For:jmo
Printed on:Tue, Oct 21, 1997 13:25:28
From book:front_matter
Document:Title_Page
Last saved on:Tue, Oct 21, 1997 13:11:34
Document:proprietary_copy/trade
Last saved on:Tue, Oct 21, 1997 13:11:34
Document:Rec_of_Rev
Last saved on:Tue, Oct 21, 1997 13:11:53
Document:Preface
Last saved on:Tue, Oct 21, 1997 13:11:35
Document:TOC
Last saved on:Tue, Oct 21, 1997 13:11:36
Document:1_Overview
Last saved on:Tue, Oct 21, 1997 13:01:48
Document:2_MCE
Last saved on:Tue, Oct 21, 1997 13:07:47
Document:3_ENV0
Last saved on:Tue, Oct 21, 1997 13:01:52
Document:4_ENV 1& 2
Last saved on:Tue, Oct 21, 1997 13:09:07
Document:5_LME
Last saved on:Tue, Oct 21, 1997 13:01:59
Document:6_CBP
Last saved on:Tue, Oct 21, 1997 13:02:00
(...)

CRAY C90™ Series Mainframe Offline Diagnostic Manual

CDM-0505-0D0

Cray Research, Inc.

Requests for copies of Cray Research, Inc. publications should be directed to:

CRAY RESEARCH, INC.
Logistics
6251 South Prairie View Road
Chippewa Falls, WI 54729

Comments about this publication should be directed to:

CRAY RESEARCH, INC.
Hardware Publications and Training
890 Industrial Blvd.
Chippewa Falls, WI 54729

Record of Revision

Each time this manual is updated with a change packet, a change to part of a text page is indicated by a change bar in the margin directly opposite the change. A change bar in the footer of a text page indicates that most, if not all, of the text is new. A change bar in the footer of a page composed primarily of a table and/or figure may indicate that a change was made to that table/figure or, it could indicate that the entire table/figure is new. Change packets are assigned a numerical designator, which is indicated in the publication number on each page of the change packet.

Each time this manual is fully revised and reprinted, all change packets to the previous version are incorporated into the new version, and the new version is assigned an alphabetical revision level, which is indicated in the publication number on each page of the manual. A revised manual does not usually contain change bars.

REVISION	DESCRIPTION
	October 1992. Original printing.
001	December 1992. This change packet adds an index and updates the Table of Contents.
A	June 1993. This revision reorganizes and updates the manual to correspond with the diagnostics included in the ME-C2.0 offline diagnostic release. It includes new and updated information about the mainframe configuration environment, mainframe maintenance environment (environments 0, 1, and 2), logic monitor environment, command buffer parser, diagnostics and utilities, simulator and debugger, diagnostic controller, and remote support capabilities of MME. This revision changes the manual title to *CRAY C90 Series Mainframe Offline Diagnostic Reference Manual* and makes all previous versions obsolete.
B	November 1993. This revision corresponds with the diagnostics included in the ME-C2.1 offline diagnostic release. This revision adds system and CPU status and flaw chip management information to the MCE section, updates the MME sections, adds interface information to the LME section, documents the new CBP application in the CBP section, updates the diagnostic descriptions and troubleshooting information, adds an index, and makes all previous versions obsolete.
C	April 1994. This revision corresponds with the diagnostics included in the ME-C2.3 offline diagnostic release. This revision documents changes to MME and MCE for DRAM memory support, updates the command buffer parser description to focus on the CRAY C90 CBP runtime module, enhances the remote support information, includes technical and editorial changes throughout the manual, and makes all previous versions obsolete.

REVISION	DESCRIPTION
D	March 1995. This revision corresponds with the diagnostics included in the ME-C2.3.1 offline diagnostic release. This revision updates MME, MCE, LME, and bugger/debugger information and includes technical and editorial changes throughout the manual. This revision also removes Section 10, “Diagnostic Controller.” This information has been incorporated in the *CRAY C90 Series MME Reference*, publication number HDM-081-0.
0D1	September 1995. This change packet includes pages 4-81 through 4-99, which were inadvertently omitted from CDM-0505-0D0. This change packet also includes technical corrections to Table 2-3.

PREFACE

The *CRAY C90 Series Mainframe Offline Diagnostic Manual* is for Cray Research, Inc. (CRI) field engineers (FEs). Other readers include CRI Systems Test and Check-out (STCO) employees and hardware training students. Readers should have at least 2 years of technical training, a working knowledge of electrical concepts, and an understanding of the architecture of the CRAY C90 series computer system. Readers do not need extensive software knowledge but must know the basics of the UNIX operating system.

This manual and the diagnostic programs described in it also support troubleshooting for the CRAY C90D series mainframes. All references to CRAY C90 series mainframes also apply to CRAY C90D series mainframes. All references to DRAM memory are specific to troubleshooting a CRAY C90D series mainframe.

How to Use this Manual

This manual combines theory, illustrations, and procedures. Use this manual for reference in the field and in STCO when using offline diagnostics to maintain or repair a CRAY C90 series mainframe. This manual provides information about offline diagnostic programs and explains how to use the programs to identify and correct hardware problems. This manual is divided into the following sections:

- Section 1, “Overview,” provides an overview of the CRAY C90 series mainframe offline diagnostic set.
- Section 2, “Mainframe Configuration Environment,” provides information about the mainframe configuration environment (MCE) used to configure the mainframe maintenance environment (MME) and logic monitor environment (LME) programs.
- Section 3, “MME Environment 0,” provides information about how to use MME environment 0.
- Section 4, “MME Environments 1 and 2,” provides information about how to use MME environments 1 and 2.
- Section 5, “Logic Monitor Environment,” provides information about how to use LME.

- Section 6, “CRAY C90 CBP Runtime Module,” provides information about the CRAY C90 CBP runtime module used with the command buffer parser (CBP) application.

- Section 7, “Diagnostic Tests and Utilities,” provides descriptions of the diagnostic tests and utilities that can be used with MME.

- Section 8, “Simulator and Bugger/Debugger,” provides information about how to use the simulator and debugger.

- Section 9, “Troubleshooting,” provides troubleshooting examples.

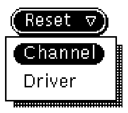
- “Appendix: CRAY C90 Remote Support” provides information on how to perform remote support from a service center through a hub.

Reader comment forms are located at the front and the back of this manual. Please use them to offer comments or suggestions. You can obtain information about other hardware and software publications by using the online publications catalog.

Notational Conventions

The following conventions are used throughout this manual:

- The base of a number is decimal unless otherwise stated. All memory references are in octal.
- `Courier` type indicates directory pathnames, filenames, commands, and screen output.
- **`Courier bold`** type indicates commands, options, and field inputs that you should enter.
- Buttons are shown as they appear in a window; for example, Go.
- Settings are shown as they appear in a window; for example, Automatic.
- Pulldown menu selections appear in the left margins when the text indicates you should choose a menu option. An example is shown at left.



- The following conventions are used for command usage descriptions:
  - The ↵ symbol indicates pressing the return key.
  - The **`-->`** symbol indicates holding the MENU mouse button down and moving the mouse pointer to the next menu item.
  - *Italic* type indicates a variable or user-supplied entry.
  - Commands must be entered as shown in the command syntax. Spaces must be included or left out as shown; do not use tabs.
  - Square brackets [ ] indicate an optional entry.
  - Angle brackets < > indicate a required entry.
  - A vertical bar | indicates a choice.

Related Information

Please refer to the following related information:

- The *CRAY C90 Series LME System Monitor Utility* document, publication number HDM-120-0, provides detailed information about the LME System Monitor (SMON) utility.
- The *CRAY C90 Series Mainframe Offline Diagnostic Booklet*, publication number CQH-0509-0A0, provides quick-reference information for the CRAY C90 series offline diagnostic set.
- The *CRAY C90 Series MME Reference* document, publication number HDM-081-0, provides detailed information about MME environments 1 and 2.
- The *CRAY C90 Series Memory Chip Flawing* document, publication number HMM-104-A, describes memory chip flawing.

The following Sun Microsystems, Inc. manuals provide advanced-user information about the features of OpenWindows software.

OpenWindows Version 3 End User's Manuals, part number 851-1390-01, which includes the following manuals:

OpenWindows Version 3 User's Guide, part number 800-6618-10

Desk Set Environment Reference Guide, part number 800-6231-10

OpenWindows Version 3 Release Notes, part number 800-6006-10

OpenWindows Version 3 Installation & Start-up Guide, part number 800-6029-10

This manual does not address the various features and functionality provided by OpenWindows software. It is assumed that the user is familiar with OpenWindows software.

CONTENTS

5 LOGIC MONITOR ENVIRONMENT

5 LOGIC MONITOR ENVIRONMENT (continued)

6 CRAY C90 CBP RUNTIME MODULE

8 SIMULATOR AND BUGGER/DEBUGGER (continued)

9 TROUBLESHOOTING

APPENDIX: CRAY C90 REMOTE SUPPORT

FIGURES (continued)

FIGURES (continued)

TABLES

TABLES (continued)

1 OVERVIEW

This manual provides information about the offline diagnostic set used to troubleshoot CRAY C90 and CRAY C90D series computer systems. The major components of this set are the mainframe configuration environment (MCE), the mainframe maintenance environment (MME), the MME keyboard processor, the logic monitor environment (LME), the command buffer parser (CBP), and the instruction level simulator (MSIM) and mainframe debugger (MDB). These maintenance workstation (MWS-E) based applications run under the OpenWindows 3.0 user environment. Table 1-1 provides a brief description of each application and shows the icons that appear on the OpenWindows desktop.

NOTE: This manual and the diagnostic programs described in it also support troubleshooting for the CRAY C90D series mainframes. All references to CRAY C90 series mainframes also apply to CRAY C90D series mainframes. All references to DRAM memory are specific to troubleshooting a CRAY C90D series mainframe.

Table 1-1. Components of the Offline Diagnostic Set

Application	Description	Icon
Mainframe configuration environment (MCE)	An X Window System based application used to configure parameters used by MME and LME, to specify concurrent or offline mode, and to create a table of flawed-chip data. (Refer to Section 2 for more information about MCE.)	CRAY RESEARCH, INC. MCE Y-MP C90
Mainframe maintenance environment (MME)	An X Window System based application that uses features of the maintenance channel such as individual CPU control and direct memory access to provide an advanced troubleshooting environment. (Refer to Sections 3 and 4 for more information about MME.)	CRAY RESEARCH, INC. MME Y-MP C90
MME keyboard processor	A Cray maintenance system (CMS)-style interface that provides control over one diagnostic test, utility, or loop. (Refer to “Using the Keyboard Processor in Environment 1 or 2” in Section 4 for more information about the MME keyboard processor.)	CRAY RESEARCH, INC. MME Keyboard Processor

If you start an application with a copy number, the copy number appears at the bottom of the MCE, MME, and LME icons.

Table 1-1. Components of the Offline Diagnostic Set (continued)

Application	Description	Icon
Logic monitor environment (LME)	An X Window System based application that provides an interface to the CRAY C90 series diagnostic monitor (DM). (Refer to Section 5 for more information about LME.)	CRAY RESEARCH, INC. LME Y-MP C90
Command buffer parser (CBP)	An X Window System based application that automates troubleshooting. (Refer to Section 6 for more information about the CRAY C90 CBP runtime module.)	CRAY RESEARCH, INC. CBP
Mainframe debugger	An X Window System based application that works with the instruction level simulator (MSIM) to provide additional learning opportunities through hardware control and simulated hardware failures. (Refer to Section 8 for more information about MSIM and MDB.)	MDB

If you start an application with a copy number, the copy number appears at the bottom of the MCE, MME, and LME icons.

Maintenance Channel Operation

The applications included in the offline diagnostic set use the maintenance channel through a direct connection to the MWS-E. Many functions are performed on the maintenance channel to facilitate troubleshooting a CRAY C90 series computer system. Therefore, it is useful to have an understanding of the maintenance channel operation.

The maintenance channel interface/controller is located on the CRAY C90 series mainframe clock module. The maintenance channel of the CRAY C90 series computer systems provides extensive control over the system and the CPUs for maintenance and problem diagnosis. The maintenance channel uses the same protocol and cables as the low-speed (LOSP) channels.

The basic functions provided by the channel are as follows:

- System level
  - I/O master clear
  - CPU master clear
  - Memory priority counter
  - Master CPU selection (software switch)
  - External control cable selection (software switch)
  - System-level status
- Individual CPU level
  - I/O master clear
  - CPU master clear and exchange
  - Memory modes (half-memory mode and 256-Kword memory mode)
  - Control of software switches on the CPU
  - Reading from and writing to memory using CA/CL
  - Interface to the diagnostic monitor
  - CPU level status
  - Loopback

The channel connection from the MWS-E communicates by means of a 16-bit parallel asynchronous interface. This interface connects to the maintenance channel interface on the clock module. The 16 data bits plus parity bits are decoded, checked for errors, and serialized for transmission to the 16 CPUs in the system. The format of the data transmitted varies, depending on the type of function being performed. Three different transfer lengths are normally used: most functions use 4 parcels, memory read functions use 8 parcels, and memory write functions or diagnostic monitor parameter write functions use 12 or more parcels. If the CPUs return data to the MWS-E, the CPU transmits serial data to the maintenance channel interface on the clock module, which converts it to the 16 data bits plus parity bits required by the channel.

Hardware and Software Switches for the Maintenance Channel

The CRAY C90 mainframe default configuration switches enable, disable, and restrict the mode of operation of the maintenance channel. In addition, other hardware switches on the default board provide initial values for the software switches. The MWS-E can initiate a function to change the setting of the software switches; as a result, the software switches override the hardware switch settings. Figure 1-1 shows the relationship of the maintenance channel enable/disable and restricted switches.

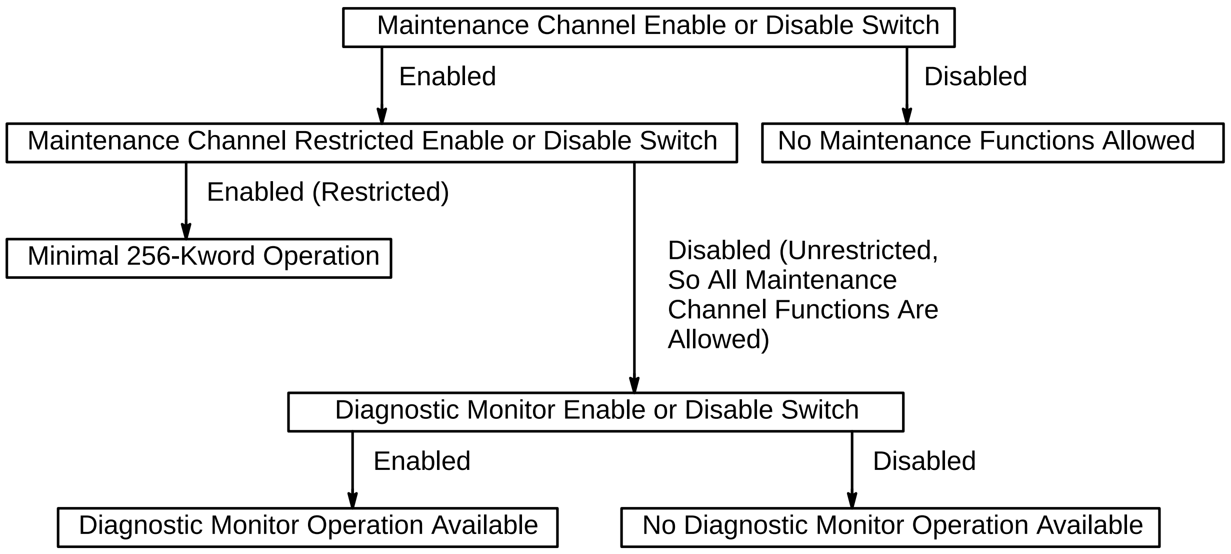

Maintenance Channel Enable or Disable Switch
Enabled
Disabled
Maintenance Channel Restricted Enable or Disable Switch
No Maintenance Functions Allowed
Enabled (Restricted)
Minimal 256-Kword Operation
Disabled (Unrestricted, So All Maintenance Channel Functions Are Allowed)
Diagnostic Monitor Enable or Disable Switch
Enabled
Disabled
Diagnostic Monitor Operation Available
No Diagnostic Monitor Operation Available

Figure 1-1. Maintenance Channel Enable/Disable and Restricted Switch Relationships

Testing Overview

You use the MCE, MME, LME, and CBP programs to troubleshoot the CRAY C90 series mainframes.

MCE

MCE is the application you use to configure the parameters used by MME and LME. Using MCE, you can configure the soft switches; mainframe, memory, SSD, and CPU parameters; and channels used by MME and LME. You can also use MCE to view the status of a CPU and to create a table of flawed-chip data. When you save this table, the operating system (OS) uses the data to specify which spare chips are used at boot time. MCE operates in concurrent (with the OS) or offline mode. MME and LME have different levels of functionality depending on the mode you choose.

MME

MME is the application you use to load and run the diagnostics used to troubleshoot a CRAY C90 series computer system. MME supports three troubleshooting environments: environment 0, environment 1, and environment 2.

MME Environment 0

Environment 0 tests specific areas of the mainframe to a degree that ensures environment 1 functionality. The areas tested are:

- The maintenance channel
- The diagnostic monitor
- The mainframe memory
- The CPU exchange mechanism
- The CPU instruction buffers

These areas are tested from the MWS-E through a sequence of maintenance channel functions.

Environment 0 has three modes of operation. In automatic mode, MME runs a series of predefined sequences for selected areas against selected CPUs. In manual mode, only one area may be under test at a time, but individual predefined sequences may be selected and the selected sequences are run against all selected CPUs. In compose mode, a customized sequence may be generated or one of the predefined sequences may be edited. Various displays show information needed to analyze hardware failures.

This environment replaces the level 0 tests and boot tests used on the previous CRAY Y-MP computer systems.

If no default configuration is set by MCE, environment 0 defaults to concurrent mode to prevent accidental crashing of the mainframe. In concurrent mode, you can switch to environment 1 or 2 to continue working in concurrent mode, or you can use MCE to switch to offline mode. Testing in environment 0 is done in offline mode. This gives environment 0 full access to the mainframe but means that no CPUs can be running the operating system (OS) or the OS will crash.

MME Environments 1 and 2

Environments 1 and 2 are similar in operation. The point of testing is moved from the MWS-E to the mainframe. Control points, generic names used to reference diagnostic programs, utilities, or loops, are loaded into mainframe memory. Memory allocation and control point configuration are controlled by MME by means of user settings and information available from the maintenance channel. CPUs are assigned to the control points, and MME provides users control for starting and stopping the CPU execution of control points. Various displays show information to analyze hardware failures.

Environments 1 and 2 can be run in two modes: concurrent and offline. Concurrent mode enables troubleshooting while the OS is running in some CPUs. Offline mode provides full access to the mainframe but requires that the OS is stopped in all CPUs.

CAUTION

Currently, UNICOS does not support MME in concurrent mode. If you apply a configuration with CPUs in 256K mode to use concurrent mode, the OS will crash. You must use offline mode for troubleshooting.

In concurrent mode, the OS is running and has control of the mainframe. MME runs in restricted mode, using the upper 256 Kwords of memory and the CPU(s) you specify as usable in the MCE `Soft Switches` window. These CPUs are taken from use by the OS and run the diagnostics and utilities you load in MME. Concurrent mode is useful for troubleshooting bad CPU(s) while the OS runs in the other CPUs.

In offline mode, you will need to stop the OS in the mainframe before you start MME or the OS will crash. Once you have stopped the OS, MME has total control of the mainframe and has access to all of memory. This enables you to perform extensive troubleshooting of the mainframe.

MME starts in concurrent mode by default to ensure that you do not accidentally crash the OS in the mainframe. You can force MME environments 1 and 2 into offline mode with the `–offline` option; refer to the descriptions of starting MME environments 1 and 2 in Section 4 for more information. You can also use MCE to switch from concurrent to offline mode; refer to the description of MCE in Section 2 for more information.

Environment 1

Environment 1 enables one diagnostic and/or one or more loops to be loaded into memory. MME loads all diagnostics at memory address 0; loops are loaded at an origin. Generally, because only one control point resides in the mainframe at a given time, it has access to all the resources, such as memory, I/O channels, and shared registers.

Environment 1 replaces the MM and MI monitors used in the previous CRAY Y-MP computer systems. Monitors are no longer required to perform these functions because the maintenance channel functions are now available, such as individual CPU control and direct memory access.

In environment 1, all control points (except loops) contain a segment of code called the interrupt router. The interrupt router provides one method of handling interrupts among all the control points.

Environment 2

Environment 2 enables one or more diagnostics or utilities to be loaded into memory. For example, you can have MME load multiple copies of the same diagnostic or utility, or you can have MME load different diagnostics or utilities. Several memory-allocation schemes are available that cause the control points to be loaded starting from the lowest memory location (bottom up), starting from the highest memory location, randomly (top down), or equally (partitioned).

A small code segment called *the controller* resides in the lower 040000_8 words of memory. Because there may be more than one control point in memory, the controller negotiates the sharing of resources, such as I/O channels and shared registers.

Environment 2 controls run system operation. The run system automatically rotates the CPUs among various control points, which simulates an operating system environment.

Environment 2 replaces the M8 and run system monitors used in the previous CRAY Y-MP computer systems.

LME

LME is the application you use to the access the diagnostic monitor (DM) hardware. LME causes the DM to start and stop recording on specific events that occur while instructions are executing. This enables you to monitor CPU(s) as instructions execute without affecting any CPUs. You can also use LME to compare the actual results with expected results that you provide.

LME runs in offline and concurrent environments (modes). In the offline environment (mode), LME can perform all DM functions, read mainframe memory through any CPU, and write mainframe memory through any CPU; the OS cannot be running in the mainframe.

In the concurrent environment (mode), LME can perform DM functions and read memory through any CPUs you specify as usable. LME can perform DM functions, read memory, and write memory through any CPUs that you specify as usable and for which you set 256K mode. Use the MCE soft-switch settings to specify CPUs that are usable and in 256K mode.

CBP

CBP is the application you use to create, run, debug, and edit files of the commands, called command buffers, used to automate troubleshooting. The CRAY C90 CBP runtime module contains the commands used to control the CRAY C90 versions of MME, MCE, and LME.

2 MAINFRAME CONFIGURATION ENVIRONMENT

The mainframe configuration environment (MCE) is an X Window System based application that enables you to configure parameters used by the mainframe maintenance environment (MME) and the logic monitor environment (LME). Using MCE, you can configure the soft switches; mainframe, memory, SSD, and CPU parameters; and channels used by MME and LME. You can also use MCE to view the status of a CPU and to create a table of flawed-chip data. When you save this table, the operating system (OS) uses the data to specify which spare chips are used at boot time. MCE also specifies whether MME and LME run in concurrent or offline mode.

Starting MCE

You can start MCE from MME or LME, from the OpenWindows desktop workspace menu, or from a UNIX command prompt.

Information about how to start MCE from a service center through a hub is included in the appendix to this material.

From MME Environments 0, 1, or 2

To start MCE from MME environments 0, 1, or 2; choose **`Utilities ––> Configuration`**.

From LME

To start MCE from the LME base window, choose **`Utilities ––> Configuration (MCE)`**.

From the OpenWindows Desktop Workspace Menu

To start MCE as a stand-alone program from the OpenWindows desktop workspace menu, perform one of the following actions:

- Choose **`Maintenance Tools ––> MME ––> MCE ––> Copy#`** to start MCE with the copy number specified by the `Copy#` selection.

The copy number option enables you to differentiate between multiple independent MCE sessions that are supported from the same MWS-E. Copy numbers 0, 1, 2, and 3 are available from the workspace menu. The copy number does not affect performance of MCE; it serves as an identifier only.

- Choose **`Maintenance Tools --> MME Simulator --> MCE`** to start MCE with the simulator.

From a UNIX Command Prompt

To start MCE from a UNIX prompt, type **`mce`** followed by any appropriate command line options and press the return (↵) key. Table 2-1 describes the available command line options. In the following list of commands, parameters in angle brackets < > are required, parameters in brackets [ ] are optional, and a vertical bar indicates an either /or choice.

```
mce     [–copy <num>]
        [–kill]
        [–config <file>]
        [–concurrent | –offline]
        [–chn<num> | -sim]
        [–remote <host> | –client | –server]
```

NOTE: You can also enter any of the command line options shown in Table 2-1.

Table 2-1. Command Line Options

Option	Description
`–chn<`*`num`*`>`	Use FEI channel specified by <*num*>, which can range from 0 to 7. The default channel is channel 1.
`–client`	Start the client only
`–concurrent`	Use concurrent mode
`–config <`*`file`*`>`	Configure MCE with the configuration data stored in the file specified by <*file*>
`–copy <`*`num`*`>`	Connect to maintenance software assigned to the copy number specified by <*num*>. This option allows you to differentiate which system is being supported by this session of the software.
`–kill`	Kill all MME, MCE, and LME processes
`–offline`	Use offline mode

Table 2-1. Command Line Options (continued)

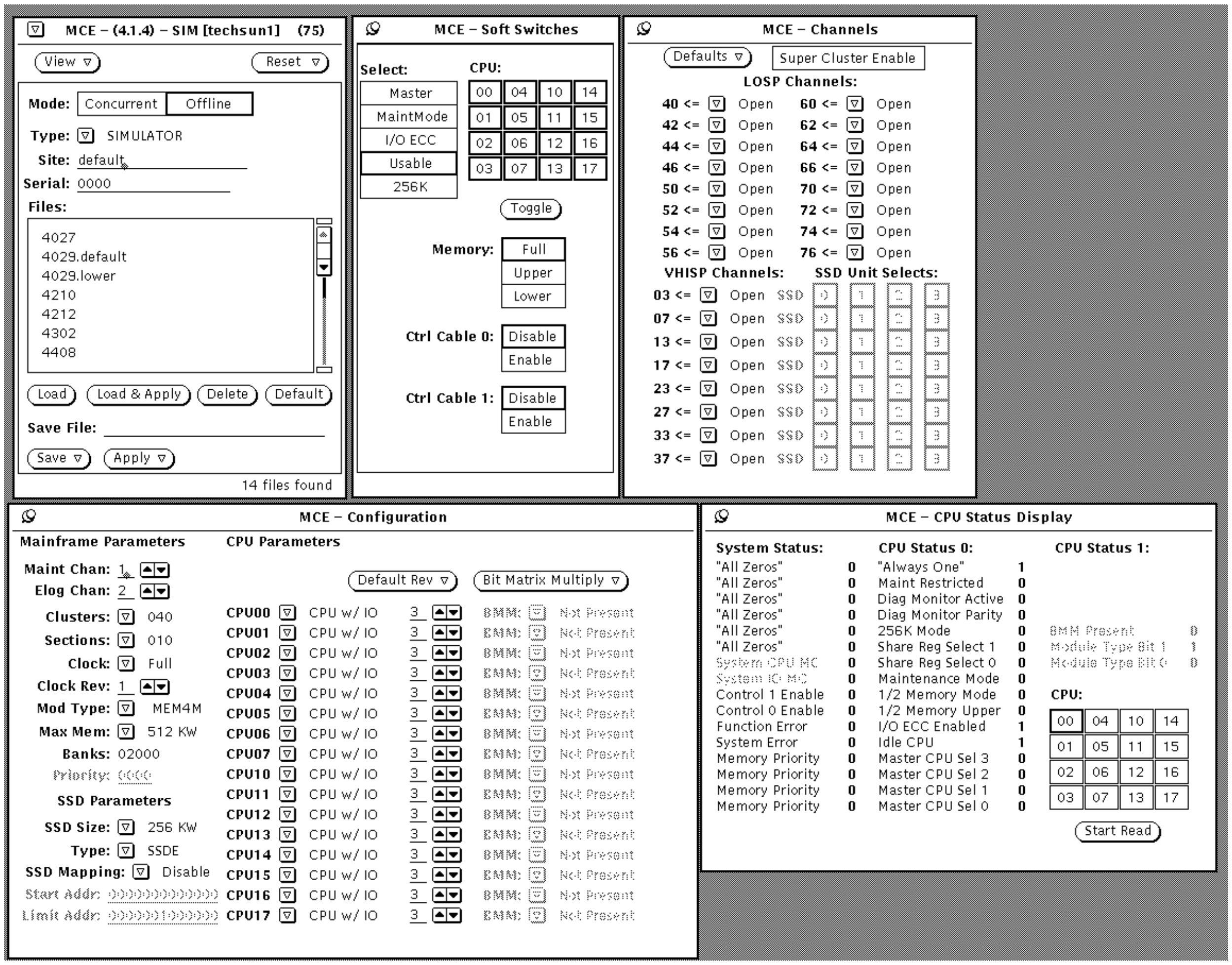
Option	Description
`–remote <host>`	Start client only, connect to remote host
`–server`	Start server only
`–sim`	Use simulator

For example, enter **`mce –copy 5 –chn2`**↵ to start MCE using copy number 5 and FEI channel 2.

