction buffer that differ from the master (simulated or actual) execution.

```
actar   = address register data results
actsr   = scalar register data results
actsb   = sb0-sb7 register data results
actst   = st0-st7 register data results
actsm   = semaphore register data result
```

The difference data shown below has the following format:

```
   name      + index     <offset>  = data ...
                                     data differences ....
```

```
 name:              The name of the data dumped on this line.
 index:             The index into the data starting at name.  Optional, default: 0.
 offset:            The offset into the data buffer.
 data:              The actual data dumped.
                    The differences are marked with an asterisk (*) preceding the
                    data word.
 data differences:  The bits in difference between the actual results and the
                    expected results.
```

```
*** Differences ***   cpu A (master)

Source data buffer at 7200 in Memory copied to save buffer at 113755 in Memory
   Memory address in source data buffer = <offset> + 7200 (source data buffer)
   Memory address in save data buffer   = <offset> + 113755 (save data buffer)


actual random buffer execution results

actst    + 0004    <     114>  = *0000000000000000000000   . . .
                                  1777777777777777777777   . . .


The first address (FADD) of the diagnostic is 200a


olsbt reached maximum error limit with 4 passes and 1 errors at Wed Dec  7 14:27:00
1988
```

If **olsbt** determines that the initial load of the semaphores failed, the test produces a dump and terminates.

Output on an error:

```
olsbt cpu a,b,c,d
olsbt started in cpus A, B, C, D with master cpu A on Wed Dec  7 15:12:29 1988
CRAY Y-MP MODE

execute: an error was detected in the initial load of the semaphore register
olsbt: restart file written to A60249-olsbt
name            <        200> =  'olsbt   '
rev             <        201> =  '5.0     '
date            <        202> =  '12/07/88'
pass            <        203> =  0
error           <        204> =  1
seed            <        205> =  33
failpat         <       1774> =  'bits    '
failcln         <        220> =  2
numins          <        206> =  20

TASK  0 random instruction buffer executed in CPU A


   2175a     073102                                     SM         S1
   2175b     072202                                     S2         SM
   2175c     046012                                     S0         S1\S2


initial address register data for TASK 0
initar0         <       5210> =  0000000000000000000000   . . .
initar0   + 0004 <       5214> =  0000000000000000000000   . . .


initial scalar register data for TASK 0
initsr0         <       5200> =  0000000000000000000760   . . .
initsr0   + 0004 <       5204> =  0000777777777777777777   . . .


initial shared B register data
initsb          <       5300> =  0000000000000000000000   . . .
initsb    + 0004 <       5304> =  0000000000000000000000   . . .


initial shared T register data
initst          <       5310> =  0000000000000000000020   . . .
initst    + 0004 <       5314> =  1777776000000000000007   . . .
```

Output (continued):

```
initial semaphore register data
initsm          <       5320> =  1106721617240000000000

simulated random instruction buffer results
The expected data shown below has the following format:

   name       + index     <offset>  = data ...

 name:              The name of the data dumped on this line.
 index:             The index into the data starting at name.  Optional, default: 0.
 offset:            The offset into the data buffer.
 data:              The actual data dumped.

*** Expected Results ***   cpu A (master)

Source data buffer at 6200 in Memory
   Memory address in source data buffer = <offset> + 6200 (source data buffer)



simulated address register data results for TASK 0
actar0               <      10>  =  0000000000000000000000   . . .
actar0    + 0004     <      14>  =  0000000000000000000000   . . .


simulated scalar register data results for TASK 0
actsr0               <       0>  =  0000000000000000000000   . . .
actsr0    + 0004     <       4>  =  0000000000000000000000   . . .


simulated shared B register data results
actsb                <     100>  =  0000000000000000000000   . . .
actsb     + 0004     <     104>  =  0000000000000000000000   . . .


simulated shared T register data results
actst                <     110>  =  0000000000000000000000   . . .
actst     + 0004     <     114>  =  0000000000000000000000   . . .


simulated semaphore register data results
actsm                <     120>  =  1106721617240000000000
```

Output (continued):

Differences are the results from actual execution of the random instruction buffer that differ from the master (simulated or actual) execution.

```
actar   = address register data results
actsr   = scalar register data results
actsb   = sb0-sb7 register data results
actst   = st0-st7 register data results
actsm   = semaphore register data result
```

The difference data shown below has the following format:

```
   name      + index     <offset>  = data ...
                                     data differences ....

 name:              The name of the data dumped on this line.
 index:             The index into the data starting at name.  Optional, default: 0.
 offset:            The offset into the data buffer.
 data:              The actual data dumped.
                    The differences are marked with an asterisk (*) preceding the
                    data word.
 data differences:  The bits in difference between the actual results and
                    the expected results.
```

```
*** Differences ***   cpu A (master)

Source data buffer at 6200 in Memory
   Memory address in source data buffer = <offset> + 6200 (source data buffer)



*** Differences ***   cpu A (master)

Source data buffer at 7200 in Memory copied to save buffer at 113755 in Memory
   Memory address in source data buffer = <offset> + 7200 (source data buffer)
   Memory address in save data buffer   = <offset> + 113755 (save data buffer)


actsm                <     120>  = *1000000000000000000000
                                    0106721617240000000000


The first address (FADD) of the diagnostic is 200a


olsbt reached maximum error limit with 0 passes and 1 errors
at Wed Dec  7 15:12:30 1988
```

3.7.5 TEST MESSAGES

The **olsbt** test produces the following types of messages:

- Test mode
- Informative
- Error

These messages are listed in the subsections that follow.

3.7.5.1 Test mode messages

During test execution, one of the following messages is displayed to indicate the test mode:

CRAY Y-MP MODE
    Indicates that the mainframe is a CEA system (Y-mode).

CRAY X-MP MODE
    Indicates that the mainframe is a CRAY X-MP computer system.

3.7.5.2 Informative messages

If no error occurs, the test generates two messages, one at start-up time and the other at test termination.

If the **+verbose** option is enabled, a message is sent to **stdout** (standard output device) after each pass through the test loop. On an error, the test provides information such as the following:

- Pass and error counts
- Seed at the beginning of the pass on which the error occurred
- Cluster number for the error that occurred
- Contents of the instruction buffers and in which CPU each instruction buffer was executed
- Initial data
- Resulting data from the simulated instruction execution in the master CPU
- Differences between the simulation execution results from the master CPU and the actual execution results from all of the selected CPUs

3.7.5.3 Error messages

The following error message is sent to **stderr** (standard error device) if an invalid command option is entered:

```
olsbt: no data pattern(s) selected
   All data patterns were deselected (-bits -onezero -random).
   Correct and rerun.
```

The following messages are sent to **stderr** if **olsbt** detects an unexpected error. Select a different master CPU and rerun the test. If the problem persists, contact your CRI representative.

```
olsbt:  generate:  (software error)  The instruction does not have a
generation routine.

olsbt:  simulate:  (software error)  a deadlock was encountered
during simulation.

olsbt:  simulate:  (software error)  gh field is not valid.

olsbt:  simulate:  (software error)  ijk field is not valid.

olsbt:  simulate:  (software error)  The instruction does not have a
simulation routine.
```

The following error message is sent to **stderr** if **olsbt** detects an error in the initial load of the semaphore register. Contact your CRI representative.

```
execute:  an error was detected in the initial load of the semaphore
register.
```

4. MAINTENANCE TEST AND MONITOR OVERVIEW

The on-line maintenance tests provide error detection and isolation. These on-line tests are variants of the off-line diagnostic tests.

This section provides an overview of the following information:

- Maintenance monitor (**olmon**†)
- Program synopsis
- Test execution
- Test-specific requirements
- Test termination
- Test examples
- Test messages
- Diagnostic memory image

For a brief description of each maintenance test, refer to appendix A, On-line Diagnostic Programs. For a list of test execution times, refer to appendix B, Test Execution Times. For additional information on the maintenance tests, refer to the on-line diagnostic listings.

4.1 MAINTENANCE MONITOR (**olmon**)

The **olmon** monitor is a C program monitor for the on-line maintenance tests. The loader program attaches **olmon** to a slightly modified version of an off-line diagnostic test to create an on-line maintenance program.

The **olmon** monitor provides the interface to the on-line maintenance tests. By accepting and interpreting command options and arguments, **olmon** allows you to do the following:

- Set the diagnostic information block (DIB) locations in the diagnostic

- Set limits on the maximum number of passes and errors allowed (**maxerr** *n* and **maxp** *n*)

- Set limits on test execution time, in CPU time (**cputime** *h*:*m*:*s*) or elapsed (wall-clock) time (**time** *h*:*m*:*s*)

---

† CEA (X-mode) and CX/1 systems only

- Allocate memory for memory tests
- Select the CPU to be tested
- Send test results to **stdout** (standard output device) by default or to a file by indicating output redirection on the command line

4.2 PROGRAM SYNOPSIS

Before a test can be started, UNICOS must be running in the CPU to be tested. The **olmon** command options can be entered in any order. If an option is omitted, the program uses the default value.

Synopsis:

test [**chkpnt** *mode*] [**cpu** *x*] [**cputime** *h*:*m*:*s*] [**data** *x*:*y*] [*dib x*]

[**help**] [**maxerr** *n*] [**maxp** *n*] [**time** *h*:*m*:*s*] [**+/-verbose**] [**words** *n*]

chkpnt *mode*

Indicates whether restart files are to be generated. Restart files cannot be created unless output is directed to a disk file.

mode is one of the following arguments:

Argument	Description
first	Generates a restart file for the first failure detected (default)
all	Generates a restart file for each failure detected, including failures detected during error isolation
none	Does not generate restart files

The default generates a restart file for the first failure detected.

For additional information, refer to the following: **chkpnt**(1), **restart**(1), **chkpnt**(2), and **restart**(2).

cpu *x*

Selects **cpu** *x*. *x* can be **a**, **b**, **c**, **d**, **e**, **f**, **g**, or **h**. The default is **cpu a**.

cputime *h*:*m*:*s*

Sets the test execution time in CPU time. The time is specified in hours (*h*), minutes (*m*), and seconds (*s*); minutes and seconds; or just seconds. Use colons as delimiters, as follows: *h*:*m*:*s*.

Generally, actual execution time is within one second of the specified CPU time. If **cputime** is allowed to default (or is set to 0), the test uses the **maxp** value. However, if set to a value other than 0, **cputime** overrides **maxp**.

data *x*:*y* Stores data *y* (octal) at location *x* (octal) before the diagnostic is started; no length check is performed on *x*.

dib x Allows you to set the following diagnostic information block (DIB) options in the diagnostic:

Option	Description
modes *x*	Test mode
secs *x*	Section select
stop *x*	Stop condition bits
.	.
.	.
.	.
option x	Refer to the on-line listings for additional DIB descriptions.

In addition to the previously listed options, you can set the following options for **olcmx** only (refer to subsection 4.4.2, **olcmx**):

Option	Description
param *x*	Test control bits
rcp *x*	Repeat current pass
reqi *x*	Number of parcels requested
rislp *x*	Repeat isolation loop
rnum *x*	Initial random number
rpass *x*	Starting pass count (**maxp** *n* must be greater than **rpass** *x*)

To determine the *dib x* settings, refer to the on-line diagnostic listings.

help Generates an on-line help display containing a synopsis and brief description of the command options and arguments. If **help** is entered with a test name, help information is written to **stdout**, and the test terminates.

maxerr *n* Sets the maximum number of errors. *n* is an octal value. The default for *n* is 1.

maxp *n* Sets the maximum number of passes. *n* is an octal value. The default for *n* is O'1000. If **cputime** or **time** is set to a value other than 0, the specified option overrides **maxp**.

time *h*:*m*:*s*

Sets the test execution time in elapsed (wall-clock) time. The time is specified in hours (*h*), minutes (*m*), and seconds (*s*); minutes and seconds; or just seconds. Use colons as delimiters, as follows: *h*:*m*:*s*.

Generally, actual execution time is within one second of the specified elapsed time. If **time** is allowed to default (or is set to 0), the test uses the **maxp** value. However, if specified to a value other than 0, **time** overrides **maxp**.

+/-verbose

Enables (**+verbose**) or disables (**-verbose**) the generation of informational messages. The **+verbose** option causes a line of output to be generated after each pass of the diagnostic. The default is **-verbose**.

words *n* Allocates words for memory testing, and sets the DIB locations **mfrst** and **mlast** (the first and last memory addresses to be tested). *n* is an octal value. If **words** *n* is not entered, the diagnostic sets the test limits by default. Default values are test-dependent (refer to the on-line diagnostic listings).

4.3 TEST EXECUTION

To start a single diagnostic test, enter the following:

- *test*
- Monitor command options

To run a sequence of diagnostics, use the **runsequence** utility described in section 7, Utility Programs.

4.4 TEST-SPECIFIC REQUIREMENTS

This subsection provides information on test-specific requirements and command line entries. You must observe these requirements to ensure that the indicated test executes properly.

4.4.1 **olaht**

To run **olaht**† (on-line A register indexing test), you must set **cput** *n* (the DIB option to set the CPU type), as follows:

Value	CPU Type
10	CRAY X-MP/1
20	CRAY X-MP/2
40 (default)	CRAY Y-MP CRAY X-MP EA (X-mode) CRAY X-MP/4

To execute **olaht** on a CRAY X-MP/2 or CRAY X-MP/1 computer system, you must set **cput** as previously indicated (rather than allow it to default) or the test will generate invalid results.

To ensure that the test automatically selects the appropriate **cput** value, do the following:

1. Rename **olaht** to **olaht1** or **olaht2.**
2. Create a shell script called **olaht.**
3. Enter the following information into the **olaht** shell script:

```
olaht1 cput 10 $*

       or

olaht2 cput 20 $*
```

4.4.2 **olcmx**

To run **olcmx**† (on-line random instruction and operand test) on a Cray computer system without compressed indexing capabilities, you must set **param** *n* (DIB option to set the test control bits) so that the vector compressed indexing instructions are disabled. To disable these instructions, set **param** as follows:

```
olcmx param 400000001
```

The default value for **param** is 1 (stop on isolated error). If you allow **param** to default, and the Cray computer system does not have compressed indexing capabilities, the test does not run properly.

† CRAY X-MP EA (X-mode) and CRAY X-MP computer systems only.

To ensure that the test automatically disables the vector compressed indexing instructions, do the following:

1. Rename **olcmx** to **olcmxa**.

2. Create a shell script called **olcmx**.

3. Enter the following information into the **olcmx** shell script:

   olcmxa param 400000001 $*

4.4.3 olibz

To run **olibz**† (on-line instruction buffer test), you must set **cput** (the DIB option to set the CPU type), as follows:

Value	CPU Type
10 (default)	CRAY X-MP/1
20	CRAY X-MP/2
40	CRAY X-MP EA (X-mode) CRAY X-MP/4

The default value for **cput** is 10, indicating a CRAY X-MP/1 computer system. If you allow **cput** to default, and you attempt to run **olibz** on a mainframe other than the CRAY X-MP/1, the test executes but it generates invalid error information. Therefore, ensure that the appropriate **cput** value is set.

To ensure that the test automatically selects the appropriate **cput** value, do the following:

1. Rename **olibz** to **olibz4** or **olibz2**.

2. Create a shell script called **olibz**.

3. Enter the following information into the **olibz** shell script:

   olibz4 cput 40 $*

   or

   olibz2 cput 20 $*

† CRAY X-MP EA (X-mode) and CRAY X-MP computer systems only.

4.5 TEST TERMINATION

A test stops under the following conditions:

- The test successfully completes the maximum number of passes (**maxp** *n*).
- The test reaches the specified CPU time (**cputime** *h*:*m*:*s*) or elapsed (wall-clock) time (**time** *h*:*m*:*s*).
- The test detects the maximum number of errors (**maxerr** *n*). If **maxerr** is set to a value greater than 1, **stop** (DIB option to set stop condition bits) must be set to **0** (continue on error). Error reports are automatically sent to **stdout** (standard output device), but they can be redirected to an error file.
- The test detects an error and **stop** is set to **1** (stop on error).
- The **help** option is entered with a test name, help information is written to **stdout**, and the test terminates.
- The monitor or test detects an error in a command line entry and writes a message to **stderr** (standard error device). Only the first error detected is reported.

4.6 TEST EXAMPLES

The following example executes **olvrx** with two DIB options set: **secs 3** executes test sections 0 and 1; **stop 0** directs the program to continue on error. To exit a continue on error, enter the **kill**(1) command to terminate test execution.

Example:

```
olvrx secs 3 stop 0
```

The following example executes **olvrx** with two DIB options set: **secs 3** executes test sections 0 and 1; **data 205:77** stores the value O'77 at location O'205.

Example:

```
olvrx secs 3 data 205:77
```

The following example executes **olvrx** with one DIB option: **secs 1** executes test section 0.

Example:

```
olvrx secs 1
```

The following example executes *test* in CPU c, sets the maximum error limit to 3, and redirects the output to ***test*.logc**.

Example:

```
test cpu c maxerr 3 > test.logc
```

The following example displays test results from ***test*.logc** one page at a time (press the RETURN key to display the next page).

Example:

```
pg test.logc
```

The following example executes **olcmx** in CPU b for 500,000 passes, starting at pass 500,000. Output is redirected to **olcmx.log**. The **nohup**(1) command allows the program to continue executing after you log off the system. You can later log on to check the test's progress. The ampersand (&) causes the entire command to execute in the background, so that another prompt is immediately displayed and you can continue to use the system.

Example:

```
nohup olcmx cpu b maxp 1000000 rpass 500000 > olcmx.log &
```

The following example shows the help information that is displayed if **help** is entered with a test name.

Example:

```
olaht help
```

Help display:

```
olaht help
olaht [help] [chkpnt mode] [cpu x] [cputime h:m:s] [data x:y] [maxerr n]
      [maxp n] [time h:m:s] [+/-verbose] [words n] [dib x]
 chkpnt mode   - Checkpoint mode:  none, first, or all.  (Default: first)
 cpu x         - Selects CPU x.  (Default: a)
 cputime h:m:s- Set amount of CPU time to execute.
 data x:y      - Stores data y at diagnostic location x before the
                 diagnostic is started.
 maxerr n      - Sets maximum number of errors.  (Default: 1)
 maxp n        - Sets maximum number of passes.  (Default: O'1000)
 time h:m:s    - Set amount of wall clock time to execute.
 +/-verbose    - Send (+verbose)/do not send (-verbose) informational
                 messages to output.  (Default: -verbose)
 words n       - Allocates x words for Central Memory testing.
                 MFRST (sta) and MLAST (lim) are set with the appropriate
                 values.
 dib x         - Sets the DIB location to x.
                 Refer to the individual test to determine which
                 DIBs are available for the test.
 NOTE: Actual results of setting a DIB location are test-dependent.
```

The following example shows the output that is displayed when the test is run with all default values.

Example:

```
olsr3
```

Output:

```
olsr3
olsr3: started running in cpu A on Thu Dec 17 09:10:05 1987
olsr3 reached maximum pass limit with 1000 passes and 0 errors
on Thu Dec 17 09:10:05 1987
```

The following example shows the output that is displayed if **+verbose** is specified and **maxp** reaches 10.

Example:

```
olsr3 +verbose maxp 10
```

Output:

```
olsr3 +verbose maxp 10
olsr3: started running in cpu A on Thu Dec 17 09:10:48 1987
olsr3: pass =          1, error =          0  Thu Dec 17 09:10:48 1987
olsr3: pass =          2, error =          0  Thu Dec 17 09:10:48 1987
olsr3: pass =          3, error =          0  Thu Dec 17 09:10:48 1987
olsr3: pass =          4, error =          0  Thu Dec 17 09:10:48 1987
olsr3: pass =          5, error =          0  Thu Dec 17 09:10:48 1987
olsr3: pass =          6, error =          0  Thu Dec 17 09:10:48 1987
olsr3: pass =          7, error =          0  Thu Dec 17 09:10:48 1987
olsr3: pass =         10, error =          0  Thu Dec 17 09:10:48 1987
olsr3 reached maximum pass limit with 10 passes and 0 errors
on Thu Dec 17 09:10:48 1987
```

The following example shows the output that is displayed if **olsr3** is run for 2 minutes (CPU time) in CPU c only.

Example:

```
    olsr3 cpu c cputime 2:00
```

Output:

```
olsr3 cpu c cputime 2:00
olsr3: started running in cpu C on Fri Dec 4 09:11:45 1987
olsr3 reached maximum cputime limit with 1114656 passes and 0 errors
on Fri Dec 4 09:13:49 1987
```

The following example shows the output that is displayed if **maxerr** reaches 1 (default).

Example:

```
    oltrb
```

Output:

```
oltrb
oltrb started running in cpu A at Wed Jan  6 15:30:34 1988
oltrb: pass =            0, error =          1  Wed Jan 6 15:30:34 1988
oltrb: restart file written to A55663-oltrb
NAME        <   630> = 'TRB     '
REV         <   632> = 'X3.0    '
DATE        <   634> = '12/07/87'
MODES       <   636> = 'TB RU   '
MTRT        <   642> = 16                 000000 000000 000000 000016
SECS        <   241> = 7654321            000000 000000 000037 054321
```

Output (continued):

```
PASS           <    64> = 0                    000000 000000 000000 000000
STOP           <    66> = 1                    000000 000000 000000 000001
ERROR          <    63> = 1                    000000 000000 000000 000001
ERA            <    65> = 1576                 000000 000000 000000 001576
ACT            <    61> = 1777777777777777777  177777 177777 177777 177777
EXP            <    62> = 1                    000000 000000 000000 000001
DIF            <    60> = 1777777777777777776  177777 177777 177777 177776
CF             <    67> = 0                    000000 000000 000000 000000
IBUF           <  1440> = 1777777777777777777  177777 177777 177777 177777
IBUF   + 0001  <  1441> = 1777777777777777777  177777 177777 177777 177777
  .
  .
  .
IBUF   + 0077  <  1537> = 1777777777777777777  177777 177777 177777 177777
OBUF           <  1540> = 1777777777777777777  177777 177777 177777 177777
OBUF   + 0001  <  1541> = 1777777777777777777  177777 177777 177777 177777
  .
  .
  .
OBUF   + 0077  <  1637> = 1777777777777777777  177777 177777 177777 177777
SAVA0          < 27616> = 0000000000000000001  000000 000000 000000 000001
SAVA0  + 0001  < 27617> = 0000000000000000100  000000 000000 000000 000100
SAVA0  + 0002  < 27620> = 0000000000000000076  000000 000000 000000 000076
SAVA0  + 0003  < 27621> = 0000000000000000077  000000 000000 000000 000077
SAVA0  + 0004  < 27622> = 0000000000000034772  000000 000000 000000 034772
SAVA0  + 0005  < 27623> = 0000000000000037035  000000 000000 000000 037035
SAVA0  + 0006  < 27624> = 0000000000000037027  000000 000000 000000 037027
SAVA0  + 0007  < 27625> = 0000000000000000001  000000 000000 000000 000001
SAVBR          < 30640> = 0000000000000001576  000000 000000 000000 001576
SAVBR  + 0001  < 30641> = 0000000000000001311  000000 000000 000000 001311
SAVBR  + 0002  < 30642> = 0000000000000001576  000000 000000 000000 001576
SAVBR  + 0003  < 30643> = 0000000000000001471  000000 000000 000000 001471
SAVBR  + 0004  < 30644> = 0000000000000036711  000000 000000 000000 036711
SAVBR  + 0005  < 30645> = 0000000000000000000  000000 000000 000000 000000
SAVS0          < 27626> = 0000000000000000000  000000 000000 000000 000000
SAVS0  + 0001  < 27627> = 0000000000000000000  000000 000000 000000 000000
SAVS0  + 0002  < 27630> = 1777777777777777777  177777 177777 177777 177777
SAVS0  + 0003  < 27631> = 0000000000000000004  000000 000000 000000 000004
SAVS0  + 0004  < 27632> = 0000000000000000000  000000 000000 000000 000000
SAVS0  + 0005  < 27633> = 0000000000000000102  000000 000000 000000 000102
SAVS0  + 0006  < 27634> = 0000000000000000001  000000 000000 000000 000001
SAVS0  + 0007  < 27635> = 0000000000000000001  000000 000000 000000 000001
SAVVL          < 30636> = 0000000000000000003  000000 000000 000000 000003
SAVVM          < 30637> = 0000000000000000000  000000 000000 000000 000000
```

Output (continued):

```
SAVTR        < 30740> = 1777777777777777777  177777 177777 177777 177777
SAVTR + 0001 < 30741> = 1777777777777777777  177777 177777 177777 177777
   .
   .
   .
SAVTR + 0077 < 31037> = 1777777777777777777  177777 177777 177777 177777

The first address (FADD) of the diagnostic is 40a
oltrb reached maximum error limit with 0 passes and 1 errors
on Wed Jan  6 15:30:35 1988
```

4.7 TEST MESSAGES

Each test sends messages to **stdout** (standard output device) by default or to a file when UNICOS output redirection is indicated on the command line. When a test detects an error, the following information is displayed:

- DIBs
- Absolute addresses of the DIBs
- DIB values in word and parcel formats

The following error messages are sent to **stderr** (standard error device):

test: Illegal argument x.
Argument x is invalid. Correct and rerun.

test: Error selecting cpu x.
CPU x is unavailable. Contact your CRI representative.

test: Error allocating memory:
number of words = *n*, error = 0.
The test cannot allocate memory. Decrease the amount of memory requested by the **words** *n* option, or regenerate the diagnostic, and rerun. If the problem persists, contact your CRI representative.

test: Cannot write restart file. errno = *n*.
The test cannot write a restart file. Contact your CRI representative.

4.8 DIAGNOSTIC MEMORY IMAGE FOR MAINTENANCE TESTS

Figure 4-1 shows a sample memory image of a diagnostic that is executing. The diagnostic test is relocated to start at the first address (FADD) of the test. FADD must be subtracted from the error address if the diagnostic fails. After an error occurs, FADD is displayed in the following format:

The first address (FADD) of the diagnostic is xa

The value *x* is determined by the length of the on-line monitor program.

The on-line maintenance tests call the following monitor routines:

Routine	Description
UERROR()	The test calls the UERROR() routine when an error is detected. The monitor dumps the DIB and examines a DIB macro at the end of the diagnostic for memory areas to be dumped.
UPASS()	The test calls the UPASS() routine on each successful pass.

If an error is detected, the following occurs:

1. The test does the following:

   - Creates a restart file

   - Saves the CPU registers using the SAVEREG macro, defined in the common deck OLMAC

   - Calls the monitor error function routine, UERROR()

   - Restores the CPU registers using the RESTORE macro, defined in the common deck OLMAC

   For additional information on the restart file, refer to the following system calls: **chkpnt**(2) and **restart**(2). The SAVEREG and RESTORE code is assembled into the on-line maintenance test, but the memory required to save the registers is allocated to the following monitor arrays: SAVA0, SAVBR, SAVS0, SAVTR, SAVV0, SAVVL, and SAVVM.

2. The system produces a core dump of the diagnostic test area.

	Location Names	Memory Image
Base Address		
	UERROR() UPASS()	Monitor program (**olmon**)
	SAVA0 SAVBR SAVS0 SAVTR SAVV0 SAVVL SAVVM	Data area for storing register data
FADD	START () DIB SAVEREG RESTORE	Diagnostic program
mfrst		Memory allocated for a memory test
mlast		C library routines
		Unused area
		System stack
Limit Address		

Figure 4-1. Sample Diagnostic Memory Image

5. DOWN-DEVICE PROGRAMS

The down-device programs provide on-line CPU and peripheral testing. The hardware is removed from normal system operations and can be accessed and exercised only by the down-device programs.

This section describes the following programs:

Program	Description
donut	On-line disk maintenance program
oldmon†	Down CPU monitor
unitap	On-line magnetic tape test

5.1 donut

The **donut** program is an interactive, menu-driven diagnostic program for testing and maintaining DD-10, DD-19, DD-29, DD-39, DD-40, and DD-49 disk drives. The **donut** program cannot be run off-line.

The **donut** program can be used to perform the following functions:

- Buffer testing†
- Error correction code (ECC) testing††
- Flaw table maintenance
- Formatting
- ID verification††
- Surface analysis

The subsections that follow describe the following topics:

- Disk selection
- Disk mode
  - System mode
  - Maintenance mode
- Warnings and messages
- Menu displays
- Program execution
- Menus
- Program execution examples

† Multiple-CPU Cray computer systems only
†† Not available for DD-19 or DD-29 disk drives

5.1.1 DISK SELECTION

The **donut** program can test only one disk at a time. However, multiple copies of **donut** can be executed simultaneously to test different disk drives.

To access a disk, **donut** uses the same logical device name as that assigned during system configuration. To select the disk to be exercised, define the logical device name by doing one of the following:

- Enter **dev** from the Main menu (refer to subsection 5.1.6.3, Commands to Set Arguments)
- Enter **a** from the Parameter menu (refer to subsection, 5.1.13, Parameter Menu)

The **donut** program attempts to open and retrieve iobuf information for the specified device, to determine whether the specified logical device name is valid.

If the logical device name is valid, **donut** determines the device type and adjusts the other arguments accordingly. As a precaution, **donut** sets the initial cylinder argument to point to a scratch cylinder. **donut** reads and verifies the disk flaw tables for the device, and displays an appropriate message if any abnormalities are detected.

If the logical device name is invalid, **donut** does not accept disk requests and the device argument is set as follows:

```
* none *
```

Reenter a valid logical device name and continue.

5.1.2 DISK MODE

A disk in the system configuration can be in one of the following modes:

Mode	Description
System	UNICOS system routines and all user jobs can access the disk
Maintenance	Only UNICOS system routines and **donut** can access the disk

The current mode is displayed under the MODE heading in the argument banner of various menus (refer to subsection 5.1.4, Menu Displays).

To change the mode, do the following:

1. Select the mode by doing one of the following:

   - Enter **mode** from the Main menu (refer to subsection 5.1.6.3, Commands to Set Arguments)

   - Enter t from the Parameter menu (refer to subsection 5.1.13, Parameter Menu)

//

WARNING

The **donut** program can write to any of the cylinders on a disk. Therefore, device labels and flaw tables are vulnerable to accidental destruction. It is recommended that writes and surface analysis not be performed on the CE cylinders that contain the flaw tables (typically, cylinder 0 and the second-to-last cylinder on a device) unless absolutely necessary, and then only if backup procedures are used. Before writing to a disk, **donut** displays a message that flaw table information will be destroyed on those cylinders that contain information.

//

5.1.2.1 System mode

In system mode, **donut** and other user jobs have equal access to the disk. The following operations are supported:

- **donut** can read from and write to CE cylinders only
- **donut** can perform ID verification (except on DD-19s and DD-29s)
- Flaw tables can be updated

5.1.2.2 Maintenance mode

In maintenance mode, only UNICOS and **donut** requests can access the disk. All **donut** functions are valid.

If a maintenance mode function is requested while the disk is in system mode, the function aborts and **donut** displays the following message:

```
     *** DIAGNOSTIC TASK ERROR CODE ***
1 - Device not in Maintenance mode
```

5.1.3 WARNINGS AND MESSAGES

The **donut** program displays various warnings and messages. For example, the following warning is displayed if you are about to overwrite the User Flaw Table in **donut**'s area of central memory:

```
-------------------------------------------------------

                        WARNING

        USER flaw table in memory will be altered.
                   Enter go to continue
              or enter anything else to abort.

-------------------------------------------------------
```

If an invalid command is entered, an error message is displayed and the menu from which the command was entered is redisplayed. If an invalid argument is entered, an informative message is displayed. After some of the informative messages, the following prompt is displayed:

```
---> Enter anything to continue <---
```

Some of the **donut** messages require a response. For example, the following message requires a response to ensure that read, write, and surface analysis operations are performed on only selected sectors.

```
                 L I M I T S   C H E C K

          -------------------------------------------
           Check CYLINDER, HEAD and SECTOR limits.
                   Enter go to initiate.
             Enter any other character to abort.
          -------------------------------------------
```

5.1.4 MENU DISPLAYS

At the top of the various menus is the argument banner displaying the arguments used in the program. A sample argument banner is as follows:

```
=============================================================== 09:50:10 =
=     DEVICE   CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE               =
=    --------  ---------  -----  -------  ----  ----  ------              =
=    * none *    0 - 0    0 - 0   0 - 0     0   none  ~~~~~~              =
==========================================================================
```

By default, arguments are displayed in decimal. The cylinder, head, and sector values must be entered in decimal unless otherwise indicated.

To generate an octal display, enter **oct** from any of the menus (enter **dec** to return to a decimal display).

If you generate an octal display, the following applies:

- The argument banner displays the heading **(OCTAL)** to the left of the arguments
- The cylinder, head, and sector information is entered and displayed in octal

5.1.5 PROGRAM EXECUTION

The **donut** program resides in **/ce/bin directory.** To execute **donut,** enter the following:

/ce/bin/donut

The initial **donut** screen display is as follows:

```
Welcome to X-MP UNICOS DONUT

        Version 2.0


   ---> Enter anything to continue <---
```

To continue, press any key. The program displays the Main menu. From the Main menu, you can get to various other menus. Menu commands are not case sensitive. They can be entered in uppercase or lowercase. In this document, the menus show commands in uppercase; however, the descriptions show them in lowercase and **bold**, according to UNICOS conventions.