MCE uses the following six windows:

- `MCE` base window
- `MCE – Soft Switches` window
- `MCE – Configuration` window
- `MCE – Channels` window
- `MCE – CPU Status Display` window
- `MCE – Flaw Chip Management` window

By default, the first five of these windows are displayed when MCE is started, as shown in Figure 2-1. To open the sixth window (`Flaw Chip Management` window), choose **`View ––> Flaw Chip`**.



MCE – (4.1.4) – SIM [techsun1] (75)

Mode: Concurrent | Offline
Type: SIMULATOR
Site: default
Serial: 0000
Files:
4027
4029.default
4029.lower
4210
4212
4302
4408
Save File:
14 files found

MCE – Soft Switches

Select: Master, MaintMode, I/O ECC, Usable, 256K

CPU:

00	04	10	14
01	05	11	15
02	06	12	16
03	07	13	17

Memory: Full, Upper, Lower
Ctrl Cable 0: Disable, Enable
Ctrl Cable 1: Disable, Enable

MCE – Channels

Super Cluster Enable

LOSP Channels:

40 <=	Open	60 <=	Open
42 <=	Open	62 <=	Open
44 <=	Open	64 <=	Open
46 <=	Open	66 <=	Open
50 <=	Open	70 <=	Open
52 <=	Open	72 <=	Open
54 <=	Open	74 <=	Open
56 <=	Open	76 <=	Open

VHISP Channels:		SSD Unit Selects:				
03 <=	Open	SSD	0	1	2	3
07 <=	Open	SSD	0	1	2	3
13 <=	Open	SSD	0	1	2	3
17 <=	Open	SSD	0	1	2	3
23 <=	Open	SSD	0	1	2	3
27 <=	Open	SSD	0	1	2	3
33 <=	Open	SSD	0	1	2	3
37 <=	Open	SSD	0	1	2	3

MCE – Configuration

Mainframe Parameters
Maint Chan: 1
Elog Chan: 2
Clusters: 040
Sections: 010
Clock: Full
Clock Rev: 1
Mod Type: MEM4M
Max Mem: 512 KW
Banks: 02000
Priority: 000

SSD Parameters
SSD Size: 256 KW
Type: SSDE
SSD Mapping: Disable
Start Addr: 000000000000
Limit Addr: 000000100000

CPU Parameters

CPU00	CPU w/ IO	3	BMM:	Not Present
CPU01	CPU w/ IO	3	BMM:	Not Present
CPU02	CPU w/ IO	3	BMM:	Not Present
CPU03	CPU w/ IO	3	BMM:	Not Present
CPU04	CPU w/ IO	3	BMM:	Not Present
CPU05	CPU w/ IO	3	BMM:	Not Present
CPU06	CPU w/ IO	3	BMM:	Not Present
CPU07	CPU w/ IO	3	BMM:	Not Present
CPU10	CPU w/ IO	3	BMM:	Not Present
CPU11	CPU w/ IO	3	BMM:	Not Present
CPU12	CPU w/ IO	3	BMM:	Not Present
CPU13	CPU w/ IO	3	BMM:	Not Present
CPU14	CPU w/ IO	3	BMM:	Not Present
CPU15	CPU w/ IO	3	BMM:	Not Present
CPU16	CPU w/ IO	3	BMM:	Not Present
CPU17	CPU w/ IO	3	BMM:	Not Present

MCE – CPU Status Display

System Status:		CPU Status 0:		CPU Status 1:	
"All Zeros"	0	"Always One"	1		
"All Zeros"	0	Maint Restricted	0		
"All Zeros"	0	Diag Monitor Active	0		
"All Zeros"	0	Diag Monitor Parity	0		
"All Zeros"	0	256K Mode	0	BMM Present	0
"All Zeros"	0	Share Reg Select 1	0	Module Type Bit 1	1
System CPU MC	0	Share Reg Select 0	0	Module Type Bit 0	0
System IO MC	0	Maintenance Mode	0		
Control 1 Enable	0	1/2 Memory Mode	0		
Control 0 Enable	0	1/2 Memory Upper	0		
Function Error	0	I/O ECC Enabled	1		
System Error	0	Idle CPU	1		
Memory Priority	0	Master CPU Sel 3	0		
Memory Priority	0	Master CPU Sel 2	0		
Memory Priority	0	Master CPU Sel 1	0		
Memory Priority	0	Master CPU Sel 0	0		

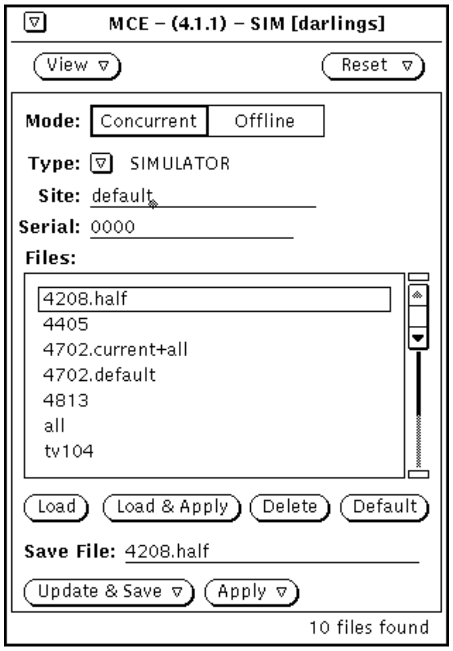
CPU:

00	04	10	14
01	05	11	15
02	06	12	16
03	07	13	17

Figure 2-1. MCE Windows (Initial Setup)

MCE Base Window

Figure 2-2 shows the MCE base window. Using this window, you can view the `Soft Switches` window, the `Configuration` window, the `Channels` window, or the `Flaw Chip Management` window. You can also load, save, delete, and apply the configuration files used by MCE.



Mode: Concurrent | Offline

Type: SIMULATOR

Site: default

Serial: 0000

Files:

4208.half
4405
4702.current+all
4702.default
4813
all
tv104

Save File: 4208.half

10 files found

Figure 2-2. MCE Base Window

NOTE: A default file extension indicates a default file; refer to the description of the button in Table 2-3.

In the MCE base window, you can set the mode under which MME environments 0, 1, and 2 run. Click on `Mode:` Concurrent to run in concurrent mode, which enables you to troubleshoot the CPUs you select in the `Soft Switches` window while the operating system is running in the other CPUs. Click on `Mode:` Offline to troubleshoot the entire mainframe; in offline mode, no CPUs can be running the operating system. Many MCE functions are grayed-out in concurrent mode because the maintenance channel is placed in restricted mode.

CAUTION

Do not configure a usable CPU in 256K mode. UNICOS will crash when you apply the configuration. Currently, UNICOS does not support concurrent maintenance.

Use the MCE base window to specify the type of mainframe you are troubleshooting; choose one of the following options from the `Type:` ▽ to specify the mainframe type. Refer to Table 2-2 for descriptions of the mainframe types that are available.

Table 2-2. Mainframe Type Settings

Setting	Mainframe Type
`mf16 (4000)`	4000 series mainframe
`mf2  (4200)`	4200 series mainframe
`mf4  (4400)`	4400 series mainframe
`mf6  (4600)`	4600 series mainframe
`mf8  (4800)`	4800 series mainframe
`tv22`	Test vehicle with 2 CPUs and 2 memory modules
`tv12`	Test vehicle with 1 CPU and 2 memory modules
`tv11`	Test vehicle with 1 CPU and 1 memory module
`mf2D (4300)`	4300 series mainframe
`mf4D (4700)`	4700 series mainframe
`mf8D (4900)`	4900 series mainframe
`tv12 DRAM`	Test vehicle with 1 CPU and 2 DRAM memory modules
`SIMULATOR`	Mainframe simulator (msim)
`UNKNOWN`	Unknown mainframe type

NOTE: Although you can enter anything in the `Site` and `Serial` fields, it is recommended that you enter your actual site and serial number. The `Channels` window requires the proper serial number information to display the correct channel number information for your mainframe.

Refer to Table 2-3 for descriptions of the buttons in the MCE base window.

Table 2-3. MCE Base Window Buttons

Button	Description
View ▽	Views all windows or the soft switches, configuration, channels, CPU status display, or flaw chip window(s).
Reset ▽	Resets all configuration data to the last applied values (choose **`Reset --> MCE to last configuration`**) or to the current hardware configuration (choose **`Reset --> MCE to hardware`**) that MCE can detect.
Load	Loads the configuration file selected in the scroll box.
Load & Apply	Loads the configuration file selected in the scroll box and sends the information to MME, LME, and the soft switches in the mainframe.
Delete	Deletes the configuration file selected in the scroll box.
Default	Makes the selected configuration file the default file that loads when MCE is started; the default file is indicated by a `.default` file extension. You can turn off the default status of a file by clicking on this button again.
Save ▽	Saves the configuration and writes the spare-chip data files (choose **`Save --> Write OS and EASE files`**). This command writes an ASCII file of spare-chip data on the OWS-E for the operating system (OS). It writes an ASCII file of spare-chip data on the MWS-E for `xelog`. It also writes a binary file of spare-chip data on the MWS-E for the Error Acquisition SoftwarE program (EASE). This command saves the current configuration, including spare-chip data, under the name specified in the `Save File` field. or Saves the configuration without writing the OS, `xelog`, and EASE files (choose **`Save --> Do not write OS and EASE files`**). This command saves the current configuration, including spare-chip data, under the name specified in the `Save File` field.
Apply ▽	Applies the current configuration and writes the spare-chip data files (**`Apply --> Write EASE file`**). This selection sends the current system configuration data to MME, LME, and the soft switches in the mainframe; performs an I/O master clear to enable the spare-chip latches to be written; initializes the maintenance channel; and writes spare-chip data to memory to close the latches. It writes an ASCII file of spare-chip data on the MWS-E for `xelog`. It also writes a binary file of spare-chip data on the MWS-E for EASE. or Configures the hardware without writing the `xelog` and EASE files, (choose **`Apply --> Do not write Ease file`**). This command sends the current system configuration data to MME, LME (in concurrent mode), and the soft switches in the mainframe; performs an I/O master clear to enable the spare-chip latches to be written, initializes the maintenance channel; and writes spare-chip data to memory to close the latches. **NOTE:** In concurrent mode, MCE sends the configuration data and spare-chip data to MME and LME but does not send this data to the mainframe.

Soft Switches Window

The `Soft Switches` window, shown in Figure 2-3, enables you to configure the mainframe software switches. Use this window to select the CPUs for the `Master`, `MaintMode`, `I/O ECC`, `256K`, and `Usable` configurations; to select `Full`, `Upper`, or `Lower` memory; and to `Disable` or `Enable` control cables 0 and 1.

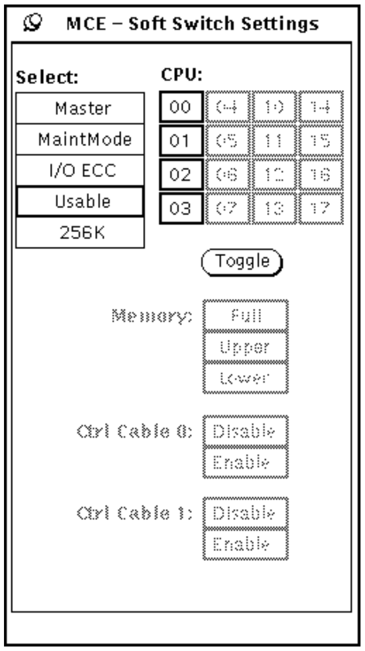


Figure 2-3. MCE Soft Switches Window

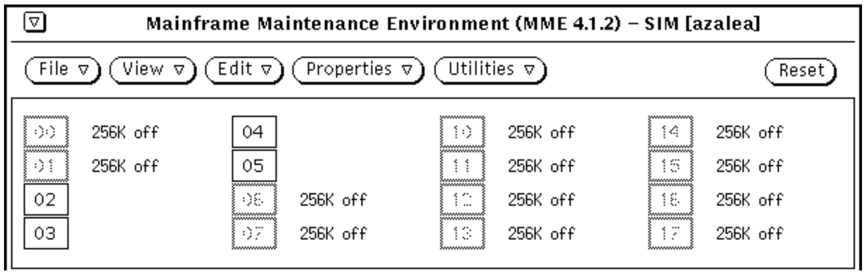
NOTE: Use the Toggle button to toggle the state of the CPUs.

When you switch between full- and half- (upper or lower) memory modes, the `Sections` and `Banks` parameters are automatically updated on the `Configuration` window.

In offline mode, the Usable setting specifies which CPUs are usable for testing. Click on the CPUs you want to use.

In concurrent mode, the 256K setting specifies which CPUs are available to run the offline diagnostics while the other CPUs run the OS. Click on the CPUs you want to use; remember, these CPUs cannot run the OS. Before you can load any control points in MME, you must specify the usable CPUs.

A CPU that is not set as usable in concurrent mode cannot be used for testing. MME displays `256K off` for any CPUs that are not usable in concurrent mode, as shown in Figure 2-4.


Mainframe Maintenance Environment (MME 4.1.2) – SIM [azalea]
File
View
Edit
Properties
Utilities
Reset

00	256K off	04		10	256K off	14	256K off
01	256K off	05		11	256K off	15	256K off
02		06	256K off	12	256K off	16	256K off
03		07	256K off	13	256K off	17	256K off

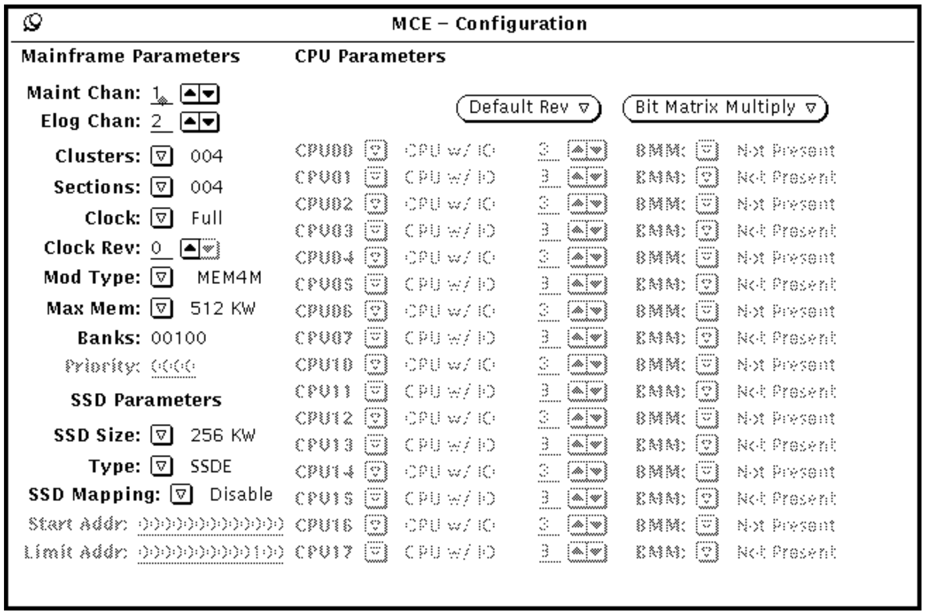
Figure 2-4. Usable and Unusable CPUs in Concurrent Mode

CAUTION

Do not configure a usable CPU in 256K mode. UNICOS will crash when you apply the configuration. Currently, UNICOS does not support concurrent maintenance.

Configuration Window

The `Configuration` window, shown in Figure 2-5, enables you to set mainframe, SSD, and CPU parameters. The options for each of the parameter groups are discussed in the following subsections.



```
MCE – Configuration
Mainframe Parameters          CPU Parameters

  Maint Chan: 1
   Elog Chan: 2                          Default Rev ▽      Bit Matrix Multiply ▽

    Clusters: ▽ 004           CPU00 ▽ CPU w/ IO   3   BMM: ▽ Not Present
    Sections: ▽ 004           CPU01 ▽ CPU w/ IO   3   BMM: ▽ Not Present
       Clock: ▽ Full          CPU02 ▽ CPU w/ IO   3   BMM: ▽ Not Present
   Clock Rev: 0               CPU03 ▽ CPU w/ IO   3   BMM: ▽ Not Present
    Mod Type: ▽  MEM4M        CPU04 ▽ CPU w/ IO   3   BMM: ▽ Not Present
     Max Mem: ▽ 512 KW        CPU05 ▽ CPU w/ IO   3   BMM: ▽ Not Present
       Banks: 00100           CPU06 ▽ CPU w/ IO   3   BMM: ▽ Not Present
    Priority: 0000            CPU07 ▽ CPU w/ IO   3   BMM: ▽ Not Present
                              CPU10 ▽ CPU w/ IO   3   BMM: ▽ Not Present
      SSD Parameters          CPU11 ▽ CPU w/ IO   3   BMM: ▽ Not Present
                              CPU12 ▽ CPU w/ IO   3   BMM: ▽ Not Present
    SSD Size: ▽ 256 KW        CPU13 ▽ CPU w/ IO   3   BMM: ▽ Not Present
        Type: ▽ SSDE          CPU14 ▽ CPU w/ IO   3   BMM: ▽ Not Present
 SSD Mapping: ▽ Disable       CPU15 ▽ CPU w/ IO   3   BMM: ▽ Not Present
 Start Addr: 0000000000000    CPU16 ▽ CPU w/ IO   3   BMM: ▽ Not Present
 Limit Addr: 0000000000100    CPU17 ▽ CPU w/ IO   3   BMM: ▽ Not Present
```

Figure 2-5. MCE Configuration Window

Mainframe Parameters

Set the `Maintenance Channel`, `Elog Channel`, `Clusters`, `Sections`, `Clock`, `Clock Rev`, `ModType`, `Max Mem`, and `Priority` parameters for your mainframe configuration. The `Banks` parameter is set automatically.

- `Elog Chan:` Error logger channel number
- `FEI Chan:` Front-end interface channel number
- `Clusters:` Number of clusters: `004`, `010`, `020`, `024`, or `040`; select your choice from the `Clusters:` ▽.
- `Sections:` Number of memory sections: `004` or `010`; select your choice from the `Sections` ▽.
- `Clock:` Type of clock module: `Full` or `Half`; select your choice from the `Clock:` ▽.

- `Clock Rev:` Revision of the clock module: enter the value in the `Clock Rev` field and press return (↵), or increase or decrease the value by clicking on the ▲ or ▼ buttons.

- `ModType:` Memory module type: `MEM1M` (for 1 Mword full memory modules), `MEM4M` (for 4 Mword full memory modules), `HDRM` (for DRAM memory modules), or `HM4M` (for 4 Mword half memory modules). Select your choice from the `Mod Type:` ▽.

- `Max Mem:` Mainframe maximum memory size: `Auto`, `128 KW`, `256 KW`, `512 KW`, `1 MW`, `2 MW`, `4 MW`, `8 MW`, `16 MW`, `32 MW`, `64 MW`, `128 MW`, `256 MW`, `512 MW`, `1 GW`, `2 GW`, or `4 GW`; select your choice from the `Max Mem:` ▽. `Auto` causes MCE to read the hardware to get the maximum memory.

- `Banks:` Number of memory banks: `02000`, `01000`, `00400`, `00200`, `00100`, or `00040`; this value is set automatically.

- `Priority:` Priority: enter the value in the `Priority` field and press return (↵)· This parameter specifies how often the priority counter is incremented. By default, the priority is set to 100 clock periods. This parameter is available for the revision 3 or greater clock modules and the revision 0 or greater half-clock (Hclock) modules.

SSD Parameters

Set the `SSD Size`, `Type`, and `SSD Mapping` parameters for your SSD configuration.

- `SSD Size:` Address limit for the SSD: `128 KW`, `256 KW`, `512 KW`, `1 MW`, `2 MW`, `4 MW`, `8 MW`, `16 MW`, `32 MW`, `64 MW`, `128 MW`, `256 MW`, `512 MW`, `1 GW`, `2 GW`, `4 GW`, `8 GW`, `16 GW`, or `32 GW`; select your choice from the `SSD Size:` ▽.

- `Type:` SSD type: `SSDE` (for an SSD-E), `32I` (for an SSD-E/32i), or `128I` (for an SSD-E/128i); select your choice from the `Type:` ▽.

- `SSD Mapping:` Map SSD memory selector: `Enable` (to force MME to use a specific portion of SSD memory for control points) or `Disable` (to allow MME to use any portion of SSD memory for control points); select your choice from the `SSD Mapping:` ▿.

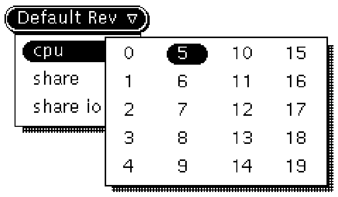
  If SSD mapping is enabled, set the starting and limit addresses to use in the `Start Addr` and `Limit Addr` fields. The limit address must be at least 100 words larger than the start address. These values are used to define the address parameters in the MME `SSD Resource Allocation` window; refer to "Allocating SSD Resources in Environment 1 or 2" in Section 4 for more information.

CPU Parameters

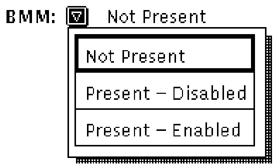
Set the module type, CPU revision, and bit matrix multiply (BMM) CPU parameters for each CPU in your mainframe.

- `CPU 00` – `CPU 17` Field that contains the module revision number. The field name changes for each module type (`CPU w/ IO`, `CPU w/o IO`, `SHARED IO`, `SHARED`, or `EMPTY`). Enter the revision number for each module in the corresponding field.

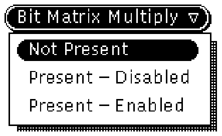
- ▿ Module type: `CPU w/ IO`, `CPU w/o IO`, `SHARED IO`, `SHARED`, or `EMPTY`. Select your choice from the ▿.



  You can use the Default Rev ▿ menu button to set all module configurations to the same revision number; for example, to set all CPU module configurations to revision, choose **`Default Rev --> cpu --> 5`**, as shown at the left.



- `BMM:` Configuration of the bit matrix multiply functional unit; for each CPU choose **`Not Present`**, or **`Present – Disabled`**, **`Present – Enabled`**, from `BMM:` ▿, as shown at the left.

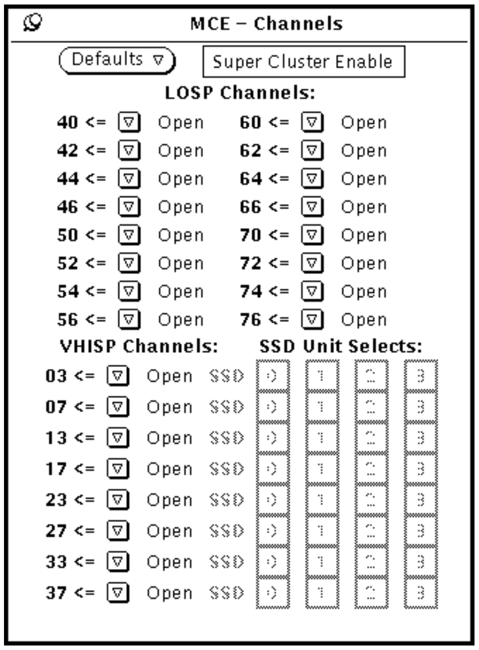


You can use the Bit Matrix Multiply ▿ menu button to set all BMM configurations to the same state; for example, to set all BMM configurations to `Not Present`, choose **`Bit Matrix Multiply --> Not Present`**, as shown at the left.

For CPUs lower than revision 5, the `BMM:` parameter automatically defaults to `Not Present` because these CPUs do not have a BMM functional unit.

Channels Window

The `Channels` window, shown in Figure 2-6, enables you to define the low-speed (LOSP) and very high-speed (VHISP) channel parameters. The settings for these parameters are described in the following subsections.



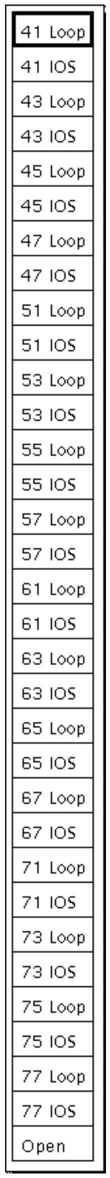
```
MCE – Channels

Defaults ▿    Super Cluster Enable
LOSP Channels:
40 <= ▿ Open    60 <= ▿ Open
42 <= ▿ Open    62 <= ▿ Open
44 <= ▿ Open    64 <= ▿ Open
46 <= ▿ Open    66 <= ▿ Open
50 <= ▿ Open    70 <= ▿ Open
52 <= ▿ Open    72 <= ▿ Open
54 <= ▿ Open    74 <= ▿ Open
56 <= ▿ Open    76 <= ▿ Open
VHISP Channels:    SSD Unit Selects:
03 <= ▿ Open  SSD  0  1  2  3
07 <= ▿ Open  SSD  0  1  2  3
13 <= ▿ Open  SSD  0  1  2  3
17 <= ▿ Open  SSD  0  1  2  3
23 <= ▿ Open  SSD  0  1  2  3
27 <= ▿ Open  SSD  0  1  2  3
33 <= ▿ Open  SSD  0  1  2  3
37 <= ▿ Open  SSD  0  1  2  3
```

The VHISP channels available in the `Channels` window for mainframes with serial number 4003 and below are 03, 07, 13, 17, 23, 27, 33, and 37. The VHISP channels available in the `MCE - Channels` window for mainframes with serial number 4004 and above are 11, 13, 15, 17, 21, 23, 25, and 27. These values depend on the value in the `Serial` field in the MCE base window.

Figure 2-6. MCE Channels Window

LOSP Channel Parameters

The even-numbered LOSP channels (40 through 76) can be set to the following values:

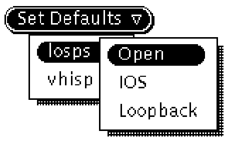


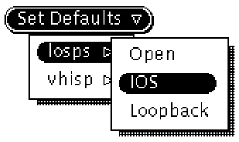
This menu gives three options for each even-numbered LOSP channel:

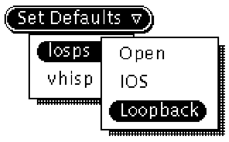
1. `Loop` indicates that the channel is looped back to another channel, specified by the number before `Loop`.

2. `IOS` indicates that the channel is connected to an IOS channel, specified by the number before `IOS`.

3. `Open` indicates that the channel is not connected to anything.







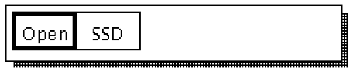
You can set all even-numbered LOSP channels to `Open` by choosing **`Set Defaults --> losps --> Open`**, as shown at the left.

You can set all even-numbered LOSP channels to be connected to the next odd-numbered IOS channel by choosing **`Set Defaults --> losps --> IOS`**, as shown at the left.

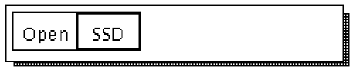
You can set all even-numbered LOSP channels to be looped back to the next odd-numbered channel by choosing **`Set Defaults --> losps --> Loopback`**, as shown at the left.