The menu structure is as follows:

Main Menu Command	Description
a	Displays disk information
b	Displays Buffer Utility menu
e	Displays Error Utility menu

Buffer Utility menu (main menu command **b**):

Command	Description
a	Displays Write Buffer menu
b	Displays Read Buffer menu

Error Utility menu (main menu command **e**):

Command	Description
a	Displays Error Table menu
b	Displays Error Log menu

Error Table menu (command **a**):

- **a** Adds the displayed error to the Found Flaw Table
- **b** Adds all errors to the Found Flaw Table
- **d** Deletes the displayed error record from the Error Table
- **e** Prints the error record to a file

Error Log menu (command **b**):

- **a** Adds top entry to the Found Flaw Table
- **b** Adds all entries to the Found Flaw Table
- **c** Prints the entire error log
- **e** Deletes all error log entries

Main Menu Command	Description
f	Displays Formatting menu

Command	Description
b	Displays argument banner with warning. Enter **go** to format IDs with flaw handling.
c	Displays argument banner with warning. Enter **go** to format IDs with no flaw handling.
e	Displays Examine Data Buffer menu
f	Verifies track IDs using the User Flaw Table
g	Verifies track IDs without using the User Flaw Table
z	Displays Parameter menu

Main Menu Command	Description
s	Displays Surface Tests menu

Command	Description
a	Displays Write Data menu
b	Displays Read Data and Compare menu
c	Displays argument banner with warning. Enter **go** to perform a read exercise.
d	Displays Surface Analysis menu
e	Displays Examine Data Buffer menu
f	Displays argument banner with warning. Enter **go** to execute a read absolute operation.
g	Displays argument banner with warning. Enter **go** to execute a write current data buffer operation.
z	Displays Parameter menu

Main Menu Command	Description
t	Displays Flaw Table Utility menu

Command	Description
a	Displays Factory Flaw Table menu
b	Displays User Flaw Table menu
c	Displays System Flaw Table menu
d	Displays Found Flaw Table menu

w	Executes the Error Correction Code test
z	Displays Parameter menu

Command	Description
a	Sets logical device name
b	Sets cylinder limits
c	Sets head limits
d	Sets sector limits
e	Sets diagnostic flags
t	Toggles disk mode

q	Exits **donut**

In addition, there are various commands that can be entered from the Main menu or various other menus. These commands are described in the following subsections:

- Subsection 5.1.6.1, Commands to Display Submenus
- Subsection 5.1.6.2, Commands to Select Display Format
- Subsection 5.1.6.3, Commands to Set Arguments
- Subsection 5.1.6.4, Commands to Display the Data Buffer
- Subsection 5.1.6.5, Commands to Display Flaw Table Menus
- Subsection 5.1.6.6, Commands to Change the Data Buffer
- Subsection 5.1.6.7, Commands to Change the Type of Write Command Used
- Subsection 5.1.6.8, Commands to Display Commands List

5.1.6 MAIN MENU

Figure 5-1 shows **donut**'s Main menu.

```
================================================================ 09:50:10 =
=     DEVICE   CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE              =
=    --------  ---------  -----  -------  ----  ----  ------             =
=    * none *    0 - 0    0 - 0   0 - 0    0    none  ~~~~~~             =
==========================================================================

              D I S K   O N L I N E   U T I L I T Y  (DONUT)

                A  -  Disk Information
                B  -  Buffer tests
                E  -  Review Errors
                F  -  Formatting and ID analysis
                S  -  Surface tests
                T  -  Flaw Table Utility
                W  -  Error Correction Test
                Z  -  Reset Parameters
                Q  -  Exit DONUT - (Quit)

              Enter command ==>
```

Figure 5-1. Main Menu for **donut**

5.1.6.1 Commands to display submenus

Table 5-1 lists the Main menu commands, which are used to do the following:

- Display disk information (enter **a** from the Main menu or enter **info** from any menu)
- Display various submenus
- Execute the Error Correction Code test

Table 5-1. Main Menu Commands

Command	Description
a	Displays disk information
b	Displays the Buffer Utility menu
e	Displays the Error Utility menu
f	Displays the Formatting menu
s	Displays the Surface Tests menu
t	Displays the Flaw Table Utility menu
w	Executes the Error Correction Code test
z	Displays the Parameter menu
q	Quit; exits **donut**.

5.1.6.2 Commands to select display format

The following commands for selecting the display format can be entered from any menu:

Command	Description
oct	Displays the cylinder, head, and sector information in octal
dec	Displays the cylinder, head, and sector information in decimal (default)

5.1.6.3 Commands to set arguments

Table 5-2 lists the commands to set arguments from the Main menu or any of the subsequent menus (except the data pattern menus). Alternatively, you can set arguments by entering **z** (reset parameters) from the Main menu or various other menus.

Table 5-2. Commands to Set Arguments

Command	Description
cyl	Sets the cylinder range
dev	Sets the logical device name
flags	Sets diagnostic flags
hed	Sets the head range
mode	Sets the disk mode to **system** or **maintenance**
sec	Sets the sector range

5.1.6.4 Commands to display the data buffer

The **donut** program keeps a record of the 1-track buffer used during the last disk operation. When **donut** writes data or IDs, the buffer contains data for the last track written. When **donut** reads data or IDs or performs surface analysis, the buffer contains data for the last track read. The buffer is reused during the next disk operation.

To display the data buffer from any menu, enter the following command:

data

The data buffer can also be displayed by entering **e** from the Formatting menu (subsection 5.1.9) or the Surface Tests menu (subsection 5.1.10).

5.1.6.5 Commands to display flaw table menus

To display a flaw table without going through the Flaw Table Utility menu, enter one of the following commands from the Main menu or any of the flaw table menus, as appropriate:

Command	Description
fac	Factory Flaw Table menu
fnd	Found Flaw Table menu
sys	System Flaw Table menu
usr	User Flaw Table menu

For additional information on flaw tables, refer to subsection 5.1.11, Flaw Table Utility Menus.

5.1.6.6 Commands to change the data buffer

To change the contents of the **donut** data buffer, the following commands can be used:

Command	Description
clr	Fills all sectors of the data buffer selected in the sectors section of the argument banner with 0's
fill	Fills all sectors of the data buffer selected in the sectors section of the argument banner with 1's

5.1.6.7 Commands to change the type of write command used

To change the type of write command used during write operations to the disk, the following commands can be used. These commands need only be used for DD-40 type disks.

Command	Description
wrt	Sets the write command to perform a write (function code 4) during write operations. The write function is the default.
fill	Sets the write command to issue a write immediate (function code 22 octal) during write operations. This function code is valid only for DD-40 disks. It may be used when a controller releases control after all data is received but before the data is written to the disk and an error occurs when the remaining data is finally written.

5.1.6.8 Commands to display commands list

Entering the **help** command displays a list of global commands that can be entered from any menu:

```
Parameters changes:
DEV  - Change DEVICE Parameter
CYL  - Change CYLINDER Parameter Limits
HED  - Change HEAD Parameter Limits
SEC  - Change SECTOR Parameter Limits
MODE - Toggle Disk MODE (System/Maint.)

Flaw tables:
FAC  - Factory Flaw Table     SYS  - System Flaw Table
FND  - Found Flaw Table       USR  - User Flaw Table

Miscellaneous:
CLR  - Clear Data Buffer To Zeros
DATA - Display Data Buffer
FILL - Fill Data Buffer With Ones
HELP - Display This Help Information
INFO - Display Disk Information
MAIN - Main Menu
WRT  - Select Write Function (WRT=4)
WRIM - Select Write Immediate Function (WRTIM=22 oct)
```

5.1.7 BUFFER UTILITY MENU

Figure 5-2 shows the Buffer Utility menu (not applicable to DD-19 or DD-29 disk drives). Table 5-3 lists the Buffer Utility menu commands. These commands display the following submenus:

- Write Buffer menu
- Read Buffer menu

From the submenus, you can execute a write or read function in the controller's 16-parcel buffer. To exercise the basic Cray-to-disk communication path, put the disk in maintenance mode and execute a write followed by a read and compare (if the disk is in system mode, other jobs may be using the buffer and the test may not be effective).

```
=================================================================== 09:54:01 =
=      DEVICE   CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE                 =
=     --------  ---------  -----  -------  ----  ----  ------                =
=     49-2-24A   10 - 20   0 - 7   0 - 41    2   DD49  Maint.                =
==============================================================================

                          B U F F E R   U T I L I T Y

                  A  -  Write Buffer
                  B  -  Read Buffer and compare
                  R  -  Return
```

Figure 5-2. Buffer Utility Menu

Table 5-3. Buffer Utility Menu Commands

Command	Description
a	Displays the Write Buffer menu, from which you can select a data pattern to perform a 16-parcel write to the buffer
b	Displays the Read Buffer menu, from which you can compare actual data to the selected data pattern
r	Returns to previous menu

Figure 5-3 shows the Write Buffer menu. Figure 5-4 shows the Read Buffer menu. Table 5-4 lists the commands for the Write Buffer and Read Buffer menus.

```
================================================================ 09:53:52 =
=      DEVICE    CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE             =
=     --------   ---------  -----  -------  ----  ----  ------            =
=     49-2-24A   10 - 20    0 - 7  0 - 41    2    DD49  Maint.            =
===========================================================================

                        W R I T E   B U F F E R

   0  -  All zeros                         1  -  All ones
   A  -  Addressing pattern                B  -  Bump
   C  -  Alternating 0,1 pattern
   E  -  Hole                              F  -  Fixed data
   S  -  Sequential data                   T  -  Peak shift
   Z  -  Reset Parameters                  R  -  Return

            Input the data pattern ==>
```

Figure 5-3. Write Buffer Menu

```
================================================================ 09:54:07 =
=      DEVICE    CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE             =
=     --------   ---------  -----  -------  ----  ----  ------            =
=     49-2-24A   10 - 20    0 - 7  0 - 41    2    DD49  Maint.            =
===========================================================================

                        R E A D   B U F F E R

   0  -  All zeros                         1  -  All ones
   A  -  Addressing pattern                B  -  Bump
   C  -  Alternating 0,1 pattern
   E  -  Hole                              F  -  Fixed data
   S  -  Sequential data                   T  -  Peak shift
   Z  -  Reset Parameters                  R  -  Return

            Input the data pattern ==>
```

Figure 5-4. Read Buffer Menu

Table 5-4. Commands for the Write Buffer and Read Buffer Menus

Command	Description
0	All 0's
1	All 1's
a	Addressing pattern in a Cray word: Parcel / Value 0 Cylinder number 1 Head number 2 Sector number 3 Word number
b	Bump. This is a repeating 4-word pattern: Word / Octal / Hexadecimal 0 0525252525242104252525 5555 5555 1111 5555 1 0525250421052525252525 5555 1111 5555 5555 2 0104212525252525252525 1111 5555 5555 5555 3 0525252525252525210421 5555 5555 5555 1111
c	Alternating 0's and 1's. This is a repeating 2-word pattern: Word / Octal / Hexadecimal 0 1252525252525252525252 AAAA AAAA AAAA AAAA 1 0525252525252525252525 5555 5555 5555 5555
e	Hole. This is a repeating 4-word pattern: Word / Octal / Hexadecimal 0 0525252525256735652525 5555 5555 7777 5555 1 0525356735252525252525 5555 7777 5555 5555 2 0735672525252525252525 7777 5555 5555 5555 3 0525252525252525273567 5555 5555 5555 7777
f	Fixed data. This is a 1-word, user-input pattern.

Table 5-4. Commands for the Write Buffer and Read Buffer Menus (continued)

Command	Description
s	Sequential data pattern: Word / Description 0 Random number *n* Word 0 + *n*
t	Peak shift. This is a repeating 3-word pattern: Word / Octal / Hexadecimal 0 0631466735667356663146 6666 DDDD BBBB 6666 1 1567355673554631556735 DDDD BBBB 6666 DDDD 2 1356733146333567335673 BBBB 6666 DDDD BBBB
z	Displays the Parameter menu
r	Return to previous menu

5.1.8 ERROR UTILITY MENU

Figure 5-5 shows the Error Utility menu. Table 5-5 lists the Error Utility menu commands. These commands display the following submenus:

- Error Table menu
- Error Log menu

```
============================================================ 09:54:21 =
=      DEVICE   CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE         =
=     --------  ---------  -----  -------  ----  ----  ------        =
=     49-2-24A   10 - 20   0 - 7   0 - 41    2   DD49  Maint.        =
=======================================================================

                        E R R O R   U T I L I T Y

              A  -  Review details of the latest Error Table
              B  -  Review Error Log
              R  -  Return
```

Figure 5-5. Error Utility Menu

Table 5-5. Error Utility Menu Commands

Command	Description
a	Displays an error record and the Error Table menu
b	Displays the error log and the Error Log menu
r	Returns to previous menu

5.1.8.1 Error Table menu

When a disk request generates an error (such as a seek, read, or write error), the IOS sends **donut** an error record containing information such as function, address, status, and syndromes. The **donut** program interprets these records and stores them in the Error Table. If no error is detected in the disk function, no error record is returned. The error table is only valid for the latest call-in-error and is overwritten during each disk function call.

Figure 5-6 shows an error record for a DD-39 read time-out error, and the Error Table menu. Table 5-6 lists the Error Table menu commands.

```
~~~~~~~~~~~~~~~~ E R R O R   R E C O R D      1 of  1 (octal data) ~~~~~~~~~~~
 Dev Type     000004  IOP number    0001  Channel #    000032 Major Err    Read
 Expect CYL   001511  Fin Err St Unrecov  Expect HED   000001 Expect SEC   000017
 Disk Funct LMA Rg1   Retry Cnt   000000  Orig Cntlr   007611 Orig GenSt   041600
 Sel Stat 0   001600  Sel Stat 1  103200  Sel Stat 2   000200 Sel Stat 3   070200
 Sel Stat 4   000200  Unit numbr  000000  Offset dir   None   Err Correc   Is off
 C3 Cor Msk   000000  C3 Cor Off  000000  C2 Cor Msk   000000 C2 Cor Off   000000
 C1 Cor Msk   000000  C1 Cor Off  000000  C0 Cor Msk   000000 C0 Cor Off   000000
 Expect LMA   000000  Actual LMA  000000  Fin ctl st   007611 Fin gen st   041600
 Fin Dsk Fn Unknown   Orig Recov DN ~set  Finl Recov Unknown
 C3 Syn upr   000000  C3 Syn low  000000  C2 Syn upr   000000 C2 Syn low   000000
 C1 Syn upr   000000  C1 Syn low  000000  C0 Syn upr   000000 C0 Syn low   000000
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

              A  -  Add THIS error to FOUND Flaw Table
              B  -  Add ALL errors to FOUND Flaw Table
              D  -  Delete THIS error record
              E  -  Erase ALL error records
              R  -  Return
             Enter Command or Error Number ==>
```

Figure 5-6. Error Table Menu

Table 5-6. Error Table Menu Commands

Command	Description
a	Adds the displayed error record to the Found Flaw Table
b	Adds all error records in the Error Table to the Found Flaw Table
d	Deletes the displayed error record from the Error Table
e	Creates a file called **ERRECRD** in the current directory. The error record is saved in this file.
r	Returns to previous menu

5.1.8.2 Error Log menu

The **donut** program maintains a log of all disk errors detected during a session. For each error, the log contains an error summary with the time, device, address, function, and pattern. The Error Log is deleted if you exit or abort **donut**, or if you enter **e** from the Error Table menu. Figure 5-7 shows a typical Error Log display and the Error Log menu. Table 5-7 lists the Error Log menu commands.

```
                         E R R O R   L O G                       LAST=   17
---------------------------------------------------------------------------
 TIME     NUM LOG DEV  CYL HEAD SEC   CHANNEL    ERROR DISK FUNC   TEST
-------- --- -------- --- ---- --- -----------  ----- ---------  -------
09:58:56   1 49-2-24A  10   0     0 -- B1 -- --  Read  LMA Reg 1 Compare
09:58:58   2 49-2-24A  11   0     0 -- B1 -- --  Read  LMA Reg 1 Compare
09:59:02   3 49-2-24A  12   0     0 -- B1 -- --  Read  LMA Reg 1 Compare
09:59:04   4 49-2-24A  13   0     0 -- B1 -- --  Read  LMA Reg 1 Compare
09:59:24   5 49-2-24A  10   0     0 -- B1 -- --  Read  LMA Reg 1 Compare
09:59:25   6 49-2-24A  11   0     0 -- B1 -- --  Read  LMA Reg 1 Compare
09:59:28   7 49-2-24A  12   0     0 -- B1 -- --  Read  LMA Reg 1 Compare
09:59:31   8 49-2-24A  13   0     0 -- B1 -- --  Read  LMA Reg 1 Compare
10:08:19   9 49-2-24A  10   0     0 -- B1 -- --  Read  LMA Reg 1 Compare
10:08:20  10 49-2-24A  11   0     0 -- B1 -- --  Read  LMA Reg 1 Compare
             A  -  Add TOP entry to FOUND Flaw Table
             B  -  Add ALL entries to FOUND Flaw Table
             C  -  Print out entire log
             E  -  Erase ALL log entries
             R  -  Return
              Enter Command or Entry Number ==>
```

Figure 5-7. Error Log Menu

Table 5-7. Error Log Menu Commands

Command	Description
a	Adds the top entry in the Error Log to the Found Flaw Table. Duplicate entries are skipped.
b	Adds all entries in the Error Log to the Found Flaw Table. Duplicate entries are skipped.
c	Creates a file called **DONULOG** in the current directory. The Error Log is saved in this file.
e	Deletes all Error Log entries.
r	Returns to previous menu.

5.1.9 FORMATTING MENU

Figure 5-8 shows the Formatting menu. Table 5-8 lists the Formatting menu commands. These commands display the following submenus:

- Examine Data Buffer menu
- ID Analysis Results menu
- Parameter menu

```
============================================================= 09:57:18 =
=      DEVICE   CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE           =
=      --------  ---------  -----  -------  ----  ----  ------         =
=     49-2-24A   10 - 20   0 - 7   0 - 41    2   DD49  Maint.          =
========================================================================

                           F O R M A T T I N G

                B  -  Format with USER Flaw Table
                C  -  Format with NO flaw handling
                E  -  Examine Buffer
                F  -  Verify IDs with USER flaw table
                G  -  Verify IDs with NO flaw handling
                Z  -  Reset Parameters
                R  -  Return

              Enter Command ==>
```

Figure 5-8. Formatting Menu

Table 5-8. Formatting Menu Commands

Command	Description
b	Uses the User Flaw Table to format IDs
c	Formats IDs without using the User Flaw Table (**donut** assumes there are no flaws)
e	Displays the Examine Data Buffer menu
f	Reads track IDs and does ID verification based on the assumption that IDs were formatted with the User Flaw Table (DD-10, DD-39, DD-40, and DD-49 disk drives only)
g	Reads track IDs and does ID verification based on the assumption that IDs were formatted without the User Flaw Table (DD-39, DD-40, and DD-49 disk drives only)
z	Displays the Parameter menu, from which you can set the arguments in the argument banner
r	Returns to previous menu

5.1.9.1 Logical address of the sector ID

Formatting is performed on a track basis, using spare sectors if applicable and the User Flaw Table if specified. Only DD-10, DD-39, DD-40, and DD-49 disks have a User Flaw Table, and only DD-39 and DD-49 disks have spare sectors.

During formatting, the logical address is written into the sector ID field. For flawed sectors, a flawed ID is written into this field. The formatting routine does the following:

- Uses the **slip** argument to calculate the logical address
- Determines whether the User Flaw Table is to be used

When the logical address is written into the sector ID field, the type of disk drive determines how the data is affected, as follows:

Disk	Logical Address is Written to Sector ID
DD-10/39/40/49s	Data in the sector is not affected when the logical address is written to the ID field.
DD-19/29s	The entire data area is corrupted when the logical address is written to the ID field. **donut** does not automatically write to the newly formatted sectors. If a read is attempted following a formatting operation, an unrecoverable error occurs. Therefore, after completing a formatting operation, write data before performing a read.

5.1.9.2 Position field of the sector ID (DD-10s and DD-40s only)

DD-40 disk drives can contain the following types of defects:

Defect Type	Description
Hideable	Contains a defect that resides in a 16-byte field called the defect address, which is skipped during all disk operations. The defect address is written to the position field (POS) of the sector ID.
Unhideable	Contains either a defect that spans more than one address or multiple defects. These defects are not hidden because only one defect address is skipped during all disk operations. The sector ID is set to all 1's to indicate that the sector is unavailable.

If a sector has no defects, the sector ID is formatted with the position field set to D'511 (all 1's).

5.1.9.3 Examine Data Buffer menu

Figure 5-9 shows the Examine Data Buffer menu. Table 5-9 lists the Examine Data Buffer menu commands.

```
                    E X A M I N E   D A T A   B U F F E R

nn[,WPH]  -  Display sector nn (Word(8), Parcel(8) or Hex)
    A     -  Print out ALL sectors
    B,nn  -  Print out sector nn
    R     -  Return

    Input sector number or option ==>
```

Figure 5-9. Examine Data Buffer Menu

Table 5-9. Examine Data Buffer Menu Commands

Command	Description
nn[,**wph**]	Displays sector *nn* in octal words (W) or parcels (P), or in hexadecimal (H)
a	Prints all sectors to file **BUFFER** in the current directory
b,*nn*	Prints sector *nn* to file **BUFFER** in the current directory
r	Returns to previous menu

5.1.9.4 ID Analysis menu (DD-10s, DD-39s, DD-40s, and DD-49s only)

ID analysis can be performed with or without the User Flaw Table (see commands **f** and **g**, respectively, in table 5-8).

The ID analysis report contains the following field headings for both the expected and actual IDs:

Heading	Description
NUM	Entry number
CYL	Cylinder number
HED	Head number
SEC	Sector number

The ID analysis report for DD-10s and DD-40s contains the following additional headings:

Heading	Description
POS	Position field (POS) of the sector ID (contains the defect address)
SPIN	Spindle associated with the sector ID. Each DD-40 contains four spindles, each of which is associated with 12 sectors. For DD-10s, SPIN is always 0.

ID analysis (DD-39s/49s) - Figure 5-10 shows the ID Analysis menu for DD-39 and DD-49 disk drives. Table 5-10 shows the ID analysis menu commands.

The following example describes the results of an ID analysis that was performed using the User Flaw Table (enter **f** from the Formatting menu).

To verify that IDs are being written correctly, the User Flaw Table is used to read the IDs of a track containing a flawed ID.

If all IDs match, a display such as the following is generated:

```
               VERIFYING IDs
       On Cylinder =   10 at 09:58:55
       On Cylinder =   11 at 09:58:56
       On Cylinder =   12 at 09:59:01
       On Cylinder =   13 at 09:59:03
       On Cylinder =   14 at 09:59:05
       On Cylinder =   15 at 09:59:06
       On Cylinder =   16 at 09:59:06
       On Cylinder =   17 at 09:59:08
       On Cylinder =   18 at 09:59:08
       On Cylinder =   19 at 09:59:09
       On Cylinder =   20 at 09:59:09
 --------------------------------------
       All IDs checked were correct
 --------------------------------------
  ---> Enter anything to continue <---
```

If there are any unexpected IDs, such as a flawed ID or an invalid sector ID, the routine generates an ID analysis report and displays the report with the ID Analysis menu (refer to figure 5-10, ID Analysis Menu for DD-39 and DD-49 disk drives).

If an ID matches the expected value, MATCH is displayed in the RESULTS column; otherwise, MISMATCH is displayed. If a mismatch occurs, refer to the mismatch column to determine whether the error is in the ID's cylinder (C), head (H), or sector (S). An ID of -1 (0'77) represents a flawed ID.

Generally, when one ID is in error, all subsequent IDs for the track are in error. To view specific IDs in the report, enter the desired entry number (NUM).

```
V E R I F Y  I D  A N A L Y S I S  F O R  39-1-32A       15:21:55 06/23/87
   EXPECTED ID         ACTUAL ID
 NUM   CYL HED SEC    CYL HED SEC          RESULTS              MISMATCH
------ ---- --- ---   ---- --- ---   ------------------------   --------
     1  841   0   0    841   0   0          M A T C H
     2  841   0   1    841   0   1          M A T C H
     3  841   0   2     -1  -1  -1   - Uncharted flaw found -    C  H  S
     4  841   0   3    841   0   2      M I S M A T C H  --->          S
     5  841   0   4    841   0   3      M I S M A T C H  --->          S
     6  841   0   5    841   0   4      M I S M A T C H  --->          S
     7  841   0   6    841   0   5      M I S M A T C H  --->          S
     8  841   0   7    841   0   6      M I S M A T C H  --->          S
     9  841   0   8    841   0   7      M I S M A T C H  --->          S
    10  841   0   9    841   0   8      M I S M A T C H  --->          S

               A  -  Show all entry types
               B  -  Show mismatched entries:  First=     1 Last=  72
               C  -  Print out all entries
               D  -  Print only mismatched entries
               R  -  Return
              Enter Command or Entry Number ==>
```

Figure 5-10. ID Analysis Menu for DD-39 and DD-49 Disk Drives

ID analysis (DD-40s) - Figure 5-11 shows the ID Analysis Menu for DD-40 disk drives. Table 5-10 shows the ID analysis menu commands.

The following example describes the results of an ID analysis that was performed without using the User Flaw Table (enter **g** from the Formatting menu).

The ID analysis report preceding the ID Analysis menu (figure 5-11) is for logical device 40-2-30A (command **b**, 'Show mismatched entries,' was entered). The results show that three mismatched entries were detected in the position (POS) field of the sector ID.

The SEC column in the ID analysis report shows the physical sector number. To calculate the logical sector number, do the following:

1. Multiply the spindle number (SPIN) by 12 (the number of sectors in each spindle).

2. Add the result from step 1 to the physical sector number.

For example, the ID analysis report in figure 5-11 shows physical sector 5 is associated with spindle 1. Calculate the logical sector number as follows:

```
1.   1 * 12 = 12   (spindle number * number of sectors in the spindle)
2.  12 +  5 = 17   (result from step 1 * physical sector number)
```

Logical sector 17 is the equivalent of physical sector 5 on spindle 1.

```
V E R I F Y  I D  A N A L Y S I S  F O R  40-2-30A      15:16:44 05/04/88
    EXPECTED ID          ACTUAL ID
  NUM CYL  HED SEC POS   CYL  HED SEC POS SPIN      RESULTS         MISMATCH
----- ---- --- --- ---  ---- --- --- --- ----  ------------------  --------
    4 1063  0    3 511  1063  0    3 210    0  M I S M A T C H ->        P
  114 1063  2    5 511  1063  2    5   7    1  M I S M A T C H ->        P
  170 1063  3    1 511  1063  3    1 169    2  M I S M A T C H ->        P
                                E N D  O F  D A T A

               A  -  Show all entry types
               B  -  Show mismatched entries:  First=     4 Last= 170
               C  -  Print out all entries
               D  -  Print only mismatched entries
               R  -  Return
              Enter Command or Entry Number ==>
```

Figure 5-11. ID Analysis Menu for DD-40 Disk Drives

ID Analysis menu commands - Table 5-10 shows the ID analysis menu commands.

Table 5-10. ID Analysis Menu Commands

Command	Description
a	Displays all entries in the report
b	Displays only the mismatched entries (which are not necessarily contiguous). The first and last mismatched entry numbers are displayed on the command line.
c	Enters the entire report in a file called **PRINTIDS,** which is located in the current directory
d	Enters only the mismatched entries in a file called **PRINTIDS,** which is located in the current directory
r	Returns to previous menu

5.1.9.5 Parameter menu

Figure 5-23 shows the Parameter menu. Table 5-15 lists the Parameter menu commands (refer to subsection 5.1.13, Parameter Menu).

5.1.10 SURFACE TESTS MENU

Figure 5-12 shows the Surface Tests menu. Table 5-11 lists the Surface Tests menu commands. These commands display the following submenus:

- Write Data menu
- Read Data and Compare menu
- Surface Analysis menu
- Examine Data Buffer menu
- Parameter menu

Surface tests consist of the following operations: reads, writes, read absolute, and surface analysis. These operations are all performed within the cylinder, head, and sector ranges specified in the argument banner. The read absolute operation only reads from the lowest track specified.

```
=================================================================== 10:11:47 =
=     DEVICE   CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE                  =
=    --------  ---------  -----  -------  ----  ----  ------                 =
=    49-2-24A   10 - 20   0 - 7   0 - 41    2   DD49  Maint.                 =
==============================================================================

                    S U R F A C E   T E S T   C H O I C E S

                  A  -  Write data
                  B  -  Read data and compare
                  C  -  Read exercise
                  D  -  Surface Analysis
                  E  -  Examine Buffer
                  F  -  Read Absolute (one track only)
                  G  -  Write Current Data Buffer
                  Z  -  Reset parameters
                  R  -  Return

                 Enter read/write option ==>
```

Figure 5-12. Surface Tests Menu

Table 5-11. Surface Tests Menu Commands

Command	Description
a	Displays the Write Data menu, from which you can select a data pattern to perform a write operation
b	Displays the Read Data and Compare menu, from which you can read the sectors listed in the argument banner and compare the data to the selected data pattern.
c	Reads the sectors listed in the argument banner. This command can be used to verify the readability of a sector or group of sectors.
d	Displays the Surface Analysis menu, from which you can do a write-read-compare on the sectors listed in the argument banner, using the selected surface analysis pattern.
e	Displays the Examine Data Buffer menu

Table 5-11. Surface Tests Menu Commands (continued)

Command	Description
f	Executes a read absolute operation, reading the specified sectors of the track with the lowest cylinder and head numbers in the argument banner. The read is performed without checking the sector's ID field. Therefore, the program reads the physical, rather than the logical, sector addresses.
g	Writes the contents of the data buffer to the specified cylinder, head, and sector locations
t	Reads the track headers of all the tracks in the cylinder with the lowest number in the argument banner. The information is stored in the data buffer. This menu command is displayed for DD-39s only.
z	Displays the Parameter menu, from which you can set the arguments in the argument banner.
r	Return to previous menu

5.1.10.1 Write Data, Read Data and Compare, and Surface Analysis menus

Figure 5-13 shows the Write Data menu. Figure 5-14 shows the Read Data and Compare menu. Figure 5-15 shows the Surface Analysis menu. Table 5-12 lists the commands for these menus. Use the commands to select patterns to be used for various operations. For a write or a read and compare operation, select only one pattern. For a surface analysis operation, select one or more patterns.