VHISP Channel Parameters

The VHISP channels (03, 07, 13, 17, 23, 27, 33, and 37 for mainframes with serial numbers 4003 and below or 11, 13, 15, 17, 21, 23, 25, and 27 for mainframes with serial numbers 4004 and above) can be set to `Open` or `SSD`:



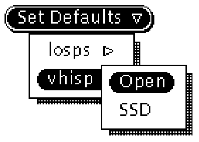
or

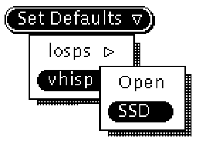


This gives you two options for each VHISP channel:

1. `Open` indicates that the channel is not connected to anything.

2. `SSD` indicates that the channel is connected to an SSD. If more than one SSD is connected to a VHISP channel, click on Super Cluster Enable to enable selection of more than one SSD. Then, click on the `SSD Unit Selects` setting(s) (0, 1, 2, and/or 3) to specify which SSD(s) to use. The default setting is 0, which selects the first SSD connected to the VHISP channel.





You can set all VHISP channels to `Open` by choosing **`Set Defaults --> VHISP --> Open`**, as shown at the left.

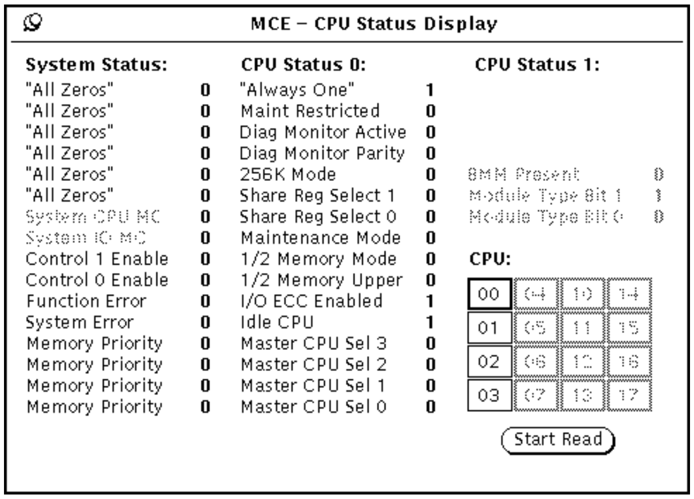
You can set all VHISP channels to `SSD` by choosing **`Set Defaults --> VHISP --> SSD`**, as shown at the left.

CPU Status Display Window

The `CPU Status Display` window enables you to view the status for any CPU. There are two versions of the `CPU Status Display` window that correspond to the two versions of the CPU status read parcel 0. These windows are shown in Figure 2-7 and Figure 2-8.

To display the system and CPU status information, click on the CPU for which you want to view the status and click on (Start Read). The system status bits, described in Table 2-4, and the CPU status bits, described in Table 2-5, are updated in this window until you click on (Stop Read). The system status bits are shown from parcel 1 of the system status read command, and the CPU status bits are shown from parcel 0 (status 0 or 1) of the maintenance channel read status.

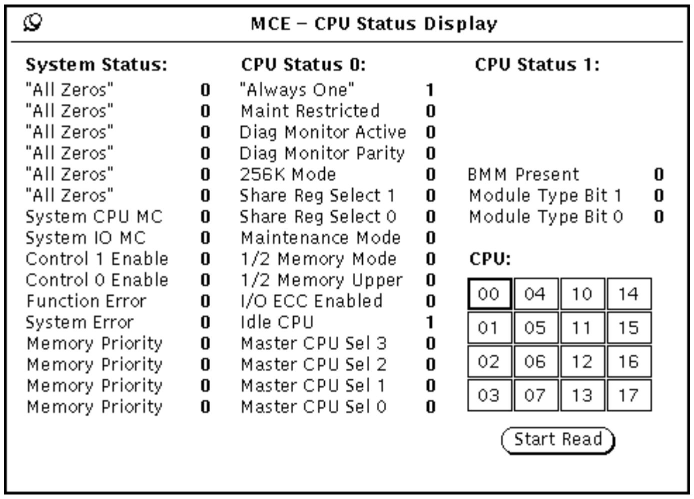
Figure 2-7 shows the `CPU Status Display` window displaying the system status read parcel (parcel 1) and the CPU status read parcel (status 0, parcel 0).



```
                         MCE – CPU Status Display

System Status:            CPU Status 0:               CPU Status 1:
"All Zeros"          0    "Always One"          1
"All Zeros"          0    Maint Restricted      0
"All Zeros"          0    Diag Monitor Active   0
"All Zeros"          0    Diag Monitor Parity   0
"All Zeros"          0    256K Mode             0    8MM Present          0
"All Zeros"          0    Share Reg Select 1    0    Module Type Bit 1    1
System CPU MC        0    Share Reg Select 0    0    Module Type Bit 0    0
System IO MC         0    Maintenance Mode      0
Control 1 Enable     0    1/2 Memory Mode       0    CPU:
Control 0 Enable     0    1/2 Memory Upper      0    00  04  10  14
Function Error       0    I/O ECC Enabled       1
System Error         0    Idle CPU              1    01  05  11  15
Memory Priority      0    Master CPU Sel 3      0
Memory Priority      0    Master CPU Sel 2      0    02  06  12  16
Memory Priority      0    Master CPU Sel 1      0
Memory Priority      0    Master CPU Sel 0      0    03  07  13  17

                                                        Start Read
```

Figure 2-7. MCE CPU Status Display for CPU Status 0

Figure 2-8 shows `CPU Status Display` window displaying the system status read parcel (parcel 1) and the CPU status read parcel (status 1, parcel 0). The status 1, parcel 0 format differs from the status 0, parcel 0 format in bits 2^{11}, 2^{10}, and 2^{9}.



```
                           MCE – CPU Status Display

System Status:             CPU Status 0:                CPU Status 1:
"All Zeros"           0    "Always One"            1
"All Zeros"           0    Maint Restricted        0
"All Zeros"           0    Diag Monitor Active     0
"All Zeros"           0    Diag Monitor Parity     0
"All Zeros"           0    256K Mode               0    BMM Present          0
"All Zeros"           0    Share Reg Select 1      0    Module Type Bit 1    0
System CPU MC         0    Share Reg Select 0      0    Module Type Bit 0    0
System IO MC          0    Maintenance Mode        0
Control 1 Enable      0    1/2 Memory Mode         0    CPU:
Control 0 Enable      0    1/2 Memory Upper        0    00  04  10  14
Function Error        0    I/O ECC Enabled         0
System Error          0    Idle CPU                1    01  05  11  15
Memory Priority       0    Master CPU Sel 3        0
Memory Priority       0    Master CPU Sel 2        0    02  06  12  16
Memory Priority       0    Master CPU Sel 1        0
Memory Priority       0    Master CPU Sel 0        0    03  07  13  17

                                                          Start Read
```

Figure 2-8. MCE CPU Status Display for CPU Status 1

Table 2-4 describes the system status bits.

Table 2-4. System Status Bits

Status Bit	Label	Description
2^{15}	`“All Zeros”`	This bit is always set to 0.
2^{14}	`“All Zeros”`	This bit is always set to 0.
2^{13}	`“All Zeros”`	This bit is always set to 0.
2^{12}	`“All Zeros”`	This bit is always set to 0.
2^{11}	`“All Zeros”`	This bit is always set to 0.
2^{10}	`“All Zeros”`	This bit is always set to 0.
2^{9}	`System CPU MC`	System CPU master clear; this status information is displayed only for revision 3 or higher clock modules and revision 0 or higher Hclock modules.
2^{8}	`System IO MC`	System I/O master clear; this status information is displayed only for revision 3 or higher clock modules and revision 0 or higher Hclock modules.
2^{7}	`Control 1 Enable`	Control cable 1 is enabled [soft switch (0 = no; 1 = yes)]

Table 2-4. System Status Bits (continued)

Status Bit	Label	Description
2^6	`Control 0 Enable`	Control cable 0 is enabled [soft switch (0 = no; 1 = yes)]
2^5	`Function Error`	Function error has occurred [(0 = no; 1 = yes) caused by a parity error in parcel 0 or parcel 1 of the maintenance channel header]
2^4	`System Error`	System error has occurred [(0 = no; 1 = yes) caused by a parcel parity error and detected by the DN option]
2^3	`Memory Priority`	Current memory priority (bit positions 2^3 through 2^0 indicate which CPU has memory priority; decode the bit combination to determine the octal CPU value.)
2^2	`Memory Priority`	Current memory priority
2^1	`Memory Priority`	Current memory priority
2^0	`Memory Priority`	Current memory priority

Table 2-5 describes the CPU status bits.

Table 2-5. CPU Status Bits

Status Bit	Label	Description
2^{15}	`"Always One"`	This bit is always set to 1.
2^{14}	`Maint Restricted`	CPU is in maintenance restricted mode (0 = no; 1 = yes)
2^{13}	`Diag Monitor Active`	CPU's diagnostic monitor is active (0 = no; 1 = yes)
2^{12}	`Diag Monitor Parity`	CPU's diagnostic monitor has a parity error (0 = no; 1 = yes)

2^{11}	`256K Mode`
	
	or
	
	`BMM Present`
		CPU is in 256-Kword memory mode (0 = no; 1 =yes)
		
		or
		
		CPU has bit matrix multiply functional unit (0 = no; 1 = yes)

† The labels of bits 2^{11} through 2^9 for CPU status 1 (`BMM Present`, `Module Type Bit 1`, and `Module Type Bit 0`) activate for the following module revisions and above: CPU revision 6, shared revision 1, and shared I/O revision 1.

Table 2-5. CPU Status Bits (continued)

Status Bit	Label	Description

2^{10}	`Share Reg Select 1`
	
	or
	
	`Module Type Bit 1`
		Shared register select 1 (0 = no; 1 = yes)
		
		or
		
		Module type – bits 2^{10} and 2^{9} form a mask that indicates which type of module is being used:
		
		Bit 2^{10} / Bit 2^{9} / Module Type
		0 / 0 / CPU
		0 / 1 / Share
		1 / 0 / Share I/O

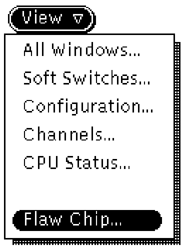
2^{9}	`Shared Reg Select 0`
	
	or
	
	`Module Type Bit 0`
		Shared register select 0 (0 = no; 1 = yes)
		
		or
		
		Module type

2^{8}	`Maintenance Mode`	CPU is in maintenance mode (0 = no; 1 = yes)
2^{7}	`1/2 Memory Mode`	CPU is in half-memory mode (0 = no; 1 = yes)
2^{6}	`1/2 Memory Upper`	CPU has upper-half memory selected (0 = no; 1 = yes)
2^{5}	`I/O ECC Enabled`	CPU has I/O ECC enabled (0 = no; 1 = yes)
2^{4}	`IDLE CPU`	CPU is idle (0 = no; 1 = yes)
2^{3}	`Master CPU Sel 3`	Master CPU select 3 (bit positions 2^{3} through 2^{0} form a mask that indicates which CPU is the master CPU; refer to Table 2-2 to decode the mask.)
2^{2}	`Master CPU Sel 2`	Master CPU select 2
2^{1}	`Master CPU Sel 1`	Master CPU select 1
2^{0}	`Master CPU Sel 0`	Master CPU select 0

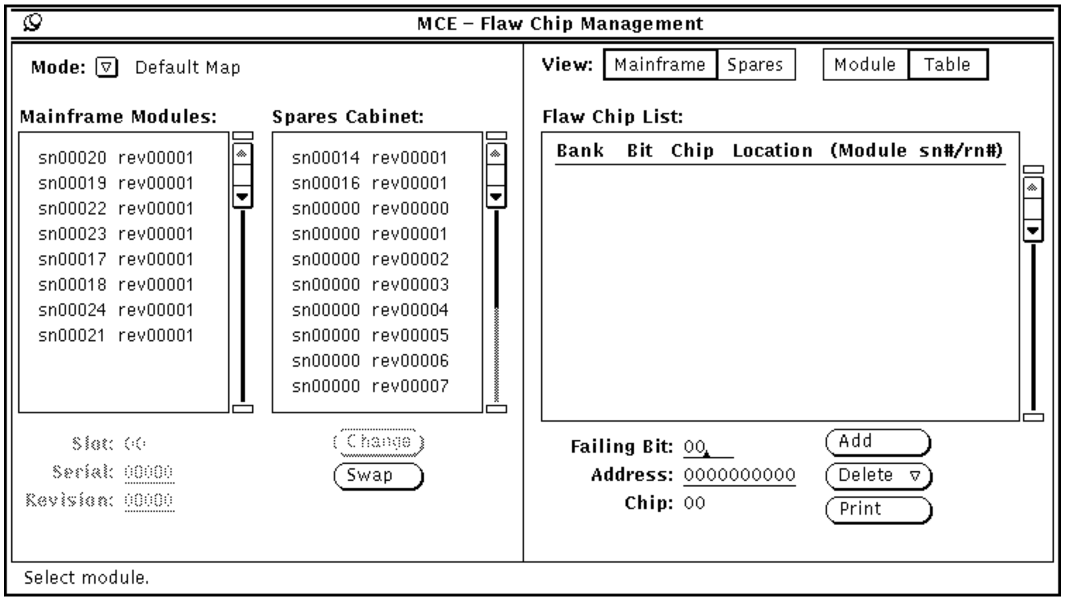
† The labels of bits 2^{11} through 2^{9} for CPU status 1 (`BMM Present`, `Module Type Bit 1`, and `Module Type Bit 0`) activate for the following module revisions and above: CPU revision 6, shared revision 1, and shared I/O revision 1.

Flaw Chip Management Window

MCE also enables you to create a table of flawed chips on the mainframe and spare memory modules if the current configuration contains modules that have spare chips (4 Mword full memory modules, 4 Mword half memory modules, and DRAM memory modules). When you save and apply the MCE Configuration table, the operating system (OS) uses the data to specify which spare chips are used at boot time. The flaw chip table is also used to identify the flawed chips when the modules are repaired.



To access the `MCE – Flaw Chip Management` window, choose **`View --> Flaw Chip`**, as shown at the left, in the MCE base window. If memory modules containing spare chips have been specified as the module type by the `ModType` parameter (`MEM4M`, `HDRM`, or `HM4M`) in the `MCE – Configuration` window, the window shown in Figure 2-9 appears, indicating flaw chip management is available. If memory modules without spare chips have been specified (`MEM1M`), the window shown in Figure 2-10 appears, indicating flaw chip management is not available.


MCE – Flaw Chip Management
View: Mainframe Spares Module Table
Flaw Chip List:
Bank Bit Chip Location (Module sn#/rn#)
Change
Swap
Add
Delete
Print

Mode: Default Map

Mainframe Modules:

```
sn00020 rev00001
sn00019 rev00001
sn00022 rev00001
sn00023 rev00001
sn00017 rev00001
sn00018 rev00001
sn00024 rev00001
sn00021 rev00001
```

Spares Cabinet:

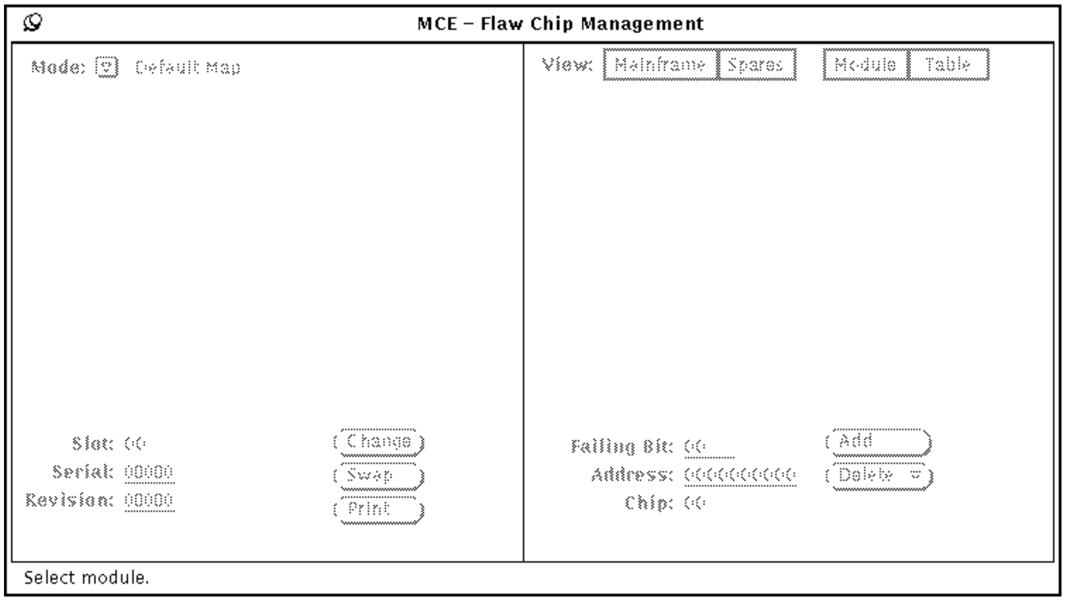
```
sn00014 rev00001
sn00016 rev00001
sn00000 rev00000
sn00000 rev00001
sn00000 rev00002
sn00000 rev00003
sn00000 rev00004
sn00000 rev00005
sn00000 rev00006
sn00000 rev00007
```

Slot: 00
Serial: 00000
Revision: 00000

Failing Bit: 00
Address: 0000000000
Chip: 00

Select module.

Figure 2-9. MCE Flaw Chip Management Window



Select module.

Figure 2-10. Spare Chips Not Present

The `MCE - Flaw Chip Management` window provides several options for viewing the flaw information you have entered. You can look at the information entered for just the mainframe modules (Mainframe) or just the spare modules (Spares). You can look at a table of the flaws for all mainframe or spare modules (Table) or look at the flaws for just one module (Module).

For more complete information about the spare chip feature, refer to *CRAY C90 Series Memory Chip Flawing* document and its companion videotape, which are both included in kit number HMK-113-0.

3 MME ENVIRONMENT 0

MME environment 0 tests the mainframe from the MWS-E through a sequence of maintenance channel functions to ensure environment 1 functionality. The areas tested are the maintenance channel, the diagnostic monitor, the mainframe memory, the CPU exchange mechanism, and the CPU instruction buffers. Three modes – automatic, manual, and compose – are available to test the basic functions of the mainframe. Automatic mode runs a predefined series of sequences against user-selected areas and CPUs. Manual mode runs user-selected sequences against a single user-selected area and CPUs. Compose mode runs a user-defined sequence.

This section provides information about MME environment 0. This section includes information about starting MME environment 0, using the MME environment 0 interface, performing tasks in MME environment 0, and testing in MME environment 0.

Starting MME Environment 0

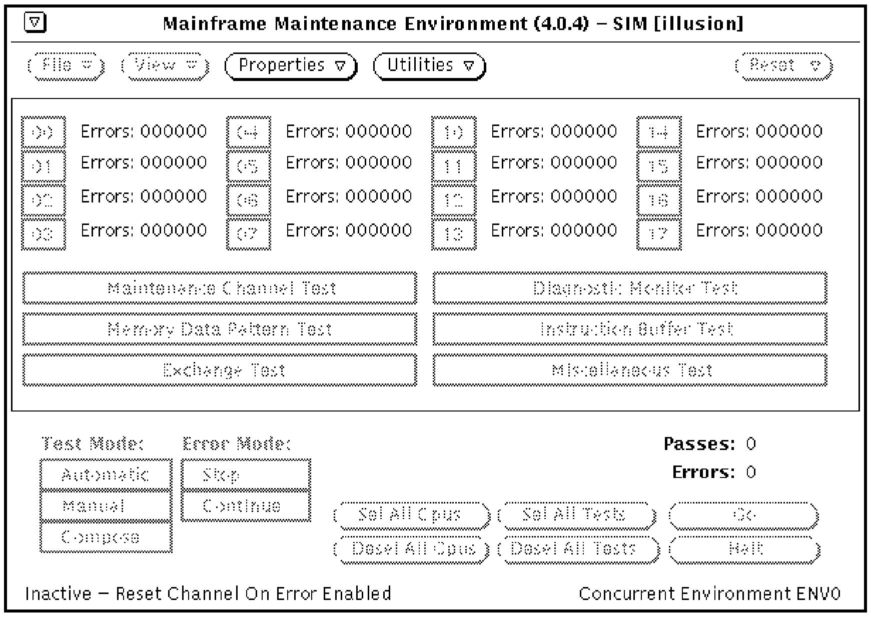
CAUTION

Do not start MME environment 0 when UNICOS is running. Several functions that MME environment 0 performs will crash UNICOS if it is running.

If no default configuration exists and offline mode is not specified, MME environment 0 defaults to concurrent mode, as shown in Figure 3-1, to prevent accidental crashing of the operating system. When environment 0 is in concurrent mode, two options are available:

- Use the (Properties ▽) button to change to environment 1 or 2 to continue testing in concurrent mode.
- Use the (Utilities ▽) button to access MCE to switch to offline mode to continue testing in environment 0; do not use offline mode when the OS is running in the mainframe.

For information about running MME from a service center through a hub refer to “Appendix: CRAY C90 Remote Support” in this manual.



Mainframe Maintenance Environment (4.0.4) – SIM [illusion]

00	Errors: 000000	04	Errors: 000000	10	Errors: 000000	14	Errors: 000000
01	Errors: 000000	05	Errors: 000000	11	Errors: 000000	15	Errors: 000000
02	Errors: 000000	06	Errors: 000000	12	Errors: 000000	16	Errors: 000000
03	Errors: 000000	07	Errors: 000000	13	Errors: 000000	17	Errors: 000000

Passes: 0
Errors: 0

Inactive – Reset Channel On Error Enabled

Concurrent Environment ENV0

Figure 3-1. Environment 0 in Concurrent Mode

From the OpenWindows Workspace Menu

To start MME environment 0 from the OpenWindows workspace menu, perform one of the following actions:

- Choose **`Maintenance Tools --> MME --> MME env 0 --> Copy#`** to start MME environment 0 with a specified copy number.

  The copy number option enables you to differentiate between multiple independent MME sessions that are supported from the same MWS-E. Copy numbers 0, 1, 2, and 3 are available from the workspace menu. The copy number does not affect performance; it serves as an identifier only.

- Choose **`Maintenance Tools --> MME Simulator --> MME env 0 --> Simulator`** to start MME environment 0 with the simulator.

- Choose **`Maintenance Tools --> MME Simulator --> MME env 0 --> Simulator with Debugger`** to start MME environment 0 with the simulator and debugger.

Starting from a UNIX Command Prompt

To start MME environment 0 from a UNIX prompt, type **`mme –0`** followed by any appropriate command line options:

```
mme –0  [–copy <num>]
        [–remote <host> | –client | –server]
        [–kill] [–io<num>]
        [–config <file>] [–l <file>]
        [–concurrent | –offline]
        [–chn<num> | -sim | -debug]
```

Table 3-1 describes the available command line options.

Table 3-1. Command Line Options

Option	Description
`–chn<num>`	Use front-end interface (FEI) channel specified by <*num*>, which can range from 0 to 7. The default channel number is 1.
`–client`	Start the client only
`–concurrent`	Use concurrent mode
`–config <file>`	Configure MCE with the configuration data stored in the file specified by <*file*>
`–copy <num>`	Connect to maintenance software assigned to the copy number specified by <*num*>
`–debug`	Use the simulator and bugger/debugger
`–io<num>`	Use the CPU specified by <*num*> as the I/O CPU
`–kill`	Kill all MME, MCE, and LME processes
`–l <file>`	Load a layout file
`–offline`	Use offline mode
`–remote <host>`	Start client only and connect to remote host
`–server`	Start server only
`–sim`	Use the simulator

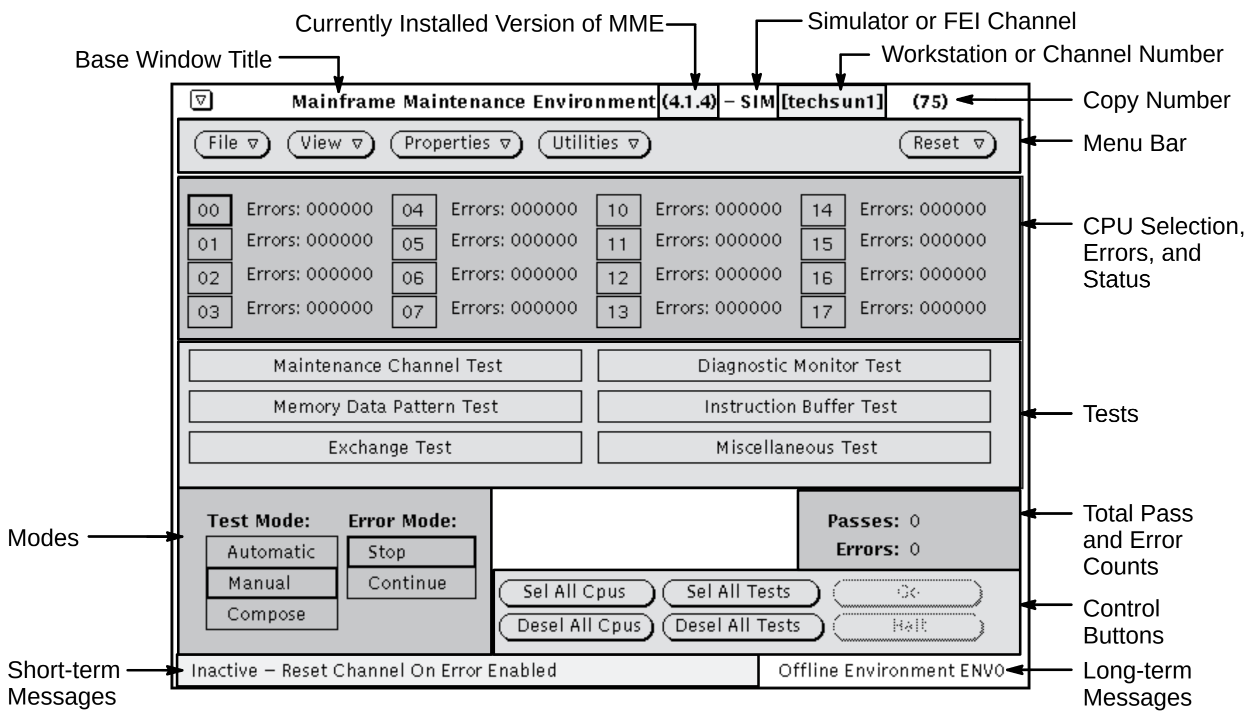
For example, enter **`mme –0 –io1 –chn2`**↵ to start MME environment 0 using CPU 1 as the I/O CPU and using FEI channel 2.

MME Environment 0 Interfaces

When MME environment 0 is started, the MME environment 0 base window and `MME Message Log` window are displayed. The MME environment 0 base window contains the interface for controlling all MME environment 0 activities. There are two interfaces associated with environment 0: the environment 0 interface in automatic or manual mode and the environment 0 interface in compose mode. This subsection describes the interfaces and the functions of their components.

MME Environment 0 Base Window in Automatic or Manual Mode

When MME environment 0 is in automatic or manual mode, the MME environment 0 base window contains the components and information shown in Figure 3-2.



00	Errors: 000000	04	Errors: 000000	10	Errors: 000000	14	Errors: 000000
01	Errors: 000000	05	Errors: 000000	11	Errors: 000000	15	Errors: 000000
02	Errors: 000000	06	Errors: 000000	12	Errors: 000000	16	Errors: 000000
03	Errors: 000000	07	Errors: 000000	13	Errors: 000000	17	Errors: 000000

Maintenance Channel Test
Diagnostic Monitor Test
Memory Data Pattern Test
Instruction Buffer Test
Exchange Test
Miscellaneous Test

Passes: 0
Errors: 0

Inactive – Reset Channel On Error Enabled
Offline Environment ENV0

Figure 3-2. Environment 0 Interface in Automatic or Manual Mode

The components of the MME environment 0 interface in automatic or manual mode are described as follows. These components are described as they appear in Figure 3-2 (clockwise, from the upper-left corner).

Base Window Title

The base window title displays the name of the program: `Mainframe Maintenance Environment`.

Currently Installed Version of MME

The currently installed version of MME is a number in parentheses that indicates which version of MME you are running.

Simulator or FEI Channel

The simulator or FEI channel indicator shows you are running MME with the simulator (indicated by `SIM`) or an FEI channel (indicated by `FEI CHN 0` for channel 0, `FEI CHN 1` for channel 1, or `FEI CHN 2` for channel 2).

Workstation or Channel Number

The workstation or channel number indicator lists the name of the workstation or the channel number that MME is running on.

Copy Number

The copy number identifies the copy of MME you are using. Because you may run more than one session of MME at a time from a single MWS-E, the copy number differentiates the sessions. To set the copy number, start MME with the `–copy` option. If you start MME with the default copy number of 0, the MME base window does not display a copy number. The copy number is used for identification only and will not affect performance. For more information about starting MME Environment 0 with the `–copy` option, refer to “Starting MME Environment 0” earlier in this section.

Menu Bar

The menu bar contains the menu buttons for controlling many functions of MME environment 0. There are five menu buttons: (File ▽), (View ▽), (Properties ▽), (Utilities ▽), and (Reset ▽). For descriptions of the tasks you can perform with these menu buttons, refer to “Performing Tasks in Environment 0” later in this section.

CPU Selection, Errors, and Status

The CPU selection, errors, and status information area is where you select which CPU(s) will run the test(s) and where MME displays the error counts for each CPU. You can click on the CPUs (00 through 17) to select them. The error count (`Errors`) for each CPU is displayed next to the CPU number. The error counts are in decimal.

Tests

The test area of the interface contains nonexclusive settings for selecting which tests to run. There are six environment 0 tests that can be run in automatic or manual mode:

Maintenance Channel Test	Diagnostic Monitor Test
Memory Data Pattern Test	Instruction Buffer Test
Exchange Test	Miscellaneous Test

Click on a test to select it. For more information about running these tests, refer to “Testing in Environment 0” later in this section.

Total Pass and Error Counts

The total pass count (`Passes`) indicates the number of passes a test has completed. The total error count (`Errors`) indicates the total number of errors found during the current test(s) executions. The pass and error counts are in decimal.

Control Buttons

The six buttons for controlling CPU selection, test selection, and test execution are: Sel All Cpus, Desel All Cpus, Sel All Tests, Desel All Tests, Go, and Halt. Use these buttons to perform the following functions:

Button	Function
Sel All Cpus	Select all CPUs for testing
Desel All Cpus	Deselect all CPUs
Sel All Tests	Select all tests (automatic mode only)
Desel All Tests	Deselect all tests (automatic mode only)

Button	Function
Go	Start testing
Halt	Stop testing

Long-term Messages

The long-term message area displays which environment you are working in for the current base window.

Short-term Messages

The short-term message area on the interface displays messages about the current state of the MME program. The following messages are displayed:

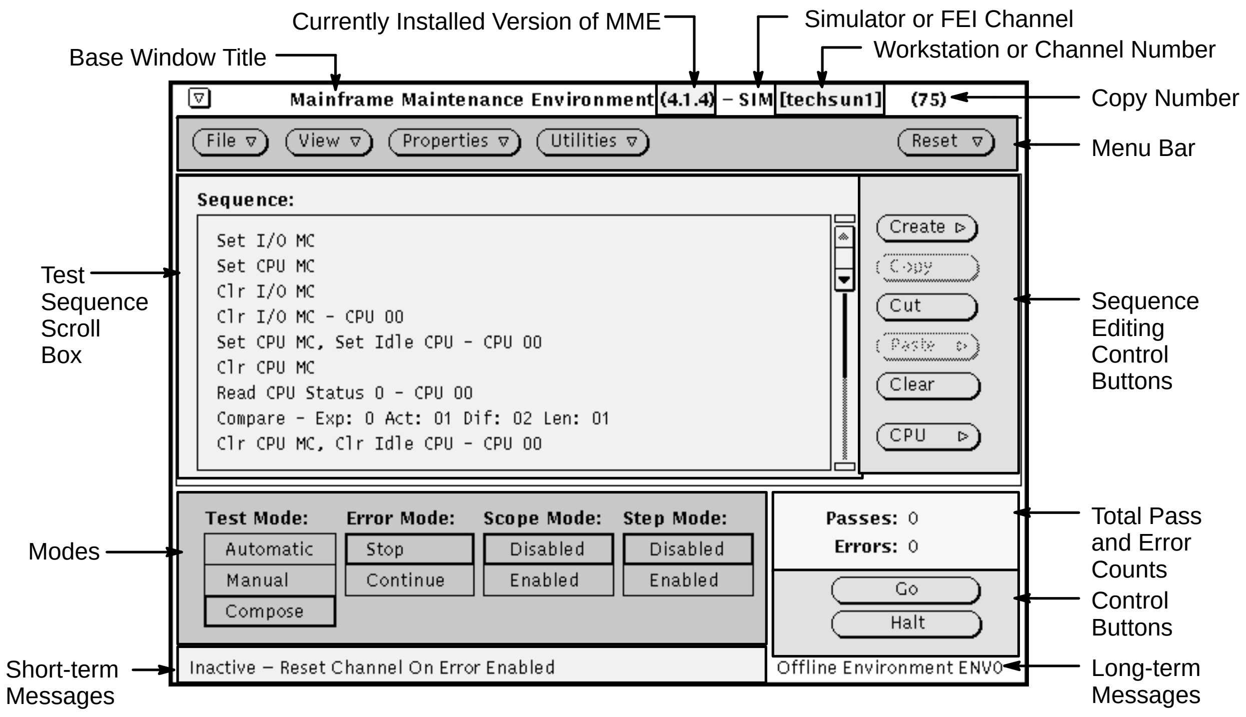
Message	Description
`Active`	Test(s) are running
`Inactive`	No test(s) are running
`Reset Channel On Error Enabled`	Channel resets on error (this is set in the `MME Resource Allocation` window)
`Disabled Reset Channel on Error`	Channel does not reset on error (this is set in the `MME Resource Allocation` window)

Modes

The modes area contains the exclusive settings for the `Test Mode` and `Error Mode`. You can set the `Test Mode` to Automatic, Manual, or Compose (refer to “Testing in Environment 0” later in this section for more information). You can set the `Error Mode` to Stop (stop on error) or Continue (continue on error). Click on the modes of your choice.

MME Environment 0 Base Window in Compose Mode

When MME environment 0 is in compose mode, the MME environment 0 base window contains the components and information shown in Figure 3-3.



```
Set I/O MC
Set CPU MC
Clr I/O MC
Clr I/O MC - CPU 00
Set CPU MC, Set Idle CPU - CPU 00
Clr CPU MC
Read CPU Status 0 - CPU 00
Compare - Exp: 0 Act: 01 Dif: 02 Len: 01
Clr CPU MC, Clr Idle CPU - CPU 00
```

```
Inactive – Reset Channel On Error Enabled
Offline Environment ENV0
```

Figure 3-3. Environment 0 Interface in Compose Test Mode

The components of the MME environment 0 base window in compose mode are described below. These components are described in the same order as they appear in Figure 3-3 (clockwise, from the upper-left corner).

Base Window Title

The base window title displays the name of the program: `Mainframe Maintenance Environment`.

Currently Installed Version of MME

The currently installed version of MME is a number in parentheses that indicates which version of MME you are running.

Simulator or FEI Channel

The simulator or FEI channel indicator shows that you are running MME with the simulator (indicated by `SIM`) or an FEI channel (indicated by `FEI CHN 0` for channel 0, `FEI CHN 1` for channel 1, or `FEI CHN 2` for channel 2).

Workstation or Channel Number

The workstation or channel number indicator indicates the name of the workstation or the channel number that MME is running on.

Copy Number

The copy number identifies the copy of MME you are using. Because you may run more than one session of MME at a time from a single MWS-E, the copy number differentiates the sessions. To set the copy number, start MME with the `–copy` option. If you start MME with the default copy number of 0, the MME base window does not display a copy number. The copy number is used for identification only and will not affect performance. For more information about starting MME Environment 0 with the `–copy` option, refer to "Starting MME Environment 1 or Environment 2" earlier in this section.

Menu Bar

The menu bar contains the menu buttons for controlling many functions of MME environment 0. There are five menu buttons: File ▽, View ▽, Properties ▽, Utilities ▽, and Reset ▽. For descriptions of the tasks you can perform with the commands contained in these menu buttons, refer to following subsection "Performing Tasks in Environment 0."

Sequence Editing Control Buttons

The sequence editing control buttons allow you to manipulate the placement of functions within a test sequence. These buttons are: Create ▷, Copy, Cut, Paste ▷, Clear, and CPU ▷. Use these buttons to perform the following functions:

Button	Function
Create ▷	Create a function entry in the composed sequence
Copy	Copy a sequence entry (this button is not implemented)

Button	Function
Cut	Cut a sequence from the composed sequence scroll box
Paste	Paste a sequence entry (this button is not implemented)
Clear	Clear all sequences from the test sequence scroll box
CPU	Set the CPU used by compose sequences

Total Pass and Error Counts

The total pass count (`Passes`) indicates the number of passes a test has completed. The total error count (`Errors`) indicates the total number of errors found during the current test(s) executions. The pass and error counts are in decimal.

Control Buttons

The control buttons in compose mode are Go and Halt. Use these buttons to perform the following functions.

Button	Function
Go	Start testing
Halt	Stop testing

Long-term Messages

The long-term message area displays which environment you are working in for the current base window.

Short-term Messages

The short-term message area on the interface displays messages about the current state of the MME program. The following messages are displayed:

Message	Description
`Active`	Test(s) are running
`Inactive`	No test(s) are running

Message	Description
`Reset Channel On Error Enabled`	Channel resets on error (this is set in the `MME Resource Allocation` window)

Modes

The modes area contains the exclusive settings for the `Test Mode`, `Error Mode`, `Scope Mode`, and `Step Mode`. You can set the `Test Mode` to Automatic, Manual, or Compose (refer to “Testing in Environment 0” later in this section for more information). You can set the `Error Mode` to Stop (stop on error) or Continue (continue on error). You can set `Scope Mode` to Enabled (enable scope mode) or Disabled (disable scope mode). You can set Step Mode to Enabled (enable step mode) or Disabled (disable step mode). Click on the modes of your choice.

Test Sequence Scroll Box

The test sequence scroll box contains the sequences you have chosen to run. These sequences will run in the order in which they are displayed in the scroll box (top to bottom) when you click on Go.

Performing Tasks in Environment 0

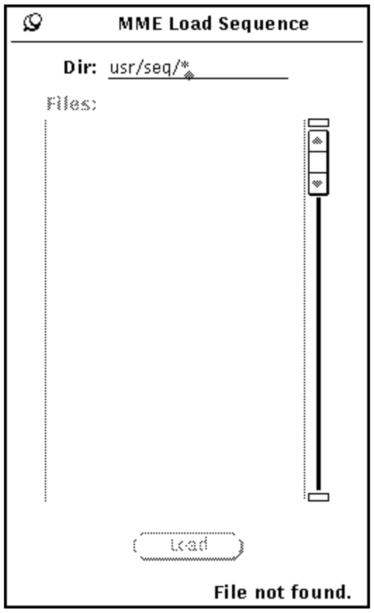
This subsection provides procedures that describe the tasks you can perform with the environment 0 menu buttons.

Loading a Sequence in Environment 0

If you have created or customized a sequence in compose mode of environment 0 and then saved the sequence (refer to "Saving a Sequence in Environment 0"), you may want to use the sequence again. To do so, perform the following procedure:

NOTE: This procedure must be performed in compose mode.

1. Choose **`File --> Load --> Sequence`**, as shown at the left. MME displays the `MME Load Sequence` window:



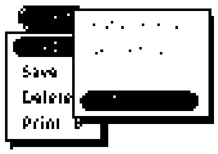
2. Change the directory, if necessary, by triple clicking on the `Dir` field, typing the name of the directory you want to use, and pressing the return (↵) key.

3. Click on the sequence that you want in the `Files` list.

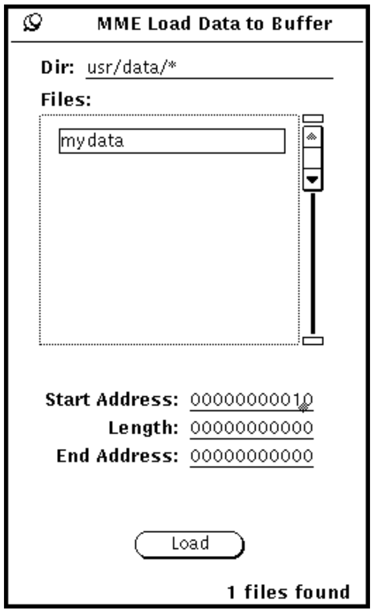
4. Click on Load ; MME loads the sequence.

Loading Data in Environment 0

To load data you have previously saved (refer to "Saving Data in Environment 0"), perform the following procedure:



1. Choose **`File --> Load --> Data`**, as shown at the left. MME displays the `MME Load Data` window:



Dir: usr/data/*
Files:
mydata
Start Address: 00000000010
Length: 00000000000
End Address: 00000000000
1 files found

2. Change the directory, if necessary, by triple clicking on the `Dir` field, typing the name of the directory you want to use, and pressing the return (↵) key.

3. Click on the file you want.

4. Change the values of the parameters to specify the addresses you want to use (after you perform any two of the three following steps, MME updates the third field).

   - Double click on the `Start Address` field. Type the starting address you want in octal.
   - Double click on the `Length` field. Type the length in octal.
   - Double click on the `End Address` field. Type the ending address in octal.

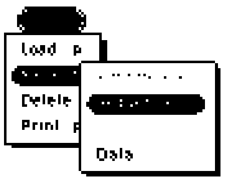
5. Click on Load; MME loads the data into memory.

Saving a Sequence in Environment 0

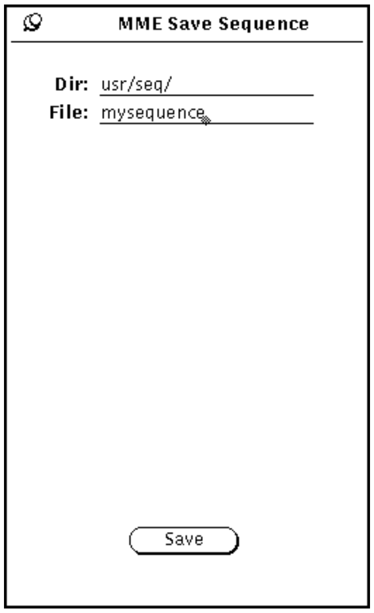
If you have created or customized a test sequence using compose mode in environment 0, you may want to save the sequence so you do not need to create or customize the same sequence again (refer to “Loading a Sequence in Environment 0”).

To save a sequence, perform the following procedure:

NOTE: This procedure must be performed in compose mode.



1. Choose **`File --> Save --> Sequence`**, as shown at the left. MME displays the `MME Save Sequence` window:



Dir: usr/seq/
File: mysequence

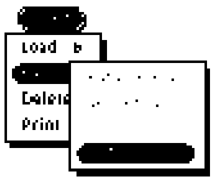
2. Change the directory, if necessary, by triple clicking on the `Dir` field, typing the name of the directory you want to use, and pressing the return (↵) key.

3. Double click on the `Sequence File` field. Type the name of the file in which you want to save the sequence.

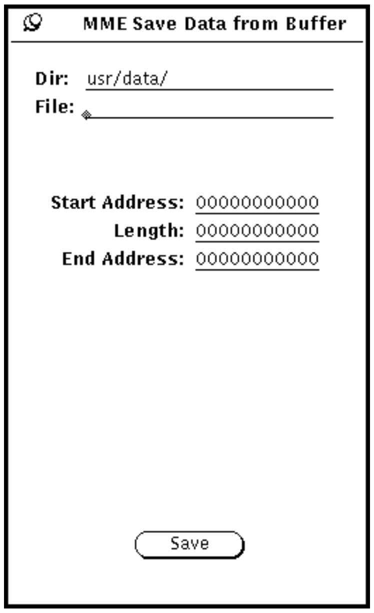
4. Click on (Save); MME saves the sequence.

Saving Data in Environment 0

To save data you have loaded for later use (refer to “Loading Data in Environment 0”), perform the following procedure:



1. Choose **`File --> Save --> Data`**, as shown at the left. MME displays the `MME Save Data from Buffer` window:



MME Save Data from Buffer

Dir: usr/data/

File:

Start Address: 00000000000

Length: 00000000000

End Address: 00000000000

2. Change the directory, if necessary, by triple clicking on the `Dir` field, typing the name of the directory you want to use, and pressing the return (↵) key.

3. Double click on the `File` field. Type the name of the file you want to use.

4. Change the values of the parameters to specify the addresses you want to use (after you perform any two of the three following steps, MME updates the third field).

   - Double click on the `Start Address` field. Type the starting address in octal.

   - Double click on the `Length` field. Type the data length in octal.

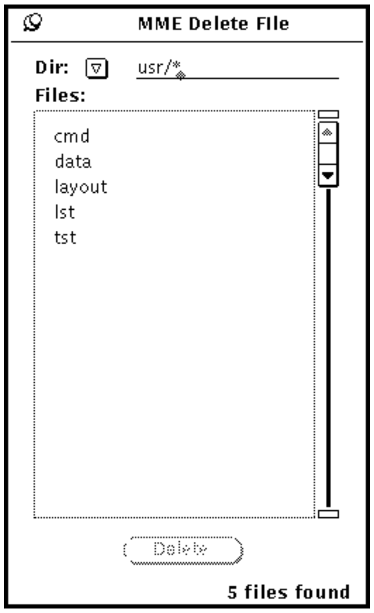
   - Double click on the `End Address` field. Type the ending address in octal.

5. Click on Save; MME saves the data.

Deleting a File in Environment 0

To delete a file that you no longer need, perform the following procedure:

1. Choose **`File --> Delete`**, as shown at the left. MME displays the `MME Delete File` window:



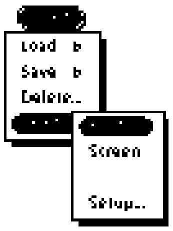
2. Change the directory, if necessary, by selecting a directory from the `Dir:` ▽ or by triple clicking on the `Dir` field, typing the name of the directory you want to use, and pressing the return (↵) key.

3. Click on the file you want to delete.

4. Click on (Delete); MME deletes the file.

Printing the Root Window in Environment 0

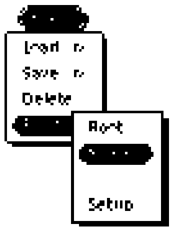
Before performing this procedure, you must first set up printing. Refer to “Setting Up or Changing Where MME Data Is Printed in Environment 0.”



To print an image of everything contained in the root window, choose **`File --> Print --> Root`**, as shown at the left. Use this command to print the MME base window.

Printing a Screen or Panel in Environment 0

Before performing this procedure, you must first set up printing. Refer to “Setting Up or Changing Where MME Data is Printed in Environment 0.”



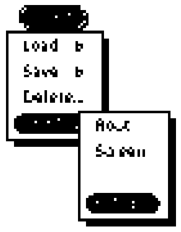
To print a window or an icon, choose **`File --> Print --> Screen`**, as shown at the left. When you choose this command, the cursor becomes a plus symbol. Move the cursor to the window or icon you want to print an image of and click on a mouse button.

NOTE: You cannot print the MME base window with this command. To print the MME base window, you must choose **`File --> Print --> Root`**.

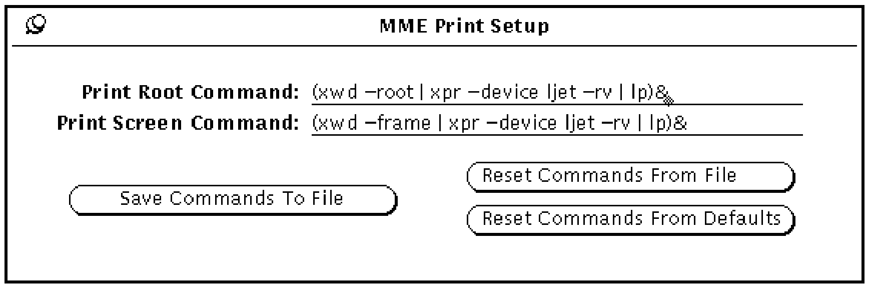
Setting Up or Changing Where MME Data is Printed in Environment 0

Before you print a root or screen, you must set up the printer options by entering the appropriate UNIX commands in the `Print Root Command` and `Print Screen Command` fields of the `MME Print Setup` window. MME uses the specified commands for the print functions to ensure that output goes to the proper printer.

NOTE: This process uses the `xwd`, `xpr`, and `lp` UNIX commands. The `xwd` command dumps an image of an X window. The `xpr` command prints an image of the X window dump. The `lp` command sends a request to the printer. For detailed information about these commands, refer to the UNIX man pages (enter **`man xwd`**, **`man xpr`**, or **`man lp`** at a UNIX prompt).



To set up where you want data printed, choose **`File --> Print --> Setup`**, as shown at the left. MME displays the `MME Print Setup` window:



MME Print Setup

Print Root Command: (xwd –root | xpr –device ljet –rv | lp)&

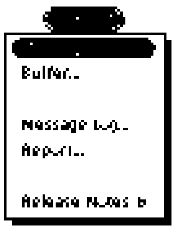
Print Screen Command: (xwd –frame | xpr –device ljet –rv | lp)&

The buttons in this window are described in the following list:

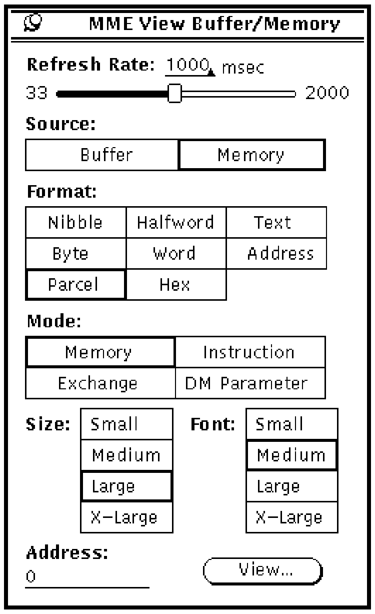
- Click on (Save Commands To File) to save your current printer setup commands for later use.
- Click on (Reset Commands From File) to load the printer setup commands you saved previously.
- Click on (Reset Commands From Defaults) to load the default printer setup commands provided with the MME program.

Viewing Memory in Environment 0

You can view the memory to track the status of the diagnostic test that you are running. To view mainframe memory, perform the following procedure:



1. Choose **`View --> Memory`**, as shown at the left. MME displays the `MME View Memory Setup` window:


MME View Buffer/Memory

Refresh Rate: 1000 msec
33 — 2000

Source: Buffer [✓ Memory]

Format: Nibble Halfword Text Byte Word Address [✓ Parcel] Hex

Mode: [✓ Memory] Instruction Exchange DM Parameter

Size: Small Medium [✓ Large] X-Large

Font: Small [✓ Medium] Large X-Large

Address: 0

View...

NOTE: You can set the interval at which memory windows are updated by moving the `Refresh Rate` slider or by double clicking on the `Refresh Rate` field, typing a new value, and pressing the return (↵) key. Setting this value too low can monopolize the workstation CPU.

2. Click on a `Format` [Nibble, Halfword, Text, Byte, Word, Address, Parcel, or Hex (hexadecimal)] to specify the format in which you want the memory displayed.

3. Click on a `Mode` (Memory, Instruction, Exchange, or DM Parameters) to specify the way you want the data represented.

   Memory mode displays normal memory, exchange mode displays exchange information, instruction mode decodes the memory into instructions, and DM parameter mode translates the memory into DM parameter information.

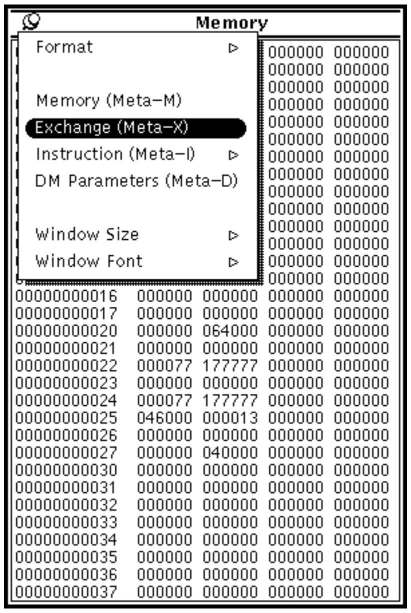
4. Click on a `Size` [Small, Medium, Large, or X-Large (extra large)] to specify the size of the display window. This affects only the memory and instruction windows.

5. Click on a `Font` [Small, Medium, Large, or X-Large (extra large)] to specify the font size to display in the window.

6. Change the starting address, if necessary, by double clicking on the `Address` field and typing a new value.

7. Click on View... . MME displays the specified memory window.

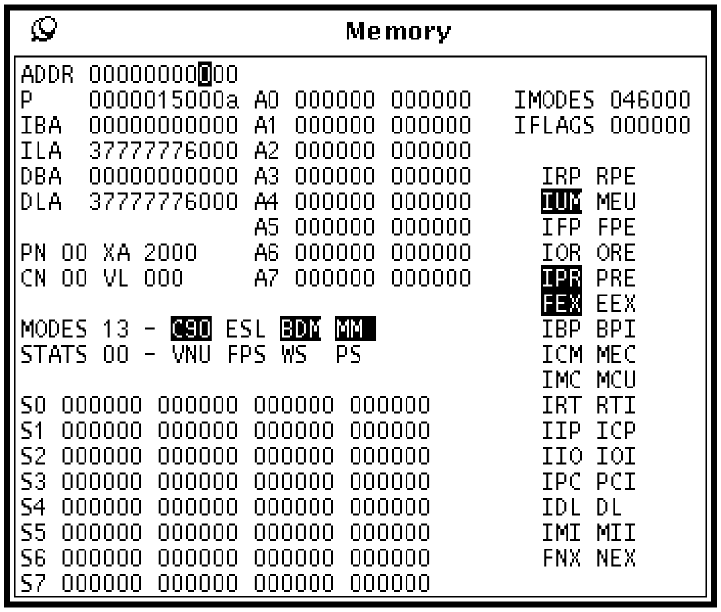
If you want to change the `Format`, `Memory`, `Exchange`, `Instruction`, `DM` (diagnostic monitor) `Parameters`, `Window Size`, or `Window Font` from the `Memory` window, press the MENU mouse button and move the mouse pointer to choose the menu item you want:



```
                                      000000  000000
                                      000000  000000
                                      000000  000000
                                      000000  000000
                                      000000  000000
                                      000000  000000
                                      000000  000000
                                      000000  000000
                                      000000  000000
                                      000000  000000
                                      000000  000000
                                      000000  000000
                                      000000  000000
                                      000000  000000
00000000016   000000  000000  000000  000000
00000000017   000000  000000  000000  000000
00000000020   000000  064000  000000  000000
00000000021   000000  000000  000000  000000
00000000022   000077  177777  000000  000000
00000000023   000000  000000  000000  000000
00000000024   000077  177777  000000  000000
00000000025   046000  000013  000000  000000
00000000026   000000  000000  000000  000000
00000000027   000000  040000  000000  000000
00000000030   000000  000000  000000  000000
00000000031   000000  000000  000000  000000
00000000032   000000  000000  000000  000000
00000000033   000000  000000  000000  000000
00000000034   000000  000000  000000  000000
00000000035   000000  000000  000000  000000
00000000036   000000  000000  000000  000000
00000000037   000000  000000  000000  000000
```

In this example, instead of using the MENU mouse button, you can use the diamond-shaped (◇) meta key to the left of the space bar and one of the following alphabetical keys: `a` for address format, `n` for nibble format, `b` for byte format, `p` for parcel format, `h` for halfword format, `w` for word format, `e` for hexadecimal format, `t` for text format, `i` for instruction mode, `x` for exchange mode, `d` for diagnostic monitor mode, and `m` for memory mode.