```
============================================================ 15:21:55 =
=     DEVICE   CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE           =
=    --------  ---------  -----  -------  ----  ----  ------          =
=    39-1-32A  841 - 841  0 - 4  0 - 23     1   DD39  System          =
=======================================================================

                        W R I T E   D A T A

  0  -  All zeros                         1  -  All ones
  A  -  Addressing pattern                B  -  Bump
  C  -  Alternating 0, 1 pattern
  E  -  Hole                              F  -  Fixed data
  G  -  Random
  S  -  Sequential data                   T  -  Peak shift
  Z  -  Reset Parameters                  R  -  Return

          Input the data pattern ==>
```

Figure 5-13. Write Data Menu

```
============================================================ 15:21:55 =
=     DEVICE   CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE           =
=    --------  ---------  -----  -------  ----  ----  ------          =
=    39-1-32A  841 - 841  0 - 4  0 - 23     1   DD39  System          =
=======================================================================

               R E A D   B U F F E R   &   C O M P A R E

  0  -  All zeros                         1  -  All ones
  A  -  Addressing pattern                B  -  Bump
  C  -  Alternating 0, 1 pattern
  E  -  Hole                              F  -  Fixed data
  S  -  Sequential data                   T  -  Peak shift
  Z  -  Reset Parameters                  R  -  Return

          Input the data pattern ==>
```

Figure 5-14. Read Data and Compare Menu

```
============================================================== 15:21:55 =
=      DEVICE   CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE            =
=     --------  ---------  -----  -------  ----  ----  ------           =
=     39-1-32A  841 - 841  0 - 4  0 - 23    1    DD39  System           =
=========================================================================

                    S U R F A C E   A N A L Y S I S

   0  -  All zeros                            1  -  All ones
   A  -  Addressing pattern                   B  -  Bump
   C  -  Alternating 0, 1 pattern
   D  -  All patterns but F
   E  -  Hole                                 F  -  Fixed data
   G  -  Random
   S  -  Sequential data                      T  -  Peak shift
   Z  -  Reset Parameters                     R  -  Return

            Input the data pattern ==>
```

Figure 5-15. Surface Analysis Menu

Table 5-12. Commands for the Write Data, Read Data and Compare, and Surface Analysis Menus

Command	Description
0	All 0's
1	All 1's
a	Addressing pattern in a Cray word: Parcel / Value 0 — Cylinder number 1 — Head number 2 — Sector number 3 — Word number
b	Bump. This is a repeating 4-word pattern: Word / Octal / Hexadecimal 0 — 0525252525242104252525 — 5555 5555 1111 5555 1 — 0525250421052525252525 — 5555 1111 5555 5555 2 — 0104212525252525252525 — 1111 5555 5555 5555 3 — 0525252525252525210421 — 5555 5555 5555 1111

Table 5-12. Commands for the Write Data, Read Data and Compare, and Surface Analysis Menus (continued)

Command	Description
c	Alternating 0's and 1's. This is a repeating 2-word pattern: Word / Octal / Hexadecimal 0 / 1252525252525252525252 / AAAA AAAA AAAA AAAA 1 / 0525252525252525252525 / 5555 5555 5555 5555
d	All patterns except the fixed data pattern (F)
e	Hole. This is a repeating 4-word pattern: Word / Octal / Hexadecimal 0 / 0525252525256735652525 / 5555 5555 7777 5555 1 / 0525356735252525252525 / 5555 7777 5555 5555 2 / 0735672525252525252525 / 7777 5555 5555 5555 3 / 0525252525252525273567 / 5555 5555 5555 7777
f	Fixed data. This is a 1-word, user-input pattern.
g	Random data
s	Sequential data pattern: Word / Description 0 / Random number *n* / Word 0 + *n*
t	Peak shift. This is a repeating 3-word pattern: Word / Octal / Hexadecimal 0 / 0631466735667356663146 / 6666 DDDD BBBB 6666 1 / 1567355673554631556735 / DDDD BBBB 6666 DDDD 2 / 1356733146333567335673 / BBBB 6666 DDDD BBBB
z	Displays the Parameter menu
r	Return to previous menu

5.1.10.2 Examine Data Buffer menu

Figure 5-9 shows the Examine Data Buffer menu. Table 5-9 lists the Examine Data Buffer menu commands (refer to subsection 5.1.10.2, Examine Data Buffer Menu).

5.1.10.3 Parameter menu

Figure 5-23 shows the Parameter menu. Table 5-15 lists the Parameter menu commands (refer to subsection 5.1.13, Parameter Menu).

5.1.11 FLAW TABLE UTILITY MENUS

Figure 5-16 shows the Flaw Table Utility menu. Table 5-13 lists the Flaw Table Utility menu commands. These commands display the following submenus:

- Factory Flaw Table
- User Flaw Table
- System Flaw Table
- Found Flaw Table

```
============================================================== 10:11:53 =
=      DEVICE    CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE           =
=     --------   ---------  -----  -------  ----  ----  ------          =
=     49-2-24A    10 - 20   0 - 7   0 - 41    2   DD49  Maint.          =
=========================================================================

               F L A W   T A B L E   U T I L I T Y

                A  -  FACTORY Flaw Table
                B  -  USER Flaw Table
                C  -  SYSTEM Flaw Table
                D  -  FOUND Flaw Table
                R  -  Return

               Choose a flaw table ==>
```

Figure 5-16. Flaw Table Utility Menu

Table 5-13. Flaw Table Utility Menu Commands

Command	Description
a	Displays the Factory Flaw Table (not used for DD-19/29 disks). This table contains the factory flaws originally found on the disk.
b	Displays the User Flaw Table (not used for DD-19/29 disks). This table contains the physical addresses of the flawed sectors.
c	Displays the System Flaw Table (sometimes called the System EFT). This table contains the flaws used by UNICOS when creating the UNICOS Flaw Map.
d	Displays the Found Flaw Table, which resides in **donut.** This table contains flaws detected during surface analysis.
r	Returns to the previous menu

The flaw table utilities allow you to read, edit, write, or print the disk flaw tables. Flaw tables can be edited in **donut**'s area of central memory only. However, **donut** does not automatically write the edited tables to disk; you must enter **f** (Write flaw table to disk) from either the User or System flaw table, as appropriate. Any function that requires flaw tables (such as formatting) uses the tables currently in **donut**'s area of central memory (the tables must be read into **donut** before they can be referenced).

To display a flaw table without going through the Flaw Table Utility menu, enter one of the following commands from the Main menu or any of the flaw table menus, as appropriate:

Command	Description
FAC	Displays the Factory Flaw Table menu
USR	Displays the User Flaw Table menu
SYS	Displays the System Flaw Table menu
FND	Displays the Found Flaw Table menu

For example, if your current screen display shows the User Flaw Table menu, you can enter **sys** to display the System Flaw Table menu. To return to the User Flaw Table menu, enter **r**.

The main heading in each flaw table menu contains the following information:

- Logical device name
- Flaw table name
- Number of entries

Below the main heading are the following field headings:

Heading	Description
NUM	Entry number
CHANNEL	Channel number
CYL	Cylinder number
HEAD	Head number
SEC	Sector number
USER	User-input-flaw bit

The User and Found Flaw Tables for DD-10s and DD-40s contain the following additional headings (and no channel number heading):

Heading	Description
U/H	Hideable/unhideable defects. For additional information, refer to subsection 5.1.9.2, Position Field of the Sector ID (DD-10s and DD-40s only).
Position	Position field (POS) of the sector ID. The POS field contains the defect address.

In the System Flaw Table, the field heading contains a contiguous (CONTIG) number, which is always a value of 1, instead of a channel number and no USER bit heading; however, this field is not used under UNICOS.

Each flaw table display lists up to 18 flaws, two per line. From any of the flaw tables, you can do the following:

- Enter a menu command to perform a specific function

- Enter the number of the first flaw that you want to appear in a display of any contiguous group of flaws

- Enter + (plus) or - (minus) to scroll forward or backward, respectively

For additional flaw information, refer to the Disk Systems Hardware Reference Manual, CRI publication HR-0077.

The flaw tables are shown in the following figures:

Figure	Title
5-17	Factory Flaw Table Menu
5-18	User Flaw Table Menu for DD-39 and DD-49 Disk Drives
5-19	User Flaw Table Menu for DD-10 and DD-40 disk drives
5-20	System Flaw Table Menu
5-21	Found Flaw Table Menu for DD-19/29/39/49 Disk Drives
5-22	Found Flaw Table Menu for DD-10 and DD-40 Disk Drives

Table 5-14 shows the commands for the flaw table menus. These commands apply to all of the flaw tables unless otherwise indicated.

```
 49-2-24A        F A C T O R Y   F L A W   T A B L E         LAST=  249
------------------------------------------------------------------------
 NUM    CHANNEL     CYL HEAD SEC   USER NUM    CHANNEL     CYL HEAD  SEC  USER
 ---- -------------- ---- ---    ---- ---- ------------ ---- ---- ---- ----
   1 -- -- A2 --    0    0   0    0   10 -- -- A2 --    8    0    0    0
   2 -- -- A2 --    1    0   0    0   11 -- -- A2 --    9    0    0    0
   3 -- -- A2 --    2    0   0    0   12 -- -- A2 --   10    0    0    0
   4 -- -- A2 --    3    0   0    0   13 -- -- A2 --   11    0    0    0
   5 -- -- A2 --    4    0   0    0   14 -- -- A2 --   12    0    0    0
   6 -- -- A2 --    5    0   0    0   15 -- -- A2 --   13    0    0    0
   7 -- -- A2 --    6    0   0    0   16 -- -- A2 --   40    0    0    0
   8 -- -- A2 --    7    0   0    0   17 -- -- A2 --   41    0    0    0
   9 B2 -- -- --    7    5   1    0   18 B2 -- -- --   43    5   21    0

                B   - Read flaw table from disk
                C   - Check flaw table validity
                E   - Erase flaw table from memory
                V   - Print out flaw table
                X n - Start display at cylinder n
                R   - Return
               Enter Command or Flaw Number ==>
```

Figure 5-17. Factory Flaw Table Menu

```
49-2-24A          U S E R   F L A W   T A B L E              LAST=  249
------------------------------------------------------------------------
NUM  CHANNEL     CYL HEAD SEC  USER  NUM  CHANNEL      CYL HEAD  SEC  USER
---- ----------- ---- ---  ---- ---- ------------ ---- ---- ---- ----
  1 -- -- A2 --   0    0    0    0   10 -- -- A2 --   8    0    0    0
  2 -- -- A2 --   1    0    0    0   11 -- -- A2 --   9    0    0    0
  3 -- -- A2 --   2    0    0    0   12 -- -- A2 --  10    0    0    0
  4 -- -- A2 --   3    0    0    0   13 -- -- A2 --  11    0    0    0
  5 -- -- A2 --   4    0    0    0   14 -- -- A2 --  12    0    0    0
  6 -- -- A2 --   5    0    0    0   15 -- -- A2 --  13    0    0    0
  7 -- -- A2 --   6    0    0    0   16 -- -- A2 --  40    0    0    0
  8 -- -- A2 --   7    0    0    0   17 -- -- A2 --  41    0    0    0
  9 B2 -- -- --   7    5    1    0   18 B2 -- -- --  43    5   21    0

A   - Add a flaw                      B  - Read flaw table from disk
C   - Check flaw table validity       D  - Delete a flaw
E   - Erase flaw table from memory    F  - Write flaw table to disk
G   - Merge FACTORY flaws into USER   V  - Print out flaw table
X n - Start display at cylinder n     R  - Return
          Enter Command or Flaw Number ==>
```

Figure 5-18. User Flaw Table Menu for DD-39 and DD-49 Disk Drives

```
40-1-36A     U S E R   F L A W   T A B L E     HIDEABLE = 425  LAST=1165
------------------------------------------------------------------------
NUM  CYL HEAD SEC USER U/H POSITION NUM  CYL HEAD SEC USER U/H POSITION
---- ---- ---- --- ---- --- -------- ---- ---- ---- --- ---- --- --------
418 1055   15  28    0 U        511 427    3   14  13    0 H        151
419 1057    6  16    0 U        511 428    3   15  11    0 H        214
420 1058    1  37    0 U        511 429    4   12  35    0 H        148
421 1059   15  16    0 U        511 430    4   14  13    0 H        151
422 1060    2  27    0 U        511 431    4   15  11    0 H        215
423 1060   15  16    0 U        511 432    6   12  12    0 H        256
424 1063   15  16    1 U        511 433    7    1   1    1 H        117
425    1    1   2    1 H         69 434    7    6  19    0 H         95
426    1    8  12    0 H        199 435    7   12  12    0 H        256

A   - Add a flaw                      B   - Read flaw table from disk
C   - Check flaw table validity       D   - Delete a flaw
E   - Erase flaw table from memory    F   - Write flaw table to disk
V   - Print out flaw table            X n - Display unhideables at CYL n
Y n - Display hideables at CYL n      R   - Return
          Enter Command or Flaw Number ==>
```

Figure 5-19. User Flaw Table Menu for DD-10 and DD-40 Disk Drives

```
 49-2-24A          S Y S T E M   F L A W   T A B L E         LAST=    1
-------------------------------------------------------------------------
 NUM   # CONTIG     CYL HEAD  SEC         NUM   # CONTIG    CYL HEAD  SEC
---- ------------ ---- ---- ----        ---- ------------ ---- ---- ----
   1        1      495    7   41          10
   2                                      11
   3                                      12
   4                                      13
   5                                      14
   6                                      15
   7                                      16
   8                                      17
   9                                      18

 A   - Add a flaw                       B    - Read flaw table from disk
 C   - Check flaw table validity        D    - Delete a flaw
 E   - Erase flaw table from memory     F    - Write flaw table to disk
 G   - Make SYSTEM table from FACTORY   H    - Make SYSTEM table from USER
 V   - Print out flaw table             X n  - Start display at cylinder n
 R   - Return
             Enter Command or Flaw Number ==>
```

Figure 5-20. System Flaw Table Menu

```
 49-2-24A           F O U N D   F L A W   T A B L E          LAST=    1
---------------------------------------------------------------------------
 NUM   CHANNEL     CYL HEAD SEC  USER  NUM   CHANNEL     CYL HEAD  SEC  USER
----- ------------- ---- ---  ---- ---- ----------- ---- ---- ---- ----
    1 B2 B1 A2 A1   1     2    3    1    10
    2                                    11
    3                                    12
    4                                    13
    5                                    14
    6                                    15
    7                                    16
    8                                    17
    9                                    18

A    - Add a flaw                     C  - Check flaw table validity
D    - Delete a flaw                  E  - Erase flaw table from memory
V    - Print out flaw table           G  - Merge FOUND flaws into USER flaw table
X n  - Start display at cylinder n    R  - Return

             Enter Command or Flaw Number ==>
```

Figure 5-21. Found Flaw Table Menu for DD-19/29/39/49 Disk Drives

```
 40-1-36A   F O U N D   F L A W   T A B L E     HIDEABLE =   2  LAST=   2
------------------------------------------------------------------------
 NUM  CYL HEAD SEC USER U/H POSITION  NUM  CYL HEAD SEC USER U/H POSITION
---- ---- ---- --- ---- --- -------- ---- ---- ---- --- ---- --- --------
   1 1055   15  28    0 U        511
   2    1    1   2    1 H         69

A    - Add a flaw                    C  - Check flaw table validity
D    - Delete a flaw                 E  - Erase flaw table from memory
V    - Print out flaw table          G  - Merge FOUND flaws into USER flaw table
X n - Start display at cylinder n    R  - Return

             Enter Command or Flaw Number ==>
```

Figure 5-22. Found Flaw Table Menu for DD-10 and DD-40 Disk Drives

Table 5-14. Commands for the Flaw Table Menus

Command	Description
a	Adds a flaw; issues prompts for the flaw arguments and inserts valid flaws in their proper order in the flaw table. Flaws cannot be added to the Factory Flaw Table.
b	Reads the flaw table from disk to central memory, after first deleting the table currently in central memory. • System Flaw Table menu When the System Flaw Table is read from disk, the table is compared to the UNICOS Flaw Map and any mismatches are displayed on the screen.
c	Verifies that the flaw table is in order, that no duplicate entries exist, that values are within a valid range, and that the table is terminated correctly. If a problem exists in any of these areas, a message is displayed indicating the first entry in error.
d	Deletes a flaw; issues prompts for the entry number of the flaw to be deleted. The flaw is only removed from the table currently in central memory (does not affect the disk-resident table). Factory flaws cannot be deleted.

Table 5-14. Commands for the Flaw Table Menus (continued)

Command	Description
e	Deletes flaw table from central memory (does not affect the disk-resident table)
f	Writes flaw table from central memory to disk, overwriting the disk-resident table. Factory and Found flaw tables cannot be written to disk. • System Flaw Table menu In addition to writing the table from central memory to disk, the UNICOS Flaw Map (used by UNICOS to define alternate sectors for flawed sectors) will be updated to reflect the new System Flaw Table.
g	Merges flaws from one flaw table into another. The menu from which the **g** command is entered and (in some cases) the device type being exercised determine which flaw tables are merged. You can enter **g** from the following menus: • Found Flaw Table menu For DD-39, DD-40, and DD-49 disk drives: Copies the Found Flaw Table entries into the User Flaw Table. Duplicate entries are skipped. Entries are added in their proper order. For DD-19 and DD-29 disk drives: Copies the Found Flaw Table entries into the System Flaw Table (this does not overwrite the current System Flaw Table) • User Flaw Table menu Copies the Factory Flaw Table entries into the User Flaw Table. Duplicate entries are skipped. Entries are added in their proper order.

Table 5-14. Commands for the Flaw Table Menus (continued)

Command	Description
	• System Flaw Table menu: Creates a System Flaw Table from the Factory Flaw Table entries. The **SLIP** argument determines which entries are made in the System Flaw Table.
h	Creates a System Flaw Table from the User Flaw Table entries (**h** is entered from the System Flaw Table menu only). The **SLIP** argument determines which entries are made in the System Flaw Table.
v	Creates a file with the name of the flaw table (FACTORY, USER, SYSTEM, or FOUND) in the current directory.
x *n*	Displays flaws starting at cylinder *n*. For DD-40s, the flaws displayed are unhideable defects.
y *n*	Displays hideable defects starting at cylinder *n* (DD-40s only)
+	Displays the next screen of flaws
-	Displays the previous screen of flaws
r	Returns to previous menu

5.1.12 ERROR CORRECTION CODE TEST

The Error Correction Code (ECC) test does the following:†

1. Writes a 512-word buffer of random data with 0's for ECC.
2. Reads the data, expecting an ECC error.
3. Writes the same data with standard ECC.
4. Reads the data, expecting no errors.

† The ECC test cannot be performed on DD-19 or DD-29 disk drives.

5. Compares the data read with that written in step 3.

6. Displays a message indicating whether the ECC test passed or failed. If the test failed, the message also indicates the word-in-error.

The ECC test uses the **DISK** and **DEVICE** arguments (displayed in the argument banner) and the following software CE cylinder numbers (instead of the numbers in the argument banner):

Cylinder	= *n*	Scratch cylinder; typically the last cylinder on the device.
Head	= 0	
Sector	= 0	

5.1.13 PARAMETER MENU

Figure 5-23 shows the Parameter menu, from which you can define the logical device name and set the arguments (parameters) in the argument banner. Table 5-15 lists the Parameter menu commands.

```
============================================================ 09:50:28 =
=      DEVICE   CYLINDERS  HEADS  SECTORS  SLIP  DISK   MODE          =
=     --------  ---------  -----  -------  ----  ----  ------         =
=     * none *    0 - 0    0 - 0   0 - 0     0   none  ~~~~~~         =
=======================================================================

                            P A R A M E T E R S

                 A  -  Logical Device
                 B  -  Cylinder limits
                 C  -  Head limits
                 D  -  Sector limits
                 E  -  Diagnostic flags (not displayed)
                 T  -  Toggle disk mode (system/maintenance)
                 R  -  Return
                Enter Command ==>
```

Figure 5-23. Parameter Menu

Table 5-15. Parameter Menu Commands

Command	Parameter	Description
a	DEVICE	Sets the logical device name. You must respond to the prompts.
b	CYLINDERS	Sets the cylinder range. You must respond to the prompts.
c	HEADS	Sets the head range. You must respond to the prompts.
d	SECTORS	Sets the sector range. You must respond to the prompts.
e	FLAGS	Sets diagnostic flags related to IOS error handling and read-ahead/write behind operations. You can set any combination of the following flags: Flag Description **a** Returns the error record to the diagnostic error logger, **diagerr** **b** Disables error recovery. The IOS does not attempt a retry. **c** Disables error reporting. The IOS does not log errors in the error logger. **d** Disables read-ahead/write behind operations If no flags are set, all flags are enabled.
t	MODE	Sets the disk mode to **system** or **maintenance**
r		Returns to previous menu

5.1.14 EXITING **donut**

To exit **donut**, enter **q** from the Main menu. The exit process does not change the disk mode or write any edited flaw tables to disk. (It is assumed that these operations are performed prior to exiting.) The final **donut** screen display is as follows:

```
G o o d b y e   f r o m   D O N U T
```

5.1.15 PROGRAM EXAMPLES

This subsection contains various **donut** execution examples, all of which originate from the Main menu.

Example 1 shows how to enable maintenance mode for a DD-39 disk with a logical device name of 39-2-27A.

Example 1:

1. Enter **z** (reset parameters) from the Main menu.
2. Enter **a** (logical device) from the Parameter menu.
   - Enter **39-2-27A** for the logical device name.
3. Enter **t** (toggle disk mode) from the Parameter menu.
   - Enter **go** to acknowledge the warning.
   - The following message is displayed and remains on the screen until the disk is offloaded and in maintenance mode:

     Please wait while 39-2-27A is entering MAINTENANCE mode
4. Enter **r** to return to the Main menu.

Example 2 shows the procedure to do the following:

- Read the User Flaw Table from disk
- Add the following flaw to the table:

  CYLINDER=25, HEAD=2, SECTOR=19, all surfaces

- Write the modified User Flaw Table to the disk
- Print the User Flaw Table in octal format

Example 2:

1. Enter **t** (Flaw Table Utility) from the Main menu.

2. Enter **b** (USER Flaw Table) from the Flaw Table Utility menu.

3. Enter **b** (Read flaw table from disk) from the User Flaw Table menu.

   - Enter **go** to acknowledge the warning.

4. Enter **a** (Add a flaw) from the User Flaw Table menu.

   - Enter **25** for the cylinder number.
   - Enter **2** for the head number.
   - Enter **19** for the sector number.
   - Enter **a** for all surfaces.

5. Enter **f** (Write flaw table to disk) from the User Flaw Table menu.

   - Enter **go** to acknowledge the warning.

6. Enter **v** (Print out flaw table) from the User Flaw Table menu.

   - Enter **c** for octal format.
   - Enter **r** to return to the User Flaw Table menu.

7. Enter **r** to return to the Main menu.

Example 3 shows the procedure to do the following:

- Format the track of CYLINDER=25, HEAD=2 (using the User Flaw Table)
- Verify that the IDs were written correctly

Example 3:

1. Enter **f** (formatting and ID analysis) from the Main menu.

2. Enter **z** (reset parameters) from the Formatting menu.

Example 3 (continued):

3. Enter **b** (cylinder limits) from the Parameter menu.

   - Enter **25** for the lower cylinder number.
   - Enter **25** for the upper cylinder number.

4. Enter **c** (head limits) from the Parameter menu.

   - Enter **2** for the lower head number.
   - Enter **2** for the upper head number.
   - Enter **r** to return to the Formatting menu.

5. Enter **b** (Format with USER Flaw Table) from the Formatting menu.

   - Enter **go v** after checking the formatting limits.

   - After formatting, the IDs are checked. If all IDs match their expected values, a message to that effect is displayed with the following prompt:

     ---> Enter anything to continue <---

   - If an ID error occurs, the ID Analysis Results menu is displayed. Check the results and/or obtain a printout. Enter **r** to return to the Formatting menu.

6. Enter **r** to return to the Main menu.

Example 4 shows how to perform surface analysis on cylinder 25, using the default patterns and executing the random pattern 50 times with a seed value of 6065.

Example 4:

1. Enter **z** (reset parameters) from the Main menu.

2. Enter **b** (cylinder limits) from the Parameter menu.

   - Enter **25** for the lower cylinder number.
   - Enter **25** for the upper cylinder number.

3. Enter **c** (head limits) from the Parameter menu.

   - Enter **a** for all heads.

4. Enter **d** (sector limits) from the Parameter menu.

   - Enter **a** for all sectors.

Example 4 (continued):

5. Enter **r** to return to the Main menu.

6. Enter **s** (surface tests) from the Main menu.

7. Enter **d** (surface analysis) from the Surface Tests menu.

   - Enter **d** to execute all patterns except the fixed data pattern.

   - Enter **50** for the number of random passes.

   - Enter **6065** for the seed value.

   - Enter **go** after checking the arguments.

   - The display changes as the program analyzes each track. After all tracks are analyzed, the program displays a message indicating the number of flaws added to the Found Flaw Table. This signals the end of the surface analysis operation.

   - Respond to the following prompt:

     ```
     ---> Enter anything to continue <---
     ```

   - Enter **r** to return to the Surface Tests menu.

8. Enter **r** to return to the Main menu.

Example 5 shows the procedure to do the following:

- Read the User Flaw Table for the DD-49 disk with a logical device name of 49-1-24A.

- Add the following flaw to the User Flaw Table:

  ```
  Cylinder = 1507 (octal)
  Head     =    3
  Sector   =   17 (octal)
  Channel  =   A2
  ```

- Generate a printout of the User Flaw Table (in octal).

- Write the User Flaw Table to disk.

- Generate the System Flaw Table from the User Flaw Table.

- Generate a printout of the System Flaw Table (in octal).

- Write the System Flaw Table to disk.
- Reformat Cylinder = 1507, Head = 3, using the User Flaw Table in central memory.

Example 5:

1. Enter **oct** (octal display) from the Main menu.
2. Enter **dev** from the Main menu to change the logical device name.
3. Enter **49-1-24A**.
4. Enter **usr** (display the User Flaw Table) from the Main menu.
5. Enter **b** (read flaw table from disk) from the User Flaw Table menu.
   - Enter **go** to acknowledge the warning.
6. Enter **a** (Add a flaw) from the User Flaw Table menu.
   - Enter **1507** for the cylinder number.
   - Enter **3** for the head number.
   - Enter **17** for the sector number.
   - Enter **a2** for the channel number.
7. Enter **v** (Print out flaw table) from the User Flaw Table menu.
   - Enter **c** for octal printout.

8 Enter **f** (Write flaw table to disk) from the User Flaw Table menu.

   - Enter **go** to acknowledge the warning.

9. Enter **sys** (display the System Flaw Table) from the User Flaw Table menu.
10. Enter **h** (Make SYSTEM table from USER) from the System Flaw Table menu.
11. Enter **v** (Print out flaw table) from the System Flaw Table menu.
    - Enter **c** for an octal printout.
12. Enter **f** (Write flaw table to disk) from the System Flaw Table menu.
    - Enter **go** to acknowledge the warning.
13. Enter **r** to return to the Main menu.

14. Enter **cyl** (set cylinder range) from the Main menu.

    - Enter **1507** for the lower cylinder number.
    - Enter **1507** for the upper cylinder number.

15. Enter **hed** (set head range) from the Main menu.

    - Enter **3** for the lower head number.
    - Enter **3** for the upper head number.

16. Enter **f** (Formatting and ID analysis) from the Main menu.

17. Enter **b** (Format with USER Flaw Table) from the Formatting menu.

    - Enter **go** after checking the argument limits.

    - After formatting, the IDs are checked. If all IDs match their expected values, a message to that effect is displayed with the following prompt:

      ---> Enter anything to continue <---

    - If an ID error occurs, the ID Analysis Results menu is displayed. Check the results and/or obtain a printout. Enter **r** to return to the Formatting menu.

18. Enter **r** to return to the Main menu.

Example 6 shows how to return the disk to system mode before exiting **donut**.

Example 6:

1. Enter **z** (reset parameters) from the Main menu.

2. Enter **t** (toggle disk mode) from the Parameter menu.

(Alternatively, you can enter **mode** from the Main menu instead of steps 1 and 2 and proceed with step 3.)

3. Enter **go** to acknowledge the request.

   The following message is displayed and remains on the screen until the disk is in system mode:

   Please wait while 39-2-27A is entering SYSTEM mode

4. Enter **r** to return to the Main menu.

5. Enter **q** to exit **donut**.

5.2 oldmon

The **oldmon**† monitor is the down CPU monitor, which initiates, controls, and monitors the down CPU tests. These tests execute under **oldmon** in multiple-CPU environments only. For a list of the down CPU tests, refer to appendix A, On-line Diagnostic Programs. For information on the down CPU interface to UNICOS, refer to **cpu**(4D).

5.2.1 DOWN CPU TESTS

The down CPU tests are executed in a down CPU from an operational CPU. Down CPU tests cannot be executed in monitor mode; consequently, they cannot perform I/O operations. A CPU other than the down CPU initiates I/O activity and all CPUs other than the down CPU are favored for external interrupts. If the down CPU receives interrupts, it redirects them to another CPU. For additional information on interrupts and monitor mode, refer to the following manuals, as appropriate:

CSM0111000	CRAY X-MP/1 System Programmer Reference Manual
CSM0110000	CRAY X-MP/2 System Programmer Reference Manual
CSM0112000	CRAY X-MP/4 System Programmer Reference Manual
CSM-0400-000	CRAY Y-MP System Programmer Reference Manual

To execute in a down CPU, a program must meet the following requirements:

- Must be an absolute binary
- Must not require any operating system support (the program cannot allow screen output, keyboard input, disk reading, or disk writing)

The **oldmon** monitor does the following:

- Downs the CPU
- Loads a down CPU test from a file into central memory
- Monitors and controls the execution of a down CPU test
- Loads central memory areas from files
- Allows an operator to modify the central memory image of a down CPU test

† Multiple-CPU Cray computer systems only

* Displays central memory areas in various data formats

* Writes central memory areas to files

* Dumps central memory areas in a variety of formats to files or to the expander printer

* Executes user-defined program loops

5.2.2 PROGRAM SYNOPSIS

The **oldmon** monitor resides in **/ce/bin.** Log on interactively at the system console or any other supported front-end station (refer to the appropriate front-end station reference manual).

Synopsis:

oldmon [**-d** *cpulist*] [**-q**] [**-u** *cpulist*]

-d *cpulist*

Down CPUs immediately. *cpulist* is entered in the following format:

n,*n*,...,*n*

n is a value in one of the following ranges:

0,1,2,...,*n* or **a,b,c,...,x**

If allowed to default, no CPUs are downed.

-q Exit **oldmon** after processing the command line entry. This command option should be entered with other options.

-u *cpulist*

Return CPUs to normal system operations. *cpulist* is entered in the following format:

n,*n*,...,*n*

n is a value in one of the following ranges:

0,1,2,...,*n* or **a,b,c,...,x**

Table 5-16 lists the **oldmon** commands. For additional information on these commands, refer to subsections 5.2.5.2 through 5.2.5.17.

Table 5-16. **oldmon** Commands

Command	Description
a	Appends a formatted central memory dump to a file
c	Specifies a new default CPU
d	Dumps a formatted central memory dump to a file
e	Enters a value at a specific address
f	Fills consecutive central memory locations
g	Starts a test in a CPU
h	Halts test execution in a down CPU
l	Loads a test into a CPU's central memory buffer
o	Sets test options
q	Exits **oldmon**
r	Redraws the display
s	Updates the current Exchange Package of the current CPU
u	Returns a down CPU to normal system operations
v	Views a formatted area of central memory
w	Writes an area of central memory to a binary file
x	Executes a command buffer containing **oldmon** commands

5.2.3 PROGRAM EXECUTION

When **oldmon** is started, it does the following:

1. Allocates an area of central memory to each CPU

2. Loads the test loop code into each CPU's memory area

3. Executes **$HOME/.oldmonrc** (a profile file containing any **oldmon** commands)

4. Displays the Main menu for **oldmon** (refer to figure 5-24)

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
```

Figure 5-24. Main Menu for **oldmon**

The following subsections describe program execution under **oldmon**:

- Down CPU tests (listed in appendix A, On-line Diagnostic Programs)
- Test loop code
- Environment variables

5.2.3.1 Down CPU tests

The down CPU tests reside in **/ce/oldmon**. Two types of down CPU tests run under **oldmon**: confidence tests and maintenance tests. The down CPU confidence tests are on-line confidence tests that have been converted to run under **oldmon** (off-line). The down CPU maintenance tests are taken from the off-line diagnostic release.†

The initial Exchange Package starts each test. The current Exchange Package allows a test to continue from the point at which it is interrupted.

For a list of the off-line diagnostics (down CPU tests) that run under **oldmon**, refer to Appendix A, On-line Diagnostic Programs.

† The down CPU maintenance tests are deferred for CEA systems.

Modifications to the off-line diagnostic test base - The down CPU tests are derived from the off-line diagnostic release X3.0. Some of the off-line diagnostics require modifications before they can be executed in a down CPU test environment. A configuration file containing a list of **oldmon** commands is used to make the necessary modifications.

When **oldmon** is executed, it attempts to access the configuration file **oldmon.cf**. If **oldmon.cf** is found, **oldmon** uses the information in the **oldmon.cf** configuration file to automatically configure a loaded diagnostic to execute in a down CPU environment; if **oldmon.cf** is not found, **oldmon** uses the default configuration file.

If **oldmon.cf** is not found, you can initialize it by entering **y** (yes) in response to the following prompt:

```
Cannot find configuration file oldmon.cf, should I initialize it?
Enter Yes or No (y/n)>
```

If you enter **n** (no), **oldmon** does not initialize **oldmon.cf**.

Default configuration files - The default configuration files are used to make the necessary modifications to the off-line diagnostics tests, so that they can execute in a down CPU test environment:

The following is the default configuration file for a CRAY X-MP computer system.

```
# OLDMON configuration file for X-MP off-line diagnostics.
#
aht:     e k cput 40      # Set CPU type, 20 for X-MP/2, 40 for X-MP/4
         e k mlast 7777   # Set last address to be tested
         o l 7777         # Set limit address
arb:     e 40a 005000     # Change MTA I/O routine to return
         e 140 100000000000          # Set P in SEXP
         e 143 160000000000000       # Set mode bits in SEXP
         e 144 1000000000000000000000 # Set EMA bit in SEXP
arm:                      # Nothing to configure
brb:     o l 1577         # Set limit address
cmp:                      # Nothing to configure
cmx:     e 26c 1000       # Run CMX with cluster 0
         o l 44777        # Set limit address
         e 1152c 001000   # Change monitor req. exch to pass
gth:     e k mlast 33777  # Set last address to be tested
         o l 33777        # Set limit address
ibz:     e k cput 40      # Set CPU type, 20 for X-MP/2, 40 for X-MP/4
         e k secs 62      # Can only run sections 1, 4 and 5
         o l 400777       # Set limit address
         e k cpun 1       # Set number of CPUs to 1
mit:     e k cput 40      # Set CPU type, 20 for X-MP/2, 40 for X-MP/4
         e k mlast 7777   # Set last address to be tested
         o l 7777         # Set limit address
```

Default configuration file (continued):

```
sfa:      e 205c 177777    # Disable timing portion of test
sfm:      e 205c 177777    # Disable timing portion of test
sfr:      e k secs 65432   # Disable section 1 of test
sis:                       # Nothing to configure
sr3:      o 1 6277         # Set limit address
sra:                       # Nothing to configure
srb:                       # Nothing to configure
srl:      e 40a 005000     # Change MTA I/O routine to return
          e 140 100000000000           # Set P in SEXP
          e 143 160000000000000        # Set mode bits in SEXP
          e 144 1000000000000000000000 # Set EMA bit in SEXP
srs:      e 40a 005000     # Change MTA I/O routine to return
          e 140 100000000000           # Set P in SEXP
          e 143 160000000000000        # Set mode bits in SEXP
          e 144 1000000000000000000000 # Set EMA bit in SEXP
stan:                      # Nothing to configure
svc:      o 1 1577         # Set limit address
trb:      o 1 1577         # Set limit address
vpp:      e 205c 177777    # Disable timing portion of test
vra:      o 1 2077         # Set limit address
          e 205c 177777    # Disable timing portion of test
vrl:      e 40a 005000     # Change MTA I/O routine to return
          e 140 100000000000           # Set P in SEXP
          e 143 160000000000000        # Set mode bits in SEXP
          e 144 1000000000000000000000 # Set EMA bit in SEXP
          e 205b 177777    # Disable timing portion of test
vrn:      o 1 2777         # Set limit address
vrr:      o 1 4777         # Set limit address
vrs:      o 1 2077         # Set limit address
          e 205d 177777    # Disable timing portion of test
vrx:      o 1 23777        # Set limit address
olcrit:   o 1 60000        # Set limit address
olcsvc:   o 1 50000        # Set limit address
olcfpt:   o 1 40000        # Set limit address
olibuf:   o 1 30000        # Set limit address
olcm:     o 1 40000        # Set limit address
```

The following is the default configuration file for a CEA system.

```
# OLDMON configuration file for Y-MP off-line diagnostics.
#
olcrit:   o 1 60000        # Set limit address
olcsvc:   o 1 50000        # Set limit address
olcfpt:   o 1 40000        # Set limit address
olibuf:   o 1 30000        # Set limit address
olcm:     o 1 40000        # Set limit address
```

5.2.3.2 Test loop code

The test loop code can be used to build a failing loop. The initial Exchange Package resides at address O'140. Use either the Enter or Fill command to overwrite the PASS instructions (instruction 001000 at address O'500a) with the suspected failing code. The suspected failing code (at address O'500a) is executed with the test loop. The program then jumps to a check routine.