For example, the following `Memory` window appears if you choose the `Exchange` format:



Memory

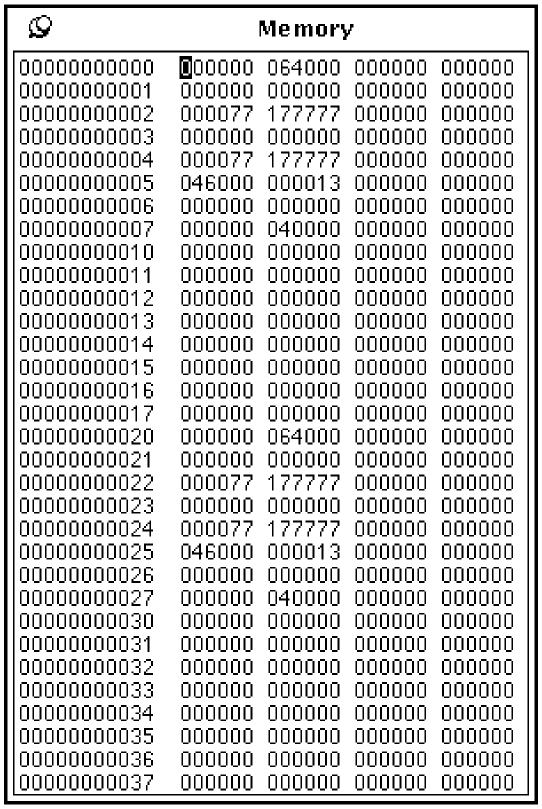
```
ADDR 00000000000
P    0000015000a A0 000000 000000   IMODES 046000
IBA  00000000000 A1 000000 000000   IFLAGS 000000
ILA  37777776000 A2 000000 000000
DBA  00000000000 A3 000000 000000     IRP RPE
DLA  37777776000 A4 000000 000000     IUM MEU
                 A5 000000 000000     IFP FPE
PN 00 XA 2000    A6 000000 000000     IOR ORE
CN 00 VL 000     A7 000000 000000     IPR PRE
                                      FEX EEX
MODES 13 - C90 ESL BDM MM             IBP BPI
STATS 00 - VNU FPS WS  PS             ICM MEC
                                      IMC MCU
S0 000000 000000 000000 000000        IRT RTI
S1 000000 000000 000000 000000        IIP ICP
S2 000000 000000 000000 000000        IIO IOI
S3 000000 000000 000000 000000        IPC PCI
S4 000000 000000 000000 000000        IDL DL
S5 000000 000000 000000 000000        IMI MII
S6 000000 000000 000000 000000        FNX NEX
S7 000000 000000 000000 000000
```

You can change the window `Format`, `Memory`, `Instruction`, `DM (Diagnostic Monitor) parameters`, `Window Size`, or `Window Font` in a manner similar to changing a window format.

Changing Memory in Environment 0

After you have displayed memory (refer to “Viewing Memory in Environment 0”), you may want to change the value at a memory address for a test you want to run in environment 0. To change memory from the `Memory` window, perform the following procedure.

1. Refer again to “Viewing Memory in Environment 0” to view the following `Memory` window (use all the default values on the `MME View Buffer/Memory` window):



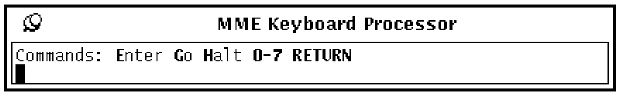
```
00000000000  000000 064000 000000 000000
00000000001  000000 000000 000000 000000
00000000002  000077 177777 000000 000000
00000000003  000000 000000 000000 000000
00000000004  000077 177777 000000 000000
00000000005  046000 000013 000000 000000
00000000006  000000 000000 000000 000000
00000000007  000000 040000 000000 000000
00000000010  000000 000000 000000 000000
00000000011  000000 000000 000000 000000
00000000012  000000 000000 000000 000000
00000000013  000000 000000 000000 000000
00000000014  000000 000000 000000 000000
00000000015  000000 000000 000000 000000
00000000016  000000 000000 000000 000000
00000000017  000000 000000 000000 000000
00000000020  000000 064000 000000 000000
00000000021  000000 000000 000000 000000
00000000022  000077 177777 000000 000000
00000000023  000000 000000 000000 000000
00000000024  000077 177777 000000 000000
00000000025  046000 000013 000000 000000
00000000026  000000 000000 000000 000000
00000000027  000000 040000 000000 000000
00000000030  000000 000000 000000 000000
00000000031  000000 000000 000000 000000
00000000032  000000 000000 000000 000000
00000000033  000000 000000 000000 000000
00000000034  000000 000000 000000 000000
00000000035  000000 000000 000000 000000
00000000036  000000 000000 000000 000000
00000000037  000000 000000 000000 000000
```

Notice that the word at memory address 0, parcel a, appears highlighted in the `Memory` window.

NOTE: To change the format, mode, window size, font size, or address, you can use the MENU mouse button or you can use the diamond-shaped (◇) meta key to the left of the space bar and one of the following alphabetical keys: `a` for address format, `n` for nibble format, `b` for byte format, `p` for parcel format, `h` for halfword format, `w` for word format, `e` for hexadecimal format, `t` for text format, `i` for instruction mode, `x` for exchange mode, `d` for diagnostic monitor mode, and `m` for memory mode. For an example of how to change the format, mode, window size, font size, or address, refer to "Viewing Buffer Data in Environment 0" or "Viewing Memory in Environment 1 or 2."

2. Perform one of the following actions:

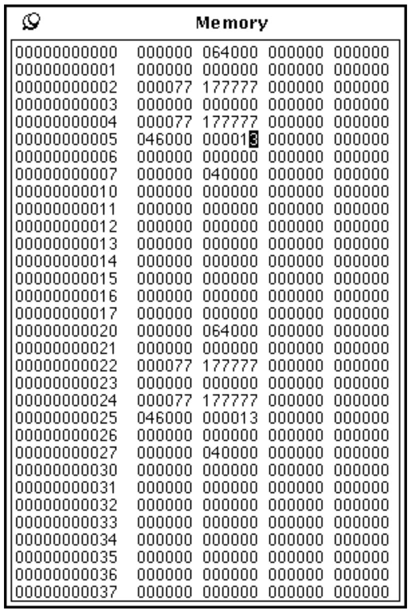
- Press and release the space bar; the `MME Keyboard Processor` window appears:



```
MME Keyboard Processor
Commands: Enter Go Halt 0-7 RETURN
```

Using this window, you can change (convert) the format; enter data in memory; start or halt tests; or view memory at a specific address. When you type the letter or number shown in bold, the command runs. The next window prompts you for any additional commands or information you must enter.

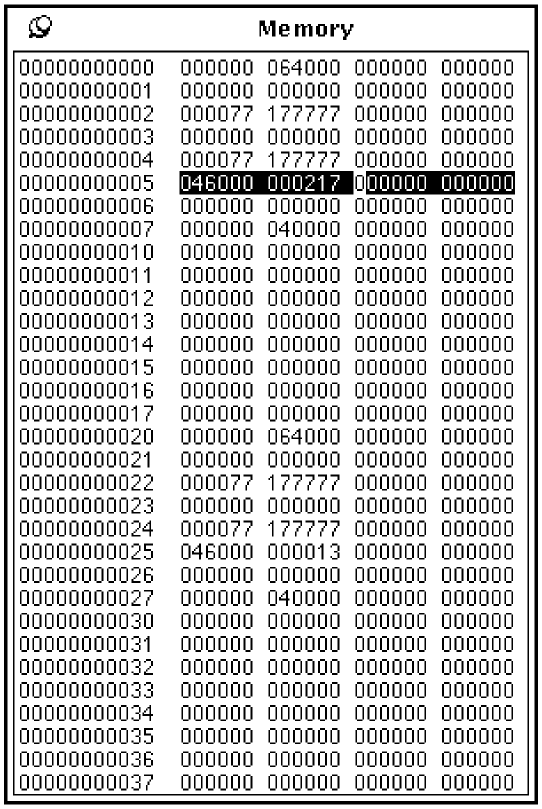
- Move the mouse pointer to the word in memory you want to change using the PgUp (page up) or PgDn (page down) keys and the cursor movement keys. Click on the word that you want so that the word appears in reverse video. In this example, the user clicked on parcel 000005b:



```
00000000000   000000 064000 000000 000000
00000000001   000000 000000 000000 000000
00000000002   000077 177777 000000 000000
00000000003   000000 000000 000000 000000
00000000004   000077 177777 000000 000000
00000000005   046000 000013 000000 000000
00000000006   000000 000000 000000 000000
00000000007   000000 040000 000000 000000
00000000010   000000 000000 000000 000000
00000000011   000000 000000 000000 000000
00000000012   000000 000000 000000 000000
00000000013   000000 000000 000000 000000
00000000014   000000 000000 000000 000000
00000000015   000000 000000 000000 000000
00000000016   000000 000000 000000 000000
00000000017   000000 000000 000000 000000
00000000020   000000 064000 000000 000000
00000000021   000000 000000 000000 000000
00000000022   000077 177777 000000 000000
00000000023   000000 000000 000000 000000
00000000024   000077 177777 000000 000000
00000000025   046000 000013 000000 000000
00000000026   000000 000000 000000 000000
00000000027   000000 040000 000000 000000
00000000030   000000 000000 000000 000000
00000000031   000000 000000 000000 000000
00000000032   000000 000000 000000 000000
00000000033   000000 000000 000000 000000
00000000034   000000 000000 000000 000000
00000000035   000000 000000 000000 000000
00000000036   000000 000000 000000 000000
00000000037   000000 000000 000000 000000
```

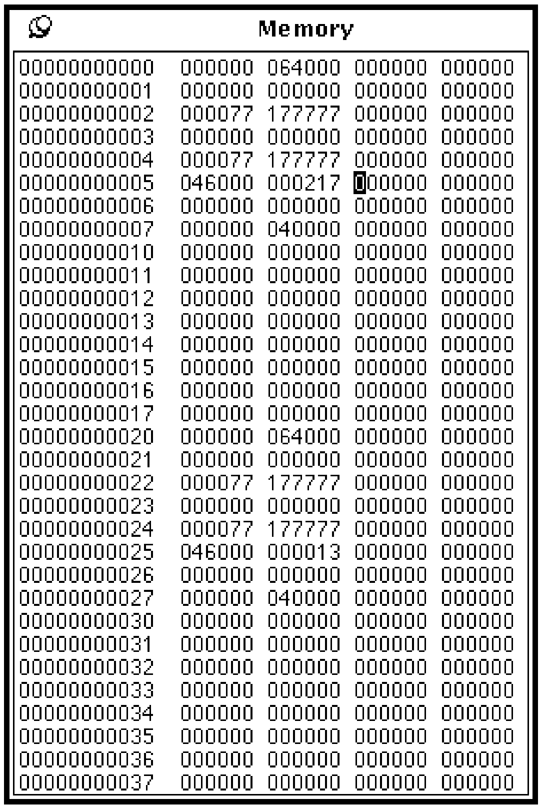
3. Press and release the Esc key. Type the new word value. The entire word is highlighted, which enables you to change the entire word.

   When you type a new word value and press the return (↵) key, it automatically changes the value in mainframe memory. In the following example, the `Memory` window shows the word value changed from 000013 to 000217 at memory address 000005, parcel b:



```
Memory
00000000000   000000 064000 000000 000000
00000000001   000000 000000 000000 000000
00000000002   000077 177777 000000 000000
00000000003   000000 000000 000000 000000
00000000004   000077 177777 000000 000000
00000000005   046000 000217 000000 000000
00000000006   000000 000000 000000 000000
00000000007   000000 040000 000000 000000
00000000010   000000 000000 000000 000000
00000000011   000000 000000 000000 000000
00000000012   000000 000000 000000 000000
00000000013   000000 000000 000000 000000
00000000014   000000 000000 000000 000000
00000000015   000000 000000 000000 000000
00000000016   000000 000000 000000 000000
00000000017   000000 000000 000000 000000
00000000020   000000 064000 000000 000000
00000000021   000000 000000 000000 000000
00000000022   000077 177777 000000 000000
00000000023   000000 000000 000000 000000
00000000024   000077 177777 000000 000000
00000000025   046000 000013 000000 000000
00000000026   000000 000000 000000 000000
00000000027   000000 040000 000000 000000
00000000030   000000 000000 000000 000000
00000000031   000000 000000 000000 000000
00000000032   000000 000000 000000 000000
00000000033   000000 000000 000000 000000
00000000034   000000 000000 000000 000000
00000000035   000000 000000 000000 000000
00000000036   000000 000000 000000 000000
00000000037   000000 000000 000000 000000
```

4. When you press and release the return (↵) key, memory is updated. The following window shows the updated memory:

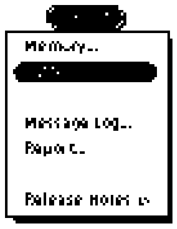

Memory

```
00000000000   000000 064000 000000 000000
00000000001   000000 000000 000000 000000
00000000002   000077 177777 000000 000000
00000000003   000000 000000 000000 000000
00000000004   000077 177777 000000 000000
00000000005   046000 000217 000000 000000
00000000006   000000 000000 000000 000000
00000000007   000000 040000 000000 000000
00000000010   000000 000000 000000 000000
00000000011   000000 000000 000000 000000
00000000012   000000 000000 000000 000000
00000000013   000000 000000 000000 000000
00000000014   000000 000000 000000 000000
00000000015   000000 000000 000000 000000
00000000016   000000 000000 000000 000000
00000000017   000000 000000 000000 000000
00000000020   000000 064000 000000 000000
00000000021   000000 000000 000000 000000
00000000022   000077 177777 000000 000000
00000000023   000000 000000 000000 000000
00000000024   000077 177777 000000 000000
00000000025   046000 000013 000000 000000
00000000026   000000 000000 000000 000000
00000000027   000000 040000 000000 000000
00000000030   000000 000000 000000 000000
00000000031   000000 000000 000000 000000
00000000032   000000 000000 000000 000000
00000000033   000000 000000 000000 000000
00000000034   000000 000000 000000 000000
00000000035   000000 000000 000000 000000
00000000036   000000 000000 000000 000000
00000000037   000000 000000 000000 000000
```

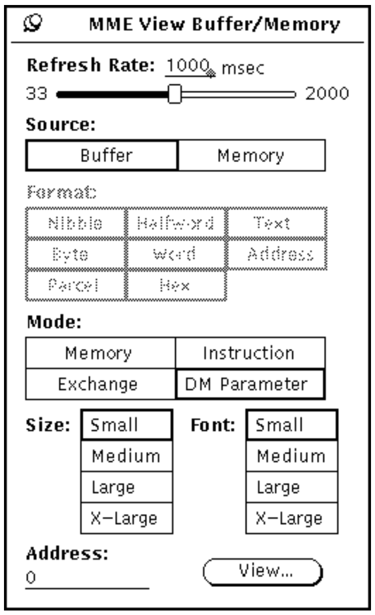
5. Repeat Steps 1 through 4 until you finish changing mainframe memory.

Viewing Buffer Data in Environment 0

To view buffer data in environment 0, perform the following procedure:



1. Choose **`View --> Buffer`**, as shown at the left. MME displays the `MME View Buffer/Memory` window:



MME View Buffer/Memory

Refresh Rate: 1000 msec
33 2000

Source: [✓ Buffer] Memory

Format: Nibble Halfword Text Byte Word Address Parcel Hex

Mode: Memory Instruction Exchange [✓ DM Parameter]

Size: [✓ Small] Medium Large X-Large

Font: [✓ Small] Medium Large X-Large

Address: 0

View...

NOTE: You can set the interval at which buffer data windows are updated by moving the `Refresh Rate` slider or by double clicking on the `Refresh Rate` field, typing a new value, and pressing the return (↵) key. Setting this value too low can monopolize the workstation CPU.

2. Click on a `Format` [Nibble, Halfword, Text, Byte, Word, Address, Parcel, or Hex (hexadecimal)] to specify the format in which you want the buffer data displayed.

3. Click on a `Mode` (Memory, Instruction, Exchange, or DM Parameter) to specify which way you want the data represented.

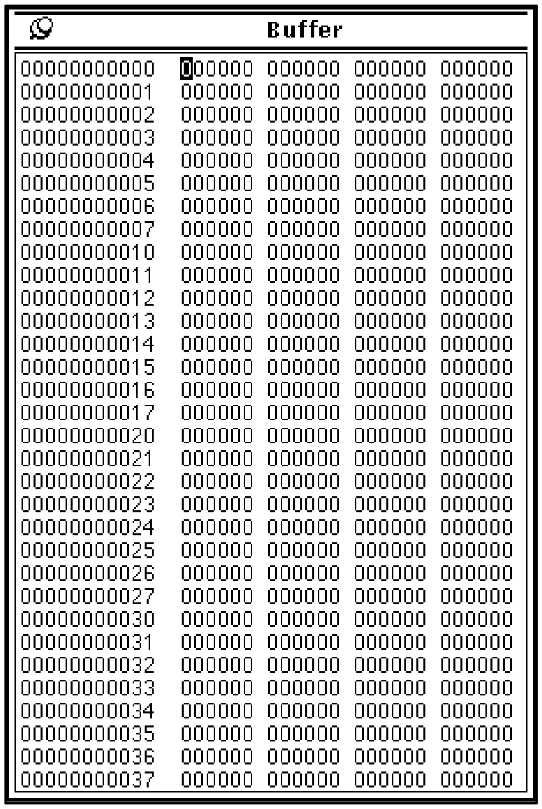
   Memory mode displays normal memory, exchange mode displays exchange information, instruction mode decodes the memory into instructions, and DM parameter mode translates the memory into DM parameter information.

4. Click on a `Size` [Small, Medium, Large, or X-Large (extra large)] to specify the size of the display window. This affects only the memory and instruction windows.

5. Click on a `Font` [Small, Medium, Large, or X-Large (extra large)] to specify the font size to display in the window.

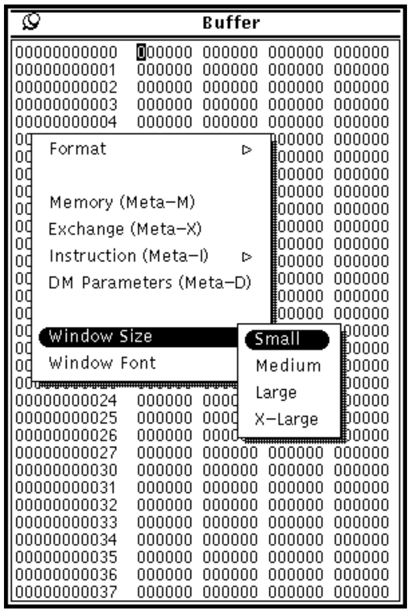
6. Change the starting address, if necessary, by double clicking on the `Address` field and typing a new value.

7. Click on View...; MME displays the `Buffer` window:



```
Buffer
00000000000   000000 000000 000000 000000
00000000001   000000 000000 000000 000000
00000000002   000000 000000 000000 000000
00000000003   000000 000000 000000 000000
00000000004   000000 000000 000000 000000
00000000005   000000 000000 000000 000000
00000000006   000000 000000 000000 000000
00000000007   000000 000000 000000 000000
00000000010   000000 000000 000000 000000
00000000011   000000 000000 000000 000000
00000000012   000000 000000 000000 000000
00000000013   000000 000000 000000 000000
00000000014   000000 000000 000000 000000
00000000015   000000 000000 000000 000000
00000000016   000000 000000 000000 000000
00000000017   000000 000000 000000 000000
00000000020   000000 000000 000000 000000
00000000021   000000 000000 000000 000000
00000000022   000000 000000 000000 000000
00000000023   000000 000000 000000 000000
00000000024   000000 000000 000000 000000
00000000025   000000 000000 000000 000000
00000000026   000000 000000 000000 000000
00000000027   000000 000000 000000 000000
00000000030   000000 000000 000000 000000
00000000031   000000 000000 000000 000000
00000000032   000000 000000 000000 000000
00000000033   000000 000000 000000 000000
00000000034   000000 000000 000000 000000
00000000035   000000 000000 000000 000000
00000000036   000000 000000 000000 000000
00000000037   000000 000000 000000 000000
```

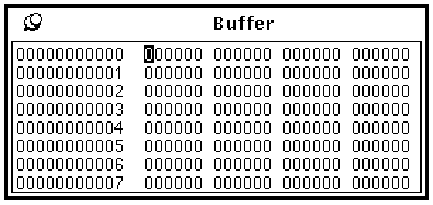
If you want to change the `Format`, `Window Size`, or `Window Font` from the `Buffer` window, press the MENU mouse button and move the mouse pointer to change to the menu item you want.



```
00000000000  000000 000000 000000 000000
00000000001  000000 000000 000000 000000
00000000002  000000 000000 000000 000000
00000000003  000000 000000 000000 000000
00000000004  000000 000000 000000 000000
00000000024  000000 000
00000000025  000000 000
00000000026  000000 000
00000000027  000000 000000 000000 000000
00000000030  000000 000000 000000 000000
00000000031  000000 000000 000000 000000
00000000032  000000 000000 000000 000000
00000000033  000000 000000 000000 000000
00000000034  000000 000000 000000 000000
00000000035  000000 000000 000000 000000
00000000036  000000 000000 000000 000000
00000000037  000000 000000 000000 000000
```

NOTE: In this example, instead of using the MENU mouse button, you can also use the diamond-shaped (◇) meta key to the left of the space bar and one of the following alphabetical keys: `a` for address format, `n` for nibble format, `b` for byte format, `p` for parcel format, `h` for halfword format, `w` for word format, `e` for hexadecimal format, `t` for text format, `i` for instruction mode, `x` for exchange mode, `d` for diagnostic monitor mode, and `m` for memory mode.

The following `Buffer` window appears when you choose a small buffer window size:



```
00000000000  000000 000000 000000 000000
00000000001  000000 000000 000000 000000
00000000002  000000 000000 000000 000000
00000000003  000000 000000 000000 000000
00000000004  000000 000000 000000 000000
00000000005  000000 000000 000000 000000
00000000006  000000 000000 000000 000000
00000000007  000000 000000 000000 000000
```

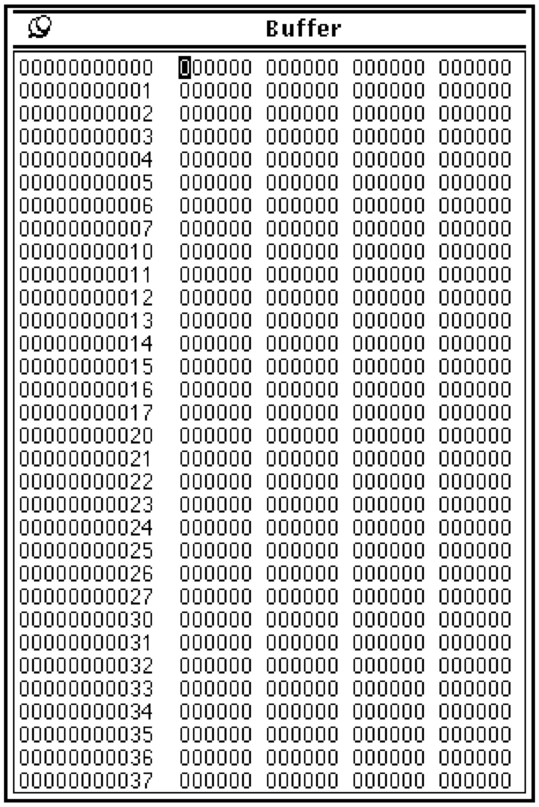
You can also change the window format and font in a manner similar to changing the window size.

Changing Buffer Data in Environment 0

After you have displayed buffer data (refer to “Viewing Buffer Data in Environment 0”), you may want to change the buffer data in environment 0 for a test you want to run.

To change buffer data from the `Buffer` window, perform the following procedure:

1. Refer to “Viewing Memory in Environment 0” to view the following `Buffer` window (use all the default values on the `MME View Buffer/Memory` window):

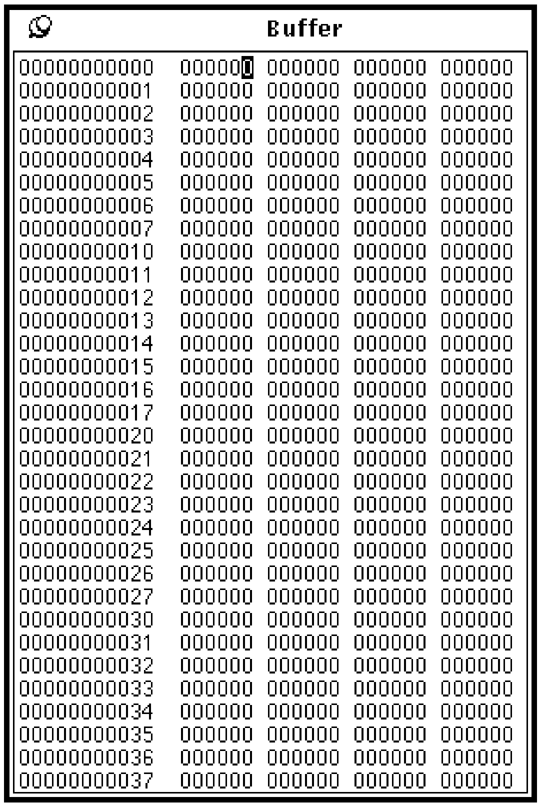


```
00000000000  000000 000000 000000 000000
00000000001  000000 000000 000000 000000
00000000002  000000 000000 000000 000000
00000000003  000000 000000 000000 000000
00000000004  000000 000000 000000 000000
00000000005  000000 000000 000000 000000
00000000006  000000 000000 000000 000000
00000000007  000000 000000 000000 000000
00000000010  000000 000000 000000 000000
00000000011  000000 000000 000000 000000
00000000012  000000 000000 000000 000000
00000000013  000000 000000 000000 000000
00000000014  000000 000000 000000 000000
00000000015  000000 000000 000000 000000
00000000016  000000 000000 000000 000000
00000000017  000000 000000 000000 000000
00000000020  000000 000000 000000 000000
00000000021  000000 000000 000000 000000
00000000022  000000 000000 000000 000000
00000000023  000000 000000 000000 000000
00000000024  000000 000000 000000 000000
00000000025  000000 000000 000000 000000
00000000026  000000 000000 000000 000000
00000000027  000000 000000 000000 000000
00000000030  000000 000000 000000 000000
00000000031  000000 000000 000000 000000
00000000032  000000 000000 000000 000000
00000000033  000000 000000 000000 000000
00000000034  000000 000000 000000 000000
00000000035  000000 000000 000000 000000
00000000036  000000 000000 000000 000000
00000000037  000000 000000 000000 000000
```

Notice that, by default, the first part of the word at buffer address 0, parcel a, appears highlighted in the `Buffer` window.

2. Perform one of the following actions:

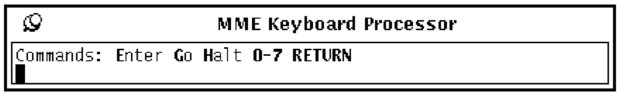
   - Move the mouse pointer to the word in the buffer you want to change using the PgUp (page up) or PgDn (page down) keys and the cursor movement keys. Click on the word you want so that the word appears in reverse video.



```
00000000000   000000 000000 000000 000000
00000000001   000000 000000 000000 000000
00000000002   000000 000000 000000 000000
00000000003   000000 000000 000000 000000
00000000004   000000 000000 000000 000000
00000000005   000000 000000 000000 000000
00000000006   000000 000000 000000 000000
00000000007   000000 000000 000000 000000
00000000010   000000 000000 000000 000000
00000000011   000000 000000 000000 000000
00000000012   000000 000000 000000 000000
00000000013   000000 000000 000000 000000
00000000014   000000 000000 000000 000000
00000000015   000000 000000 000000 000000
00000000016   000000 000000 000000 000000
00000000017   000000 000000 000000 000000
00000000020   000000 000000 000000 000000
00000000021   000000 000000 000000 000000
00000000022   000000 000000 000000 000000
00000000023   000000 000000 000000 000000
00000000024   000000 000000 000000 000000
00000000025   000000 000000 000000 000000
00000000026   000000 000000 000000 000000
00000000027   000000 000000 000000 000000
00000000030   000000 000000 000000 000000
00000000031   000000 000000 000000 000000
00000000032   000000 000000 000000 000000
00000000033   000000 000000 000000 000000
00000000034   000000 000000 000000 000000
00000000035   000000 000000 000000 000000
00000000036   000000 000000 000000 000000
00000000037   000000 000000 000000 000000
```

For example, the first 0 of the word at buffer address 0, parcel a, appears highlighted in the previous `Buffer` window.

- Press and release the space bar; the `MME Keyboard Processor` window appears:

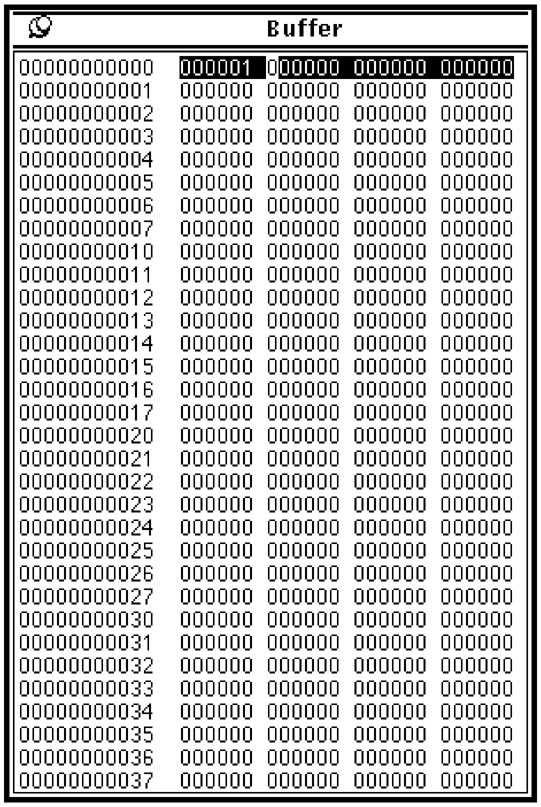


```
MME Keyboard Processor
Commands: Enter Go Halt 0-7 RETURN
```

Using this window, you can change (convert) the format; enter data in the buffer; start or halt a test; or view buffer data at a specific address. When you type the letter shown in bold, the command runs. The next window prompts you for any additional commands or information you must enter.

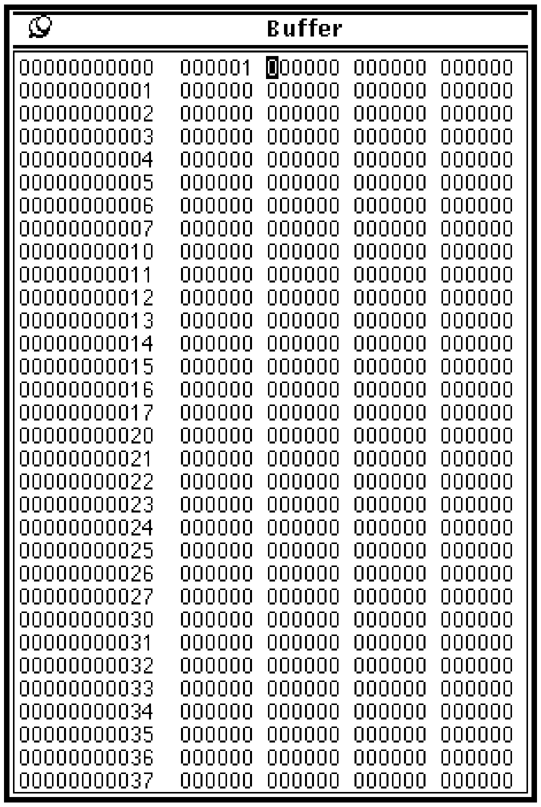
3. Press and release the Esc key. Type the new word value. The entire word is highlighted, which enables you to change the entire word.

   When you type a new word value and use the return (↵) key in Step 4, it automatically changes the value in the buffer. In this example, the `Buffer` window shows the word value changed from 000000 to 000001 at memory address 0, parcel a:



```
Buffer
00000000000   000001 000000 000000 000000
00000000001   000000 000000 000000 000000
00000000002   000000 000000 000000 000000
00000000003   000000 000000 000000 000000
00000000004   000000 000000 000000 000000
00000000005   000000 000000 000000 000000
00000000006   000000 000000 000000 000000
00000000007   000000 000000 000000 000000
00000000010   000000 000000 000000 000000
00000000011   000000 000000 000000 000000
00000000012   000000 000000 000000 000000
00000000013   000000 000000 000000 000000
00000000014   000000 000000 000000 000000
00000000015   000000 000000 000000 000000
00000000016   000000 000000 000000 000000
00000000017   000000 000000 000000 000000
00000000020   000000 000000 000000 000000
00000000021   000000 000000 000000 000000
00000000022   000000 000000 000000 000000
00000000023   000000 000000 000000 000000
00000000024   000000 000000 000000 000000
00000000025   000000 000000 000000 000000
00000000026   000000 000000 000000 000000
00000000027   000000 000000 000000 000000
00000000030   000000 000000 000000 000000
00000000031   000000 000000 000000 000000
00000000032   000000 000000 000000 000000
00000000033   000000 000000 000000 000000
00000000034   000000 000000 000000 000000
00000000035   000000 000000 000000 000000
00000000036   000000 000000 000000 000000
00000000037   000000 000000 000000 000000
```

4. When you press and release the return (↵) key, the buffer data is updated, as shown in the following updated `Buffer` window:



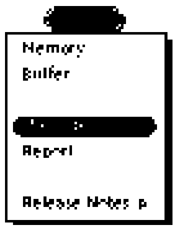
```
Buffer
00000000000   000001 000000 000000 000000
00000000001   000000 000000 000000 000000
00000000002   000000 000000 000000 000000
00000000003   000000 000000 000000 000000
00000000004   000000 000000 000000 000000
00000000005   000000 000000 000000 000000
00000000006   000000 000000 000000 000000
00000000007   000000 000000 000000 000000
00000000010   000000 000000 000000 000000
00000000011   000000 000000 000000 000000
00000000012   000000 000000 000000 000000
00000000013   000000 000000 000000 000000
00000000014   000000 000000 000000 000000
00000000015   000000 000000 000000 000000
00000000016   000000 000000 000000 000000
00000000017   000000 000000 000000 000000
00000000020   000000 000000 000000 000000
00000000021   000000 000000 000000 000000
00000000022   000000 000000 000000 000000
00000000023   000000 000000 000000 000000
00000000024   000000 000000 000000 000000
00000000025   000000 000000 000000 000000
00000000026   000000 000000 000000 000000
00000000027   000000 000000 000000 000000
00000000030   000000 000000 000000 000000
00000000031   000000 000000 000000 000000
00000000032   000000 000000 000000 000000
00000000033   000000 000000 000000 000000
00000000034   000000 000000 000000 000000
00000000035   000000 000000 000000 000000
00000000036   000000 000000 000000 000000
00000000037   000000 000000 000000 000000
```

5. Repeat Steps 1 through 4 until you finish changing buffer data.

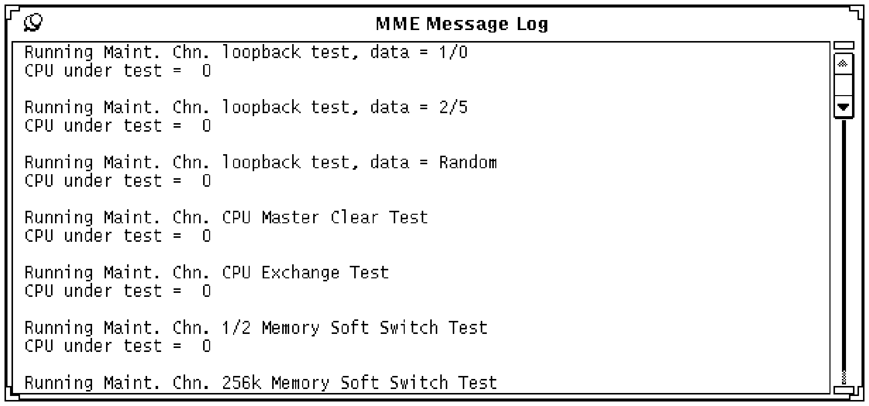
Viewing the Message Log in Environment 0

You can use the `MME Message Log` window to perform the following tasks:

- View errors from the diagnostic program
- Identify where any errors occurred
- Ensure the diagnostic program is running properly

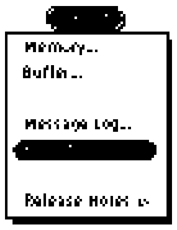


To view the message log, choose **`View --> Message Log`**, as shown at the left. MME displays the `MME Message Log` window:

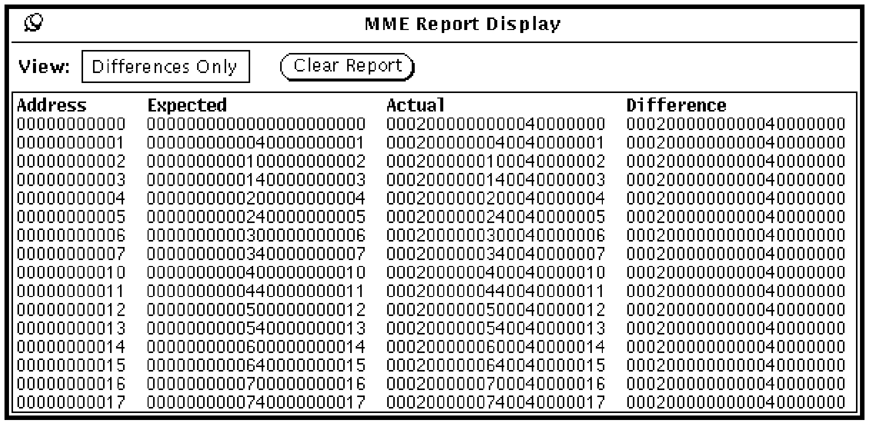


```
Running Maint. Chn. loopback test, data = 1/0
CPU under test =  0

Running Maint. Chn. loopback test, data = 2/5
CPU under test =  0

Running Maint. Chn. loopback test, data = Random
CPU under test =  0

Running Maint. Chn. CPU Master Clear Test
CPU under test =  0

Running Maint. Chn. CPU Exchange Test
CPU under test =  0

Running Maint. Chn. 1/2 Memory Soft Switch Test
CPU under test =  0

Running Maint. Chn. 256k Memory Soft Switch Test
```

Viewing a Report for Memory and Instruction Buffer Errors in Environment 0



To view a `Report` window in order to find memory errors or instruction buffer errors, choose **`View --> Report`**, as shown at the left. MME displays the `MME Report Display`:

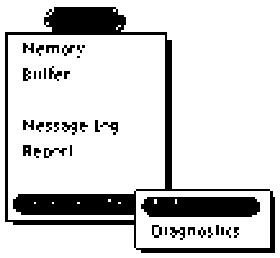


Address	Expected	Actual	Difference
00000000000	0000000000000000000000	0002000000000040000000	0002000000000040000000
00000000001	0000000000040000000001	0002000000040040000001	0002000000000040000000
00000000002	0000000000100000000002	0002000000100040000002	0002000000000040000000
00000000003	0000000000140000000003	0002000000140040000003	0002000000000040000000
00000000004	0000000000200000000004	0002000000200040000004	0002000000000040000000
00000000005	0000000000240000000005	0002000000240040000005	0002000000000040000000
00000000006	0000000000300000000006	0002000000300040000006	0002000000000040000000
00000000007	0000000000340000000007	0002000000340040000007	0002000000000040000000
00000000010	0000000000400000000010	0002000000400040000010	0002000000000040000000
00000000011	0000000000440000000011	0002000000440040000011	0002000000000040000000
00000000012	0000000000500000000012	0002000000500040000012	0002000000000040000000
00000000013	0000000000540000000013	0002000000540040000013	0002000000000040000000
00000000014	0000000000600000000014	0002000000600040000014	0002000000000040000000
00000000015	0000000000640000000015	0002000000640040000015	0002000000000040000000
00000000016	0000000000700000000016	0002000000700040000016	0002000000000040000000
00000000017	0000000000740000000017	0002000000740040000017	0002000000000040000000

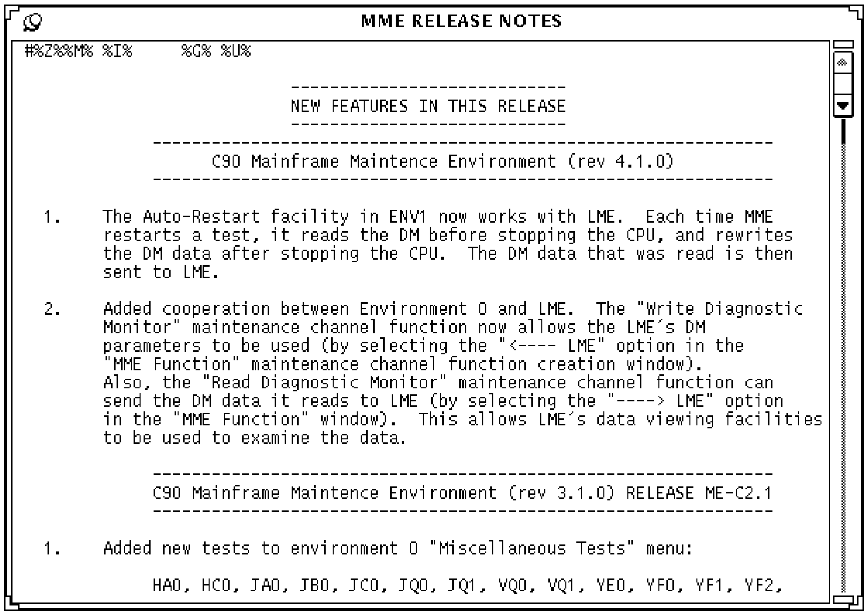
You can use the buttons in this window to perform the following actions:

- Click on Differences Only to display only the addresses where differences have occurred.
- Click on Clear Report to clear the `MME Report Display` of data.

Viewing MME Release Notes in Environment 0

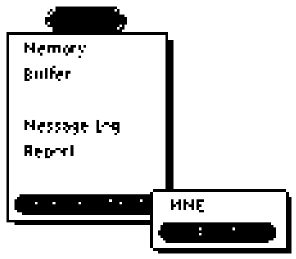


To view information about the most recently installed release of MME, choose **`View --> Release Notes --> MME`**, as shown at the left. The `MME RELEASE NOTES` window appears:

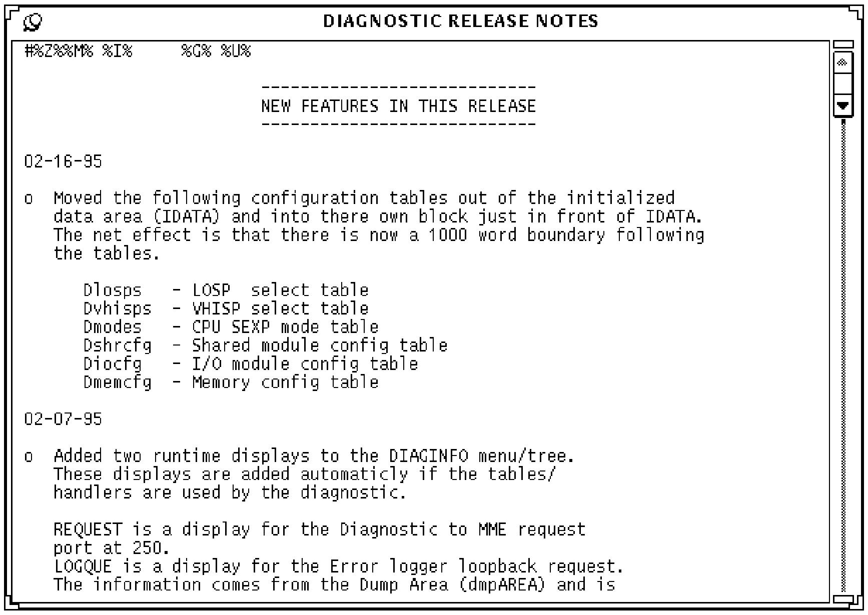


```
#%Z%%M% %I%     %G% %U%

                          ----------------------------
                          NEW FEATURES IN THIS RELEASE
                          ----------------------------
              ---------------------------------------------------------------
                    C90 Mainframe Maintence Environment (rev 4.1.0)
              ---------------------------------------------------------------

   1.    The Auto-Restart facility in ENV1 now works with LME.  Each time MME
         restarts a test, it reads the DM before stopping the CPU, and rewrites
         the DM data after stopping the CPU.  The DM data that was read is then
         sent to LME.

   2.    Added cooperation between Environment 0 and LME.  The "Write Diagnostic
         Monitor" maintenance channel function now allows the LME's DM
         parameters to be used (by selecting the "<---- LME" option in the
         "MME Function" maintenance channel function creation window).
         Also, the "Read Diagnostic Monitor" maintenance channel function can
         send the DM data it reads to LME (by selecting the "----> LME" option
         in the "MME Function" window).  This allows LME's data viewing facilities
         to be used to examine the data.

              ---------------------------------------------------------------
              C90 Mainframe Maintence Environment (rev 3.1.0) RELEASE ME-C2.1
              ---------------------------------------------------------------

   1.    Added new tests to environment 0 "Miscellaneous Tests" menu:

              HA0, HC0, JA0, JB0, JC0, JQ0, JQ1, VQ0, VQ1, YE0, YF0, YF1, YF2,
```

Viewing MME Diagnostic Program Notes in Environment 0



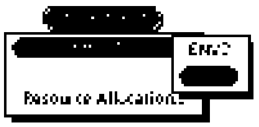
To view information about the most recently installed release of diagnostic programs, choose **`View --> Release Notes --> Diagnostics`**, as shown at the left. The `DIAGNOSTIC RELEASE NOTES` window appears:



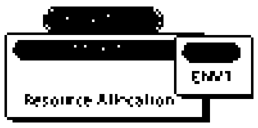
```
#%Z%%M% %I%     %G% %U%

                        ----------------------------
                        NEW FEATURES IN THIS RELEASE
                        ----------------------------

02-16-95

o  Moved the following configuration tables out of the initialized
   data area (IDATA) and into there own block just in front of IDATA.
   The net effect is that there is now a 1000 word boundary following
   the tables.

      Dlosps   - LOSP  select table
      Dvhisps  - VHISP select table
      Dmodes   - CPU SEXP mode table
      Dshrcfg  - Shared module config table
      Diocfg   - I/O module config table
      Dmemcfg  - Memory config table

02-07-95

o  Added two runtime displays to the DIAGINFO menu/tree.
   These displays are added automaticly if the tables/
   handlers are used by the diagnostic.

   REQUEST is a display for the Diagnostic to MME request
   port at 250.
   LOGQUE is a display for the Error logger loopback request.
   The information comes from the Dump Area (dmpAREA) and is
```

Changing from Environment 0

In order to change the testing method (MWS-based testing in environment 0, low-level mainframe-based testing in environment 1, or confidence-level mainframe-based testing in environment 2), you must change environments.



- To change from environment 0 to environment 1, choose **`Properties --> Environment --> ENV1`**, as shown at the left.



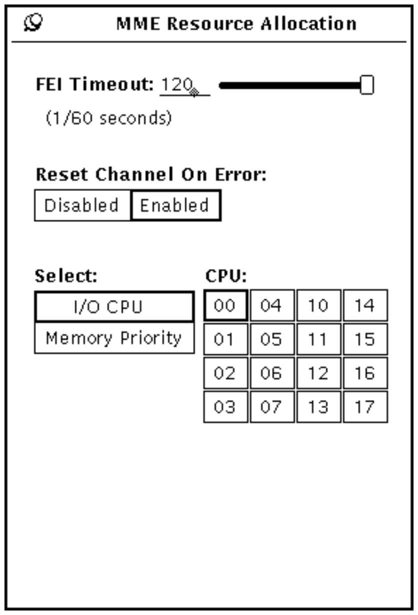
- To change from environment 0 to environment 2, choose **`Properties --> Environment --> ENV2`**, as shown at the left.

NOTE: Before you change environments, save all control points and test lists. All control points and test lists are lost unless you save them before changing environments.

Allocating Resources in Environment 0

You can change resources to perform the following activities: set the front-end interface (FEI) timeout for scoping or normal diagnostic troubleshooting; specify which CPU writes to and which CPU reads from mainframe memory; and specify the hardware areas tested. To change the resource allocation, perform the following procedure:

1. Choose **`Properties --> Resource Allocation`**, as shown at the left. MME displays the `MME Resource Allocation` window for environment 0.



FEI Timeout: 120
(1/60 seconds)

Reset Channel On Error:
Disabled | Enabled

Select:
I/O CPU
Memory Priority

CPU:

00	04	10	14
01	05	11	15
02	06	12	16
03	07	13	17

2. Change the `FEI Timeout` value, if necessary, by clicking on the `FEI Timeout` field, typing the timeout value you want to use, and pressing the return (↵) key or by dragging the `FEI Timeout` slider to the timeout value you want to use.

   NOTE: Use a low value (3) for scoping and a high value (120) for normal troubleshooting.

3. Click on Enabled to enable resetting the maintenance channel when an error occurs, or click on Disabled to disable resetting the maintenance channel when an error occurs.

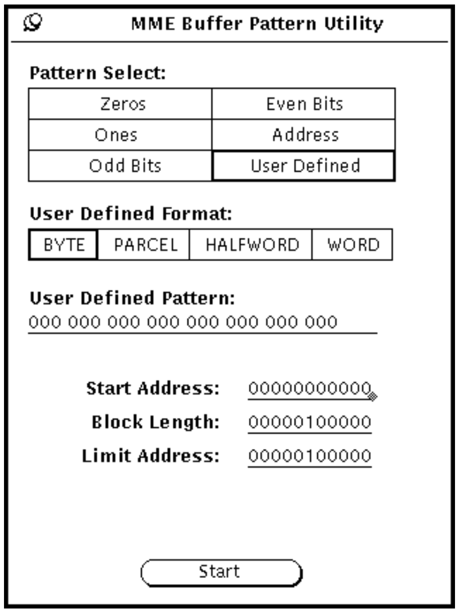
4. Select an I/O CPU, if desired, by clicking on I/O CPU and clicking on a CPU (MME uses `CPU 00` as the default).

5. Specify memory priority, if desired, by clicking on Memory Priority and clicking on a CPU (MME uses `CPU 00` as the default).

Using the Buffer Pattern Utility in Environment 0

The buffer pattern utility writes a specified pattern of data into the buffer. To enter a pattern of zeros, ones, odd bits, even bits, random data, addresses, or user-defined data into the buffer for testing purposes, perform the following procedure:

1. Choose **`Utilities --> Pattern`**, as shown at the left. MME displays the `MME Buffer Pattern Utility` window:



MME Buffer Pattern Utility

Pattern Select:

Zeros	Even Bits
Ones	Address
Odd Bits	User Defined

User Defined Format:

BYTE	PARCEL	HALFWORD	WORD

User Defined Pattern:
000 000 000 000 000 000 000 000

Start Address:	00000000000
Block Length:	00000100000
Limit Address:	00000100000

Start

2. Select a pattern to write to the buffer by clicking on one of the following: Zeros, Ones, Odd Bits, Even Bits, Address, or User Defined.

   If you clicked on User Defined, perform the following three actions:

   - Indicate the format you want to use (click on BYTE, PARCEL, HALFWORD, or WORD)

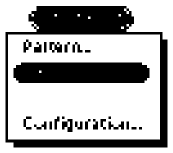
   - Enter the pattern [triple click on the `User Defined Pattern` field, enter the pattern, and press the return (↵) key]

- Enter the correct data in two of the three fields: `Start Address`, `Block Length`, and `End Address` [double click on the field, enter the value, and press the return (↵) key].

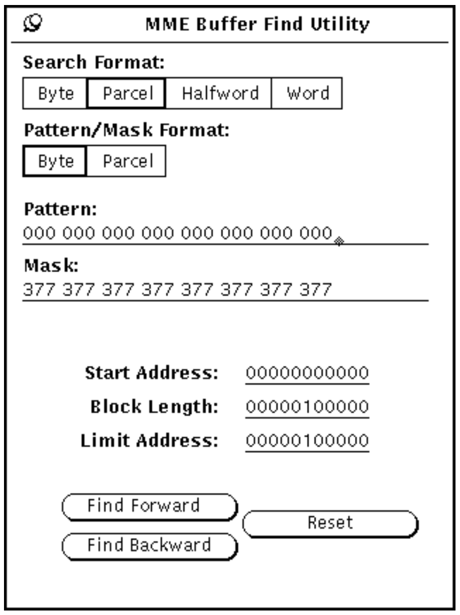
3. Click on Start. MME starts the buffer utility.

Using the Find Utility in Environment 0

To use the find utility to locate a word, parcel, 9-bit pattern, or check-bit pattern in the buffer, perform the following procedure:



1. Choose **`Utilities --> Find`**, as shown at the left. MME displays the `MME Buffer Find Utility`:



MME Buffer Find Utility

Search Format: Byte | Parcel | Halfword | Word

Pattern/Mask Format: Byte | Parcel

Pattern:
000 000 000 000 000 000 000 000

Mask:
377 377 377 377 377 377 377 377

Start Address: 00000000000
Block Length: 00000100000
Limit Address: 00000100000

Find Forward
Find Backward
Reset

2. Click on the `Search Format` [Byte, Parcel (Default), Halfword, or Word] you want to use.

3. Click on Byte or Parcel to indicate the pattern and mask format.

4. Double click on the `Pattern` field. Type the data pattern you want to locate.

5. Click on the `Mask` field and perform one of the following actions:

   - Use the default value of 177777 to search for the data you entered in the `Pattern` field
   - Type `000000` to prevent the search.

6. After you perform two of the following three steps, MME automatically updates the third field:

   - Double click on the `Start Address` field. Type the starting address (in octal) of the data search.
   - Double click on the `Block Length` field. Type the number of blocks in octal you want to search.
   - Double click on the `Limit Address` field. Type the last address (in octal) you want to search.

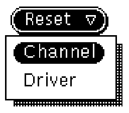
7. Start the search by performing one of the following actions:

   - Click on Find Forward to search from the starting (lowest memory) address to the last (lowest memory) address.
   - Click on Find Backward to search from the last (highest memory) address to the starting (lowest memory) address.
   - Click on Reset to change the values back to the previous values.

Configuring MME in Environment 0

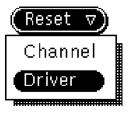
To view or change the MME configuration for your system, choose **`Utilities --> Configuration`**, as shown at the left. This starts the mainframe configuration environment (MCE). For more information about configuring MME with MCE, refer to Section 2, “Mainframe Configuration Environment,” in this manual.

Resetting the Channel in Environment 0



To reset the channel in environment 0, choose **`Reset --> Channel`**, as shown at the left. This initializes the driver, initializes the channel, and reasserts the configuration.

Resetting the Driver in Environment 0



To reset the driver in environment 0, choose **`Reset --> Driver`**, as shown at the left. This initializes the driver.

Testing in Environment 0

There are three test modes you can use in environment 0: automatic, manual, and compose. You select these modes by clicking on Automatic; Manual; and Compose, respectively. Automatic mode runs a predefined series of sequences against user-selected areas and CPUs. Manual mode runs user-selected sequences against a single user-selected area and user-selected CPUs. Compose mode runs a user-defined sequence.

NOTE: You can use compose mode to determine which function failed in a test sequence when an error occurs in automatic or manual mode. To do this, click on `Error Mode:` Stop, which stops test execution when an error occurs. Then, after an error occurs, click on `Test Mode:` Compose. The interface for compose mode is displayed, and the failing function in the test sequence is highlighted in a box.

Using Automatic and Manual Test Modes

This subsection gives step-by-step procedures that describe how to run each of the environment 0 tests in automatic and manual test modes. For a description of the environment 0 tests, refer to Section 7, "Diagnostic Tests and Utilities," in this manual.