The check routine does the following:

1. Compares the actual results in S1 to the expected results in S2

2. Increments the PASS and ERROR counts

3. Jumps to the suspected failing code sequence (at address O'500a) to loop

The current Exchange Package resides at address O'120. It allows the loop to continue from the point at which it is interrupted.

The test loop code is as follows:

```
START    =         *              ; Initialize values.
         S0        0
         PASS,     S0
         ERROR,    S0
         ACT,      S0
         EXP,      S0
         DIF,      S0

MAINLOOP =         *
         J         TESTCODE       ; Jump to testcode provided by user.

*
*  Test code provided by user should return here.  The test code can
*  use all registers.  It should return with s1 containing the
*  actual value, and s2 containing the expected value.
*
```

Test loop code (continued):

```
    TESTRTN  =        *
             SO       S1\S2         ; Compare actual and expected.
             JSZ      CONTIN        ; No failure, increment pass count.

             ACT,     S1            ; Save actual result
             EXP,     S2            ; Save expected result
             DIF,     S0            ; Save difference

             S6       ERROR,        ; Increment error count
             S7       1
             S6       S6+S7
             ERROR,   S6

             S6       STOP,         ; check stop flag
             S0       S6\S7
             JSN      CONTIN

             ERR                    ; Stop on error

    CONTIN   =        *
             S6       PASS,         ; Increment pass count
             S7       1
             S6       S6+S7
             PASS,    S6
             J        MAINLOOP
```

The following gives the locations of items within the test code.

	CRAY X-MP Computer System	CEA System
START	200	2000
TESTCODE	500	2100
PASS	24	1104
ERROR	23	1103
ACT	21	1101
EXP	22	1102
DIF	20	1100
STOP	26	1010

Location TESTCODE contains a series of PASS instructions, followed by an unconditional jump to TESTRTN. You can create a test loop by overwriting the PASS instructions at TESTCODE with the suspected failing instructions. Before the jump to TESTRTN, the actual value should be in S1, and the expected value in S2.

5.2.3.3 Environment variables

The **oldmon** environment can be modified by setting certain environment variables. These variables are as follows:

Variable	Description
DMONPATH:	Enter a list of directories to search when opening a file for reading. Separate directories with a colon. When **oldmon** tries to read a file, it first checks the current directory for that file. If the file is not found, **oldmon** checks **$HOME/oldmon.** If the file is not found, the program searches the directories specified by the DMONPATH environment variable. If the file is not found in any of those directories, the program searches the directory **/ce/oldmon.** If the file is still not found, **oldmon** issues an error message.
OLDMON_PRINTER:	Command used to print output. The data to be printed is sent to **stdin** (the command's standard input). If this variable is not defined, **exlp**(1) is used.
TERM:	Terminal type being used. The terminal specified must be defined in the **terminfo**(4F) database.

Set the environment variables before entering **oldmon.** If you are running under the Bourne shell, **sh**(1), enter the following:

```
VAR=value
export VAR
```

If you are running under the C shell, **csh**(1), enter the following:

```
setenv VAR value
```

Examples:

To specify a VT100 terminal type while running under **csh**(1), enter the following at the **csh**(1) prompt:

```
% setenv TERM vt100
```

To specify an **oldmon** search path while running under **sh**(1), enter the following at the **sh**(1) prompt:

```
$ DMONPATH=search-path-one:search-path-two
$ export DMONPATH
```

To specify a different print command while running under **sh**(1), enter the following at the **sh**(1) prompt:

```
$ OLDMON_PRINTER='remsh remsys /usr/ucb/lpr'
$ export OLDMON_PRINTER
```

In the preceding example, the single quotes are necessary because the command contains spaces. When **oldmon** wants to print output, it will execute this command and send the data to be printed to the standard input (**stdin**) of this command. In this example, the **remsh** command will initiate a remote shell on the *remsys* system and execute the **/usr/ucb/lpr** command on the remote system. This allows **oldmon** output to be sent to a printer attached to a remote system. See **remsh**(1) for more information.

5.2.4 DISPLAY MODES

The following subsections describe the **oldmon** display modes:

- Scroll mode display
- Screen mode display

The **oldmon** display contains the following information:

Information	Description
Command menu	Lists input values
Command prompt	Prompts user for information
Error messages	Identifies error condition

Information	Description
CPU status	Displays the following information for the current CPU:

- State of the CPU:
  - Up
  - Down
  - Down, idle
  - Down, running
- Name of the diagnostic in the current CPU
- Program register (P) of the current Exchange Package of the current CPU
- Status bits (S) of the current Exchange Package of the current CPU

  For CRAY X-MP computer systems:

  f fff mm mm c

  f fff indicates flags.
  mm mm indicates mode bits.
  c indicates the cluster number.

  For CEA systems:

  ffff mmmm cc

  ffff indicates flags.
  mmmm indicates modes.
  cc indicates the cluster number.

Down CPU list	List of the down CPUs
Time	Current date and time
Display area	Display area for the portion of central memory associated with the current CPU. The display area can be divided into separate displays, showing different areas of central memory. In addition, each central memory display can be formatted differently. For additional information, refer to subsection 5.2.5.16, View command (v).

5.2.4.1 Scroll mode display

Figure 5-25 shows a scroll mode display.

```
CPU   B: Down, running Name: offcrit                        Wed Oct 19 14:13:14 1988
P             0a        B           0    Downed CPUs:
S  0000 0000 00         L           0         B
    DIB display for olcrit                  00,000,004,000
name        ='olcrit  '                  00 0675543067115135020040 olcrit
rev         ='5.0     '                  01 0324561402004010020040 5.0
date        ='10/12/88'                  02 0304601363046213634070 10/12/88
pass        = 252                        03 0000000000000000000252 ........
error       = 0                          04 0000000000000000000000 ........
seed        = 1206302764022300543002     05 1206302764022300543002 .._@....
failpat     ='onezero '                  06 0000000000000000000000 ........
failcln     = 0                          07 0000000000000000000200 ........
isop        = 1000                          00,000,003,600
numins      = 200                        00 running .......................
                                         04 ...............................
ibuff       12000a S5        S7+S5       10 ...............................
                                         14 ...............................
jbuff       12400a A0        B00         20 single cpu mode................
jbuff       12400b 32300,0   A0          24 ...............................
jbuff       12401a J         B00         30 ...............................
jbuff       12401b ERR                   34 ...............................

A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
```

Figure 5-25. Scroll Mode Display

The following information is displayed (in the order listed):

1. Current CPU status; time; down CPU list

2. Central memory display area

3. Error messages

4. Command menu

5. Command prompt

The following information applies to command line entries:

- Enter commands after the command prompt.
- If a command string is executed, the display scrolls upward and a new display appears.
- If a command is entered without a required argument, the argument menu is displayed with a command prompt. Enter an argument after the prompt. After all commands are executed, the display scrolls upward and a new display appears.

5.2.4.2 Screen mode display

Figure 5-26 shows a screen mode display.

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute


CPU    B: Down, running Name: offcrit                     Wed Oct 19 14:13:14 1988
P              0a         B            0    Downed CPUs:
S  0000 0000 00           L            0         B
    DIB display for olcrit                     00,000,004,000
name       ='olcrit  '                      00 0675543067115135020040 olcrit
rev        ='5.0     '                      01 0324561402004010020040 5.0
date       ='10/12/88'                      02 0304601363046213634070 10/12/88
pass       = 252                            03 0000000000000000000252 ........
error      = 0                              04 0000000000000000000000 ........
seed       = 1206302764022300543002         05 1206302764022300543002 .._@....
failpat    ='onezero '                      06 0000000000000000000000 ........
failcln    = 0                              07 0000000000000000000200 ........
isop       = 1000                              00,000,003,600
numins     = 200                            00 running .......................
                                            04 ...............................
ibuff       12000a S5         S7+S5         10 ...............................
                                            14 ...............................
jbuff       12400a A0         B00           20 single cpu mode................
jbuff       12400b 32300,0    A0            24 ...............................
jbuff       12401a J          B00           30 ...............................
jbuff       12401b ERR                      34 ...............................
```

Figure 5-26. Screen Mode Display

To execute in screen mode, your terminal type must be defined in the **terminfo**(4F) database. See **terminfo**(4F) and **curses**(3X) for more information.

The TERM environment variable sets the default terminal type. If TERM is set to a valid terminal type, **oldmon** executes in screen mode; if not, **oldmon** executes in scroll mode. For information on the TERM environment variable, refer to **sh**(1).

If your terminal type is not defined or is invalid, **oldmon** does not enter screen mode; instead, an error message is displayed.

In screen mode, the display is updated (overwritten) rather than scrolled. The following information is displayed (in the order listed):

1. Command menu
2. Command prompt
3. Error messages
4. Current CPU status; time; down CPU list
5. Central memory display area

The following information applies to command line entries:

- Enter commands after the command prompt.
- If a command string is executed, the entire display is updated.
- If a command is entered without a required argument, the argument menu is displayed with a command prompt. Enter an argument after the prompt. After all commands are executed, the entire display is updated.

5.2.5 PROGRAM COMMANDS

The **oldmon** commands are entered from a front-end terminal or an IOS station console. Figure 5-24 shows the Main menu for **oldmon**.

Unless a complete command string is entered from the Main menu (with all of the required arguments), the program displays various menus with prompts for additional entries. If you enter an invalid argument, the program displays a menu listing the valid arguments. Reenter a valid argument and continue.

Between argument entries, the menu, prompt, and message lines are updated. After a command is executed with all of the required arguments, the entire display is redrawn.

The following guidelines apply to all command entries:

- Select commands from the command menu by entering the first letter of the command. Depending on the command, the program displays various menus with prompts for arguments.

- Enter all inputs in uppercase, lowercase, or a combination of both.

- Press the Return key to receive a prompt for the next required argument or to execute the command if all of the required arguments are entered.

- Enter the less-than key (<) to return to the preceding menu. This allows you to reenter an argument.

- Enter the greater-than key (>) to abort the current command and return to the Main menu.

- Use a semicolon (;) to combine commands. The following applies to a combined command entry:

  - If any of the command entries are incomplete, the program issues a prompt for additional arguments for the first incomplete command.

  - If an error is detected in the command list, the program displays the menu for the first incorrect command. This allows you to reenter the menu commands and any subsequent commands.

  - If you have not yet pressed the Return key to execute the command list, you can abort the last command in the list by pressing the greater-than key (>). All commands in the list are executed except the last entry, and the program returns to the Main menu.

- Use white space (blank spaces, tabs, and newline characters) to indicate the end of an address or file name.

- Enter a pound sign (#) to start a comment in a command buffer.

5.2.5.1 Common arguments

Several of the **oldmon** commands accept the following arguments:

Argument	Description
address	Enter an octal address, or press K (Key) followed by a diagnostic information block (DIB) entry (refer to the off-line diagnostic listings for a list of DIB entries). All addresses are relative to the central memory image of the current CPU. The related menus indicate whether a parcel or word address is expected. - If a parcel address is required, enter the word address followed by a parcel designator (do not leave a space between them). The parcel designator can be **a**, **b**, **c**, or **d**; the default is **a**. - If a parcel address is not required and no parcel designator is specified, the address is assumed to be a word address.
cpu	CPU number. *cpu* is a value in one of the following ranges: **0, 1, 2, 3, 4, 5, 6, 7** or **a, b, c, d, e, f, g, h** The default is the current CPU.
file	Enter a valid file name. Full and relative path names are valid file names. If a relative path name is specified, the program searches for the file in the current directory. If the file is not found, the program uses the DMONPATH environment variable to search. For information on the DMONPATH environment variable, refer to **sh**(1).

Argument	Description
format	Enter one of the following arguments to select the display format for the Dump (**d**) and View (**v**) commands:

Argument	Format
d	DIB format (View command only); displays the DIB of the diagnostic in the current CPU.
i	Instruction format; displays central memory in disassembled instructions. The program issues a prompt for a word or parcel address.
p	Parcel format; displays central memory in 6-digit octal parcels. The program issues a prompt for a word address.
r	Register format (View command only); displays the registers of the current CPU when the CPU is down and idle.
t	Text format; displays central memory in ASCII. The program issues a prompt for a word address.
w	Word format; displays central memory in 22-digit octal words. The program issues a prompt for a word address.
x	Exchange Package format; displays central memory as an Exchange Package (View command only). The program issues a prompt for a word address or an Exchange Package value. The Exchange Package arguments are as follows:

Argument	Exchange Package
c	Current (default)
s	Starting

5.2.5.2 Append (a) and Dump (d) commands

To append or dump a formatted central memory dump to a file (commands **a** and **d**, respectively), use the following command synopses.

Synopsis (Append command):

a *start-address end-address format file*

Synopsis (Dump command):

d *start-address end-address format file*

You must have permission to write to the specified file. The file is created if it does not already exist. Before writing the dump to the file, the program issues a prompt for comments to precede the dump.

To print the dump, enter an asterisk (*) for *file*. See subsection 5.2.3.3, Environment Variables, for more information.

To set append or dump arguments, use the following command synopses.

Synopsis (Append command):

a *argument file*

Synopsis (Dump command):

d *argument file*

argument Enter one of the following values for *argument*:

Argument	Description
d	Appends or dumps the DIB of the diagnostic in the current CPU to *file*
r	Appends or dumps the registers of the current CPU to *file* (the CPU must be down and idle)
s	Appends or dumps the current screen to *file*

5.2.5.3 CPU command (c)

To specify a new default CPU, use the following command synopsis.

Synopsis:

c *cpu*

The default CPU's memory area can be displayed in the memory display area. The Status command is valid for the default CPU only. The Go, Halt, and Load commands assume the default CPU if a different CPU is not specified. The initial default CPU is the first CPU downed from the command line or CPU **a** if no CPU was downed.

5.2.5.4 Enter command (e)

To enter a value at a specific address, use the following command synopsis.

Synopsis:

e *address value*

If *address* is a parcel address and *value* exceeds O'177777, the program displays an error message. Reenter and continue.

5.2.5.5 Execute command (x)

To execute a command buffer containing **oldmon** commands, use the following command synopsis.

Synopsis:

x *file*

5.2.5.6 Fill command (f)

To fill consecutive central memory locations, use either of the following command synopses.

Synopsis:

f *address value...value*

address Indicates the first central memory location to be filled with the first value specified. Each consecutive value is placed in the next consecutive central memory location. Depending on the address specified, the program fills the memory location with words or parcels.

Press the Return key after *address* and after each value. If you press the Return key without first entering a value, the current central memory location remains unchanged and the next value specified is placed in the next consecutive memory location.

To return to the preceding word or parcel location, press the less-than key (<). You can modify the word or parcel value before proceeding to the next location.

To signal the completion of the consecutive entries, enter a period (.) or the greater-than key (>).

To fill memory in a specified range with a specific data pattern, use the following command synopsis:

Synopsis:

fp *start-address end-address value*

If parcel addresses are specified, each parcel in the given range is filled with the given data value. If word addresses are given, the given range of words is filled with the given data value.

5.2.5.7 Go command (g)

To start a test in a CPU, use the following command synopsis.

Synopsis:

g [*cpu*] [*exchange-package*]

exchange-package
Enter one of the following arguments for *exchange-package*:

Argument	Exchange Package	CX/CEA Location	CEA Location
c	Current	120	1200
s	Starting (default)	140	740
address			

If the CPU is not down, the program issues a prompt for you to verify the request to down the CPU. Enter **y** (yes) to down the CPU and start the test. Enter **n** (no) to cancel the Go command.

5.2.5.8 Halt command (h)

To halt test execution in a down CPU, use the following command synopsis.

Synopsis:

h [*cpu*]

The CPU idles until the Go or Up command is executed.

5.2.5.9 Load command (l)

To load a test into a CPU's central memory buffer, use the following command synopsis.

Synopsis:

l [*cpu*] *address file*

file Enter one of the following arguments for *file*:

Argument	Description
file	File containing the test to be executed
*	Test loop

5.2.5.10 Options command (o)

To set test options, use the following command synopsis.

Synopsis:

o *option argument*

option The values for *option* are as follows (the *argument* value is dependent on *option*):

Option	Description
c	Generates a display that is continuously refreshed at a specified interval (in seconds). Use the following command synopsis: **o c** *seconds* *seconds* is the number of seconds; a value in the range **1** through **9**. To return to the Main menu, an interrupt must be sent to **oldmon**. Typically, pressing the Control-C keys sends an interrupt to **oldmon**. See the appropriate front-end station guide and **stty**(1).
d	Downs a specified CPU. Use the following command synopsis: **o d** *cpu*

option (continued):

Option	Description
	cpu defaults to the current CPU. The CPU is downed and left idle. (The Go command also downs the CPU.)
l	Sets a new limit address for the current CPU. Use the following command synopsis: **o l** *address* The new limit address is rounded up to the next O'1000 word boundary.
t	Specifies the terminal type (required for screen mode; refer to subsection 5.2.4.2, Screen mode display). Use the following command synopsis: **o t** *type* *type* is one of the terminal types defined in the **terminfo**(4F) database. The TERM environment variable sets the default terminal type. For information on the TERM environment variable, refer to **sh**(1).

5.2.5.11 Quit command (q)

To exit **oldmon**, enter one of the following commands:

Command	Description
eof	End-of-file (typically, press the Control-d keys). Enter from any menu. A prompt is displayed before the request is processed. To verify or cancel the request, enter **y** (yes) or **n** (no), respectively.
q	Quit. Enter from the Main menu only. A prompt is displayed before the request is processed. To verify or cancel the request, enter **y** (yes) or **n** (no), respectively.

5.2.5.12 Redraw command (r)

To redraw the display, enter **r**.

5.2.5.13 Shell escape command (!)

To execute a shell command, use the following command synopsis.

Synopsis:

> **!** [*shell-command*]

The **oldmon** monitor will execute *shell-command* in a subshell. If *shell-command* is omitted, **oldmon** will execute **/bin/sh**. You must exit this shell to continue **oldmon**. See **sh**(1) for more information.

5.2.5.14 Status command (**s**)

To update the current Exchange Package of the current CPU, enter **s**. If the current CPU is not down, an error message is displayed.

5.2.5.15 Up command (**u**)

To return a down CPU to normal system operations, use the following command synopsis.

Synopsis:

> **u** [*cpu*]

5.2.5.16 View command (**v**)

To view a formatted area of central memory on all or part of the display area, use the following command synopsis.

Synopsis:

> **v** *display format address*

> *display* Enter one of the following arguments for *display*:

Argument	Description
f	Full display
l	Left half of the display
r	Right half of the display
tl	Top left quadrant
tr	Top right quadrant
bl	Bottom left quadrant
br	Bottom right quadrant

To display the DIB of the current diagnostic, use the following synopsis.

Synopsis:

v *display* **d** *argument*

argument Enter one of the following arguments:

Argument	Description
RETURN	Displays the DIB starting at the beginning.
d	Displays the differences section of the DIB (confidence tests only)
k key	Displays the DIB starting with DIB

To display the current values of the CPU's registers, use the following synopsis.

Synopsis:

v *display* **r**

To scroll the display areas forward or backward, use the plus (+) or minus (-) parameters, respectively. The command synopses are as follows.

Synopsis:

v [*display*] +[*n*] or **v** [*display*] -[*n*]

display Enter the display to be scrolled. If omitted, all display areas are scrolled.

n Number of lines to scroll. The default for *n* is 8 if display is **tl**, **tr**, **bl**, or **br**. Otherwise, the default is 16 (the number of lines in the display area).

5.2.5.17 Write command (w)

To write an area of central memory to a binary file, use the following command synopsis.

Synopsis:

w *start-address end-address file*

5.2.6 PROGRAM EXAMPLE

This subsection contains a commented **oldmon** execution example.

Example:

```
$ oldmon -d b
Do you really want to down CPU b?
Type y or n> y

     ***************************************************************

       The -d b command line option requests that oldmon down
       CPU B immediately.  Enter y to confirm the request.

     ***************************************************************

Cannot find configuration file oldmon.cf, should I initialize it?
Enter Yes or No (y/n)> y

     ***************************************************************

       The oldmon monitor cannot locate the configuration file
       oldmon.cf.  Enter y to initialize oldmon.cf.

     ***************************************************************
```

Example (continued):

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
v

CPU    B: Down, idle      Name: ** none **                  Wed Oct 19 13:21:18 1988
P             0a          B             0    Downed CPUs:
S   0000 0000 00          L             0         B


              OLDMON Version 1.0 - Online Down CPU Monitor

                  CRAY Y-MP Down CPU Monitor for the
                       UNICOS Operating System.

       Copyright (c) Cray Research, Inc. Unpublished - All rights
         reserved under the copyright laws of the United States.

                          CRAY PROPRIETARY
```

****

The Main menu for **oldmon** is displayed. CPU B is the default CPU. It is displayed as down and idle. Enter **v** to set the View command.

****

```
Display: Full, Top, Bottom, Left, Right; Scroll: + -
View 1
```

****

The choice of input values is displayed. Enter **1** to select the left half of the screen as the display area.

****

Example (continued):

```
Format: Dib, Instr, Parcel, Word, Register, Text, eXchange pkg; Scroll: + -
View Left in d
```

****

The choice of input values is displayed. Enter **d** to select the DIB format.

****

```
RETURN for DIB; Differences; Key
View Left in DIB format RETURN
```

****

The choice of input values is displayed. Press RETURN to display the beginning of the DIB.

****

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
1

CPU   B: Down, idle     Name: ** none **                 Wed Oct 19 13:22:49 1988
P           0a          B           0    Downed CPUs:
S  0000 0000 00         L           0         B
    DIB display unavailable
```

Example (continued):

****

The Main menu for **oldmon** is redisplayed. Enter **1** to load a diagnostic into the common memory buffer for CPU B.

****

Enter word address
Load cpu B at 0 **RETURN**

****

Enter the address within the buffer where the diagnostic is to be loaded. Pressing RETURN without entering an address will default to zero.

****

Enter file name, * for testloop
Load cpu B at 0 from **offcrit**

****

Enter a file name. In this example, **offcrit** (off-line version of **olcrit**) is specified.

****

Example (continued):

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
v r w 4000

CPU   B: Down, idle    Name: offcrit                     Wed Oct 19 13:24:39 1988
P             0a       B           0    Downed CPUs:
S  0000 0000 00        L           0         B
    DIB display for olcrit
name       ='olcrit  '
rev        ='5.0     '
date       ='10/12/88'
pass       = 0
error      = 0
seed       = 33
lmstart    = 0
failpat    ='         '
isop       = 1000
numins     = 200

ibuff  17000a EXIT      00

jbuff  17400a EXIT      00

inita0     = 0000000000000000000000
inita1     = 0000000000000000000000
```

****

The command string to set the right half of the display is entered. The blank space between each entry is optional.

1. Enter **v** to select the View command.

2. Enter **r** to select the right half of the display.

3. Enter **w** to select word format.

4. Enter **4000** to specify the display address.

****

Example (continued):

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
e

CPU   B: Down, idle     Name: offcrit                       Wed Oct 19 14:10:33 1988
P             0a         B           0    Downed CPUs:
S  0000 0000 00          L           0         B
    DIB display for olcrit                   00,000,004,000
name      ='olcrit  '                     00 0675543067115135020040 olcrit
rev       ='5.0     '                     01 0324561402004010020040 5.0
date      ='10/12/88'                     02 0304601363046213634070 10/12/88
pass      = 0                             03 0000000000000000000000 ........
error     = 0                             04 0000000000000000000000 ........
seed      = 33                            05 0000000000000000000033 ........
failpat   ='        '                     06 0000000000000000000000 ........
failcln   = 0                             07 0000000000000000000200 ........
isop      = 1000
numins    = 200                           10 1000000000000000037777 ......?.
                                          11 0000000000000000000000 ........
ibuff      12000a ERR                     12 0000000000000000000007 ........
                                          13 0000000000000000000000 ........
jbuff      12400a ERR                     14 0000000000000000000000 ........
                                          15 0000000000000000000000 ........
inita0    = 0000000000000000000000        16 0000000000000000001000 ........
inita1    = 0000000000000000000000        17 0000000000000000000000 ........
```

****

The new display is shown. Use the Enter command to set a location within the memory buffer.

****

Key <address>
Enter at **k**

****

Enter a **k** to specify that a DIB key will be given for the entry location.

****

Example (continued):

```
Enter <key> <key+offset>; Press RETURN when complete
Enter at Key seed
```

****

Enter **seed** to specify that the seed DIB entry is to be used.

****

```
The current value at Key seed is 0000000000000000000033
Enter at Key seed the value of   1206302764022300543002
```

****

Enter the value **1206302764022300543002.** Presumably, this is the seed from an on-line failure of **olcrit.**

****

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
e 4017 1

CPU   B: Down, idle    Name: offcrit                  Wed Oct 19 14:12:59 1988
P            0a        B           0    Downed CPUs:
S  0000 0000 00        L           0        B
    DIB display for olcrit                 00,000,004,000
name       ='olcrit  '                  00 0675543067115135020040 olcrit
rev        ='5.0     '                  01 0324561402004010020040 5.0
date       ='10/12/88'                  02 0304601363046213634070 10/12/88
pass       = 0                          03 0000000000000000000000 ........
error      = 0                          04 0000000000000000000000 ........
seed       = 1206302764022300543002     05 1206302764022300543002 .._@....
failpat    ='        '                  06 0000000000000000000000 ........
failcln    = 0                          07 0000000000000000000200 ........
isop       = 1000
numins     = 200                        10 1000000000000000037777 ......?.
                                        11 0000000000000000000000 ........
ibuff       12000a ERR                  12 0000000000000000000007 ........
                                        13 0000000000000000000000 ........
jbuff       12400a ERR                  14 0000000000000000000000 ........
                                        15 0000000000000000000000 ........
inita0     = 0000000000000000000000     16 0000000000000000001000 ........
inita1     = 0000000000000000000000     17 0000000000000000000000 ........
```

Example (continued):

****

The Enter command is used again to enter a **1** at location 4017. This sets the repeat flag for **offcrit**. (Refer to the **offcrit** listing for more information.)

****

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
g

CPU   B: Down, idle     Name: offcrit                        Wed Oct 19 14:12:59 1988
P            0a         B            0    Downed CPUs:
S  0000 0000 00         L            0        B
    DIB display for olcrit                   00,000,004,000
name       ='olcrit  '                    00 0675543067115135020040 olcrit
rev        ='5.0     '                    01 0324561402004010020040 5.0
date       ='10/12/88'                    02 0304601363046213634070 10/12/88
pass       = 0                            03 0000000000000000000000 ........
error      = 0                            04 0000000000000000000000 ........
seed       = 1206302764022300543002       05 1206302764022300543002 .._@....
failpat    ='        '                    06 0000000000000000000000 ........
failcln    = 0                            07 0000000000000000000200 ........
isop       = 1000
numins     = 200                          10 1000000000000000037777 ......?.
                                          11 0000000000000000000000 ........
ibuff       12000a ERR                    12 0000000000000000000007 ........
                                          13 0000000000000000000000 ........
jbuff       12400a ERR                    14 0000000000000000000000 ........
                                          15 0000000000000000000000 ........
inita0     = 0000000000000000000000       16 0000000000000000001000 ........
inita1     = 0000000000000000000000       17 0000000000000000000001 ........
```

****

The Go command is entered to start the diagnostic executing in CPU B.

****

Example (continued):

```
Press RETURN to continue
Go cpu B  with starting Exchange Package
```

****

The format of the Go command is: **g** *cpu exchange-package*. Press the RETURN key to process the Go command in CPU B (default), using the starting Exchange Package (default). Alternatively, you could have entered **g b s** from the Main menu.

****

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
v t r t 3600; v b r d d

CPU   B: Down, running Name: offcrit                     Wed Oct 19 14:13:14 1988
P             0a        B            0    Downed CPUs:
S  0000 0000 00         L            0         B
    DIB display for olcrit                    00,000,004,000
name      ='olcrit  '                     00 0675543067115135020040 olcrit
rev       ='5.0     '                     01 0324561402004010020040 5.0
date      ='10/12/88'                     02 0304601363046213634070 10/12/88
pass      = 252                           03 0000000000000000000252 ........
error     = 0                             04 0000000000000000000000 ........
seed      = 1206302764022300543002        05 1206302764022300543002 .._@....
failpat   ='onezero '                     06 0000000000000000000000 ........
failcln   = 0                             07 0000000000000000000200 ........
isop      = 1000
numins    = 200                           10 1000000000000000037777 ......?.
                                          11 0000000000000000000000 ........
ibuff      12000a S5        S7+S5         12 0000000000000000000007 ........
                                          13 0000000000000000000000 ........
jbuff      12400a A0        B00           14 0000000000000000000001 ........
jbuff      12400b 32300,0   A0            15 0000000000000000000000 ........
jbuff      12401a J         B00           16 0000000000000000001000 ........
jbuff      12401b ERR                     17 1777777777777777777777 ........
```

Example (continued):

****

The **offcrit** test is executing in CPU B. Note that P, S, B, and L are still zero. They are only updated when the down CPU performs an exchange. The Main menu for **oldmon** is redisplayed. Use a command string to set the View command to view the message display area, and the differences section of the DIB:

1. Enter **v t r t 3600** to execute the command View Top Right Text at 3600.
2. Enter **;** to separate the two commands.
3. Enter **v b r d d** to execute the command View Bottom Right Dib Differences.

****

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
o c 3

CPU   B: Down, running Name: offcrit                      Wed Oct 19 14:13:31 1988
P              0a         B            0    Downed CPUs:
S  0000 0000 00           L            0         B
    DIB display for olcrit                     00,000,003,600
name       ='olcrit  '                      00 running .......................
rev        ='5.0     '                      04 ...............................
date       ='10/12/88'                      10 ...............................
pass       = 1342                           14 ...............................
error      = 0                              20 single cpu mode................
seed       = 12063027640223005430O2         24 ...............................
failpat    ='bits    '                      30 ...............................
failcln    = 0                              34 ...............................
isop       = 1000                              DIB display for olcrit
numins     = 200

ibuff      12000a S5         S7+S5

jbuff      12400a A0         B00
jbuff      12400b 32300,0    A0
jbuff      12401a J          B00
jbuff      12401b ERR
```

Example (continued):

****

To generate a continuous display that is refreshed at a specific interval, use a command string to set the Options command:

1. Enter **o** the select the Options command.
2. Enter **c** to select continuous display mode.
3. Enter **3** to specify a 3-second interval.

****

```
Console interrupt to continue
Options, Continuous display update 3 seconds

CPU   B: Down, running Name: offcrit                    Wed Oct 19 14:13:39 1988
P            0a        B            0    Downed CPUs:
S  0000 0000 00        L            0       B
    DIB display for olcrit                  00,000,003,600
name      ='olcrit  '                    00 running .......................
rev       ='5.0     '                    04 ...............................
date      ='10/12/88'                    10 ...............................
pass      = 1714                         14 ...............................
error     = 0                            20 single cpu mode................
seed      = 1206302764022300543002       24 ...............................
failpat   ='random  '                    30 ...............................
failcln   = 0                            34 ...............................
isop      = 1000                             DIB display for olcrit
numins    = 200

ibuff      12000a S5         S7+S5

jbuff      12400a A0         B00
jbuff      12400b 32300,0    A0
jbuff      12401a J          B00
jbuff      12401b ERR
```

****

The **oldmon** monitor will now update the display every three seconds. This will continue until the CPU exits, or an interrupt is sent to **oldmon**.

****

Example (continued):

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
d d crit.dump
CPU B error exit
CPU    B: Down, idle     Name: offcrit                      Wed Oct 19 14:14:53 1988
P          2527c         B    10666000    Downed CPUs:
S  0002 1670 00          L    10746400         B
    DIB display for olcrit                        00,000,003,600
name       ='olcrit  '                    00 cpu(s) halted - max error reache
rev        ='5.0     '                    04 d.............................
date       ='10/12/88'                    10 ..............................
pass       = 32731                        14 ..............................
error      = 1                            20 single cpu mode...............
seed       = 12063027640223005430002      24 ..............................
lmstart    = 0                            30 ..............................
failpat    ='bits    '                    34 ..............................
isop       = 1000                            DIB display for olcrit
numins     = 200                          s6         E= 0000000000000000000001
 nrandom instruction buffer                          A= 0000000000000000000000
ibuff       12000a S5       S7+S5                    D= 0000000000000000000001
ibuff       12000b PASS                   v0    +000E= 0000000000000000000001
ibuff       12000c A0       S5                       A= 0000000000000000000000
ibuff       12000d A6       00000032267              D= 0000000000000000000001
ibuff       12001c V7       V5*IV7        v0    +001E= 0000000000000000000001
ibuff       12001d A3    37777777757,A6              A= 0000000000000000000000
```

****

The **offcrit** test detected an error and exited. The **oldmon** monitor automatically ends continuous display mode. In order to dump the DIB to a file for further analysis, the Dump command, **d d crit.dump**, is used.