Running the Maintenance Channel Test in Automatic or Manual Test Mode

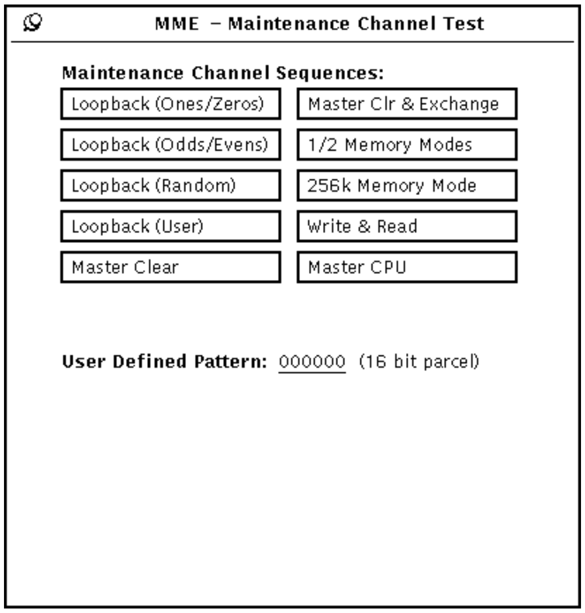
To run the maintenance channel test, perform the following procedure:

1. Click on Automatic to run the test in automatic mode or Manual to run the test in manual mode.

2. Click on the CPU(s) you want to use.

3. Click on Maintenance Channel Test to select the maintenance channel test.

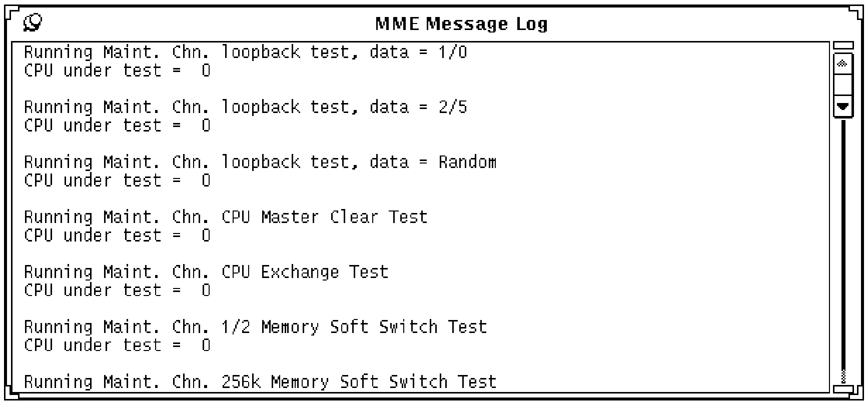
   If you are in manual test mode (you clicked on Manual), the following `MME – Maintenance Channel Test` window appears.



Click on the maintenance channel sequences (Loopback (Ones/Zeros), Loopback (Odds/Evens), Loopback (Random), Loopback (User), Master Clear, Master Clr & Exchange, 1/2 Memory Modes, 256k Memory Mode, Write & Read, or Master CPU) you want to run.

NOTE: If you clicked on Loopback (User), double click on the `User Defined Pattern` field, type the pattern you want to use, and press the return (↵) key.

4. Click on Go on the `Mainframe Maintenance Environment` window. The maintenance channel test runs with the `Passes` field incrementing for each pass completed and the `Errors` field incrementing for each error detected. The `MME Message Log` displays results of the test:



```
Running Maint. Chn. loopback test, data = 1/0
CPU under test =  0

Running Maint. Chn. loopback test, data = 2/5
CPU under test =  0

Running Maint. Chn. loopback test, data = Random
CPU under test =  0

Running Maint. Chn. CPU Master Clear Test
CPU under test =  0

Running Maint. Chn. CPU Exchange Test
CPU under test =  0

Running Maint. Chn. 1/2 Memory Soft Switch Test
CPU under test =  0

Running Maint. Chn. 256k Memory Soft Switch Test
```

Running the Diagnostic Monitor Test in Automatic or Manual Test Mode

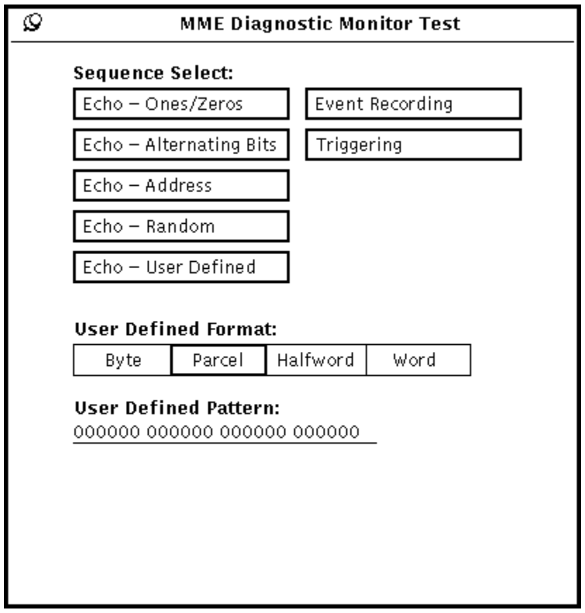
To run the diagnostic monitor test, perform the following procedure:

1. Click on Automatic to run the test in automatic mode or Manual to run the test in manual mode.

2. Click on the CPU(s) you want to use.

3. Click on Diagnostic Monitor Test to select the diagnostic monitor test.

4. If you are in manual test mode (you clicked on Manual), the `MME Diagnostic Monitor Test` window appears:

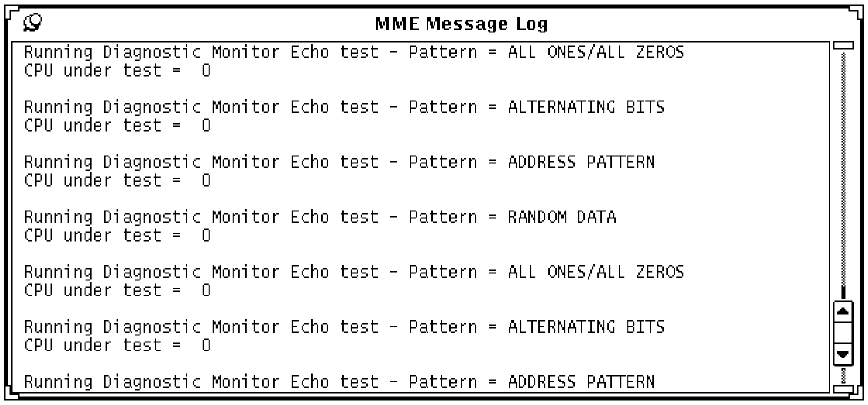


000000 000000 000000 000000

Click on the diagnostic monitor sequences (Echo – Ones/Zeros, Echo – Alternating Bits, Echo – Address, Echo – Random, Echo – User Defined, Event Recording, and Triggering) you want to run.

NOTE: If you clicked on Echo – User Defined, click on the format (Byte, Parcel, Halfword, or Word) you want to use. Double click on the `User Defined Pattern` field, type the pattern you want to use, and press the return (↵) key.

5. Click on Go on the `Mainframe Maintenance Environment` window. The diagnostic monitor test runs with the `Passes` field incrementing for each pass completed and the `Errors` field incrementing for each error detected. The `MME Message Log` displays results of the test:



```
Running Diagnostic Monitor Echo test - Pattern = ALL ONES/ALL ZEROS
CPU under test =  0

Running Diagnostic Monitor Echo test - Pattern = ALTERNATING BITS
CPU under test =  0

Running Diagnostic Monitor Echo test - Pattern = ADDRESS PATTERN
CPU under test =  0

Running Diagnostic Monitor Echo test - Pattern = RANDOM DATA
CPU under test =  0

Running Diagnostic Monitor Echo test - Pattern = ALL ONES/ALL ZEROS
CPU under test =  0

Running Diagnostic Monitor Echo test - Pattern = ALTERNATING BITS
CPU under test =  0

Running Diagnostic Monitor Echo test - Pattern = ADDRESS PATTERN
```

Running the Memory Data Pattern Test in Automatic or Manual Test Mode

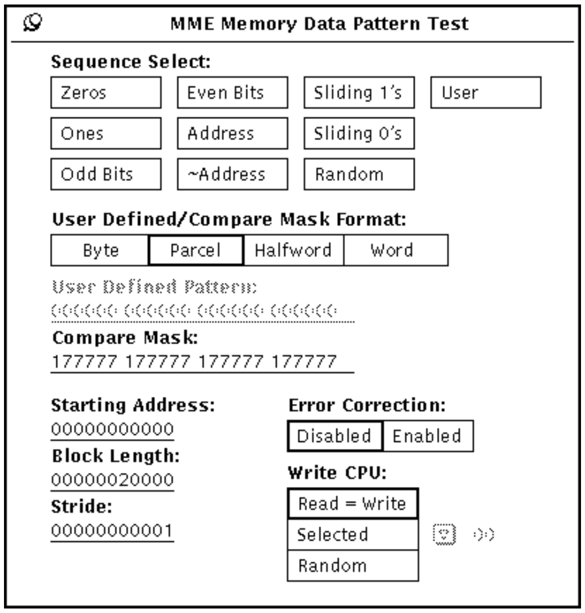
To run the memory data pattern test, perform the following procedure:

1. Click on Automatic to run the test in automatic mode or Manual to run the test in manual mode.

2. Click on the CPU(s) you want to use.

3. Click on Memory Data Pattern Test to select the memory data pattern test. If you are in automatic mode (you clicked on Automatic), go to Step 10. If you are in manual mode (you clicked on Manual), continue with Step 4.

4. Click on the memory test sequence(s) [Zeros, Ones, Odd Bits, Even Bits, Address, ~Address (complement address), Sliding 1's, Sliding 0's, Random, and User] you want to run on the `MME Memory Data Pattern Test` window that appears:



MME Memory Data Pattern Test

Sequence Select:

Zeros | Even Bits | Sliding 1's | User
Ones | Address | Sliding 0's
Odd Bits | ~Address | Random

User Defined/Compare Mask Format:

Byte | Parcel | Halfword | Word

User Defined Pattern:
000000 000000 000000 000000

Compare Mask:
177777 177777 177777 177777

Starting Address:
00000000000

Block Length:
00000020000

Stride:
00000000001

Error Correction:
Disabled | Enabled

Write CPU:
Read = Write
Selected
Random

If you clicked on a user-defined sequence or a compare mask sequence, continue with Step 5. If you did not click on a user-defined sequence or a compare mask sequence, go to Step 10.

5. Click on the format (Byte, Parcel, Halfword, or Word) you want to use. If you clicked on User, double click on the `User Defined Pattern` field, and type the pattern you want to use.

6. Double click on the `Compare Mask` field, and type the compare mask you want to use. Use the default value to compare or use zeros to prevent a compare operation.

7. Double click on the `Starting Address` field, type the starting address you want to use, and press the return (↵) key.

8. Double click on the `Block Length` field, type the length of the block you want to use, and press the return (↵) key.

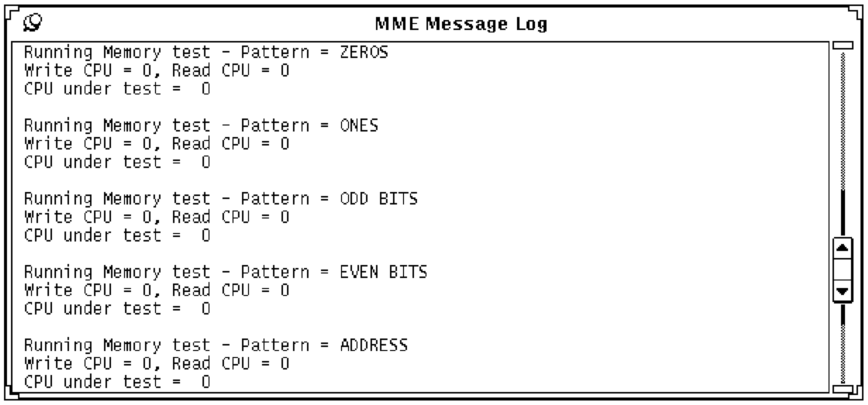
9. Specify the write CPU for the test by performing one of the following tasks:

   - Click on Read = Write to have the same CPU write the data to the instruction buffers and read the data back again.

   - Click on Selected to specify the write CPU. Choose the selected write CPU from ▽, or double click on the `Selected` field and type the CPU number in octal.

   - Click on Random to enable MME to select the write CPU randomly.

10. Click on Go on the `Mainframe Maintenance Environment` window. The memory data pattern test runs with the `Passes` field incrementing for each pass completed and the `Errors` field incrementing for each error detected. The `MME Message Log` displays results of the test:

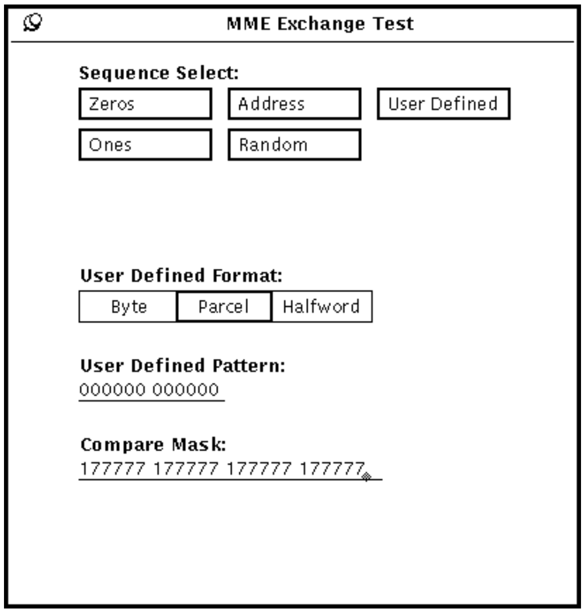


```
Running Memory test - Pattern = ZEROS
Write CPU = 0, Read CPU = 0
CPU under test =  0

Running Memory test - Pattern = ONES
Write CPU = 0, Read CPU = 0
CPU under test =  0

Running Memory test - Pattern = ODD BITS
Write CPU = 0, Read CPU = 0
CPU under test =  0

Running Memory test - Pattern = EVEN BITS
Write CPU = 0, Read CPU = 0
CPU under test =  0

Running Memory test - Pattern = ADDRESS
Write CPU = 0, Read CPU = 0
CPU under test =  0
```

Running the Exchange Test in Automatic or Manual Test Mode

To run the exchange test, perform the following procedure:

1. Click on Automatic to run the test in automatic mode or Manual to run the test in manual mode.
2. Click on the CPU(s) you want to use.
3. Click on Exchange Test to select the exchange test.
4. If you are in manual test mode (you clicked on Manual), the `MME Exchange Test` window appears:



MME Exchange Test

Sequence Select:
Zeros | Address | User Defined
Ones | Random

User Defined Format:
Byte | Parcel | Halfword

User Defined Pattern:
000000 000000

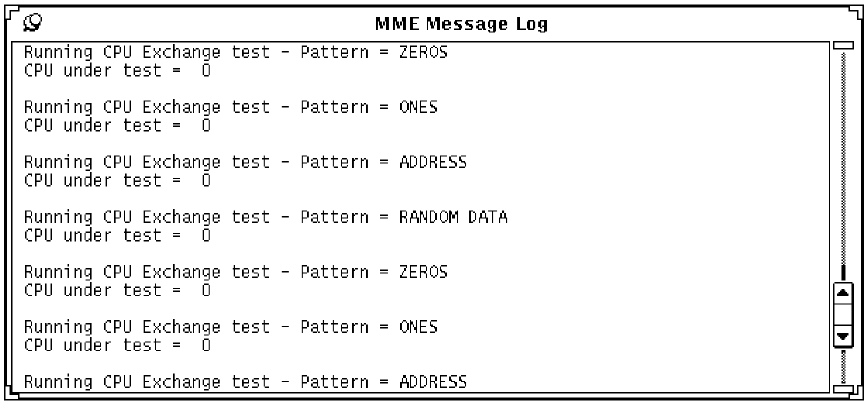
Compare Mask:
177777 177777 177777 177777

Click on the exchange test sequence(s) (Zeros, Ones, Address, Random, or User Defined) you want to run.

If you clicked on User Defined, click on the format [Byte, Parcel (default), or Halfword] you want to use for the pattern, double click on the `User Defined Pattern` field, type the pattern you want to use, and press the return (↵) key.

If you want to prevent a comparison, click on the `Compare Mask` field, and type **`000000`** in the field. If you want to run a comparison, use the default value (`177777`).

5. Click on Go on the `Mainframe Maintenance Environment` window. The exchange test runs with the `Passes` field incrementing for each pass completed and the `Errors` field incrementing for each error detected. The `MME Message Log` window displays the results of the test:



```
Running CPU Exchange test - Pattern = ZEROS
CPU under test =  0

Running CPU Exchange test - Pattern = ONES
CPU under test =  0

Running CPU Exchange test - Pattern = ADDRESS
CPU under test =  0

Running CPU Exchange test - Pattern = RANDOM DATA
CPU under test =  0

Running CPU Exchange test - Pattern = ZEROS
CPU under test =  0

Running CPU Exchange test - Pattern = ONES
CPU under test =  0

Running CPU Exchange test - Pattern = ADDRESS
```

Running the Instruction Buffer Test in Automatic or Manual Test Mode

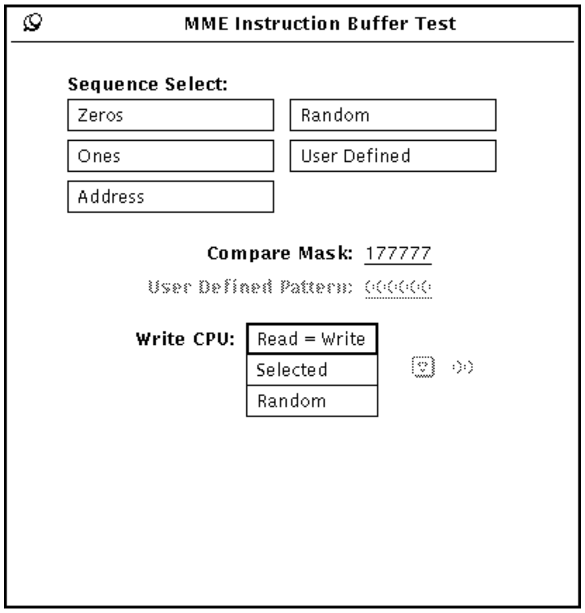
To run the instruction buffer test, perform the following procedure:

1. Click on [Automatic] to run the test in automatic mode or [Manual] to run the test in manual mode.

2. Click on the CPU(s) you want to use.

3. Click on [Instruction Buffer Test] to select the instruction buffer test.

4. If you are in manual mode (you clicked on [Manual]), the `MME Instruction Buffer Test` window appears:



Compare Mask: 177777
User Defined Pattern: 000000

Click on the instruction buffer test sequence(s) ([Zeros], [Ones], [User Defined], [Random], or [User Defined]) you want to use.

If you clicked on [User Defined], double click on the `User Defined Pattern` field, type the pattern you want to use, and press the return (↵) key if you do not want to use the default value of zeros.

If you want to prevent a comparison, click on the `Compare Mask` field, and type **`000000`** in the field. If you want to run a comparison, use the default value (`177777`).

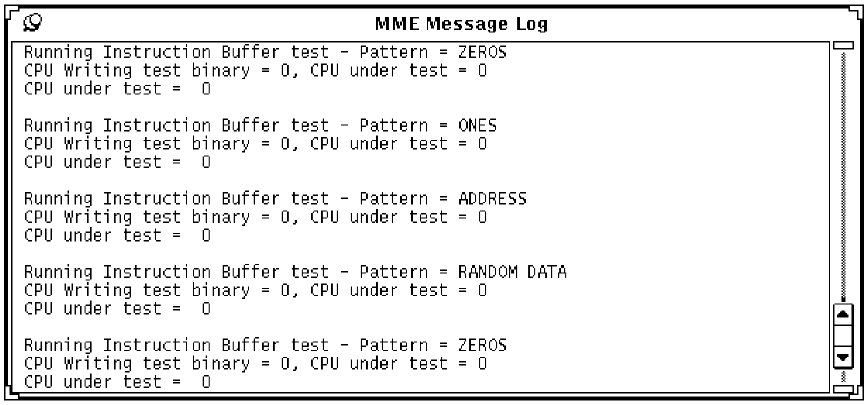
5. Specify the write CPU for the test by performing one of the following tasks:

   - Click on Read = Write to have the same CPU write the data to the instruction buffers and read the data back again.

   - Click on Selected to specify the write CPU. Choose the selected write CPU from ▽, or double click on the `Selected` field and type the CPU number in octal.

   - Click on Random to enable MME to select the write CPU randomly.

6. Click on Go on the `Mainframe Maintenance Environment` window. The instruction buffer test runs with the `Passes` field incrementing for each pass completed and the `Errors` field incrementing for each error detected. The `MME Message Log` window displays the results of the test:



```
Running Instruction Buffer test - Pattern = ZEROS
CPU Writing test binary = 0, CPU under test = 0
CPU under test =  0

Running Instruction Buffer test - Pattern = ONES
CPU Writing test binary = 0, CPU under test = 0
CPU under test =  0

Running Instruction Buffer test - Pattern = ADDRESS
CPU Writing test binary = 0, CPU under test = 0
CPU under test =  0

Running Instruction Buffer test - Pattern = RANDOM DATA
CPU Writing test binary = 0, CPU under test = 0
CPU under test =  0

Running Instruction Buffer test - Pattern = ZEROS
CPU Writing test binary = 0, CPU under test = 0
CPU under test =  0
```

Running the Miscellaneous Test in Automatic or Manual Test Mode

To run the miscellaneous test, perform the following procedure:

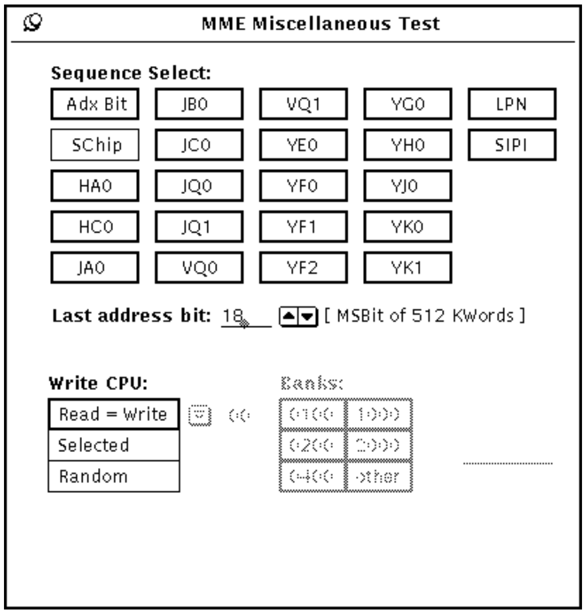
1. Click on Automatic to run the test in automatic mode or Manual to run the test in manual mode.

2. Click on the CPU(s) you want to use.

3. Click on Miscellaneous Test to run the miscellaneous test.

4. If you are in automatic mode (you clicked on Automatic), go to Step 11. If you are in manual mode (you clicked on Manual), continue with Step 5.

5. Click on the test sequence(s) you want to run (Adx Bit, SChip, HA0, HC0, JA0, JB0, JC0, JQ0, JQ1, VQ0, VQ1, YE0, YF0, YF1, YF2, YG0, YH0, YJ0, YK0, YK1, LPN, and/or SIPI) in the `MME Miscellaneous Test` window that appears for manual mode:



Last address bit: 18 [MSBit of 512 KWords]

Refer to Table 3-2 for descriptions of the miscellaneous test sequences you can select.

Table 3-2. Miscellaneous Test Sequence Descriptions

Sequence	Description
Adx Bit	Tests memory addressing by a CPU for all available memory.
SChip	Tests spare chip selection for all banks and all bit groups.
HA0	Tests the test points on the HA0 option; this tests the program counter.
HC0	Tests the test points on the HC0 option; this tests the exchange parameters.
JA0	Tests the test points on the JA0 option; this tests the A, S, and shared register issue.
JB0	Tests the test points on the JB0 option; this tests the vector functional unit and memory issue.
JC0	Tests the test points on the JC0 option; this tests the vector functional unit timing.
JQ0	Tests the test points on the JQ0 option; this tests the local semaphore (SM).
JQ1	Tests the test points on the JQ1 option; this tests the cluster number (CLN) for CPUs 0_8 through 20_8.
VQ0	Tests the test points on the VQ0 option; this tests the V0 through V3 vector register control.
VQ1	Tests the test points on the VQ1 option; this tests the V4 through V7 vector register control.
YE0	Tests the test points on the YE0 option; this tests ports A, B, and C lower addresses (2^0 to 2^5).
YF0	Tests the test points on the YF0 option; this tests port A upper addresses (2^6 to 2^{31}).
YF1	Tests the test points on the YF1 option; this tests port B upper addresses (2^6 to 2^{31}).
YF2	Tests the test points on the YF2 option; this tests port C upper addresses (2^6 to 2^{31}).
YG0	Tests the test points on the YG0 option; this tests port D lower addresses (2^0 to 2^5).
YH0	Tests the test points on the YH0 option; this tests port D upper addresses (2^6 to 2^{31}).
YJ	Tests the test points on the YJ option; this tests subsection conflict for sections 0 through 3 (the YJ0 option) and subsection conflicts for sections 4 through 7 (the YJ1 option).
YK0	Tests the test points on the YK0 option; this tests the CPU subsection conflict bank busy timing and priority counters.

Table 3-2. Miscellaneous Test Sequence Descriptions (continued)

Sequence	Description
YK1	Tests the test points on the YK1 option; this tests the CPU subsection conflict bank busy timing and priority counters.
LPN	Performs the logical processor number test; this verifies the correct logical processor number is created and stored in the exchange package for a given CPU.
SIPI	Performs the set interprocessor interrupt request (SIPI) test; this verifies that a CPU can perform a SIPI on the other CPUs. At least two CPUs must be selected for this test.

6. If you clicked on Adr. Bit in Step 5, specify the last address bit value by performing one of the following actions:

   - Use the default value.
   - Double click on the `Last address bit` field, and type the last address bit you want to compare.
   - Click on ▲ or ▼ to increment or decrement the value in the `Last address bit` field.

7. Click on the format [Byte, Parcel (default), Halfword, or Word] you want to use.

8. Double click on the `Compare Mask` field, and type the pattern you want to use if you do not want to use the default value.

   NOTE: If you want to prevent a comparison, click on the `Compare Mask` field, and type **`000000`** in the field. If you want to run a comparison, use the default value (`177777`).

9. If you clicked on Adr. Bit in Step 5, specify the write CPU for the test by performing one of the following tasks:

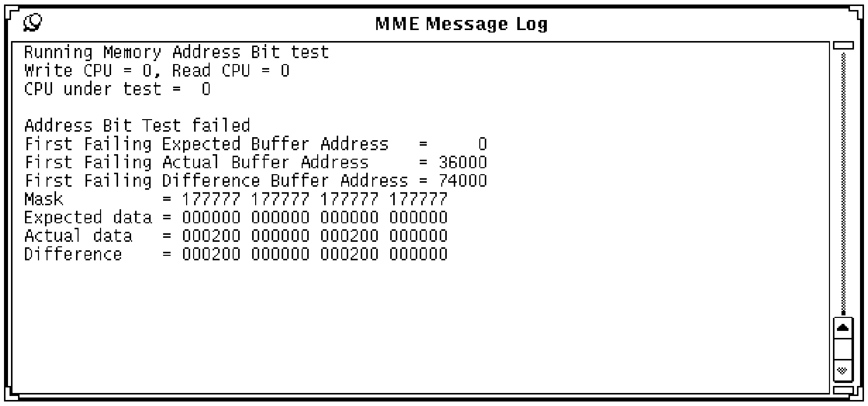
   - Click on Read = Write to have the same CPU write the data to the instruction buffers and read the data back again.

- Click on Selected to specify the write CPU. Choose the selected write CPU from ▽ or double click on the `Selected CPU` field and type the CPU number in octal.

- Click on Random to enable MME to select a write CPU randomly.