****

Example (continued):

```
A/Dump Cpu Enter Fill Go Halt Load Opts Quit Redraw Stat Up View Write Xecute
q

CPU   B: Down, idle     Name: offcrit                      Wed Oct 19 14:15:42 1988
P          2527c        B     10666000    Downed CPUs:
S  0002 1670 00         L     10746400         B
    DIB display for olcrit                   00,000,003,600
name        ='olcrit  '                   00 cpu(s) halted - max error reache
rev         ='5.0     '                   04 d...............................
date        ='10/12/88'                   10 ................................
pass        = 32731                       14 ................................
error       = 1                           20 single cpu mode.................
seed        = 1206302764022300543002      24 ................................
lmstart     = 0                           30 ................................
failpat     ='bits    '                   34 ................................
isop        = 1000                            DIB display for olcrit
numins      = 200                         s6          E= 0000000000000000000001
 nrandom instruction buffer                           A= 0000000000000000000000
ibuff       12000a S5        S7+S5                    D= 0000000000000000000001
ibuff       12000b PASS                   v0    +000E= 0000000000000000000001
ibuff       12000c A0        S5                       A= 0000000000000000000000
ibuff       12000d A6        00000032267              D= 0000000000000000000001
ibuff       12001c V7        V5*IV7       v0    +001E= 0000000000000000000001
ibuff       12001d A3     37777777757,A6              A= 0000000000000000000000
```

****

The quit command is used to exit **oldmon**.

****

Do you really want to quit?
Type y or n>**y**

****

Enter a **y** to confirm the quit. Note that CPU B will be left down since it was not explicitly returned to UNICOS with the Up command.

****

5.2.7 PROGRAM MESSAGES

This subsection lists the **oldmon** messages in alphabetical order.

Address *addr* exceeds limit address
This message is associated with the Enter (**e**) command. Reenter a valid address to continue.

Cannot access printer
If the OLDMON_PRINTER environment variable is set, its value is not a valid command. If OLDMON_PRINTER is not set, the command **exlp** cannot be executed.

Cannot allocate memory
This message is associated with the Load (**l**) or Options (**o**) command.

Cannot dump DIB of the loaded diagnostic
This message is associated with the Append (**a**) or Dump (**d**) command.

Cannot fill memory outside of buffer
This message is associated with the Fill (**f**) command. Reenter the Fill command.

Cannot find DIB entry *x*
This message is associated with the Enter (**e**) or Fill (**f**) command.

CPU *n* interrupts: *list*
This message lists all the interrupts for CPU *n*.

CPU *n* is already down
The **oldmon** monitor tried to down a CPU that it has downed already. Indicates an internal **oldmon** error. Contact your CRI representative.

CPU *n* is not down
This message is associated with the Status (**s**) or Up (**u**) command.

CPU *n* registers are unavailable and cannot be dumped
Registers cannot be dumped unless the current CPU is down and idle. This message is associated with the Append (**a**) or Dump (**d**) command.

Exception condition: *caught signal*
Refer to **signal**(2).

Exchange Package is not in the CPU's memory
This message is associated with the Go (**g**) command.

File *file* is empty
An empty file was specified when loading a diagnostic.

File *file*: *system error message*
The **oldmon** monitor had an error while accessing, reading, or writing *file*.

Invalid input *input*
The **oldmon** monitor received unexpected input.

The *ioctl-request* ioctl failed for *cpu-device*: errno *n*: *system error message*
The **oldmon** monitor made the specified request to UNICOS and the request failed.

plock: errno *n*: *system error message*
The **oldmon** monitor made a request to be locked in memory and the request failed.

Second address must be greater than first address
This message is associated with the Append (**a**), Dump (**d**), Fill (**f**), or Write (**w**) command.

Single CPU system; cannot down a CPU.
The **oldmon** monitor does not allow downing a CPU on a single CPU system.

Terminal type not set, cannot use screen mode
The TERM environment variable was not set when **oldmon** was started.

Unable to configure loaded diagnostic
This message is associated with the Load (**l**) command.

Unknown terminal *terminal*; cannot use screen mode
terminal is not defined in the **terminfo**(4F) database.

Value exceeds parcel size
This message is associated with the Enter (**e**) or Fill (**f**) command. *value* must not exceed O'**177777**.

5.3 unitap

The **unitap**[†] test is an on-line magnetic tape test that allows you to test up to 8 tape paths in parallel. It is supported in a standard configuration. You can execute **unitap** interactively or from a UNICOS shell script.[††] Interactive execution is menu-driven, with a 240-character command buffer. From each menu, you can access all of the other menus.

All user input and output is saved in a trace file for later evaluation.

To simulate passing and failing test execution examples without removing the tape device from normal system operations, you can execute **unitap** in Learn mode.

The **unitap** testing options are as follows:

Testing Option	Description
All tape tests	All of the tape tests (test sections) are executed (run time: approximately 3 minutes).
Two-channel conflict tests	A selection of tape tests are executed in parallel to exercise 2 tape paths (run time: approximately 10 minutes). The tests verify whether the channels can withstand conflict.
Three-channel conflict tests	A selection of tape tests are executed in parallel to exercise 3 tape paths (run time: approximately 10 minutes). The tests verify whether the channels can withstand conflict.
Canned test	A user-selected test is executed (for example, a byte counter test).
Test loop	A user-defined test is executed (refer to subsection 5.3.4.6, Programming Tool).

For additional information, refer to subsection 5.3.3.3, Test Menu.

---

† CX/CEA systems only.
†† Execution from a shell script is deferred.

In addition to providing error detection capabilities, **unitap** provides the following troubleshooting tools:

Troubleshooting Tool	Description
Breakpoint	Sets breakpoints in the tape tests
Channel Commands†	Issues channel commands
Compare Data Buffer	Displays data miscomparisons for the write and read data buffers
Display Memory	Displays the write and read central memory data buffers, and allows you to modify the write buffer
System Call History	Displays a history of the last 15 system calls and the last 10 *events* that preceded the current *event*. An *event* is defined as any of the following actions: - A failure occurs - A breakpoint is reached
Programming	Builds test loops
Packet Status	Displays the status of the last packet sent for each channel at the time of the last 10 *events* that preceded the current *event*

For additional information, refer to subsection 5.3.4, Debug Tools.

5.3.1 PROGRAM SYNOPSIS

You can execute **unitap** interactively or from a UNICOS shell script.†† This subsection describes how to execute **unitap** from a shell script. For a description of interactive execution, refer to subsection 5.3.2, Interactive Program Execution.

† Deferred implementation.
†† Execution from a shell script is deferred.

5.3.2 INTERACTIVE PROGRAM EXECUTION

Interactive execution is menu-driven, with a 240-character command buffer. From each menu, you can access all of the other menus.

Menu options can be entered in uppercase or lowercase.

5.3.3 PROGRAM MENUS

This subsection provides a summary of the **unitap** menu system. The following menus are described.

- Main menu
- Variable menu
- Test menu
- Canned Test menu
- Debug menu
- Global Options menu
- Hardware Layout menu

5.3.3.1 Main Menu

The Main Menu is displayed when **unitap** is initialized or when you enter **MN** from any menu (refer to figure 5-27).

```
unitap                          Main Menu

   Option                 Description

     D                    Debug Menu
     T                    Test Menu
     V                    Variable Menu

     G                    Global Options Menu
     W                    Program notes

   EXIT                   Exit the diagnostic
   HELP option            Information on option

   Note:  these menu options are global (valid from all menus).
```

Figure 5-27. Main Menu for **unitap**

The menu options are as follows:

Option	Description
D	Debug Menu (refer to subsection 5.3.3.5)
T	Test Menu (refer to subsection 5.3.3.3)
V	Variable Menu (refer to subsection 5.3.3.2)
G	Global Options Menu (refer to subsection 5.3.3.6)
W	Program notes
EXIT	Exit the diagnostic; channels dedicated to on-line diagnostic testing are released.
HELP *option*	Information on *option*

5.3.3.2 Variable Menu

The Variable Menu is displayed when you enter V from any menu (refer to figure 5-28).

---

unitap Variable Menu

Path 1 CH=20, CO=0, DV=*dv*, DN=6250, PC=1

Option	Description
CH *n*	Channel number (20-33 octal)
CO *n*	Controller number (0-F hexadecimal)
DN *n*	Density value (800, 1600, or 6250, CART)
DV *dv*	Device number (0-FFF ASCII)
P*n*	Path (1-8)
PC *n*	Pass count (decimal)
RL	Release the dedicated (reserved) path for the tape unit
G	Global Options Menu
R	Previous menu

Note: these menu options are global (valid from all menus).

Figure 5-28. Variable Menu

Each option is briefly described in the Variable Menu. However, the following descriptions provide further clarification:

Option	Description
CH *n*	Channel number. *n* is a value in the range O'20 through O'33. The default for *n* is O'20 through O'27, for paths 1 through 8, respectively.
CO *n*	Controller number. *n* is a value in the range 0 through F (hexadecimal). The default for *n* is 0.
DN *n*	Density value. *n* is one of the following values: 800, 1600, or 6250 (default), CART.
DV *dv*	Device number (required). *n* is a site-defined ASCII value.

Option	Description
P*n*	Path under test (channel, controller, and device). *n* is a value in the range 1 through 8. The default for *n* is 1.
PC *n*	Pass count. The default for *n* is 1.
RL	Release the dedicated path for the tape unit.

5.3.3.3 Test Menu

The Test Menu is displayed when you enter **T** from any menu (refer to figure 5-29).

```
unitap                      Test Menu


   Path 1   CH=20, CO=0, DV=dv, DN=6250, PC=1


   Option             Description

     A                Execute all the tape tests
     C                Display the Canned Test Menu
     2                Execute the two-channel conflict tests
     3                Execute the three-channel conflict tests

     G                Global Options Menu
     R                Previous menu

   Note:  these menu options are global (valid from all menus).
```

Figure 5-29. Test Menu

The menu options are as follows:

Option	Description
A	All tape tests. All of the tape tests are executed (run time: approximately 3 minutes).
2	Two-channel conflict tests. A selection of tape tests are executed in parallel to exercise 2 tape paths (run time: approximately 10 minutes). The tests verify whether the channels can withstand conflict.
3	Three-channel conflict tests. A selection of tape tests are executed in parallel to exercise 3 tape paths (run time: approximately 10 minutes). The tests verify whether the channels can withstand conflict.
C	Canned test. A user-selected test is executed (for example, a byte counter test).

5.3.3.4 Canned Test Menu

The Canned Test Menu is displayed when you enter C from any menu (refer to figure 5-30).

---

unitap Canned Test Menu

Path 1 CH=20, CO=0, DV=*dv*, DN=6250, PC=1

Option	Description
AC	All basic commands tests (except Read)
BC	Byte counter test (transfers up to 4 kbytes)
BF	Buffer tests (R/W 64 bits)
BN	Next byte counter test (transfers 4 to 8 kbytes)
BS	Bus test (R/W 8 bits)
LA	Ladder tests
RB	Random buffer tests (R/W 64 random bits)
ST	Stress test
TP	Tape position commands tests
G	Global Options Menu
R	Previous menu

Figure 5-30. Canned Test Menu

The menu options are as follows:

Option	Description
AC	All basic commands tests. Tests the rewind, write, write tape mark, forward block, backward block, forward tape mark, and backward tape mark tape movement commands.
BC	Byte counter test. Writes and reads 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096 bytes to the tape.
BF	Buffer tests. Writes and reads 64-bit patterns to the tape.
BN	Next byte counter test. Writes and reads 1 sector (4096 bytes) plus 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096 bytes to the tape.
BS	Bus test. Writes and reads 8-bit patterns to the tape.
LA	Ladder tests. Writes and reads 1, 2, 3, 4, 5, 6, 7, and 8 sectors to the tape.
RB	Random buffer tests. Writes and reads random data patterns to the tape.
ST	Stress test
TP	Tape position commands tests. Writes patterns to the tape, issues tape positioning commands, and then reads the patterns to verify that the positioning commands work.

5.3.3.5 Debug Menu

The Debug Menu is displayed when you enter **D** from any menu (refer to figure 5-31).

---

unitap Debug Menu

Option	Description
B	Breakpoint Tool
CC†	Channel Commands Tool
CD	Compare Data Buffer Tool
E	Fail execution (Learn mode)
H	System Call History Tool
L	Learn mode/System mode (toggle)
LO	Hardware Layout Menu
M	Memory Tool (Central Memory)
PG	Programming Tool
S	Packet Status Tool
G	Global Options Menu
R	Previous menu

Note: these menu options are global (valid from all menus).

---

† Deferred implementation

Figure 5-31. Debug Menu

For additional information, refer to subsection 5.3.4, Debug Tools.

5.3.3.6 Global Options Menu

The Global Options Menu is displayed when you enter G from any menu (refer to figure 5-32).

unitap Global Options Menu

Option	Description	Option	Description
A	All confidence tests	M	Memory Tool
B	Breakpoint Tool	MN	Main menu
C	Canned Test Menu	PG	Programming Tool
CB *n*	Command buffer pass count	PC *n*	Pass count (decimal)
CC†	Channel Commands Tool	P*n*	Path (1-8)
CD	Compare Data Buffer Tool	PT	Print screen
CH *n*	Channel number		
CO *n*	Controller number	RL	Release path
D	Debug Menu	RT	Return from breakpoint
DN *n*	Density value	S	Packet Status Tool
DV *n*	Device number	T	Test Menu
E	Error mode (Learn mode)	V	Variable Menu
H	System Call History Tool	W	Program notes
		2	Two-channel conflict test
L	Learn mode/System mode	3	Three-channel conflict test
LO	Display layout		
EXIT	Exit diagnostic	HELP *option*	Information on *option*
R	Previous menu		

† Deferred implementation

Figure 5-32. Global Options Menu

5.3.3.7 <u>Hardware Layout Menu</u>

The Hardware Layout Menu is displayed when you enter **LO** from any menu (refer to figure 5-33).

---

unitap Hardware Layout Menu

Option	Description
D	Debug Menu
BM	Block Multiplexer layout

Figure 5-33. Hardware Layout Menu

The Block Multiplexer Layout Menu for a BMC-5 is displayed when you enter **BM** from the Hardware Layout Menu (refer to figure 5-34).

---

unitap Block Multiplexer Layout Menu (BMC-5)

Option	Description
D	Debug Menu
BM	Block Multiplexer layout

Figure 5-34. Block Multiplexer Layout Menu (BMC-5)

5.3.4 DEBUG TOOLS

The **unitap** debug tools can be selected from any menu. These tools are as follows:

Tool	Menu Option
Breakpoint	B
Channel Commands†	CC
Memory Buffer	M
Compare Data Buffer	CD
System Call History	H
Programming	PG
Packet Status	S

These tools are described in the subsections that follow.

† Deferred implementation

5.3.4.1 Breakpoint Tool

The Breakpoint Tool is displayed when you enter **B** from any menu (refer to figure 5-35). This tool allows you to set a breakpoint immediately preceding or following a system call in a test. When the breakpoint is reached, the user's keyboard input is executed.

If an error is detected, information relating to the *event* is displayed. An *event* is defined as any of the following actions: a failure occurs or a breakpoint is reached. Use the System Call History and Packet Status tools to display additional information regarding an event.

---

unitap Breakpoint Tool

Breakpoint = 0 Breakpoint pass count = 1

When breakpoint is reached, the user's keyboard input is executed.

message displayed on error
Event *n* occurred after *y* system calls.

Option	Description
BP *n*	Execute a breakpoint on pass *n*
BR *n*	Set or clear a breakpoint. *n* is one of the following breakpoint numbers: 0 - Clear the breakpoint 1 - Set breakpoint prior to the system call 2 - Set breakpoint after the system call
RT	Return to test after a breakpoint (global option)
D	Debug Menu
G	Global Options Menu
R	Previous menu

Figure 5-35. Breakpoint Tool

5.3.4.2 Channel Commands Tool

The Channel Commands Tool† is displayed when you enter CC from any menu (refer to figure 5-36). This tool allows you to issue channel commands to the tape device, and to display channel status. For additional information on the channel commands, refer to the APML Reference Card for COS and UNICOS, CRI publication SQ-0059.

```
unitap                     Channel Commands Tool


   Path 1   CH=20, CO=0, DV=dv, DN=6250, PC=1

   LMAR0        = 123456     Bus in  = 123001
   LMAR1        = 123457     Tags in = 377
   Byte counter = 1000       Flags   = IDLE

 --------------------------------------------------------------------------

  Command  Description           Command  Description

   00      Clear chan control     11      Read byte counter register
   01      Reset channel          12      Read bus and status
   02      Send command           13      Read input tags
   03      Read address           14 n    Write LMAR (n: accumulator value)
   04      Single byte I/O        15 n    Write BC   (n: accumulator value)
   05      Run diagnostics        16 n    Enter Addr (n: accumulator value)
   10      Read LMAR              17 n    Write tags (n: accumulator value)

  R  Previous menu
  G  Global Options Menu
```

Figure 5-36. Channel Commands Tool

† Deferred implementation

5.3.4.3 Display Data Buffer Tool

The Display Data Buffer Tool is displayed when you enter **M** from any menu (refer to figure 5-37). This tool allows you to display the read and write data buffers, and to modify the write data buffer. Each data buffer is 16 Kwords.

```
unitap                  Display Data Buffer Tool

   message displayed on error

Write Address = 0                   Read Address = 0
 0 000000 000000 020124 044145       0  000000 000000 000000 000000
 1 000000 000000 070565 064553       1  000000 000000 000022 153207
 2 000000 000000 041162 067556       2  000000 000000 000045 126416
 3 000000 000000 021106 067570       3  000000 000000 000070 101625
 4 000000 000000 045155 070144       4  000000 000000 000113 055034
 5 000000 000000 000000 170435       5  000000 000000 000136 030243
 6 000000 000000 000001 020526       6  000000 000000 000161 003452
 7 000000 000000 000001 050617       7  000000 000000 000203 156661

------------------------------------------------------------------------
  Option                  Description
 DA n               Display address
 DF DB DP DW        Display Forward or Back in Parcel or Word format
 DI DO DD DX        Display in Ascii,Octal,Decimal,Hex
 ST  SS  SP  SK     Store adr data,Store Seeded random,Store Pattern,Store
                      Skip
 CP  LP  LN         Copy a block of data, Locate Pattern, Locate a
                      non-pattern
```

Figure 5-37. Display Data Buffer Tool (1 of 2)

unitap Display Data Buffer Tool

Command	Description
CP *addr1 addr2 n*	Copy *n* words from *addr1* to *addr2*
LP *addr pattern*	Search for *pattern* starting at *addr*
SK *addr data n y*	Store *data* in *n* words (skip *y* words between stores), starting at *addr*
SP *addr data n*	Store *data* consecutively in *n* words, starting at *addr*
SS *addr seed n*	Store random data consecutively in *n* words, starting at *addr*, using *seed* to start the random number generator
ST *addr data*	Store *data* in *addr*
D/DR/DL *addr*	Display full/right/left screen starting at *addr*
D*x*/DR*x*/DL*x*	Display *x*: F (forward), B (backward), A (ASCII), O (octal), D (decimal), X (hexadecimal), P (parcel), W (word)

Figure 5-37. Display Data Buffer Tool (2 of 2)

5.3.4.4 Compare Data Tool

The Compare Data Tool is displayed when you enter **CD** from any menu (refer to figure 5-38). This tool allows you to display the read and write data buffers, and exclusive ORs (logical differences) for the Write and Read address comparisons. Each data buffer is 16 Kwords.

```
unitap                   Compare Data Tool

    The Read compare grid is the Exclusive OR  (or logical difference) of
    the data at the Write grid address and the data at the Read grid
    address.


Write Address = 0                       READ COMPARE Address = 0

0 000000 000000 020124 044145    0                     20124 044145
1 000000 000000 705654 064553    1                     70547 137754
2 000000 000000 041162 067556    2                     41127 141140
3 000000 000000 021106 067570    3                     21176 166355
4 000000 000000 045155 070144    4                     45046 025170
5 000000 000000 000000 170435    5                       136 140676
6 000000 000000 000001 020526    6                       160 023174
7 000000 000000 000001 050617    7                       202 106076

---------------------------------------------------------------------------
Display  : Forw,Back,Oct,Dec,Hex,Parc,Word, Display Address, Locate Error
Enter      DF   DB   DO  DD  DX  DP   DW    DA                LE
```

Figure 5-38. Compare Data Tool

5.3.4.5 System Call History Tool

The System Call History Tool is displayed when you enter **H** from any menu (refer to figure 5-39). This tool allows you to display a history of the last 15 system calls (commands) and the last 10 *events* that preceded the current *event*. An *event* is defined as any of the following actions: a failure occurs or a breakpoint is reached.

---

unitap System Call History Tool

Event # 1 was on PATH 1 in the LMAR Test at label L11002 pattern=40
The diagnostic wrote 40 to the LMAR and read back 44445

Seq	Path	Chan	Cont	Dev	CMD	Sec	Blk	B Adr	Flg	ACC	Label	Pattern
14	1	20	0	0	RLMAR	0	0	0	0	10	11001	10
13	3	22	2	0	F BK	0	0	0	0	0	27008	0
12	2	21	1	0	W BUS	0	0	0	0	2	15000	2
11	1	20	0	0	WLMAR	0	0	0	0	20	11000	20
10	3	22	2	0	BK BK	0	0	0	0	0	27009	0
9	2	21	1	0	W TAG	0	0	0	0	2000	15001	2
8	1	20	0	0	RLMAR	0	0	0	0	20	11001	20
7	3	22	2	0	F BK	0	0	0	0	0	27010	0
6	2	21	1	0	W TAG	0	0	0	0	2000	15002	2
5	1	20	0	0	WLMAR	0	0	0	0	40	11000	40
4	3	22	2	0	BK BK	0	0	0	0	0	27011	0
3	2	21	1	0	R BUS	0	0	0	0	2000	15003	2
2	1	20	0	0	RLMAR	0	0	0	0	40	11001	40
1	3	22	2	0	W TAG	0	0	0	0	0	21000	0
LAST	2	21	1	0	W BUS	0	0	0	0	3	15000	3

Option	Description	Option	Description
D	Debug Menu	N or P	Previous or next event
G	Global Options Menu	S	Status tool

Figure 5-39. System Call History Tool

5.3.4.6 Programming Tool

The Programming Tool is displayed when you enter **PG** from any menu (refer to figure 5-40). This tool allows you to define a test loop with up to 32 steps and up to 8 channels performing read, write, rewind, and compare operations.

unitap Programming Tool

Path 1 CH=20, CO=0, DV=*dv*, DN=6250, PC=1

STEP	PATH	DEV	COMMAND	SECT	BLOCKS	BYTES	BUF ADR	FLAGS	JUMP TO STEP
1	1	20	WRITE	5	1	0	1234	1357	0
2	2	21	REWIND	0	0	0	0	0	0
3	1	20	REWIND	0	0	0	0	0	0
4	2	21	READ	3	2	0	7010	0	0
5	1	20	READ	0	0	0	11000	0	0
6	2	21	FORW TM	2	0	0	0	0	2
7	0	0	---	0	0	0	0	0	0
8	0	0	---	0	0	0	0	0	0

Option	Description	Option	Description
BA *n*	Buffer address	JP *n*	Jump to step *n*
BK *n*	Number of blocks	PP*n*	Path (1-8)
BY *n*	Number of bytes	SC *n*	Number of sectors
CM *n*	Tape/channel command	ST *n*	Step (1-32)
FG *n*	Flag settings		

DF/DB Scroll forward/backward HELP *option* Information on *option*
G Display global options RUN Run test for *n* passes (PC *n*)

Now loading step number *n*

Figure 5-40. Programming Tool

5.3.4.7 Packet Status Tool

The Packet Status Tool is displayed when you enter **S** from any menu (refer to figure 5-41). This tool allows you to display the status of the last packet sent for each channel at the time of the last 10 *events* that preceded the current *event*. An *event* may be either of the following actions: a failure occurs or a breakpoint is reached.

```
unitap                    Packet Status Tool

   Path 1   CH=20, CO=0, DV=dv, DN=6250, PC=1

Path 1 was in the LMAR Test at label L11002 pattern=40.
Event # 1 was on PATH 1 in the LMAR Test at label L11002 pattern=40.
The diagnostic wrote 40 to the LMAR and read back 44445

                              Last DFT        Last DFT Reply

   Requested Sector Count =      0                 0
   Requested Block Count =       0                 0
   Data buffer address =         0                 0
   Accumulator =                40             44445
   Function =                  RLMAR            RLMAR
   Diagnostic Flags =            0                 0
   DFT packet Status flag =     ---              DONE
   DFT packet Status code =     ---                0
-------------------------------------------------------------------------

   Option              Description

   G                   Global Options Menu
   H                   System Call History Tool
   P or N              Previous or next event, respectively
   Pn                  Status for path (1-8)
   R                   Previous menu
```

Figure 5-41. Packet Status Tool

5.3.5 TRACE FILE

All user input and output is saved in a trace file for later evaluation.

5.3.6 LEARN MODE

To simulate passing test execution examples without removing the tape device from normal system operations, you can execute **unitap** in Learn mode. To enter Learn mode, enter L from any menu; to return to normal system operations (system mode), enter L again.

When you execute in Learn mode, the mode is indicated at the top of all the menus.

5.3.7 PROGRAM EXAMPLES

This subsection contains **unitap** execution examples.

The following example runs all of the **unitap** tests on device 00 and then exits the program.

```
unitap dv 00 a exit
```

The following example runs the two-channel conflict tests on devices 00 and 01, and then exits the program.

```
unitap dv 00 p2 dv 01 2 exit
```

5.3.8 PROGRAM MESSAGES

The following subsections contain the **unitap** messages:

- Messages with menu displays
- Messages without menu displays

The messages are listed alphabetically in each subsection.

5.3.8.1 Messages with menu displays

The messages are listed alphabetically in this subsection.

BREAKPOINT PROCESSED

Option	Description	Option	Description
C	Canned Test Menu	O	Rerun test
D	Debug Menu	PG	Programming Tool
G	Global Options Menu	R	Previous menu
H	System Call History Tool	S	Packet Status Tool
MN	Main Menu	T	Test Menu
N	Continue testing with next pattern	V	Variable Menu

TEST FAILED

Path 1 CH=20, CO=0, DV=*dv*, DN=6250

3-channel conflict tests were executing on pass 1 at label L4
Event # 1 was flagged in the diagnostic at label DL11002

Path 2 was in the Bus test at label L15004 variable=2
Path 3 was in the Tag-Loopback test at label L21001 variable=0

The error was on Path 1 in the LMAR Test at label L11002 variable=40

The diagnostic wrote 40 to the LMAR and read 44445

Option	Description	Option	Description
C	Canned Test Menu	N	Continue testing with next pattern
D	Debug Menu	O	Rerun test
G	Global Options Menu	S	Packet Status Tool
H	System Call History Tool	T	Test Menu

F Loop on failing pattern until next error or pass count is reached
X Loop on failing pattern until abort (press the ESC-A keys)

5.3.8.2 Messages without menu displays

The messages are listed alphabetically in this subsection.

```
Invalid entry:  n
Range:  n through n (radix)
Enter a valid value to continue
or an asterisk (*) to abort.
   The value entered is invalid.  Enter a valid value.

Test passed:  test
   The test completed successfully.
```

6. I/O SUBSYSTEM DEADSTART PROGRAMS

This section describes the following I/O Subsystem (IOS) deadstart programs:

Program	Description
cleario	IOS deadstart utility. The **cleario** utility attempts to clear the IOS if the deadstart procedure fails.
dsdiag	IOS deadstart diagnostic control program. The **dsdiag** program allows the system operator to run deadstart diagnostics from tape or disk.

6.1 SYSTEM CONFIGURATION

The file **aptext** contains the system text, including the configuration information for the IOS deadstart programs. The following system components are defined during system configuration:

- Optional I/O processors (IOP-2 and IOP-3)
- IOS type (model A, B, C, or D)
- High-speed channel connections to central memory and the SSD solid-state storage device
- Low-speed channel connection from IOP-0 to the CPU
- Console channels
- Central memory size
- Buffer memory size
- SSD memory size

For information on the IOS installation parameters, refer to the I/O Subsystem (IOS) Administrator's Guide, CRI publication SG-0307.

6.2 cleario

If the IOS deadstart procedure fails, the system operator can execute **cleario** from tape or disk in an attempt to clear the IOS. For information on the IOS deadstart procedure, refer to one of the following CRI publications, as appropriate to your configuration:

SG-2005	I/O Subsystem (IOS) Operator's Guide for UNICOS
SN-3030	Operator Workstation (OWS) Guide

IOP-0 must be minimally operational to execute the tape, disk, or OWS bootstrap routine (TAPELOAD, DISKLOAD, or VMELOAD, respectively) and **cleario**.

6.2.1 PROGRAM EXECUTION

The **cleario** program does the following:

- Disables all interrupts
- Clears all of the IOS channels
- Zeros the following:
  - The exit stack, the operand registers, and local memory in each IOP
  - Buffer memory
  - The last 64 words of central memory

Use the following procedure to execute **cleario**:

1. Mount the deadstart tape or disk at the operator's station.

2. Set the IOS maintenance panel toggle switches, as follows:

	Switch Setting	
Tape/Disk Unit	Octal	Binary
Tape	22	010 010
Ampex disk	60	110 000
CDC disk	27	010 111

NOTE

If the IOS maintenance panel has a 'maintenance mode' switch, set the switch to the 'on' position. When **cleario** is completed (successfully or unsuccessfully), return the switch to the 'off' position.

3. Press the IOP-0 MC button (or the MASTER CLEAR button on a CRAY-1 A computer system) and the DEADSTART button on the Power Distribution Unit or IOS chassis maintenance panel (as appropriate for your site).

4. Respond to one of the following prompts (for tape or disk, respectively) at the IOP-0 Kernel console:

    FILE @MTO:

    or

    FILE @DKO:

NOTE

The FILE @MTO prompt is not displayed unless a tape is mounted at the operator's station.

In response to the tape prompt, enter the number of the tape file containing **cleario** and press RETURN. If a tape is written using standard Cray generation procedures, file 7 contains **cleario**.

In response to the disk prompt, enter the name of the directory and file containing **cleario** (*dir*/**cleario**) and press RETURN.

5. If **cleario** completes successfully, the following message is displayed at the IOP-0 Kernel console:

    CLEARIO COMPLETE

The operating system bootstrap program is reloaded and one of the following prompts (for tape or disk, respectively) is displayed:

    FILE @MTO:

    or

    FILE @DKO:

Proceed with the IOS deadstart procedure. For information on the IOS deadstart procedure, refer to one of the following CRI publications, as appropriate to your configuration:

SG-2005	I/O Subsystem (IOS) Operator's Guide for UNICOS
SN-3030	Operator Workstation (OWS) Guide

6. If either of the following conditions occurs, run the IOS deadstart tests to determine if an IOS hardware malfunction exists:

   - **cleario** does not complete successfully (the message 'CLEARIO TERMINATED' is displayed or there is no response within one minute).

   - The IOS deadstart procedure continues to fail after **cleario** completes execution.

6.2.2 PROGRAM MESSAGES

The **cleario** program generates the following types of messages:

- Informative
- Error

6.2.2.1 Informative messages

The following informative messages are displayed at the IOP-0 Kernel console:

CLEARIO COMPLETE
cleario completed successfully.

TAPE NOT READY
This message is displayed until the tape is ready for use.

6.2.2.2 Error messages

The following error messages are displayed at the IOP-0 Kernel console. Unless otherwise indicated, use the IOS deadstart tests to do further error isolation.

CLEARIO TERMINATED
An error in one of the IOPs prevented **cleario** from executing successfully. Check the error logger for errors and run the **dsdiag** program for more information on the failure.

BUFFER MEMORY TIMEOUT
A Done flag is not set on the buffer memory channel. Check the error logger for errors and run the **dsdiag** program for more information on the failure.

BUFFER MEMORY ERROR
A Busy flag is set on the buffer memory channel. Check the error logger for errors and run the **dsdiag** program for more information on the failure.

device ERROR, STATUS=*status*
A device error occurred while the overlay was being loaded. *device* can be TAPE or DISK. *status* is the controller status for the deadstart device. Select a different device and deadstart the IOS. If no other device is available or the failure continues, use off-line diagnostics to isolate the error.

TAPE ERROR, STATUS=*status* AFTER REWIND
A tape error occurred after the overlay was loaded. *status* is the controller status for the tape device. Use a disk device and deadstart the IOS. If a disk device is unavailable or the failure continues, use off-line diagnostics to isolate the error.

6.3 dsdiag

The **dsdiag** program is the deadstart diagnostic control program that allows the system operator to run deadstart tests from tape or disk.

The **dsdiag** program does the following:

1. Executes a series of basic IOP-0 tests

2. Loads and executes subsequent IOS tests from a diagnostic overlay file

6.3.1 PROGRAM EXECUTION

Prior to loading the IOS Kernel, the system operator can run deadstart diagnostics from tape or disk by loading and executing the deadstart diagnostic control program, **dsdiag.** IOP-0 must be minimally operational to execute the tape, disk, or OWS bootstrap routine (TAPELOAD, DISKLOAD, or VMELOAD, respectively) and **dsdiag.**

Use the following procedure to execute the IOS deadstart diagnostics:

1. Mount the deadstart tape or disk at the operator's station.

2. Set the IOS maintenance panel toggle switches, as follows:

	Switch Setting	
Tape/Disk Unit	Octal	Binary
Tape	22	010 010
Ampex disk	60	110 000
CDC disk	27	010 111

3. Press the IOP-0 MC button (or the MASTER CLEAR button on a CRAY-1 A computer system) and the DEADSTART button on the Power Distribution Unit or IOS chassis maintenance panel (as appropriate for your site).

4. Respond to one of the following prompts (for tape or disk, respectively) at the IOP-0 Kernel console:

   FILE @MT0:

   or

   FILE @DK0:

   NOTE

   The FILE @MT0 prompt is not displayed unless a tape is mounted at the operator's station.

   In response to the tape prompt, enter the number of the tape file containing **dsdiag** and press RETURN. If a tape is written using standard Cray generation procedures, file 8 contains **dsdiag**.

   In response to the disk prompt, enter the name of the directory and file containing **dsdiag** (*dir*/**dsdiag**) and press RETURN.

   Pass/fail status messages are displayed at the IOP-0 Kernel console during test execution.

5. If the diagnostic tests complete successfully, the following message is displayed:

   DIAGNOSTICS COMPLETE

   The operating system bootstrap program is reloaded and one of the following prompts (for tape or disk, respectively) is redisplayed at the IOP-0 Kernel console:

   FILE @MT0:

   or

   FILE @DK0:

Proceed with the IOS deadstart procedure. For information on the IOS deadstart procedure, refer to one of the following CRI publications, as appropriate to your configuration:

SG-2005	I/O Subsystem (IOS) Operator's Guide for UNICOS
SN-3030	Operator Workstation (OWS) Guide

6. If a diagnostic test detects a failure, the message 'DIAGNOSTICS TERMINATED' is displayed at the IOP-0 Kernel console or there is no response within one minute. The system operator should report failures to a CRI field engineer.

6.3.1.1 IOP-0 tests

Although IOP-0 must be minimally operational to perform deadstart operations, it can still contain faults. Therefore, **dsdiag** tests IOP-0 before loading the deadstart tests from an overlay file. If the IOP-0 diagnostics do not execute successfully, use off-line diagnostics to do further testing.

The IOP-0 tests exercise the following areas, in the order shown:

1. Instruction buffers
2. Exit stack
3. Operand registers
4. Local memory
5. Real-time clock

The test procedure is as follows:

Logic Tested	Test Procedure
Instruction buffers	Forces 1's and 0's through each buffer location to detect dropped and picked bits, and adder faults.

If a failure is detected, the test does not issue an error message; instead, it loops at the point of failure. Use off-line diagnostics to do further testing.

Instruction buffer addressing is not tested. However, a fault in this area is likely to prevent **dsdiag** from loading. If no messages are displayed at the IOP-0 Kernel console within a few seconds of loading, a failure exists. You can scope the IOP-0 P register before using off-line diagnostics to do further testing.

Logic Tested	Test Procedure
Exit stack[†]	Checks for basic addressing and data faults in each stack location. Using I/O instructions for access, the test detects all single-stuck addressing and data faults, and all coupled-data bit faults. It also tests return jumps and exits at all stack depths.
Operand registers[†]	Checks for basic faults in all of the registers except 0 and 1, which are used to run the test algorithm. The test detects all single-stuck addressing and data faults, and all coupled-data bit faults.
Local memory	Tests the area of local memory between the end of **dsdiag** and the highest local memory address. The test uses an algorithm with a parcel-oriented, ascending and descending, marching 1's and 0's pattern to detect all single-stuck addressing and data faults, and all coupled-data bit faults.
Real-time clock	Tests the real-time clock to ensure that an interrupt occurs approximately once every millisecond.

When all of the IOP-0 tests complete successfully, the following message is displayed at the IOP-0 Kernel console (it is not required that the real-time clock test complete successfully):

```
IOP-0 KERNEL PASSED
```

The **dsdiag** program then loads and executes the deadstart tests contained in an overlay file.

If any one of the IOP-0 tests does not complete successfully (excluding the real-time clock test), **dsdiag** does not execute any subsequent diagnostics. An error message is displayed if a test fails (with the exception of the instruction buffer test, which loops at the point of failure instead of issuing an error message). The **dsdiag** program automatically attempts to reload the deadstart bootstrap program, TAPELOAD, DISKLOAD, or VMELOAD. If the attempt is unsuccessful, **dsdiag** halts and you can use off-line diagnostics to isolate the fault.

For a list of messages, refer to subsection 6.3.2, Program Messages.

---

† The test uses a variant of the Milner fast memory test algorithm (EDN, 28, 21; Oct 13, 1983). The Milner algorithm detects dropped and picked bits in address data, and coupled-data bit faults. The algorithm uses a rotating single-bit pattern to ensure that only one bit is changed in each memory chip at each step.

6.3.1.2 I/O Subsystem tests

If all of the IOP-0 tests complete successfully (excluding the real-time clock test), **dsdiag** loads and executes subsequent IOS tests from a diagnostic overlay file.

The tests are executed in the following order:

Test	Description
dsmos16k	Test of the lower 16 Kwords of buffer memory from IOP-0 only
dsiom	Local memory addressing and data test for each IOP except IOP-0
dsiop	Instruction test for each IOP
dsmos	Buffer memory addressing and data path test for each IOP
dshsp	High-speed channel test from an IOP to central memory or to an SSD solid-state storage device
dslsp	Low-speed channel test from IOP-0 to central memory

dsmos16k - This program tests addressing and data in the first 16384 words of buffer memory from IOP-0 only. This area of buffer memory is used to load an IOP. Therefore, **dsmos16k** must complete successfully before tests can be executed in IOP-1, IOP-2, or IOP-3.

The **dsmos16k** program consists of the following test sections:

1. Address and data test
2. Block length test

The **dsmos16k** test sections are as follows:

Section	Description
1	Address and data test. This section uses an algorithm with a word-oriented, ascending and descending, marching 1's and 0's pattern to test the lower 16 Kwords of buffer memory. The block length is 1.
2	Block length test. This section tests block length bits 1 through 13 (that is, block lengths 2^1 through 2^{13}).

If **dsmos16k** completes successfully, the following message is displayed:

MOS-16K PASSED
The test completed successfully.

For a list of messages, refer to subsection 6.3.2, Program Messages.

<u>dsiom</u> - This program tests local memory addressing and data for each IOP except IOP-0. The test detects basic faults that would inhibit the proper loading of diagnostics into an IOP.

The **dsiom** program consists of the following test sections:

1. All 0's test.
2. All 1's test.
3. Address pattern test
4. All 0's test

The test uses deadstart and dead dump procedures to load and dump data patterns. In the IOP being tested, no code is executed except a jump to P + 0 at address 0. The jump is required to prevent the IOP from executing after a deadstart. (In each of the **dsiom** test sections, address 0 contains O'7000.)

The **dsiom** test sections are as follows:

Section	Description
1	All 0's test. The test data is all 0's. The background data is all 1's.
2	All 1's test. The test data is all 1's. The background data is all 0's.
3	Address pattern test. The test data for each parcel (except parcel 0) is the parcel address. The background data is all 0's.
4	All 0's test. This section is the same as section 1. Section 4 is run so that local memory is reset to all 0's at the end of the test.

Each section uses the upper half of IOP-0 and the lower 16 Kwords of buffer memory as data buffers.

If **dsiom** completes successfully, the following message is displayed at the IOP-0 Kernel console:

IOP-*n* IOM PASSED
The test completed successfully in IOP-*n*.

For a list of messages, refer to subsection 6.3.2, Program Messages.

<u>dsiop</u> - This program tests instructions and registers in IOP-1, IOP-2, and IOP-3. Part of test section 1, basic instructions and registers test, executes in all of the IOPs, including IOP-0.

The **dsiop** program consists of the following test sections:

1. Basic instructions and registers test
2. Jump instructions test
3. Operand registers test

The **dsiop** test sections are as follows:

Section	Description
1	Basic instructions and registers test. Testing starts with the simplest instructions and data paths and becomes increasingly complex. The following IOP components are tested: 1. Registers A, B, and C 2. Instructions in the range 4 through 67 (octal) 3. Add and shift networks 4. Operand registers 0 through 20, 40, 100, 200, 400, and 777 (octal) 5. Local memory addressing 6. I/O instructions on channels 0 through 5 7. E register and exit stack location 0 8. Interprocessor channels to IOP-0 In IOP-0, only areas 1, 2, and 3 are tested; testing in the other areas would conflict with resident code. IOP-0 must be minimally operational to execute **dsdiag.** Therefore, this test is run in IOP-0 only to ensure that the basic instructions and the add/shift network are tested completely. There are no jumps in this test except a jump to P + 0, which is executed when a fault is detected, causing the test to loop at the point of failure.
2	Jump instructions test. This section is not run in IOP-0. The following areas of the IOP are tested: 1. Jump instructions 070 through 137 2. Exit instruction 001 3. Operand registers 0 and 1 4. Exit stack data and addressing

Section	Description
3	Operand registers test. This section is not run in IOP-0. This test section contains two subsections, as follows:

Subsection	Description
Systematic data	Performs a comprehensive test of operand register addressing and data.† The test detects all single-stuck faults in addressing or data, and all coupled data-bit faults.
Random data	Uses random data patterns to test registers 20 through 777 (octal). The test detects pattern-sensitive faults, which normally cannot be detected by systematic data. New data patterns are used each time the test is run.

If test section 1 (basic instructions and registers test) completes successfully, the following message is displayed at the IOP-0 Kernel console:

```
IOP-n BASIC PASSED
```

If test section 2 (jump instructions test) completes successfully, the exit stack is reset to all 0's and the following message is displayed at the Kernel consoles of IOP-0 and the IOP being tested:

```
IOP-n JUMPS PASSED
```

If test section 3 (operand registers test) completes successfully, the operand registers are reset to all 0's and the following message is displayed at the Kernel consoles of IOP-0 and the IOP being tested:

```
IOP-n OPREG PASSED
```

The **dsiop** program is run in all of the IOPs, regardless of whether a fault is detected in any single IOP. However, if a fault is detected in any of the IOPs, subsequent diagnostics cannot be executed until the fault is corrected. Use off-line diagnostics to isolate the failure.

For a list of messages, refer to subsection 6.3.2, Program Messages.

---

† The test uses a variant of the Milner fast memory test algorithm (EDN, 28, 21; Oct 13, 1983).

dsmos - This program tests the address and data paths from each IOP to buffer memory. It does not test the buffer memory data chips.

The **dsmos** program consists of the following test sections:

1. Data path test
2. Local memory addressing test
3. Buffer memory addressing test

The **dsmos** test sections are as follows:

Section	Description
1	Data path test. This section tests for dropped or picked data bits by transferring a single word between address 0 of local memory and address 0 of buffer memory. Dropped address bits do not affect this test.
2	Local memory addressing test. This section transfers data between address 0 of buffer memory and selected local memory addresses, using an algorithm with an ascending and descending, marching 1's and 0's pattern. The block length is always 1. The following local memory addresses (in octal) are used for test data: 0, 100000, 100000 + 2^n (includes all values for which n is an integer in the range 2 through 14), and 177774.
3	Buffer memory addressing test. This section transfers data between local memory and selected buffer memory addresses. The block length is always 1. The test algorithm is identical to that used in section 2 (local memory addressing) except that the local memory address is fixed and the buffer memory address varies. The following buffer memory word addresses are used for test data: 0, 2^n (includes all values for which n is an integer value in the interval [0, $\log_2$(MOS@SIZ)]).

If **dsmos** completes successfully, the following message is displayed at the Kernel consoles of IOP-0 and the IOP being tested:

IOP-*n* MOS PASSED
The test completed successfully in IOP-*n*.

The **dsmos** program is run in all of the IOPs, regardless of whether a fault is detected in any single IOP. However, if a fault is detected in any of the IOPs, subsequent diagnostics cannot be executed until the fault is corrected. Use off-line diagnostics to isolate the failure.

For a list of messages, refer to subsection 6.3.2, Program Messages.

dshsp - This program is a high-speed channel test from IOP-*n* to central memory or to an SSD solid-state storage device. Although it does not test memory, **dshsp** uses part of central memory or SSD memory to test the channel. The contents in the portion of memory used for testing are saved at the start of test execution and are restored only if the test completes successfully.

The **dshsp** program consists of the following test sections:

1. Buffer addressing and data test
2. Local memory addressing test
3. Central memory or SSD addressing test

The **dshsp** test sections are as follows:

Section	Description
1	Buffer addressing and data test. This section detects all single-stuck faults and coupled-data bit faults in the high-speed channel data buffers. The test writes to and reads from a block of memory beginning at absolute address 0 in either central memory or an SSD. For central memory, the block length is fixed at 32 words (the size of the data buffers). For an SSD, the block length is fixed at 64 words (minimum block size). This test section uses an algorithm† to move a block of sliding 1's and 0's through memory in an ascending and descending pattern. The block is addressed in ascending order due to hardware constraints.
2	Local memory addressing test. This test uses an algorithm with an ascending and descending marching 1's and 0's pattern. The transfer length is always one word for central memory and 64 words for an SSD. The central memory or SSD address is always 0. The following local memory addresses are tested if the test is from IOP-*n* to central memory: 77774, 100000, 100000 + 2^n (includes all values for which *n* is an integer in the range 2 through 14), and 177774. The following local memory addresses are tested if the test is from IOP-*n* to an SSD: 77400, 100000, 100000 + 2^n (includes all values for which *n* is an integer in the range 8 through 14), and 177400.

† The test uses a variant of the Milner fast memory test algorithm (EDN, 28, 21; Oct 13, 1983).

Section	Description
3	Central memory or SSD addressing test. This section uses an algorithm with an ascending and descending marching 1's and 0's pattern. The transfer length is always one word for central memory and 64 words for an SSD. The local memory address is arbitrary because it is assumed that section 2 (local memory addressing test) passed successfully. The following central memory addresses are tested if the test is from IOP-*n* to central memory: 0, 2^n (includes all values for which *n* is an integer in the interval [0, $\log_2$(central memory size)-1]). The following SSD addresses are tested if the test is from IOP-*n* to an SSD: 0, 2^n (includes all values for which *n* is an integer in the interval [0, $\log_2$(SSD size)-1]).

If **dshsp** completes successfully, the following message is displayed at the Kernel consoles of IOP-0 and the IOP being tested:

IOP-*n* HSP CH=*ch/ch* PASSED
The test completed successfully in the high-speed channel pair *ch/ch* in IOP-*n*. The contents of central memory or the SSD are restored.

The **dshsp** program is run in all of the IOPs for which a high-speed channel is defined in $APTEXT, regardless of whether a fault is detected in any single IOP. However, if a fault is detected in any of the IOPs, subsequent diagnostics cannot be executed until the fault is corrected. Use off-line diagnostics to isolate the failure.

For a list of messages, refer to subsection 6.3.2, Program Messages.

dslsp - This program tests the low-speed deadstart channel from IOP-0 to the Cray mainframe. The **dslsp** program consists of the following test sections:

1. Deadstart data test
2. Central memory addressing test

The **dslsp** test sections are as follows:

Section	Description
1	Deadstart data test. This section uses an algorithm with a marching 1's and 0's pattern to test the lower 64 words of central memory. Each data transfer begins at address 0 of central memory for a dead load or a dead dump.

Section	Description
2	Central memory addressing test. This section uses a CPU driver for the CPU end of the low-speed channel to test all address bits. The CPU driver occupies the first 64 words of central memory. The driver manages the channel protocol; it does not check for errors. All transfers are one word in length. The test uses the following central memory addresses: 2^n (includes all values for which *n* is an integer value in the interval [5, $\log_2$(CM@SIZE/2)]). The first five address bits are tested in section 1, deadstart data test.

If **dslsp** completes successfully, the following message is displayed at the IOP-0 Kernel console:

IOP-0 LSP CH=*ch/ch* PASSED
: The test completed successfully in the low-speed channel pair *ch/ch* in IOP-0. The contents of central memory are restored.

If a fault is detected, subsequent diagnostics cannot be executed until the fault is corrected. Use off-line diagnostics to isolate the failure.

For a list of messages, refer to subsection 6.3.2, Program Messages.

6.3.2 PROGRAM MESSAGES

The **dsdiag** program generates the following types of messages:

- Informative
- Error

6.3.2.1 Informative messages

The following informative messages are displayed at the IOP-0 Kernel console unless otherwise indicated.

DIAGNOSTICS COMPLETE
: The **dsdiag** program completed successfully.

test PASSED
: *test* completed successfully. This message is displayed at the Kernel consoles of IOP-0 and the IOP being tested.

TAPE NOT READY
: This message is displayed until the tape is ready for use.

6.3.2.2 Error messages

This subsection lists the **dsdiag** error messages, which are grouped as follows:

- Messages applicable to all tests
- IOP-0 messages
- **dsmos16k** messages
- **dsiom** messages
- **dsiop** messages
- **dsmos** messages
- **dshsp** messages
- **dslsp** messages

Messages applicable to all tests - The following error messages are displayed at the IOP-0 Kernel console. Use off-line diagnostics to do further error isolation.

DIAGNOSTICS TERMINATED
An error in one of the tests prevented **dsdiag** from executing successfully. An error message from the failing test is displayed at one or more of the Kernel consoles. Use off-line diagnostics to do further error isolation.

device ERROR, STATUS=*status*
A device error occurred while the overlay was being loaded. *device* can be TAPE or DISK. *status* is the controller status for the deadstart device. Select a different device and deadstart the IOS. If no other device is available or the failure continues, use off-line diagnostics to isolate the error.

TAPE ERROR, STATUS=*status* AFTER REWIND
A tape error occurred after the overlay was loaded. *status* is the controller status for the tape device. Use a disk device and deadstart the IOS. If a disk device is unavailable or the failure continues, use off-line diagnostics to isolate the error.

OVERLAY HEADER ERROR
The **dsdiag** program detected an error in the overlay header. Select a different device and deadstart the IOS. If no other device is available or the failure continues, use off-line diagnostics to isolate the error.

ATTEMPTED TO READ PAST ADDRESS 77777
The **dsdiag** program attempted to read beyond address 77777 in the overlay. Select a different device and deadstart the IOS. If no other device is available or the failure continues, use off-line diagnostics to isolate the error.

END-OF-FILE ENCOUNTERED

While reading the overlay, **dsdiag** detected an unexpected end-of-file. Select a different device and deadstart the IOS. If no other device is available or the failure continues, use off-line diagnostics to isolate the error.

INVALID OVERLAY DIRECTORY

While reading the overlay, **dsdiag** detected an invalid overlay directory. Select a different device and deadstart the IOS. If no other device is available or the failure continues, use off-line diagnostics to isolate the error.

NO OVERLAY FILE FOUND

The **dsdiag** program did not find an overlay file. Select a different device and deadstart the IOS. If no other device is available or the failure continues, use off-line diagnostics to isolate the error.

IOP-0 messages - The following error messages are displayed at the IOP-0 Kernel console. Use off-line diagnostics to do further error isolation.

IOP-0 FAILED EXIT STACK

The test terminated after detecting a fault in the IOP-0 exit stack. The bootstrap program is not reloaded. An IOS deadstart is required.

IOP-0 FAILED OPERAND REGISTER

The test terminated after detecting a fault in an IOP-0 operand register.

IOP-0 FAILED MEMORY, P=*address*, LMA=*lma*
EXP=*exp*
ACT=*act*

The test terminated after detecting a data compare error in IOP-0 local memory. The following information is displayed:

P=*address*	Parcel address relative to the start of the test module in which the fault was detected
LMA=*lma*	Absolute parcel address in IOP-0 local memory
EXP=*exp*	Expected data
ACT=*act*	Actual data

IOP-0 FAILED REAL-TIME CLOCK

The test detected a fault in the real-time clock. Although the test continues, subsequent tests can fail as a result of an inaccurate clock. A clock failure can occur if the IOP model is not defined correctly when the deadstart tests are generated. Check the I@IOPMOD installation parameter and regenerate. If the failure continues, use off-line diagnostics to isolate the fault. For a brief description of the IOS installation parameters, refer to the I/O Subsystem (IOS) Administrator's Guide, CRI publication SG-0307.

dsmos16k messages - The following error messages are displayed at the IOP-0 Kernel console. Use off-line diagnostics to do further error isolation.

MOS-16K FAILED, P=*address*, BMA=*bma*

The test detected a hardware failure in buffer memory. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsmos16k** in IOP-0
BMA=*bma*	Absolute word address in buffer memory

MOS-16K FAILED, P=*address*, BMA=*bma*
EXP=*exp*
ACT=*act*

The test detected a data compare error in buffer memory. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsmos16k** in IOP-0
BMA=*bma*	Absolute word address in buffer memory
EXP=*exp*	Expected data
ACT=*act*	Actual data

dsiom messages - The following error messages are displayed at the IOP-0 Kernel console. Use off-line diagnostics to do further error isolation.

IOP-*n* IOM FAILED, P=*address*, LMA=*lma*

The test detected a hardware failure in IOP-*n* local memory. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsiom** in IOP-0
LMA=*lma*	Absolute parcel address in IOP-*n* local memory

IOP-*n* IOM FAILED, P=*address*, LMA=*lma*
EXP=*exp*
ACT=*act*

The test detected a data compare error in IOP-*n* local memory. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsiom** in IOP-0
LMA=*lma*	Absolute parcel address in IOP-*n* local memory
EXP=*exp*	Expected data
ACT=*exp*	Actual data

<u>dsiop messages</u> - The following error messages are displayed at the IOP-0 Kernel console unless otherwise indicated. Use off-line diagnostics to do further error isolation.

IOP-*n* *section* FAILED, NO RESPONSE

An input-channel-done signal was not received from IOP-*n* within the required time limit. *section* is one of the following test sections: BASIC, JUMPS, or OPREG. This message precedes the following message (described in this subsection):

PRESS ANY KEY TO CONTINUE WITH REGISTER DUMP

IOP-*n* *section* FAILED, P=*address*, CH=*ipc*

The test detected a time-out or a protocol error in *ipc*, the interprocessor channel from IOP-0 to IOP-*n*. *section* is one of the following test sections: BASIC, JUMPS, or OPREG. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsiop** in IOP-0
CH=*ipc*	Interprocessor channel number associated with IOP-0

This message precedes the following message (described in this subsection):

PRESS ANY KEY TO CONTINUE WITH REGISTER DUMP

IOP-*n* *section* FAILED, P=*address*, MOS ERROR, BMA=*bma*

The test detected a failure in a data transfer between local memory in one of the configured IOPs and buffer memory. *section* is one of the following test sections: BASIC, JUMPS, or OPREG. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsiop** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.
BMA=*bma*	Absolute word address in buffer memory

IOP-*n* BASIC FAILED, P=*address*, CH=*ipc*
EXP=*exp*, ACT=*act*

The BASIC test section detected a data compare error in *ipc*, the interprocessor channel from IOP-0 to IOP-*n*. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsiop** in IOP-0
CH=*ipc*	Interprocessor channel number associated with IOP-0
EXP=*exp*	Expected data
ACT=*act*	Actual data

This message precedes the following message (described in this subsection):

PRESS ANY KEY TO CONTINUE WITH REGISTER DUMP

IOP-*n* JUMPS FAILED, CODE=*code*

The JUMPS test section detected a jump instruction error in IOP-*n*. *code* is the error code returned from the accumulator of the IOP being tested. This message precedes the following message (described in this subsection):

PRESS ANY KEY TO CONTINUE WITH REGISTER DUMP

IOP-*n* OPREG FAILED, P=*address*, B=*register*
EXP=*exp*, ACT=*act*

The OPREG test section detected a data compare error in the operand register in IOP-*n*. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsiop** in IOP-0
B=*register*	B register in which the error was detected

EXP=*exp* Expected data

ACT=*act* Actual data

The message is displayed at the Kernel consoles of IOP-0 and the IOP being tested. This message precedes the following message (described in this subsection), which is displayed at the IOP-0 Kernel console only:

PRESS ANY KEY TO CONTINUE WITH REGISTER DUMP

PRESS ANY KEY TO CONTINUE WITH REGISTER DUMP
The **dsiop** program detected an error and issued the error message that preceded this message. If you press any key, **dsiop** dumps the IOP being tested to the IOP-0 Kernel console. The following information is displayed:

A=*a*, C=*c*, B=*b*, (B)=*r*, E=*e*, (E)=*s1*, (E-1)=*s2*, (E-2)=*s3*

A=*a*	Accumulator of the IOP being tested
C=*c*	Carry flag
B=*b*	B register
(B)=*r*	B register contents
E=*e*	Exit stack pointer
(E)=*s1* (E-1)=*s2* (E-2)=*s3*	Contents of the top three exit stack locations. One of the stack locations normally represents the address at which a fault was detected in the IOP being tested.

Examine the dump values to isolate the fault. Depending on the fault, some or all of the dump values can be unreliable. Therefore, check the values for consistency. Prior to taking the dump (by pressing any key), a field engineer can scope the P register of the IOP being tested to ensure reliable values. Use off-line diagnostics to isolate the fault.

dsmos messages - The following error messages are displayed at the IOP-0 Kernel console unless otherwise indicated. Use off-line diagnostics to do further error isolation.

IOP-*n* MOS FAILED, P=*address*

The test detected a failure in the path between IOP-*n* and buffer memory. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsmos** in IOP-0; or, if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.

IOP-*n* MOS FAILED, P=*address*, NO RESPONSE

IOP-0 did not receive a response from IOP-*n* following the buffer memory test. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsmos** in IOP-0; or, if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.

IOP-*n* MOS FAILED, P=*address*, MOS ERROR

The test detected a failure in the path between IOP-*n* and buffer memory. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsmos** in IOP-0; or, if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.

This message is displayed at the Kernel consoles of IOP-0 and the IOP being tested.

IOP-*n* MOS FAILED, P=*address*
LMA=*lma*, BMA=*bma*
EXP=*exp*
ACT=*act*

The test detected a data compare error in the path between IOP-*n* and buffer memory. The following information is displayed:

P=*address*	Parcel address relative to the start of **dsmos** in IOP-0; or, if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.
LMA=*lma*	Absolute parcel address in local memory
BMA=*bma*	Absolute word address in buffer memory
EXP=*exp*	Expected data
ACT=*act*	Actual data

This message is displayed at the Kernel consoles of IOP-0 and the IOP being tested.

dshsp messages - The following error messages are displayed at the IOP-0 Kernel console unless otherwise indicated. Check the error logger for double bit errors. Use off-line diagnostics to do further isolation.

IOP-0 HSP CH=*ch/ch* FAILED, P=*address*, MOS ERROR

IOP-0 tried to write the diagnostic overlay to MOS. Upon completion, both the Busy and Done flags were found to be set. The probable error is in the channel from IOP-0 to MOS memory. Run off-line diagnostics to further isolate the problem.

CH=*ch/ch*	High-speed channel pair
P=*address*	Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.

The contents of CM or SSD remain unchanged. This message is displayed on the IOP-0 console.

IOP-*n* HSP CH=*ch/ch* FAILED, P=*address*, NO RESPONSE

IOP-0 sent an overlay package to MOS, deadstarted IOP-*n*, and waited for a response. The Done flag was never set (indicating that IOP-*n* did not respond by sending a return code). The probable error is in the deadstarting of IOP-*n*, the ability of IOP-*n* to read from MOS, or the test code was corrupt (due to a hardware memory problem). Check for further test messages or run off-line diagnostics.

IOP-*n*	The IOP that would not deadstart
CH=*ch/ch*	High-speed channel pair
P=*address*	Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.

The contents of CM or SSD remain unchanged. This message is displayed on the IOP-0 console.

IOP-*n* HSP CH=*ch/ch* FAILED, P=*address*, BAD RETURN STATUS, S=*address*

IOP-0 sent a test to IOP-*n*. IOP-*n* executed the tests and returned a bad status. This indicates that the test found an error in IOP-*n*. Check the IOP-*n* console for further messages.

IOP-*n*	The IOP that sent the message to IOP-0
CH=*ch/ch*	High-speed channel pair

P=*address* — Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.

S=*address* — The address of the problem in IOP-*n* is returned. The address is relative to the start of the overlay sent to IOP-*n*.

It is unknown whether the contents of CM or SSD have been corrupted. This message is displayed on the IOP-0 console.

IOP-*n* HSP CH=*ch/ch* PASSED

IOP-0 sent a test to IOP-*n*. IOP-*n* executed the tests and returned a zero status indicating that no errors were discovered.

IOP-*n* — The IOP that sent the message to IOP-0

CH=*ch/ch* — High-speed channel pair

The contents of CM or SSD were restored to their original state. This message is displayed on the IOP-0 console.

The following messages are displayed on the IOP-*n* console.

IOP-*n* HSP CH=*ch/ch* FAILED, P=*address*, NO CONFIGURED MEMORY SIZE

IOP-*n* found a high-speed channel configured, but the configured memory size for CM or SSD attached to that channel is zero. This is not a hardware error. Correct the channel and memory size configured in $APTEXT or $IOSDEF. The test in IOP-*n* for this channel was bypassed.

IOP-*n* — The IOP being tested

CH=*ch/ch* — High-speed channel pair

P=*address* — Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.

The contents of CM or SSD remain unchanged. This message is displayed on the IOP-*n* console.

IOP-*n* HSP CH=*ch/ch* FAILED, P=*address*, CH=*ch*, *routine*, TIMEOUT SAVEMEM

IOP-*n* tried to read from CM or SSD to save the contents of the memory to be tested before beginning the test. After the read was started, the program waited for the Done flag to be set. The Done flag was never set so the program timed out. The probable error is in the channel from IOP-*n* to CM or SSD memory. Run off-line diagnostics to further isolate the problem.

IOP-*n*	The IOP being tested
CH=*ch/ch*	High-speed channel pair
P=*address*	Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.
CH=*ch*	Channel on which the error was detected
routine	The test routine executing in IOP-*n* when the error was encountered. The test routines in order are HSPBUFF, HSPLMCM, HSPLMSSD, HSPCMA, and HSPSSDA. The test routine HSPBUFF is the first time the HSP channel is used.

The contents of CM or SSD remain unchanged. This message is displayed on the IOP-*n* console.

IOP-*n* HSP CH=*ch/ch* FAILED, P=*address*, CH=*ch*, *routine*, BZ & DN SAVEMEM
LMA=*address*, CMA or SSDA=*address*
EXP=*exp*
ACT=*act*

IOP-*n* tried to read from CM or SSD to save the contents of the memory to be tested before beginning the test. Upon completion of the read (when the Done flag was set), both the Busy and Done flags were found to be set. The probable error is in the channel from IOP-*n* to CM or SSD memory. Check the error logger for double bit errors. Run off-line diagnostics to further isolate the problem.

This error can also occur if the test tries to read or write past the end of CM or SSD. Check the configured memory size of CM or SSD in $APTEXT.

IOP-*n*	The IOP being tested
CH=*ch/ch*	High-speed channel pair

P=address — Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.

CH=*ch* — Channel on which the error was detected

routine — The test routine executing in IOP-*n* when the error was encountered. The test routines in order are HSPBUFF, HSPLMCM, HSPLMSSD, HSPCMA, and HSPSSDA. The test routine HSPBUFF is the first time the HSP channel is used.

LMA=*address* — Absolute parcel address in local memory of data

CMA or SSDA=*address* — Absolute word address in central memory or SSD of the data

EXP=*exp* — Expected data

ACT=*act* — Actual data

The contents of CM or SSD remain unchanged. This message is displayed on the IOP-*n* console.

IOP-*n* HSP CH=*ch/ch* FAILED, P=*address*, CH=*ch*, *routine*, TIMEOUT

IOP-*n* tried to read/write a test pattern from/to CM or SSD. Check the channel number to determine if the error was on a read or write. After the read/write was started, the program waited for the Done flag to be set. The Done flag was never set so the program timed out. The probable error is in the channel CH=*ch* from IOP-*n* to CM or SSD memory. Run off-line diagnostics to further isolate the problem.

IOP-*n* — The IOP being tested

CH=*ch/ch* — High-speed channel pair

P=*address* — Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.

CH=*ch* — Channel on which the error was detected

routine — The test routine executing in IOP-*n* when the error was encountered. The test routines in order are HSPBUFF, HSPLMCM, HSPLMSSD, HSPCMA, and HSPSSDA.

The contents of CM or SSD may have been corrupted. This message is displayed on the IOP-*n* console.

IOP-*n* HSP CH=*ch/ch* FAILED, P=*address*, CH=*ch*, *routine*, ERROR FLAG

IOP-*n* tried to write a test pattern to CM or SSD. Upon completion of the write (when the Done flag was set), both the Busy and Done flags were found to be set. The probable error is in the channel CH=*ch* from IOP-*n* to CM or SSD memory. Check the error logger for double bit errors. Run off-line diagnostics to further isolate the problem.

IOP-*n*	The IOP being tested
CH=*ch/ch*	High-speed channel pair
P=*address*	Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.
CH=*ch*	Channel on which the error was detected
routine	The test routine executing in IOP-*n* when the error was encountered. The test routines in order are HSPBUFF, HSPLMCM, HSPLMSSD, HSPCMA, and HSPSSDA.

The contents of CM or SSD may have been corrupted. This message is displayed on the IOP-*n* console.

IOP-*n* HSP CH=*ch/ch* FAILED, P=*address*, CH=*ch*, *routine*, ERROR FLAG
LMA=*address*, CMA or SSDA=*address*
EXP=*exp*
ACT=*act*

IOP-*n* tried to read a test pattern from CM or SSD. Upon completion of the read (when the Done flag was set), both the Busy and Done flags were found to be set. The probable error is in the channel CH=*ch* from IOP-*n* to CM or SSD memory. Check the error logger for double bit errors. Run off-line diagnostics to further isolate the problem.

IOP-*n*	The IOP being tested
CH=*ch/ch*	High-speed channel pair
P=*address*	Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.
CH=*ch*	Channel on which the error was detected

routine	The test routine executing in IOP-*n* when the error was encountered. The test routines in order are HSPBUFF, HSPLMCM, HSPLMSSD, HSPCMA, and HSPSSDA.
LMA=*address*	Absolute parcel address in local memory of data
CMA or SSDA=*address*	Absolute word address in central memory or SSD of the data
EXP=*exp*	Expected data
ACT=*act*	Actual data

The contents of CM or SSD may have been corrupted. This message is displayed on the IOP-*n* console.

IOP-*n* HSP CH=*ch/ch* FAILED, P=*address*, *routine*, CH=*ch*, DATA COMPARE
LMA=*address*, CMA or SSDA=*address*
EXP=*exp*
AC=*act*

IOP-*n* wrote a test pattern to CM or SSD and then read it back. The data read from memory (ACT) did not match the original data (EXP) written to memory. The probable error is in the channel from IOP-*n* to CM or SSD memory. Run off-line diagnostics to further isolate the problem.

IOP-*n*	The IOP being tested
CH=*ch/ch*	High-speed channel pair
P=*address*	Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.
CH=*ch*	Channel on which the error was detected
routine	The test routine executing in IOP-*n* when the error was encountered. The test routines in order are HSPBUFF, HSPLMCM, HSPLMSSD, HSPCMA, and HSPSSDA.
LMA=*address*	Absolute parcel address in local memory of data

CMA or SSDA=*address*	Absolute word address in central memory or SSD of the data
EXP=*exp*	Expected data
ACT=*act*	Actual data

The contents of CM or SSD may have been corrupted. This message is displayed on the IOP-*n* console.

IOP-*n* HSP CH=*ch/ch* FAILED, P=*address*, CH=*ch*, *routine*, TIMEOUT RESTMEM

After testing, IOP-*n* tried to write to CM or SSD to restore the original contents of memory. After the write was started, the program waited for the Done flag to be set. The Done flag was never set so the program timed-out. The probable error is in the channel from IOP-*n* to CM or SSD memory. Run off-line diagnostics to further isolate the problem.

IOP-*n*	The IOP being tested
CH=*ch/ch*	High-speed channel pair
P=*address*	Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.
CH=*ch*	Channel on which the error was detected
routine	The test routine executing in IOP-*n* when the error was encountered. The test routines in order are HSPBUFF, HSPLMCM, HSPLMSSD, HSPCMA, and HSPSSDA.

The contents of CM or SSD may have been corrupted. This message is displayed on the IOP-*n* console.

IOP-*n* HSP CH=*ch/ch* FAILED, P=*address*, CH=*ch*, *routine*, BZ & DN RESTMEM

After testing, IOP-*n* tried to write to CM or SSD to restore the original contents of memory. Upon completion of the write (when the Done flag was set), both the Busy and Done flags were found to be set. The probable error is in the channel from IOP-*n* to CM or SSD memory. Check the error logger for double bit errors. Run off-line diagnostics to further isolate the problem.

IOP-*n*	The IOP being tested
CH=*ch/ch*	High-speed channel pair

P=*address*	Parcel address relative to the start of **dshsp** in IOP-0; or if IOP-0 is being tested, the parcel address relative to the start of the test module in which the fault was detected.
CH=*ch*	Channel on which the error was detected
routine	The test routine executing in IOP-*n* when the error was encountered. The test routines in order are HSPBUFF, HSPLMCM, HSPLMSSD, HSPCMA, and HSPSSDA.

The contents of CM or SSD may have been corrupted. This message is displayed on the IOP-*n* console.

<u>**dslsp** messages</u> - The error messages are displayed at the IOP-0 Kernel console. Use off-line diagnostics to do further error isolation.

In this subsection, the messages are grouped as follows:

- Time-out messages
- Channel interface status flag messages
- Data compare error messages
- Overlay messages

For information on the channel interface status flags (FLAGS=*flags*), refer to the following CRI publications, as appropriate:

HR-0030	I/O Subsystem Model B Hardware Reference Manual
HR-0081	I/O Subsystem Model C/D Hardware Reference Manual

The time-out messages follow.

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, LMA=*lma*, CH=*ch*, TIMEOUT
LSPCPUA, READ FROM CM

While attempting to read one word from central memory addresses in multiples of 10, starting at address 100 and continuing to the end of central memory, the program detected a time-out in the low-speed channel pair *ch/ch* in IOP-*n*. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
LMA=*lma*	Absolute parcel address in local memory
CH=*ch*	Low-speed channel pair
LSPCPUA	Read one word from central memory addresses in multiples of 10, starting at address 100 and continuing to the end of central memory
READ FROM CM	Read from central memory

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, LMA=*lma*, CH=*ch*, TIMEOUT
LSPCPUA, WRITE TO CM

While attempting to write one word to central memory addresses in multiples of 10, starting at address 100 and continuing to the end of central memory, the program detected a time-out in the low-speed channel pair *ch/ch* in IOP-*n*. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
LMA=*lma*	Absolute parcel address in local memory
CH=*ch*	Low-speed channel pair
LSPCPUA	Write one word to central memory addresses in multiples of 10, starting at address 100 and continuing to the end of central memory
WRITE TO CM	Write to central memory

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, LMA=*lma*, CH=*ch*,
TIMEOUT
LSPDSDD, READ FROM CM

While attempting to read blocks of various lengths from central memory address 0, the program detected a time-out in the low-speed channel pair *ch/ch* in IOP-*n*. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
LMA=*lma*	Absolute parcel address in local memory
CH=*ch*	Low-speed channel pair
LSPDSDD	Read blocks of various lengths from central memory address 0
READ FROM CM	Read from central memory

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, LMA=*lma*, CH=*ch*,
TIMEOUT
LSPDSDD, WRITE TO CM

While attempting to write blocks of various lengths to central memory address 0, the program detected a time-out in the low-speed channel pair *ch/ch* in IOP-*n*. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
LMA=*lma*	Absolute parcel address in local memory
CH=*ch*	Channel on which the error was detected
LSPDSDD	Write blocks of various lengths to central memory address 0
WRITE TO CM	Write to central memory

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, LMA=*lma*, CH=*ch*, TIMEOUT
RESTMEM, WRITE TO CM

While attempting to restore the central memory locations used in the test, the program detected a time-out in the low-speed channel pair *ch/ch* in IOP-*n*. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
LMA=*lma*	Absolute parcel address in local memory
CH=*ch*	Channel on which the error was detected
RESTMEM	Final write to central memory
WRITE TO CM	Write to central memory

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, LMA=*lma*, CH=*ch*, TIMEOUT
SAVEMEM, READ FROM CM

While attempting to save the central memory locations used in the test, the program detected a time-out in the low-speed channel pair *ch/ch* in IOP-*n*. Central memory is not corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
LMA=*lma*	Absolute parcel address in local memory
CH=*ch*	Low-speed channel pair
SAVEMEM	Initial read from central memory
READ FROM CM	Read from central memory

The status flag messages follow.

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, FLAGS=*flags*, CH=*ch*
LSPCPUA, READ FROM CM

While attempting to read one word from central memory addresses in multiples of 10, starting at address 100 and continuing to the end of central memory, the program detected a hardware error in the low-speed channel pair *ch/ch* in IOP-0. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
FLAGS=*flags*	An octal value representing one or more channel interface status flags
CH=*ch*	Channel on which the error was detected
LSPCPUA	Read one word from central memory addresses in multiples of 10, starting at address 100 and continuing to the end of central memory
READ FROM CM	Read from central memory

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, FLAGS=*flags*, CH=*ch*
LSPCPUA, WRITE TO CM

While attempting to write one word to central memory addresses in multiples of 10, starting at address 100 and continuing to the end of central memory, the program detected a hardware error in the low-speed channel pair *ch/ch* in IOP-0. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
FLAGS=*flags*	An octal value representing one or more channel interface status flags
CH=*ch*	Channel on which the error was detected
LSPCPUA	Write one word to central memory addresses in multiples of 10, starting at address 100 and continuing to the end of central memory
WRITE TO CM	Write to central memory

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, FLAGS=*flags*, CH=*ch*
LSPDSDD, READ FROM CM

While attempting to read blocks of various lengths from central memory address 0, the program detected a hardware error in the low-speed channel pair *ch/ch* in IOP-0. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
FLAGS=*flags*	An octal value representing one or more channel interface status flags
CH=*ch*	Channel on which the error was detected
LSPDSDD	Read blocks of various lengths from central memory address 0
READ FROM CM	Read from central memory

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, FLAGS=*flags*, CH=*ch*
LSPDSDD, WRITE TO CM

While attempting to write blocks of various lengths to central memory address 0, the program detected a hardware error in the low-speed channel pair *ch/ch* in IOP-0. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
FLAGS=*flags*	An octal value representing one or more channel interface status flags
CH=*ch*	Channel on which the error was detected
LSPDSDD	Write blocks of various lengths to central memory address 0
WRITE TO CM	Write to central memory

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, FLAGS=*flags*, CH=*ch*
RESTMEM, WRITE TO CM

While attempting to restore the central memory locations used in the test, the program detected a hardware error in the low-speed channel pair *ch/ch* in IOP-0. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
FLAGS=*flags*	An octal value representing one or more channel interface status flags
CH=*ch*	Channel on which the error was detected
RESTMEM	Final write to central memory
WRITE TO CM	Write to central memory

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, FLAGS=*flags*, CH=*ch*
SAVEMEM, READ FROM CM

While attempting to save the central memory locations used in the test, the program detected a hardware error in the low-speed channel pair *ch/ch* in IOP-0. Central memory is not corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
FLAGS=*flags*	An octal value representing one or more channel interface status flags
CH=*ch*	Channel on which the error was detected
SAVEMEM	Initial read from central memory
READ FROM CM	Read from central memory

The data compare error messages follow.

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, CMA=*cma*
LSPCPUA
EXP=*exp*
ACT=*act*

While writing and reading one word to and from central memory addresses in multiples of 10, starting at address 100 and continuing to the end of central memory, the program detected a data compare error in the low-speed channel pair *ch/ch* in IOP-*n*. The expected data did not match the actual data. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
CMA=*cma*	Absolute word address in central memory

LSPCPUA	Write and read one word to and from central memory addresses in multiples of 10, starting at address 100 and continuing to the end of central memory
EXP=*exp*	Expected data
ACT=*act*	Actual data

IOP-*n* LSP CH=*ch/ch* FAILED, P=*address*, CMA=*cma*
LSPDSDD
EXP=*exp*
ACT=*act*

While writing and reading blocks of various lengths to and from central memory address 0, the program detected a data compare error in the low-speed channel pair *ch/ch* in IOP-*n*. The expected data did not match the actual data. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair
P=*address*	Parcel address relative to the start of **dslsp**
CMA=*cma*	Absolute word address in central memory
LSPDSDD	Write and read blocks of various lengths to and from central memory address 0
EXP=*exp*	Expected data
ACT=*act*	Actual data

The overlay messages follow.

IOP-*n* LSP CH=*ch/ch*
FAILED - OVERLAY NOT DSLSPCP

The overlay that the test read was not DSLSPCP. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair

IOP-*n* LSP CH=*ch/ch*
FAILED - OVERLAYS NOT FOUND

The test could not find an overlay file. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair

IOP-*n* LSP CH=*ch/ch*
FAILED - OVERLAY WRONG TYPE

The test found the overlay file DSLSPCP, but it has the wrong overlay type. Central memory may have been corrupted. The following information is displayed:

IOP-*n*	IOP in which the test was executing
CH=*ch/ch*	Low-speed channel pair

7. UTILITY PROGRAMS

Utility programs are on-line diagnostic tools rather than tests. This section describes the following utilities:

- **olhpa** (hardware performance analyzer)
- **runsequence** (automatic test sequencer)

7.1 olhpa

The **olhpa** program is a hardware performance analyzer that analyzes and reports the hardware errors and statuses recorded in the system error log. The **olhpa** program displays the following types of reports:

- A report listing one line of error information for each hardware error. The error information is displayed in fields and is sorted from left to right (refer to **sort**(1)).

- A comprehensive error report similar to the **errpt**(1M) report (**-l** command option)

- A summary of total errors (**-q** command option)

- A bar graph showing total errors for the specified time interval (**-g** [**d**]*n* command option)

7.1.1 PROGRAM SYNOPSIS

This subsection contains the **olhpa** program synopsis. All of the command options except *errfiles* can be entered in any order. If *errfiles* is specified, it must be the last entry on the command line.

The **olhpa** program displays disk, memory, tape, and SSD error reports in fields. If **olhpa** is entered without command options and arguments, it is equivalent to entering the following:

```
olhpa -dmtv
```

The start time is the current time and date minus 30 days. The end time is the current time and date. The **olhpa** program reads from the error file **/usr/adm/errfile**.

Synopsis:

olhpa [-**l**] [-**q**] [-**g** [**d**]*n*] [-**d**] [-**m**] [-**t**] [-**v**] [-**D** *argument*]

[-**M** *argument*] [-**T** *argument*] [-**V** *argument*] [-**s** *start*]

[-**e** *end*] [*errfiles*]

-l Displays a long version of the selected error report. If you select **-l**, do not select **-q** or **-g** [**d**]*n*. A **-l** report contains the same information as the **errpt**(1M) report. For example, enter the following to display a long version of a memory error report:

```
olhpa -m -l
```

Long reports are not sorted.

-q Displays only the summary information of an error report. If you select **-q**, do not select **-l** or **-g** [**d**]*n*.

-g [**d**]*n* Displays a bar graph showing the total errors for the specified time interval. If you select **-g** [**d**]*n*, do not select **-l** or **-q**. A single mnemonic value represents each error, as follows:

Mnemonic	Description
R	Represents one recovered/corrected error
U	Represents one unrecovered/uncorrected error

The required argument *n* indicates the time interval that each bar in the graph represents. If the interval (*n*) is in days, precede *n* with the **d** command; otherwise it is assumed that *n* is in hours.

n can be any integer value. However, *n* should be within the limits set by the start/end times and dates (**-s** *start* and **-e** *end*, respectively). For example, if the start time is 7:00, the end time is 11:00, and *n* is 8, the interval is adjusted so that the program generates a report for one 4-hour interval.

-d Displays a report of all disk errors. The default display contains the following information in the order listed:

Field	Field Mnemonic
Date	dte
Time	tme
Error type	et
Device type	dt
IOP	iop
Channel	cha
Head	hd
Sector	sct
Cylinder	cyl
General status	gs
Status	st

-m Displays a report of all memory errors. The default display contains the following information in the order listed:

Field	Field Mnemonic
Date	dte
Time	tme
Syndrome	syn
Bank	bnk
Failing bit	bit
Chip select	chp
Failing module	loc
CPU	cpu
Current command	cmd
Count	cnt
Status	st

-t Displays a report of all tape errors. The default display contains the following information in the order listed:

Field	Field Mnemonic
Date	dte
Time	tme
Error type	et
Initial channel	ich
Initial device path	idp
Final device path	fdp
Block	blk
Retry	ret
Sense byte #00	s0
Status	st

-v Displays a report of all SSD errors. The default display contains the following information in the order listed:

Field	Field Mnemonic
Date	dte
Time	tme
Channel	cha
Status	st
SSD address	sad
Central memory address	mad
Transfer length	len
Read/write flag	rwf

-D *argument*, **-M** *argument*, **-T** *argument* **-V** *argument*

Displays a report of disk, memory, tape, or SSD errors (**-D**, **-M**, **-T**, or **-V** option, respectively). The required argument can be one of the following:

Argument	Description
P[,+],*field*[,*field*]	Replaces or adds to the default display. If entered with the plus (+) option, the specified fields are displayed in addition to the default display. If entered without the plus (+) option, the specified fields are displayed instead of the default fields (and the specified fields become the default display for the test run). *field* can be any mnemonic listed in the help menu.

-D *argument,* **-M** *argument,* **-T** *argument* **-V** *argument*
(continued)

The fields are displayed in the order in which they are entered. The error information is sorted from left to right. Refer to **sort**(1).

S,*field*=*value*[,*field*=*value*]
Displays only the records in which the fields meet all of the associated value restrictions. *field* can be any mnemonic listed in the help menu. *value* is the field assignment.

H Displays an associated help menu. The mnemonics in the menu are used to select fields for the *field* portion of the preceding arguments.

-s *start* Sets the start time and date of the report. Enter the **-s** option with one of the following required arguments:

Argument	Description
l *n*	End time and date of the report (**-e** *end*) minus *n* days
hh:*mm*,*MM*/*DD*/*YY*	Time (hours:minutes) and date (month/day/year)
hh:*mm*	Time (hours:minutes). The date is set to the current date.
MM/*DD*/*YY*	Date (month/day/year). The time is set to 00:00.

The default for *start* is the current time and date minus 30 days.

-e *end* Sets the end time and date of the report. The required argument must be in one of the following formats:

Format	Description
hh:*mm*,*MM*/*DD*/*YY*	Time (hours:minutes) and date (month/day/year)
hh:*mm*	Time (hours:minutes). The date is set to the current date.
MM/*DD*/*YY*	Date (month/day/year). The time is set to 23:59.

The default for *end* is the current time and date.

errfiles Specifies the *errfiles* to be read. *errfiles* can be one or more files created by **errdemon**(1M). The default errfile is **/usr/adm/errfile.**

7.1.2 HELP MENUS

This subsection contains the menus to use in selecting the fields for the *field* portion of the arguments associated with the **-D**, **-M**, **-T**, and **-V** options.

Figures 7-1, 7-2, 7-3, and 7-4 show the Disk, Memory, Tape, and SSD Help Menus, respectively.

```
            dte) Date                        tme) Time
            dtc) Dt-IOP-channel-unit         dt ) Device type
            iop) IOP                         ios) IOS
            cha) Channel                     et ) Error type
            hd ) Head                        sct) Sector
            sst) Spiralled sector            cyl) Cylinder
            st ) Status                      ret) Retry
            blk) Block                       sbk) Spiralled block
            cs ) Control status              gs ) General status
            df ) Disk function               s0 ) Status 00
            s1 ) Status 01                   s2 ) Status 02
                  .                                .
                  .                                .
                  .                                .
            s21) Status 21                   s22) Status 22
            s23) Status 23                   ies) Initial error status
            ecs) End controller status       eds) End drive status
            edf) End disk function           fes) Final error status

DD49 only
            a1 ) A1 - bit 5 of G.S.          a2 ) A2 - bit 6 of G.S.
            b1 ) B1 - bit 7 of G.S.          b2 ) B2 - bit 8 of G.S.
            aof) A-offset                    bof) B-offset
            b2c) B2 correction mask          b1c) B1 correction mask
            a2c) A2 correction mask          a1c) A1 correction mask
            b2o) B2 offset                   b1o) B1 offset
            a2o) A2 offset                   a1o) A1 offset
            elm) Expected LMA                alm) Actual LMA

DD39 only
            c0 ) C0 - bit 3 of G.S.          c1 ) C1 - bit 4 of G.S.
            c2 ) C2 - bit 5 of G.S.          c3 ) C3 - bit 6 of G.S.
            ofs) Offset                      sy0) Chan. 0 syndrome
            sy1) Chan. 1 syndrome            sy2) Chan. 2 syndrome
            sy2) Chan. 3 syndrome            c3c) C3 correction mask
            c2c) C2 correction mask          c1c) C1 correction mask
            c0c) C0 correction mask          c3o) C3 offset
            c2o) C2 offset                   c1o) C1 offset
            c0o) C0 offset
```

Figure 7-1. Disk Help Menu (1 of 2)

DD40 only

	ibs) Initial buffer stat.	ids) Initial drive status
	fbs) Final buffer status	fds) Final drive status
	msk) DD40 correction mask	off) DD40 offset
	dfa) Defect address	if0) Initial fault stat 0
	if1) Initial fault stat 1	if2) Initial fault stat 2
	if3) Initial fault stat 3	io0) Initial oper. stat 0
	io1) Initial oper. stat 1	io2) Initial oper. stat 2
	io3) Initial oper. stat 3	ic0) Initial FRU code 0
	ic1) Initial FRU code 1	ic2) Initial FRU code 2
	ic3) Initial FRU code 3	ef0) Ending fault stat 0
	ef1) Ending fault stat 1	ef2) Ending fault stat 2
	ef3) Ending fault stat 3	ec0) Ending FRU code 0
	ec1) Ending FRU code 1	ec2) Ending FRU code 2
	ec3) Ending FRU code 3	syn) Channel syndrome

DD29/DD19 only

	cid) Cylinder from ID	csr) Cylinder status reg.
	req) Request	fsr) Fault status reg.
	isr) Interlock stat. reg.	mrg) Margin
	ofd) Offset direction	mgn) Magnitude
	cv0) Correction vector 0	cv1) Correction vector 1
	cv2) Correction vector 2	cv3) Correction vector 3

Figure 7-1. Disk Help Menu (2 of 2)

dte) Date	tme) Time
cnt) Count	ity) Initial type
st) Status	sub) Subtype
mde) Mode	cpu) CPU
syn) Syndrome	chp) Chip-select
bnk) Bank	rh) Rh
add) Failing address	bit) Failing bit
loc) Failing module	usr) Current user
cmd) Current Command	

Figure 7-2. Memory Help Menu

```
              dte) Date                        tme) Time
              et ) Error type                  st ) Status
              ich) Initial channel             ios) IOS number
              idp) Initial device path         ids) Initial device stat.
              fch) Final channel               fdp) Final device path
              fds) Final device stat.          ifn) Initial function
              ffn) Final function              blk) Block
              dns) Density                     ret) Retry
              vol) Volume                      usr) User
              cmd) Command                     ipt) Input tags
              s0 ) SB00                        s1 ) SB01
                    .                                .
                    .                                .
                    .                                .
              s22) SB22                        s23) SB23

 IBM 3480 only
              s24) SB24                        s25) SB25
              s26) SB26                        s27) SB27
              s28) SB28                        s29) SB29
              s30) SB30                        s31) SB31
```

Figure 7-3. Tape Help Menu

```
dte) Date                   tme) Time
cha) Channel                st ) Status
sad) SSD-Address            mad) MEM-Address
len) Length                 rwf) Read/write flag
```

Figure 7-4. SSD Help Menu

7.1.3 PROGRAM EXAMPLES

This subsection contains **olhpa** execution examples. Depending on whether errors are in the current error file, it may be necessary to specify an error file. If you need assistance, contact your CRI representative.

To display disk, tape, memory, and SSD error reports, enter the following:

```
olhpa
```

To display a disk error report, enter the following:

```
olhpa -d
```

To display a disk error report for an error file, enter the following:

olhpa -d *errfile*

To display the disk help menu, enter the following:

```
olhpa -D H
```

To display a disk error report for the date, time, head, and channel fields only, enter the following:

```
olhpa -D P,DTE,TME,HD,CHA
```

To display a disk error report of only the records for which the channel is equal to 26 and the IOP is equal to 2, enter the following:

```
olhpa -D S,CHA=26,IOP=2
```

The following example searches for disk errors for a specific channel and IOP, and displays the associated error information in the specified fields. The disk error report will display the following fields for only the records for which the channel is equal to 26 and the IOP is equal to 2: date, time, device type, general status, and A1, A2, B1, and B2 of the general status. Enter the following:

```
olhpa -DS,CHA=26,IOP=2 -DP,DTE,TME,DT,GS,A1,A2,B1,B2
```

To display a bar graph showing yesterday's disk errors in 2-hour intervals, enter the following (using yesterday's date for *date*):

olhpa -d -s *date* -e *date* -g 2

7.1.4 SHELL SCRIPT GENERATION AND EXECUTION

Shell scripts can allow you to easily generate and execute **olhpa** command sequences.

The following example shows a shell script that generates a disk error report for each disk drive for which errors are logged.

Example:

```
#
#   Shell script to report errors for each disk drive.
#

echo "*************************************************************"
echo "      R E P O R T    O F    D I S K    E R R O R S          "
echo
echo "    Only devices which logged errors will create reports.    "
echo
echo "*************************************************************"

for DEV in `olhpa -DPdtc $1 |awk '{print $1}' |uniq  |grep '-'`
do
echo "*************************************************************"
echo      $DEV
echo "*************************************************************"
echo
    olhpa -DSdtc=${DEV} $1
done

echo "*************************************************************"
echo "        E N D    O F    R E P O R T                          "
echo "*************************************************************"
```

Error report output from preceding shell script:

```
*************************************************************
     R E P O R T    O F    D I S K    E R R O R S

    Only devices which logged errors will create reports.

*************************************************************


*************************************************************
40-1-34A
*************************************************************

         Cray Hardware Performance Analyzer
         Run time      : 10:26 03/02/88
         Starting time : 10:26 02/01/88
         Ending time   : 10:26 03/02/88


           Hardware Error Report For Disks

             Restrictions:
                  Dt-IOP-channel-unit = 40-1-34A
```

Error report (continued):

Date	Time	Errtyp	DT/IOP/CHA	HD	Sect	Cyl	Gen-Stat	Status
88/02/26	03:10:51	Read	40-1-34A	00	0000	0013	011426	Corre.
88/02/26	03:37:30	Read	40-1-34A	00	0000	0013	011426	Corre.
88/02/26	04:46:23	Read	40-1-34A	00	0000	0013	011426	Recov.
.								
.								
.								
88/03/01	04:14:00	Read	40-1-34A	00	0000	0011	011411	Recov.
88/03/01	04:14:13	Read	40-1-34A	00	0000	0011	011411	Recov.
88/03/01	06:24:40	Read	40-1-34A	00	0000	0013	011426	Corre.

```
Total Disk Errors         :  30
Recovered Disk Errors     :  12
Corrected Disk Errors     :  18
Unrecovered Disk Errors   :  0
Uncorrected Disk Errors   :  0
Total Retries             :  70
```

```
**************************************************************
40-2-34A
**************************************************************

         Cray Hardware Performance Analyzer
         Run time      : 10:26 03/02/88
         Starting time : 10:26 02/01/88
         Ending time   : 10:26 03/02/88

           Hardware Error Report For Disks

             Restrictions:
                  Dt-IOP-channel-unit = 40-2-34A
```

Date	Time	Errtyp	DT/IOP/CHA	HD	Sect	Cyl	Gen-Stat	Status
88/02/26	05:48:17	Read	40-2-34A	00	0000	0007	011433	Corre.
88/02/26	06:01:49	Read	40-2-34A	00	0000	0003	011442	Corre.

```
Total Disk Errors         :  2
Recovered Disk Errors     :  0
Corrected Disk Errors     :  2
Unrecovered Disk Errors   :  0
Uncorrected Disk Errors   :  0
Total Retries             :  6
```

Error report (continued):

Error information for all drives for which errors are logged is displayed.

```
*************************************************************
       E N D    O F    R E P O R T
*************************************************************
```

7.1.5 PROGRAM MESSAGES

If an invalid or nonexistent command option is entered, **olhpa** displays the incorrect entry and the complete program synopsis.

If an invalid or nonexistent error file is entered, the following message is displayed:

olhpa: Cannot open *file*

In an error report, a field can contain the following symbols:

Symbol	Description
N/A	No information was recorded in the system error log.
(x)	No information was recorded in the system error log. The field is specific to device type x.

7.2 runsequence

The **runsequence** utility is used with the **crontab**(1) command to perform automatic test sequencing (scheduling and testing withou operator intervention). Error messages are returned to specified users through the UNICOS mail. This alerts field engineers and analysts that there is an error. They can then examine the error log to determine where the error occurred. The goal is to detect and isolate failures before a system or application failure occurs.

To initiate automatic test sequencing, do the following:

1. Set the shell variables in the **runsequence** shell script.
2. Create the sequence files.
3. Create the input file for the **crontab**(1) command.
4. Execute the **crontab**(1) command.

After being called in from the **crontab**(1) input file, **runsequence** reads a file containing a list of diagnostics and related command options, executes the diagnostics (one at a time), and saves any output in a file.

After each diagnostic in the sequence file is executed, **runsequence** determines the number of lines of output generated, as follows:

- If there are more than five lines of output, **runsequence** assumes that the diagnostic detected an error and sends specified users a message.

- If no error is detected but standard error output is generated, **runsequence** sends specified users a message.

- If no error is detected, the output files from the diagnostic are removed.

7.2.1 **crontab** INPUT FILE

The **crontab**(1) input file contains the following information:

- Times at which the sequences are to be run
- Calls to **runsequence**

When defining the **crontab**(1) input file, you must include calls to **runsequence**. Each call to **runsequence** must contain an appropriate sequence file name and, optionally, a CPU designator. For additional information on the **crontab**(1) command, refer to the UNICOS User Commands Reference Manual, CRI publication SR-2011.

runsequence synopsis:

```
runsequence seqfile [cpu]
```

seqfile Indicates the name of the file containing the sequence of diagnostics to be run, the diagnostic command options, and any comments. The comments are the same as shell script comments; they start with a pound sign (#) and continue to the end of the line.

cpu Indicates the CPU in which the diagnostics are to be run. *cpu* can be **a**, **b**, **c**, **d**, **e**, **f**, **g**, or **h**. If the *cpu* option is specified, the diagnostics in the sequence file must be CPU tests. All the log and core files are placed in a subdirectory of the **DIAGLOG** directory, which is created if it does not already exist.

If the *cpu* option is not specified, the diagnostic uses the default value or you can specify the CPU option for the diagnostic in the sequence file. All log and core files are placed in the **DIAGLOG** directory instead of a subdirectory.

The following example shows a sample **crontab**(1) input file:

```
# Run in a different cpu every 15 minutes
1  * * * * $HOME/scripts/runsequence hourlyseq a
15 * * * * $HOME/scripts/runsequence hourlyseq b
30 * * * * $HOME/scripts/runsequence hourlyseq c
45 * * * * $HOME/scripts/runsequence hourlyseq d

1  * * * * $HOME/scripts/runsequence sbtseq a,b,c,d
15 * * * * $HOME/scripts/runsequence sbtseq b,c,d,a
30 * * * * $HOME/scripts/runsequence sbtseq c,d,a,b
45 * * * * $HOME/scripts/runsequence sbtseq d,a,b,c

# Run at midnight each day
1 0      * * 0-6 $HOME/scripts/runsequence dailyseq a
1 0      * * 0-6 $HOME/scripts/runsequence dailyseq b
1 0      * * 0-6 $HOME/scripts/runsequence dailyseq c
1 0      * * 0-6 $HOME/scripts/runsequence dailyseq d
1 0      * * 0-6 FSPATH=/tmp DT=DD49 $HOME/scripts/runsequence cfdtseq
1 0      * * 0-6 FINDPATH=$HOME/log $HOME/scripts/findseq
```

The minute field is set to 1 to offset the diagnostic program execution to one minute after the hour. This allows scheduled system activities to be performed at the start of each hour.

7.2.2 SEQUENCE FILES

The sequence files contain a list of the diagnostics to be executed and their related command options. You must place these files in the directory specified by the **DIAGSCRIPTS** shell variable. Before creating sequence files, refer to appendix B, Test Execution Times.

The following example shows the recommended sequence files for the **crontab**(1) input file.

Example:

```
hourlyseq:

# Run the following sequence once every 15 minutes in a different CPU.

olcrit cputime 0:0:30 +getseed     # Read seed from olcrit.seed if available
olcsvc cputime 0:0:30 +getseed     # Read seed from olcsvc.seed if available
olibuf cputime 0:0:30 +getseed     # Read seed from olibuf.seed if available
olcfpt cputime 0:0:30 +getseed     # Read seed from olcfpt.seed if available
olcm cputime 0:0:30 +getseed       # Read seed from olcm.seed if available


dailyseq:

# Run the following sequence once a day.

olcrit cputime 0:6:0 +getseed      # Read seed from olcrit.seed if available
olcsvc cputime 0:6:0 +getseed      # Read seed from olcsvc.seed if available
olibuf cputime 0:6:0 +getseed      # Read seed from olibuf.seed if available
olcfpt cputime 0:6:0 +getseed      # Read seed from olcfpt.seed if available
olcm cputime 0:6:0 +getseed        # Read seed from olcm.seed if available


sbtseq:

#
# sbtseq: This sequence tests olsbt in all cpus available
# it should be run once every 15 minutes.
#

olsbt cputime 30 +getseed


cfdtseq:

# Run the following sequence to test a mass storage device.

olcfdt maxp 50 fn $FSPATH/workfil.$$ rsz 512 sz 250 dt $DT
find $FSPATH -name 'workfil.*' -user $LOGNAME -exec rm -f {} \\;
```

Example (continued):

```
findseq:

#
# findseq: This sequence finds and removes any small log files
# or stderr files that the runsequence created.
#

TOO_OLD=180            # Number of days to save log files
#FPATH                   Path to log files. default FPATH=$HOME/log in cronfile

find $FPATH \( \( -name '*.[0-9]*[0-9]' -size -300c \) -o -name \
'stderr.*'\) -atime +0 -type f -exec rm -f {} \; 2>/dev/null 1>&2

#
#Remove any log file that has not been touched recently
#
find $FPATH -name '*.[0-9]*[0-9]' -type f -atime +$TOO_OLD \
-exec rm -f {} \;
```

Each site must determine if additional testing is desirable.

7.2.3 **runsequence** SHELL SCRIPT

The **runsequence** shell script runs under the Bourne shell and executes a series of diagnostics by reading a file containing a list of the diagnostics to be run. The diagnostics should be run with the verbose option disabled (-**verbose**), because the size of each diagnostic output file is used to determine if the diagnostic has failed.

The shell script maintains the diagnostic output and sends messages to a specified list of users when an error is detected. You can set the following variables in the **runsequence** shell script:

DIAGBIN=*path*
: Indicates the full path name of the directory where the executable binaries of the diagnostics reside. If the binaries reside in more than one directory, enter colons between each directory. The following entry defines a single directory:

    DIAGBIN=/ce/bin

  The following entry defines several directories:

    DIAGBIN=/ce/bin:$HOME/bin

DIAGLOG=*path*

Indicates the full path name of the directory where the log files are saved when a diagnostic detects an error

DIAGSCRIPTS=*path*

Indicates the path name where the sequence files reside. You can specify only one full path name.

MAILLIST="*user ...user*"

Provides a list of users to be notified when a diagnostic detects an error. Enter a space between each user name and enclose the list in double quotes. It is recommended that the list contain more than one user name.

NICE=*n*

Indicates the amount by which the diagnostic's priority in the execution queue is to be lowered. *n* can be any integer within the range 1 through 19. If a value greater than 19 is entered, it is processed as if it were 19. If a value less than 0 is entered, it has no effect.

RUNLOG=*logfile*

Indicates the name of the log file containing information on the sequence being run and any errors detected. The log file resides in the **DIAGLOG** directory.

SAVECORE=ON|OFF

Enables (**ON**) or disables (**OFF**) the option that renames and saves each core file generated. If **SAVECORE** is set to **OFF**, any new core file overwrites an existing one.

The default values for the variables in the **runsequence** shell script are as follows:

```
DIAGBIN=/ce/bin              # Location of the executable diagnostics
DIAGLOG=$HOME/log            # Location of the diagnostic log files
DIAGSCRIPT=$HOME/scripts     # Location of the diagnostic sequence lists
RUNLOG=$DIAGLOG/runlog       # Program log
NICE=4                       # Lower the diagnostic's priority by this amount
SAVECORE=OFF                 # Existing core file will be overwritten
MAILIST="$LOGNAME"           # List of people to receive error messages
```

APPENDIX SECTION

A. ON-LINE DIAGNOSTIC PROGRAMS

This appendix lists and briefly describes the following types of on-line diagnostic programs:

- Confidence tests
- Maintenance tests
- Down-device programs
- Network communications test (**olnet**)
- I/O Subsystem (IOS) deadstart programs
- Utilities
- **offmon** tests

The on-line diagnostic programs listed in this section are supported on the following computer systems:

- CEA systems
  - Y-mode (32-bit addressing)
- CRAY X-MP and CRAY-1 computer systems

A.1 CONFIDENCE TESTS

Table A-1 briefly describes each on-line confidence test.

Table A-1. Confidence Tests

Test	Description	Language
olcfdt	Mass storage device test	CFT77
olcfpt	Comprehensive floating-point test	CAL 2
olcm	Central memory test	CAL 2
olcrit	Comprehensive random instruction test	CAL 2
olcsvc	Comprehensive scalar/vector compare test	CAL 2

Table A-1. Confidence Tests (continued)

Test	Description	Language
olibuf	Instruction buffer test	CAL 2
olsbt	Semaphore, shared B and T register test	CAL 2

A.2 MAINTENANCE TESTS

Table A-2 briefly describes each on-line CPU maintenance test.

NOTE

The CPU Maintenance Tests are supported for CX/CEA systems in X-mode only.

Table A-2. CPU Maintenance Tests

Test	Description	Language
olaht	A register indexing test	CAL 2
olarb	A register data test	CAL 2
olarm	A register multiply test	CAL 2
olbrb	B register basic data test	CAL 2

Table A-2. CPU Maintenance Tests (continued)

Test	Description	Language
olcmd†	Random instruction and operand test	CAL 2
olcmp††	Vector compress instruction test	CAL 2
olcmx†††	Random instruction and operand test	CAL 2
olgth††	Scatter/gather test	CAL 2
olibz†††	Instruction buffer test	CAL 2
olmit	Moving inversions memory test	CAL 2
olsfa	Simulate floating-point add test	CAL 2
olsfm	Simulate floating-point multiply test	CAL 2
olsfr	Simulate floating-point reciprocal	CAL 2
olsis	Scalar register instruction simulation test	CAL 2
olsr3	Random instruction issue register conflicts	CAL 2
olsra	Scalar register add test	CAL 2
olsrb	Scalar register basic test	CAL 2
olsrl	Scalar register logical test	CAL 2
olsrs	Scalar register shift test	CAL 2
olstan	Standard answer functional units test	CAL 2
olsvc	Scalar and vector compare test	CAL 2
oltrb	T register basic data test	CAL 2

† CRAY-1 computer systems only
†† CEA (X-mode) and CRAY X-MP computer systems only
††† CRAY X-MP EA (X-mode) and CRAY X-MP computer systems only

Table A-2. CPU Maintenance Tests (continued)

Test	Description	Language
olvpop†	Vector population count test	CAL 2
olvpp††	Vector population count test	CAL 2
olvra	Vector register add test	CAL 2
olvrl	Vector register logical test	CAL 2
olvrn	Vector register random test	CAL 2
olvrr	Vector register random length test	CAL 2
olvrs	Vector register shift test	CAL 2
olvrx††	Vector register stress test	CAL 2

† CRAY-1 computer systems only
†† CEA (X-mode) and CRAY X-MP computer systems only

A.3 DOWN-DEVICE PROGRAMS

Table A-3 briefly describes the down-device programs, which reside on DIAGPL.

Table A-3. Down-Device Programs

Test	Description	Language
donut	On-line disk maintenance program	CFT77, C & CAL 2
oldmon†	Down CPU monitor	C & CAL 2
unitap	On-line magnetic tape test	C

† Multiple CPU Cray computer systems only

Tables A-4 and A-5 briefly describe the down CPU tests, which reside on **XMPPL** and execute under **oldmon**, the down CPU monitor. These tests run on CRAY X-MP computer systems in multiple-CPU environments only (CRAY X-MP/4 and CRAY X-MP/2 computer systems).

Table A-4. Down CPU Confidence Tests

Test	Description	Language
offcfpt	Comprehensive floating point test	CAL 2
offcm	Central memory test	CAL 2
offcrit	Comprehensive random instruction test	CAL 2
offcsvc	Comprehensive scalar/vector compare test	CAL 2
offibuf	Instruction buffer test	CAL 2

Table A-5. Down CPU Maintenance Tests

Test	Description	Language
aht	A register indexing test	CAL 2
arb	A register data test	CAL 2
arm	A register multiply test	CAL 2
brb	B register basic data test	CAL 2
cmp	Vector compress instruction test	CAL 2
cmx	Random instruction and operand test	CAL 2
gth	Scatter/gather test	CAL 2
ibz	Instruction buffer test	CAL 2
mit	Moving inversions memory test	CAL 2

Table A-5. Down CPU Maintenance Tests (continued)

Test	Description	Language
sfa	Simulate floating-point add test	CAL 2
sfm	Simulate floating-point multiply test	CAL 2
sfr	Simulate floating-point reciprocal	CAL 2
sis	Scalar register instruction simulation test	CAL 2
sr3	Random instruction issue register conflicts	CAL 2
sra	Scalar register add test	CAL 2
srb	Scalar register basic test	CAL 2
srl	Scalar register logical test	CAL 2
srs	Scalar register shift test	CAL 2
stan	Standard answer functional units test	CAL 2
svc	Scalar and vector compare test	CAL 2
trb	T register basic data test	CAL 2
vpp	Vector population count test	CAL 2
vra	Vector register add test	CAL 2
vrl	Vector register logical test	CAL 2
vrn	Vector register random test	CAL 2
vrr	Vector register random length test	CAL 2
vrs	Vector register shift test	CAL 2
vrx	Vector register stress test	CAL 2

A.4 ON-LINE NETWORK COMMUNICATIONS PROGRAM

Table A-6 briefly describes the Cray-to-front end communications test, **olnet**.

Table A-6. On-line Network Communications Program

Test	Description	Language
olnet	Cray-to-front end communications test (exercises all or part of the path between a Cray mainframe and a front end)	CFT77 & C†

† Motorola Operator Workstation (OWS) and Maintenance Workstation (MWS) only

The **olnet** test is described in the On-line Diagnostic Network Communications Program (OLNET) Maintenance Manual, CRI publication SMM-1016.

A.5 I/O SUBSYSTEM DEADSTART PROGRAMS

Table A-7 briefly describes the I/O Subsystem (IOS) deadstart programs, which reside on DIAGPL. The **cleario** program is executed independently from the other programs listed. The **dsdiag** program, the IOS deadstart diagnostic control program, loads and executes all of the programs (except **cleario**) from a diagnostic overlay file, after first executing a series of basic IOP-0 tests.

Table A-7. I/O Subsystem Deadstart Programs

Program	Description	Language
cleario	Attempts to clear the IOS if the deadstart procedure fails	APML
dsdiag	Deadstart diagnostic control program	APML
dshsp	High-speed channel test from an I/O processor (IOP) to central memory or to an SSD solid-state storage device	APML
dsiom	Local memory addressing and data test for each IOP	APML
dsiop	Instruction test for each IOP	APML
dslsp	Low-speed channel test from IOP-0 to central memory	APML and CAL 1
dsmos	Buffer memory addressing and data path test for each IOP	APML
dsmos16k	Test of the lower 16 Kbytes of buffer memory from IOP-0 only	APML

A.6 UTILITY PROGRAMS

Table A-8 briefly describes each on-line utility program.

Table A-8. Utility Programs

Utility	Description	Language
olhpa	Hardware performance analyzer	C
runsequence	Diagnostic sequencer utility	Shell script

A.7 **offmon** TESTS

Table A-9 briefly describes each **offmon** test.

Table A-9. **offmon** Tests

Confidence Test	Description	Language
offcfpt	Comprehensive floating-point test	CAL 2
offcm	Central memory test	CAL 2
offcrit	Comprehensive random instruction test	CAL 2
offcsvc	Comprehensive scalar/vector compare test	CAL 2
offibuf	Instruction buffer test	CAL 2

B. TEST EXECUTION TIMES

This appendix lists the execution times for the following types of on-line diagnostic tests:

- Confidence
- Maintenance

The tests were run at Cray Research, Inc. during normal workday operations, using a default pass count of 512 (O'1000). The times are for test execution in a single CPU of a CRAY X-MP computer system and cannot be extrapolated to determine execution times for multiple CPU runs.

NOTE

The execution times may vary depending on system load, and should not be used for CPU or benchmark comparisons.

In the test execution tables, the following times are listed in the headings:

Time	Description
Elapsed	Wall-clock time
User	CPU time
System	System overhead time

B.1 EXECUTION TIMES FOR CONFIDENCE TESTS

Table B-1 lists the execution times for the confidence tests. Each test was run with a pass count of 512 (O'1000).

Table B-1. Execution Times for Confidence Tests

Test	Elapsed Time†	User Time	System Time
olcm	65.00 s	34.25 s	0.88 s
olcfpt	23.00 s	7.15 s	0.47 s
olcrit	15.00 s	7.55 s	0.28 s
olcsvc	12.00 s	4.27 s	0.21 s
olibuf	78.00 s	21.00 s	0.11 s
olsbt††	4.66 s	2.29 s	1.43 s

† Execution times may be reduced or increased by the use of test-specific options.

†† Times are for test execution with four CPUs (cpu a,b,c,d)

B.2 EXECUTION TIMES FOR MAINTENANCE TESTS

Table B-2 lists the execution times for the maintenance tests. Each test was run with a pass count of 512 (0'1000) except **olibz** and **olsfm**; these tests were run for less than 512 (0'1000) passes, and their respective execution times were then used to extrapolate elapsed, user, and system times for 512 passes.

Table B-2. Execution Times for Maintenance Tests

Test	Elapsed Time	User Time	System Time
olaht	10.03 s	2.24 s	0.08 s
olarb	0.74 s	0.11 s	0.01 s
olarm	21.10 m	15.95 m	17.35 s
olbrb	0.69 s	0.24 s	0.01 s
olcmd†	7.10 s	2.92 s	0.04 s

† CRAY-1 computer systems only

†† CEA (X-mode) and CRAY X-MP computer systems only

Table B-2. Execution Times for Maintenance Tests (continued)

Test	Elapsed Time	User Time	System Time
olcmx†	25.35 s	2.49 s	0.1 s
olgth††	15.11 s	7.41 s	0.12 s
olibz†	6.74 h	1.62 h	1.25 m
olmit	1.61 m	42.12 s	1.58 s
olsfa	9.39 s	7.95 s	0.17 s
olsfm	117.0 h	14.3 h	12.64 m
olsfr	8.02 m	6.33 m	5.77 s
olsis	0.46 s	0.02 s	0.01 s
olsr3	0.46 s	0.18 s	0.01 s
olsra	0.96 s	0.70 s	0.04 s
olsrb	1.00 s	0.34 s	0.02 s
olsrl	1.96 s	0.05 s	0.01 s
olsrs	20.64 s	18.04 s	0.37 s
olstan	0.31 s	0.21 s	0.01 s
olsvc	0.35 s	0.17 s	0.01 s
oltrb	6.07 s	5.13 s	0.12 s
olvpop†††	0.73 s	0.57 s	0.02 s
olvpp††	0.84 s	0.62 s	0.01 s
olvra	0.82 s	0.68 s	0.02 s
olvrl	0.87 s	0.59 s	0.01 s

† CRAY X-MP EA (X-mode) and CRAY X-MP computer systems only
†† CEA (X-mode) and CRAY X-MP computer systems only
††† CRAY-1 computer systems only

Table B-2. Execution Times for Maintenance Tests (continued)

Test	Elapsed Time	User Time	System Time
olvrn	0.23 s	0.12 s	0.01 s
olvrr	0.28 s	0.12 s	0.01 s
olvrs	26.3 s	17.34 s	0.36 s
olvrx†	2.86 m	2.83 min	1.44 s

† CEA (X-mode) and CRAY X-MP computer systems only

C. ON-LINE DIAGNOSTIC PROGRAM LIBRARIES

This appendix describes the on-line diagnostic program libraries (PLs) and their contents and associated decks. The on-line diagnostic PLs are as follows:

PL	Description
DIAGPL	Contains on-line diagnostic programs that execute on CX/CEA and CRAY-1 computer systems
XMPPL	Contains diagnostic programs that execute on CX/CEA systems
CRAY1PL	Contains diagnostic programs that execute on a CRAY-1 computer system

Each deck contains source code that is used to generate a binary.

C.1 DIAGPL

DIAGPL contains on-line diagnostic programs that execute on CX/CEA and CRAY-1 computer systems. The contents of DIAGPL are as follows:

Program	Deck
bmxtap	BMXTAP
beario	CLEARIO
donut	DONUT
dsdiag	DSDIAG, DSDIAGD, DSMOS16K, DSIOM, DSIOP, DSMOS, DSHSP, DSLSP
olcm	OLCM
olcfdt	OLCFDT
olcfpt	OLCFPT
olcrit	OLCRIT
olcsvc	OLCSVC
oldmon	OLDMON
olhpa	OLHPA
olibuf	OLIBUF
olnet	OLNET
olsbt	OLSBT
runsequence	RUNSEQ

C.2 XMPPL

XMPPL contains diagnostic programs that execute on CX/CEA systems. The contents of XMPPL are as follows:

Program	Deck
olaht	AHT
olarb	ARB
olarm	ARM
olbrb	BRB
olcmp	CMP
olcmx	CMX
olgth	GTH
olibz	IBZ
olmit	MIT
olsfa	SFA
olsfm	SFM
olsfr	SFR
olsis	SIS
olsr3	SR3
olsra	SRA
olsrb	SRB
olsrl	SRL
olsrs	SRS
olstan	STAN
olsvc	SVC
oltrb	TRB
olvpp	VPP
olvra	VRA
olvrl	VRL
olvrn	VRN
olvrr	VRR
olvrs	VRS
olvrx	VRX

C.3 CRAY1PL

CRAY1PL contains diagnostic programs that execute on CRAY-1 computer systems. The contents of CRAY1PL are as follows:

Program	Deck
olaht	AHT
olarb	ARB
olarm	ARM

Program	Deck
olbrb	BRB
olcmd	CMD
olmit	MIT
olsfa	SFA
olsfm	SFM
olsfr	SFR
olsis	SIS
olsr3	SR3
olsra	SRA
olsrb	SRB
olsrl	SRL
olsrs	SRS
olstan	STAN
olsvc	SVC
oltrb	TRB
olvpop	VPOP
olvra	VRA
olvrl	VRL
olvrn	VRN
olvrr	VRR
olvrs	VRS

D. SOFTWARE PROBLEM REPORTING

This appendix describes the on-line diagnostic software problem reporting procedure.

The on-line diagnostics are released as part of the operating system software. To report problems with or request changes to the on-line diagnostic software, send the information electronically to the automated Software Technical Support database, or send a Software Problem Report (SPR) form to the Software Technical Support department.

Figure D-1 shows an SPR form. You can order these forms from the CRI Distribution Center. For additional SPR information, refer to the Software Problem Report (SPR) User's Guide, CRI publication SD-0235.

PLEASE PRESS HARD –
YOU ARE MAKING 3 COPIES

Software Problem Report

Originator's Name | Phone | Originator Number

Site Code | Mainframe Serial # | Date | Op Sys: Cos ☐ CTSS ☐ Unicos ☐ Other ☐ ______

Problem ☐ Change ☐ | Critical ☐ Major ☐ Minor ☐ Design ☐ Info Only ☐ | Cray Op Sys Version — Prerelease Y☐ N☐ | IOS Version — Prerelease Y☐ N☐

On-Site Analyst's Signature | Product | Version — Prerelease Y☐ N☐

Front end Op Sys (Station Problem) | Version — Prerelease Y☐ N☐

Title of Problem

SUPPORTING DOCUMENTATION: (Include on a PDSDUMP format 6250 bpi, non-labeled, on-line tape when possible).

TAPE INFORMATION

DUMP (ED. NO.) | SYSTEM LOG (ED. NO.) | LISTING | JOB THAT PRODUCED PROBLEM

SPR DESCRIPTION

CORRECTIVE CODE SUPPLIED: Y ☐ N ☐ TESTED: Y ☐ N ☐

TEST CASE SUPPLIED:

SEND TO:

CRAY RESEARCH, INC.
1345 Northland Drive
Mendota Heights, MN 55120

DISTRIBUTION: WHITE - CRI FILE BLUE - SPR COORDINATOR PINK - AIC

Figure D-1. SPR Form

E. SYSTEM UTILITIES

This appendix briefly describes the UNICOS system utilities that have been identified as effective diagnostic tools. These utilities are as follows:

Utility	Description
dda(1)	The **dda** command (dynamic dump analyzer) allows you to examine the contents of a program memory dump.
icrash(1M)	The **icrash** command allows you to examine the I/O Subsystem (IOS) core image.

If you know of other system utilities that should be mentioned in this appendix, please use one of the following options to forward the information to the Technical Publications department:

- Call our Technical Publications department at (612) 681-5729 during the hours of 7:30 A.M. to 6:00 P.M. (Central Time).
- Send us electronic mail from a UNICOS or UNIX system, using the following UUCP addresses:

  uunet!cray!publications

  sun!tundra!hall!publications
- Send us electronic mail from a UNICOS or UNIX system, using the following ARPAnet address:

  publications@cray.com
- Send a facsimile of your comments to the attention of "Publications" at FAX number: (612) 681-5602
- Use the postage-paid Reader's Comment form at the back of this manual.

- Write to us at the following address:

  Cray Research, Inc.
  Technical Publications Department
  1345 Northland Drive
  Mendota Heights, Minnesota 55120

We value your comments and will respond to them promptly.

F. SITE COMMUNICATIONS

This appendix describes on-line diagnostic field support. This support includes the following:

- On-line diagnostic error dumps analysis
- On-line diagnostic formatted error output analysis
- On-line diagnostic installation, usage, and availability information

Please use one of the following options to forward inquiries to the On-line Diagnostic department:

- Call our On-line Diagnostic department at (612) 681-5642 during the hours of 8:00 A.M. to 5:00 P.M. (Central Time). From 5:00 P.M. to 8:00 A.M., you can leave a recorded message. Include the following information in your message.
  - Your name
  - Telephone number
  - Site identification
  - Operating system/release level
  - On-line diagnostic release
  - Failing on-line diagnostic
  - Description of the problem
- Send us electronic mail from a UNICOS or UNIX system, using the following electronic mail address:

  oldiag@Crayamid

- Write to us at the following address:

  Cray Research, Inc.
  On-line Diagnostic Department
  1345 Northland Drive
  Mendota Heights, Minnesota 55120

G. INSTALLATION INFORMATION

Typically, the on-line diagnostics are installed as part of the system installation procedure documented in the UNICOS System Installation Bulletin (SIB). If you need to re-install the on-line diagnostics subsequent to system installation, a different procedure must be used.

This appendix describes how to install the on-line diagnostics after system installation. The following topics are discussed:

- On-line diagnostic directories
- Generating on-line diagnostic binaries and listings
- Saving off-line versions of on-line confidence tests and I/O Subsystem (IOS) deadstart programs
- Generating **olnet**
- Deleting proprietary source code

G.1 ON-LINE DIAGNOSTIC DIRECTORIES

The on-line diagnostics are located in the following directories:

Directory	Description
/usr/src/diag	Source code
/ce/bin	On-line diagnostic binaries
/ce/oldmon	Off-line diagnostic binaries for **oldmon**
/ce/olnet	**olnet** source code for front-end computer systems
/ce/scripts	**runsequence** scripts
/ce/log	Log directory for **runsequence**
/ce/ios	IOS deadstart programs for single IOS systems
/ce/iosa **/ce/iosb**	IOS deadstart programs for two IOS systems

G.2 GENERATING ON-LINE DIAGNOSTIC BINARIES

Perform the following steps to generate on-line diagnostic binaries:

1. Load the on-line diagnostic tape. This tape is normally included with the UNICOS release package. If necessary, you can order another copy from the CRI Distribution Center.

2. Enter the following commands to execute the Makefile:

   ```
   cd /usr/src/diag
   update -p diagpl -q DIAGMAKE -c diag -a m
   mv diag.m diag.mk
   Make -f diag.mk install SN=xxxx
   ```

   xxxx is your mainframe's serial number.

G.3 GENERATING ON-LINE DIAGNOSTIC LISTINGS

To generate the on-line diagnostic listings, enter the following commands:

```
cd /usr/src/diag
make -f diag.mk listings
```

NOTE

The listings include all on-line diagnostic test listings, off-line versions of CPU on-line test listings, and IOS deadstart and **cleario** test listings.

The diagnostic listings are CRAY PROPRIETARY. Print the listings or write them to tape; do not keep the listings on-line.

G.4 SAVING OFF-LINE VERSIONS OF ON-LINE CONFIDENCE TESTS

This section describes where to save off-line versions of on-line confidence tests for Maintenance Workstation-based (MWS-based) systems running the Cray Maintenance System (CMS) or expander-based systems running DSS.

G.4.1 MWS-BASED SYSTEMS RUNNING CMS

Enter the following commands to copy the off-line confidence diagnostics to the MWS:

```
rcp /ce/oldmon/offcrit mws:/CPUDIR
rcp /ce/oldmon/offcsvc mws:/CPUDIR
rcp /ce/oldmon/offcfpt mws:/CPUDIR
rcp /ce/oldmon/offibuf mws:/CPUDIR
rcp /ce/oldmon/offcm mws:/CPUDIR
```

CPUDIR is the directory on the MWS where the CPU off-line diagnostics reside. *mws* is the hostname for the MWS.

G.4.2 EXPANDER-BASED SYSTEMS RUNNING DSS

1. Enter the following commands to write the off-line confidence diagnostics to a scratch tape:

```
extd -o -r -n 0 </ce/oldmon/offcrit
extd -o -r -n 1 </ce/oldmon/offcsvc
extd -o -r -n 2 </ce/oldmon/offcfpt
extd -o -r -n 3 </ce/oldmon/offibuf
extd -o -n 4 </ce/oldmon/offcm
```

NOTE

Steps 2 and 3 cannot be performed while the operating system is running. Perform these steps the next time you shut down your system.

2. Copy the diagnostics to the off-line expander pack under FNT 4. To copy the diagnostics from the tape that was just written, enter the following commands under DSS0:

```
READ @ 5CRIT 4
READ @ 5CSVC 4
READ @ 5CFPT 4
READ @ 5IBUF 4
READ @ 5CM 4
```

3. These off-line diagnostics are dependent on the latest off-line IOPPL release P2.0. This release of the Cray Maintenance Operating System (CMOS) allows diagnostics larger than 6000 words to be loaded and deadstarted. To load and execute these diagnostics, use the CMOS command **DS L.**

G.5 SAVING I/O SUBSYSTEM (IOS) DEADSTART PROGRAMS

This section describes where to save I/O Subsystem (IOS) deadstart programs for Operator Workstation (OWS), expander tape, or expander disk UNICOS.

G.5.1 OWS UNICOS

To copy the newly created **dsdiag** and **cleario** binaries to the OWS, enter the following commands:

rcp /ce/ios/dsdiag *ows:/IOSDIR*
rcp /ce/ios/dsdiag.ov *ows:/IOSDIR*
rcp /ce/ios/cleario *ows:/IOSDIR*
rcp /ce/ios/cleario.ov *ows:/IOSDIR*

IOSDIR is a site-specific parameter that indicates the location of the IOS kernel and overlays. *ows* is the hostname for the OWS. The deadstart diagnostics should reside in the same OWS directory as the IOS kernel and overlays. Two IOS systems will store diagnostics in two OWS directories based on the IOS serial number.

NOTE

Two IOS systems store diagnostics in directories **/ce/iosa/** and **/ce/iosb/**.

The deadstart diagnostic binaries are now saved on the OWS as files called **dsdiag, dsdiag.ov, cleario,** and **cleario.ov.**

G.5.2 EXPANDER TAPE UNICOS

Write the deadstart diagnostics to the same deadstart tape as the UNICOS kernel. To write the newly created deadstart diagnostic binaries to expander tape, enter the following commands:

```
extd -o -r -n 7 < /ce/ios/cleario
extd -o -r -n 8 < /ce/ios/dsdiag
extd -o -n 9 < /ce/ios/dsdiag.ov
```

NOTE

Two IOS systems store diagnostics in directories **/ce/iosa/** and **/ce/iosb/.**

The deadstart binaries are now saved on the expander tape as files called **CLEARIO, DSDIAG,** and **DSDIAG.OV.**

G.5.3 EXPANDER DISK UNICOS

To write the newly created **dsdiag** and **cleario** binaries to expander disk pack, enter the following commands:

exdf -o /*INSTALL*/dsdiag < /ce/ios/dsdiag
exdf -o /*INSTALL*/dsdiag.ov < /ce/ios/dsdiag.ov
exdf -o /*INSTALL*/cleario < /ce/ios/cleario

INSTALL is a site-specific parameter that indicates the location of **CLEARIO, DSDIAG,** and **DSDIAG.OV** on an expander disk. The deadstart diagnostic binaries should reside in the same directory as the UNICOS kernel and overlays.

NOTE

Two IOS systems store diagnostics in directories **/ce/iosa/** and **/ce/iosb/.**

_he deadstart binaries are now saved on the expander disk pack as files called **CLEARIO**, **DSDIAG**, and **DSDIAG.OV**.

G.6 GENERATING olnet

This section describes how to generate **olnet** for computer systems with the following front-ends:

- IBM
- Sun Workstation
- Motorola workstation, OWS, or MWS

G.6.1 IBM FRONT-END

The following **olnet** build procedure is intended for sites with front-end computer systems running VM.

1. Transfer the following files created during the UNICOS build procedure:

UNICOS Name	VM Name	Description
olnet.vm.f	file name OLNET file type FORTRAN	**olnet** Fortran source code
driver.vm.a	file name OLFEIV file type ASSEMBLE	**olnet** driver (BAL code)

Perform steps 2 through 6 from the CMS user environment:

2. Compile the **olnet** Fortran source code:

```
FORTVS OLNET
```

3. Access the VM/SP macro libraries:

LINK MAINT 194 194 RR (*a password may be required*)
ACCESS 194 B
ACC 194 I
GLOBAL MACLIB OSMACRO DMSSP DMKSP CMSLIB TSOMAC

4. Assemble the VM driver:

```
ASSEMBLE OLFEIV
REL B
REL I
```

5. Link the **olnet** driver and source code modules to create an executable binary module named OLNET:

```
GLOBAL TXTLIB VLNKMLIB VFORTLIB CMSLIB
LOAD OLNET OLFEIV
GENMOD OLNET
```

NOTE

The following step is required by the **olnet** licensing agreement.

---

6. Discard the following files:

File Name	File Type
OLNET	FORTRAN
OLNET	TEXT
OLFEIV	ASSEMBLE
OLFEIV	TEXT
LOAD	MAP

G.6.2 SUN WORKSTATION FRONT-END (NSC)

The following **olnet** NSC build procedure is intended for sites with Sun Workstation front-end computer systems.

1. Transfer the following files created during the UNICOS build procedure:

UNICOS Name	Sun Name	Description
olnet.sunnsc.f	olnet.sunnsc.f	**olnet** Fortran source code
drv.sunnsc.c	drv.sunnsc.c	**olnet** driver (C code)

2. Compile the **olnet** Fortran source code and C driver:

```
f77 -o olnet olnet.sunnsc.f drv.sunnsc.c
```

NOTE

The following step is required by the **olnet** licensing agreement.

3. Remove the following files:

```
rm olnet.sunnsc.f
rm olnet.sunnsc.o
rm drv.sunnsc.c
rm drv.sunnsc.o
```

G.6.3 SUN WORKSTATION FRONT-END (VME)

The following **olnet** VME build procedure is intended for sites with Sun Workstation front-end computer systems:

1. Transfer the following files created during the UNICOS build procedure:

UNICOS Name	Sun Name	Description
olnet.sunvme.f	olnet.sunvme.f	**olnet** Fortran source code
drv.sunvme.c	drv.sunvme.c	**olnet** driver (C code)

2. Compile the **olnet** Fortran source code and C driver.

```
f77 -o olnet olnet.sunvme.f drv.sunvme.c
```

NOTE

The following step is required by the **olnet** licensing agreement.

3. Remove the following files:

```
rm olnet.sunvme.f
rm olnet.sunvme.o
rm drv.sunvme.c
rm drv.sunvme.o
```

G.6.4 MOTOROLA WORKSTATION, OWS, OR MWS FRONT-END (VME)

The following **olnet** VME build procedure is intended for sites with Motorola workstation, OWS, or MWS front-end computer systems.

1. Transfer the following file created during the UNICOS build procedure:

UNICOS Name	Sun Name	Description
olnet.mot.c	olnet.mot.c	**olnet** C source code

2. Compile the **olnet** C source code and driver.

```
cc -o olnet olnet.mot.c
```

NOTE

The following step is required by the **olnet** licensing agreement.

3. Discard the following files:

```
rm olnet.mot.c
rm olnet.mot.o
```

G.7 DELETING PROPRIETARY SOURCE CODE

The CRAY1PL, XMPPL, and DIAGPL libraries contain source code that is CRAY PROPRIETARY. Therefore, the program libraries, source code, binaries, and listings must not be maintained on system storage.

Remove the source code files, listings, binaries, and program libraries from system storage by entering the following commands:

```
cd /usr/src/diag
make -f diag.mk delete
rm -f cray1pl xmppl diagpl cray1pl.mods xmppl.mods diagpl.mods
```

INDEX

INDEX

READER'S COMMENT FORM

CRAY Y-MP, CRAY X-MP EA, CRAY X-MP, and CRAY-1 Computer Systems
UNICOS On-line Diagnostic Maintenance Manual SMM-1012 C

Your reactions to this manual will help us provide you with better documentation. Please take a moment to check the spaces below, and use the blank space for additional comments.

1) Your experience with computers: ____ 0-1 year ____1-5 years ____5+ years
2) Your experience with Cray computer systems: ____0-1 year ____ 1-5 years ____5+ years
3) Your occupation: ____ computer programmer ____ non-computer professional
____ other (please specify): ______________________________
4) How you used this manual: ____ in a class ____as a tutorial or introduction ____ as a reference guide
____ for troubleshooting

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria:

5) Accuracy _____
6) Completeness ____
7) Organization ____
8) Physical qualities (binding, printing) ____
9) Readability ____
10) Amount and quality of examples ____

Please use the space below, and an additional sheet if necessary, for your other comments about this manual. If you have discovered any inaccuracies or omissions, please give us the page number on which the problem occurred. We promise a quick reply to your comments and questions.

Name ____________________________
Title ____________________________
Company _________________________
Telephone _______________________
Today's Date _____________________

Address _________________________
City ____________________________
State/ Country ____________________
Zip Code ________________________

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention: PUBLICATIONS
1345 Northland Drive
Mendota Heights, MN 55120

FOLD

READER'S COMMENT FORM

CRAY Y-MP, CRAY X-MP EA, CRAY X-MP, and CRAY-1 Computer Systems
UNICOS On-line Diagnostic Maintenance Manual SMM-1012 C

Your reactions to this manual will help us provide you with better documentation. Please take a moment to check the spaces below, and use the blank space for additional comments.

1) Your experience with computers: ____ 0-1 year ____1-5 years ____5+ years
2) Your experience with Cray computer systems: ____0-1 year ____ 1-5 years ____5+ years
3) Your occupation: ____ computer programmer ____ non-computer professional
   ____ other (please specify): ____________________________
4) How you used this manual: ____ in a class ____as a tutorial or introduction ____ as a reference guide
   ____for troubleshooting

Using a scale from 1 (poor) to 10 (excellent), please rate this manual on the following criteria:

5) Accuracy _____
6) Completeness ____
7) Organization ____
8) Physical qualities (binding, printing) ____
9) Readability ____
10) Amount and quality of examples ____

Please use the space below, and an additional sheet if necessary, for your other comments about this manual. If you have discovered any inaccuracies or omissions, please give us the page number on which the problem occurred. We promise a quick reply to your comments and questions.

Name ________________________
Title ________________________
Company ________________________
Telephone ________________________
Today's Date ________________________

Address ________________________
City ________________________
State/ Country ________________________
Zip Code ________________________

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention: PUBLICATIONS
1345 Northland Drive
Mendota Heights, MN 55120

FOLD

CUT ALONG THIS LINE