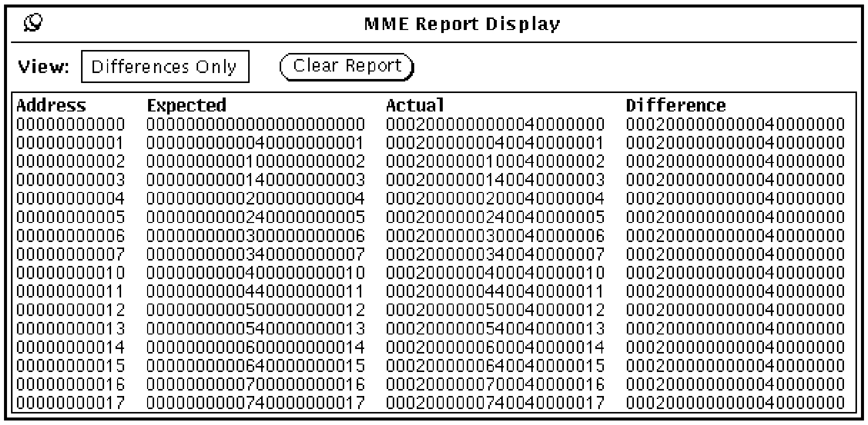
10. If you clicked on S-Chip in Step 5, specify the bank of memory you want to test. Click on `Banks:` 0100, 0200, 0400, 1000, 2000, or other. If you click on other, enter the bank number you want in the field that appears and press the return (↵) key.

11. Click on Go in the `Mainframe Maintenance Environment` window. The miscellaneous test runs with the `Passes` field incrementing for each pass completed and the `Errors` field incrementing for each error detected. The `MME Message Log` window displays the results of the test:



```
Running Memory Address Bit test
Write CPU = 0, Read CPU = 0
CPU under test =  0

Address Bit Test failed
First Failing Expected Buffer Address   =     0
First Failing Actual Buffer Address     = 36000
First Failing Difference Buffer Address = 74000
Mask          = 177777 177777 177777 177777
Expected data = 000000 000000 000000 000000
Actual data   = 000200 000000 000200 000000
Difference    = 000200 000000 000200 000000
```

MME also displays the `MME Report Display` window, which can be used to examine differences between the actual and expected memory values.



```
Address       Expected                 Actual                   Difference
00000000000   0000000000000000000000   0002000000000040000000   0002000000000040000000
00000000001   0000000000040000000001   0002000000040040000001   0002000000000040000000
00000000002   0000000000100000000002   0002000000100040000002   0002000000000040000000
00000000003   0000000000140000000003   0002000000140040000003   0002000000000040000000
00000000004   0000000000200000000004   0002000000200040000004   0002000000000040000000
00000000005   0000000000240000000005   0002000000240040000005   0002000000000040000000
00000000006   0000000000300000000006   0002000000300040000006   0002000000000040000000
00000000007   0000000000340000000007   0002000000340040000007   0002000000000040000000
00000000010   0000000000400000000010   0002000000400040000010   0002000000000040000000
00000000011   0000000000440000000011   0002000000440040000011   0002000000000040000000
00000000012   0000000000500000000012   0002000000500040000012   0002000000000040000000
00000000013   0000000000540000000013   0002000000540040000013   0002000000000040000000
00000000014   0000000000600000000014   0002000000600040000014   0002000000000040000000
00000000015   0000000000640000000015   0002000000640040000015   0002000000000040000000
00000000016   0000000000700000000016   0002000000700040000016   0002000000000040000000
00000000017   0000000000740000000017   0002000000740040000017   0002000000000040000000
```

You can click on Differences Only to display only the addresses that have differing expected and actual values. You can click on Clear Report to clear the current report.

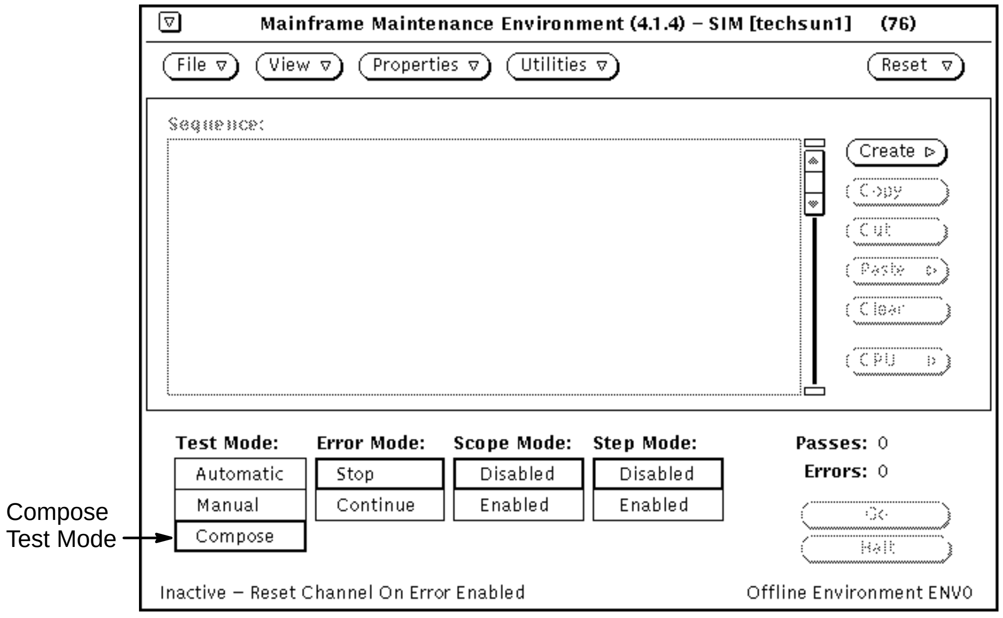
Using Compose Mode

This subsection gives step-by-step procedures that describe how to run test sequences in compose mode.

Test Sequences Using Compose Mode

When you select compose mode (refer to Figure 3-4) in environment 0, you can create customized sequences for testing. Two types of functions can be used to create a test sequence: a maintenance channel function or a pseudofunction called "compare." Refer to the `MME Function` window in Figure 3-5 or the `MME Compare` window in Figure 3-6.

NOTE: You can use compose mode to determine which function failed in a test sequence when an error occurs in automatic or manual mode. To do this, click on `Error Mode:` Stop, which stops test execution when an error occurs. Then, after an error occurs, click on `Test Mode:` Compose. The interface for compose mode is displayed, and the failing function in the test sequence is highlighted in a box.


Mainframe Maintenance Environment (4.1.4) – SIM [techsun1] (76)

Test Mode: Compose
Error Mode: Stop
Scope Mode: Disabled
Step Mode: Disabled
Passes: 0
Errors: 0

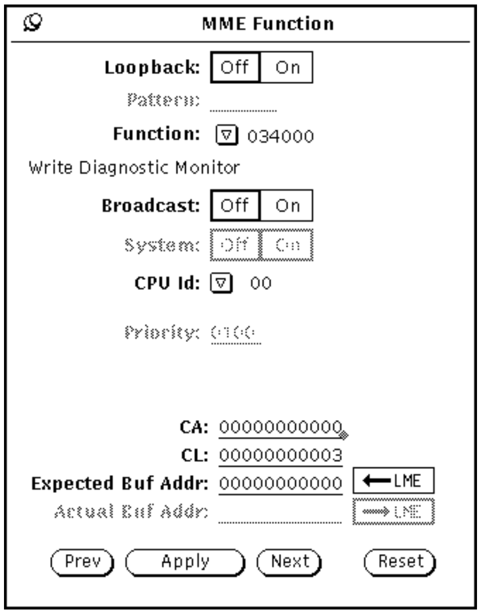
Inactive – Reset Channel On Error Enabled
Offline Environment ENV0

Figure 3-4. Compose Test Mode in Environment 0

From the `MME Function` window, you can perform the following tasks:

- Set software switches for CPU and memory control
- Control CPUs, I/O, and registers

- Control the mainframe memory and the maintenance channel
- Use the diagnostic monitor



MME Function

Loopback: [✓ Off] On

Pattern:

Function: ▽ 034000

Write Diagnostic Monitor

Broadcast: [✓ Off] On

System: Off On

CPU Id: ▽ 00

Priority: 0100

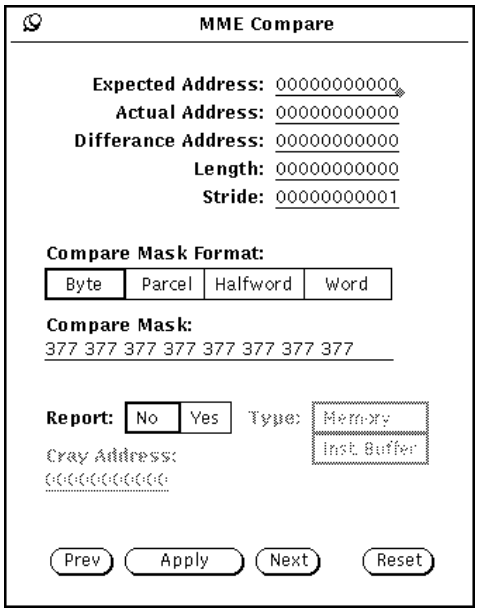
CA: 00000000000

CL: 00000000003

Expected Buf Addr: 00000000000 ⟵LME

Actual Buf Addr: ⟶LME

Figure 3-5. MME Function Window



MME Compare

Expected Address: 00000000000

Actual Address: 00000000000

Differance Address: 00000000000

Length: 00000000000

Stride: 00000000001

Compare Mask Format:

[✓ Byte] Parcel Halfword Word

Compare Mask:

377 377 377 377 377 377 377 377

Report: [✓ No] Yes Type: Memory Inst. Buffer

Cray Address:

0000000000

Figure 3-6. MME Compare Window

Maintenance Channel Functions in Compose Mode

Maintenance channel functions are divided into four categories: switch functions, control functions, memory access functions, and hardware diagnostic monitor functions. For more information about maintenance channel functions, refer to the *CRAY C90 Series Hardware Maintenance Manual*, CMM-0502-0A0.

Switch Functions

The following maintenance channel functions are available from the `Switch Functions` menu:

- Set the initial CPU software switches to hardware switches
- Set the initial cable enables to hardware switches
- Clear the control cable 0 and enable software switches
- Set the control cable 0 and enable software switches
- Clear the control cable 1 and enable software switches
- Set the control cable 1 and enable software switches
- Clear half-memory mode
- Set the half-memory size (lower)
- Set the half-memory size (upper)
- Select the master CPU
- Disable the input/output (I/O) error-correction code (ECC)
- Enable the I/O ECC
- Set the CPU test mode off
- Set the CPU test mode on
- Read the system status
- Read the CPU status
- Kill the read CPU operation or kill either system status or the loop-back test

Control Functions

The following maintenance channel functions are found on the `Control Functions` menu:

- Clear the CPU maintenance channel
- Clear the CPU maintenance channel and clear the idle CPU
- Set the CPU maintenance channel
- Set the CPU maintenance channel and set the idle CPU
- Set the CPU maintenance channel, set the idle CPU, and exchange to another diagnostic program
- Clear the I/O maintenance channel
- Set the I/O maintenance channel
- Allow full shared registers and I/O access
- Set the cluster number equal to the maximum and allow only I/O operations for this CPU
- Set the cluster number equal to the maximum and allow no I/O operations
- Allow no shared registers or I/O operations

Memory Access Functions

The following maintenance channel functions are found on the `Memory Functions` menu:

- Clear the I/O maintenance channel, set the memory priority to zero, and hold processing
- Set the I/O maintenance channel, set the memory priority to zero, and hold processing
- Release the memory priority and hold processing
- Allow advanced memory priority
- Set the highest memory priority and hold processing
- Clear the 256-Kword memory mode

- Set the 256-Kword mode CPU and the maintenance channel
- Set the 256-Kword mode CPU, I/O, and the maintenance channel
- Write to memory
- Read from memory
- Kill the read memory operation

Hardware Diagnostic Monitor Functions

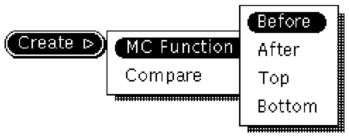
The following maintenance channel functions are found on the `Diag Monitor Functions` menu:

- Write to the diagnostic monitor
- Read from the diagnostic monitor
- Kill the read operation from the diagnostic monitor
- Reset the diagnostic monitor
- Reset the diagnostic monitor time stamp
- Stop the diagnostic monitor
- Stop the diagnostic monitor recording
- Stop the diagnostic monitor and hold the next instruction in the current instruction parcel register
- Reset the diagnostic monitor trigger, activate the diagnostic monitor, and release the next instruction from the current instruction register

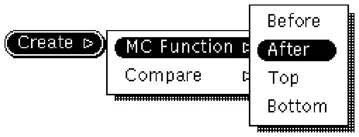
Creating Maintenance Channel Function Sequences

The recommended use of compose mode is modifying existing sequences rather than creating new sequences. To modify a sequence, start one sequence in automatic or manual mode, halt the sequence, and switch to compose mode. The sequence of functions is shown in the `Sequence` scroll box. You can modify a function by clicking on it to display a `MME Function` window, or you can add a function by performing the following procedure to create a new function in the sequence:

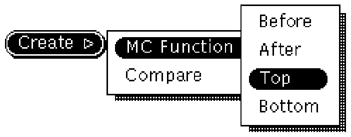
1. Choose one of the following items:



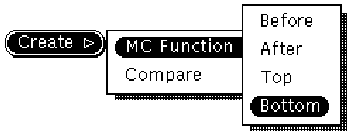
- **`Create --> MC Function --> Before`**, as shown at the left, to place the maintenance channel function sequence before the currently selected function.



- **`Create --> MC Function --> After`**, as shown at the left, to place the maintenance channel function after the currently selected function.

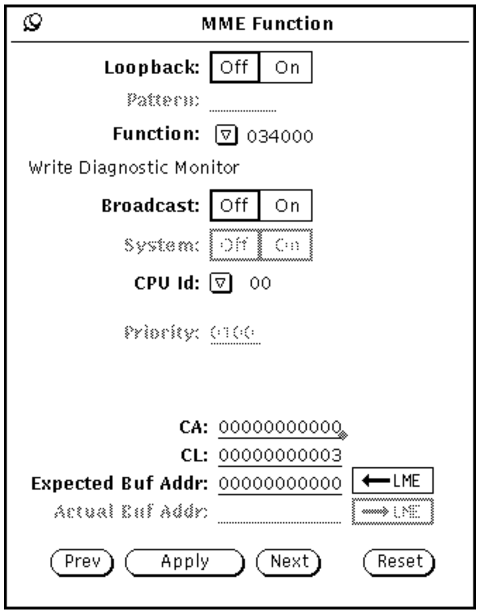


- **`Create --> MC Function --> Top`**, as shown at the left, to place the maintenance channel function first in the sequence.



- **`Create --> MC Function --> Bottom`**, as shown at the left, to place the maintenance channel function last in the sequence.

MME displays the `MME Function` window:



MME Function

Loopback: Off On

Pattern:

Function: ▽ 034000

Write Diagnostic Monitor

Broadcast: Off On

System: Off On

CPU Id: ▽ 00

Priority: 0100

CA: 00000000000

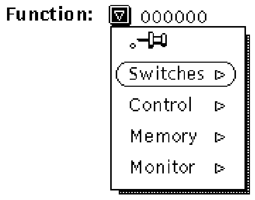
CL: 00000000003

Expected Buf Addr: 00000000000 ⟵LME

Actual Buf Addr: ⟶LME

2. Click on `Loopback` Off or On.

NOTE: If you clicked on `Loopback` On, double click on the `Pattern` field. Type the pattern you want to use. Go to Step 5. If you clicked on `Loopback` Off, continue with Step 3.



3. Choose a different maintenance function, if desired, from the options available in the `Function:` ▽, as shown at the left. The default maintenance function is `000000` (`Init CPU Soft Switches to Hard Switches`).

NOTE: If you choose one or more items you do not want, click on Cut to delete the selected item in the `Sequence` list or click on Clear to delete all the items in the `Sequence` list.

4. Set the `Broadcast` or `System` bits according to the maintenance channel functions you want to use.

   - If you clicked on `Broadcast` On or Off and `System` Off, go to Step 5.

   - If you clicked on `Broadcast` Off and `System` On, go to Step 9.

5. To change the CPU ID for all applicable functions in a sequence, click on the CPU you want.

6. Perform one of the following steps:

   - If the maintenance channel function is not a memory read, memory write, diagnostic monitor read, or diagnostic monitor write, go to Step 7.

   - If the maintenance channel function is memory read, memory write, diagnostic monitor read, or diagnostic monitor write, set the `CA` (current address) field and `CL` (current limit) fields accordingly. Go to Step 7.

7. Perform one of the following steps:

   - If this is not a memory or diagnostic monitor write, go to Step 8.

   - On memory or diagnostic monitor writes, set the `Expected Buf Addr` (expected buffer address) field to the starting MME buffer address of the data to be written.

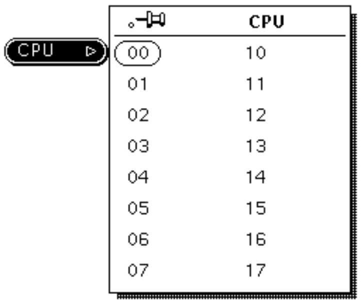
     If the maintenance channel function is a diagnostic monitor write function, you may click on ←LME to use the diagnostic monitor parameters that are set up for this CPU in LME. This option enables you to use LME to create the parameter sets. Go to Step 8.

8. Perform one of the two following steps:

   - If this is not a memory or diagnostic monitor read, go to Step 9.

   - On memory reads or diagnostic monitor reads, set the `Actual Buf Addr` (actual buffer address) field to the starting MME buffer address where the read data is to be written.

     If the maintenance channel function is a diagnostic monitor read function, you may click on →LME to send a copy of the diagnostic monitor data to LME. This option enables you to use LME to examine the diagnostic monitor data. Go to Step 9.



9. Choose **`CPU --> XX`**, where ***XX*** is the CPU number in octal, to specify the CPUs you want.

10. Click on one of the following buttons from the maintenance channel `Function` window.

    - Prev backs up (moves) from the currently selected function (enclosed in a box) to the previous function in the `Sequence` list of the `Mainframe Maintenance Environment` window.

    - Apply updates MME with the changes you entered on the maintenance channel `Function` window.

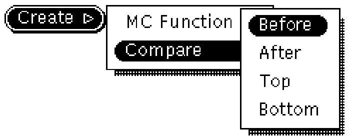
    - Next advances from the currently selected function (enclosed in a box) to the next function in the `Sequence` list of the `Mainframe Maintenance Environment` window.

    - Reset resets the settings of the options shown on the maintenance channel `Function` window to the original settings for the currently selected function.

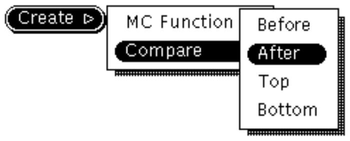
Running Comparison Functions in Compose Mode

To use the `MME Compare` window, perform the following procedure:

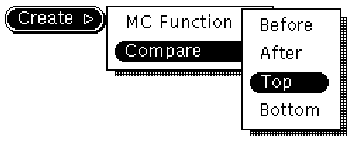
1. Choose one of the following items:



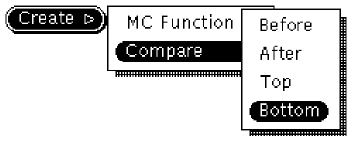
- **`Create --> Compare --> Before`**, as shown at the left, to place the compare sequence before the currently selected function. In this example, no function is currently created. However, when you create a sequence, MME shows that currently selected function enclosed in a box.



- **`Create --> Compare --> After`**, as shown at the left, to place the compare sequence after the currently selected function. In this example, no function is currently created. However, when you create a sequence, MME shows that currently selected function enclosed in a box.

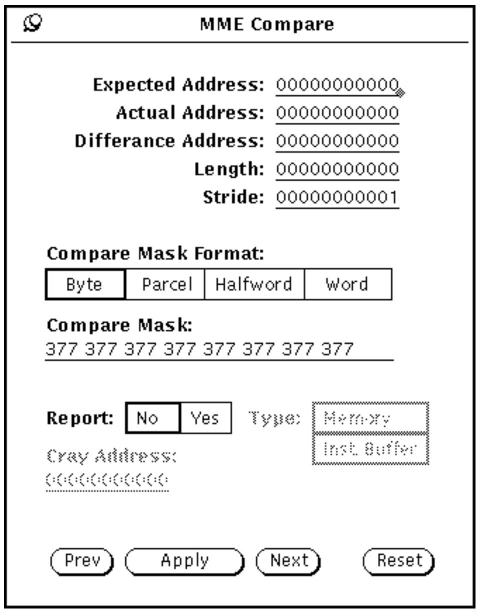


- **`Create --> Compare --> Top`**, as shown at the left, to place the compare function first in the sequence.



- **`Create --> Compare --> Bottom`**, as shown at the left, to place the compare function last in the sequence.

MME displays the `MME Compare` window:



MME Compare

Expected Address: 00000000000
Actual Address: 00000000000
Differance Address: 00000000000
Length: 00000000000
Stride: 00000000001

Compare Mask Format: [✓ Byte] Parcel Halfword Word

Compare Mask:
377 377 377 377 377 377 377 377

Report: [✓ No] Yes Type: Memory Inst. Buffer
Cray Address:
00000000000

2. Double click on the `Expected Address` field, and type the address you want.

3. Double click on the `Actual Address` field. Type the address you want.

4. Double click on the `Difference Address` field. Type the difference between the actual and the expected addresses.

5. Double click on the `Length` field. Type the address you want.

6. Double click on the `Stride` field. Type the number of address registers you want to increment within the specified length.

   NOTE: If you do not want to enter the value you typed, press and release the Esc key to reset the value to the value displayed before the edit.

7. Click on Byte, Parcel, Halfword, or Word.

8. Double click on the `Compare Mask` field. Type the mask you want. The system-supplied default mask in the `Compare Mask` field compares the address fields. If you type all zeros in the `Compare Mask Format` field, the values are not compared.

9. Perform one of following actions:

   - Click on `Report` No to prevent the results from being shown in the `Report` window.

   - Click on `Report` Yes to display the results in the `Report` window. Click on Memory to compare mainframe memory or click on Inst Buffer to compare instruction buffers that were dumped through the hardware diagnostic monitor.

10. If you clicked on the memory report format, click on the `Cray Address` field and type the mainframe address the `Actual Buf` was read from.

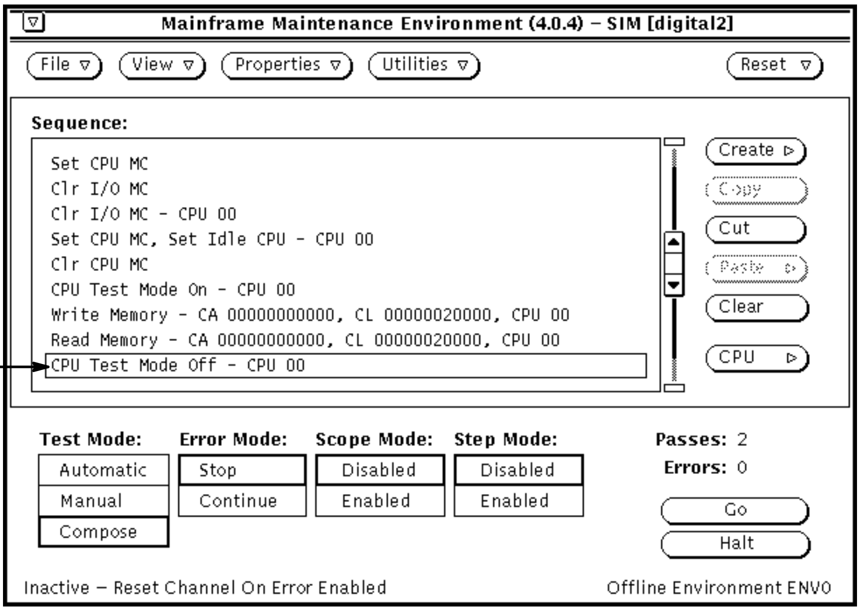
11. Use the appropriate Prev, Apply, Next, or Reset button for the `Maintenance Channel Function` menu as described under the “Maintenance Channel Functions in Compare Mode” subsection earlier in this section.

Modifying a Test Sequence

To modify an existing environment 0 test, perform the following procedure:

1. Load and run a test in automatic or manual mode.
2. Click on `Halt` to stop the current test.
3. In the MME base window, click on `Test Mode:` `Compose`.

   The instruction that was running when the test stopped is highlighted in the `Sequence` scroll box of the `Mainframe Maintenance Environment` base window, as shown in Figure 3-7.



Sequence:

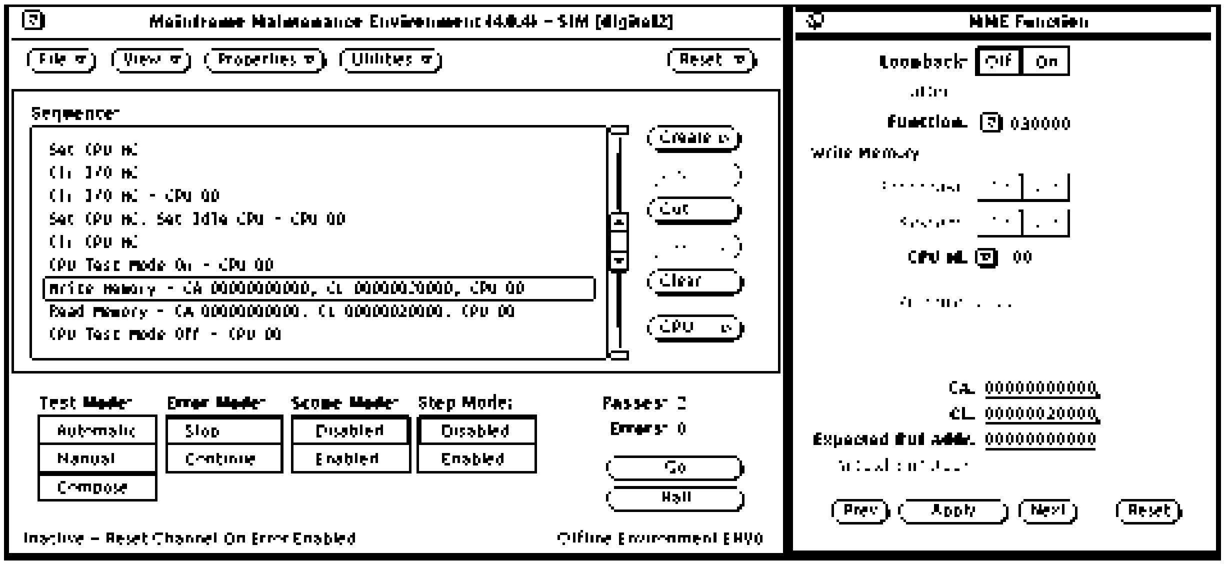
```
Set CPU MC
Clr I/O MC
Clr I/O MC - CPU 00
Set CPU MC, Set Idle CPU - CPU 00
Clr CPU MC
CPU Test Mode On - CPU 00
Write Memory - CA 00000000000, CL 00000020000, CPU 00
Read Memory - CA 00000000000, CL 00000020000, CPU 00
CPU Test Mode Off - CPU 00
```

Test Mode: Automatic / Manual / Compose
Error Mode: Stop / Continue
Scope Mode: Disabled / Enabled
Step Mode: Disabled / Enabled
Passes: 2
Errors: 0

Inactive – Reset Channel On Error Enabled
Offline Environment ENV0

Figure 3-7. Viewing the Original Sequence

4. In the `Sequence` scroll box, click on the function you want to modify. The `MME Function` window appears to the right of the `Mainframe Maintenance Environment (MME)` base window.



Sequence:

```
Set CPU MC
Clr I/O MC
Clr I/O MC - CPU 00
Set CPU MC, Set Idle CPU - CPU 00
Clr CPU MC
CPU Test Mode On - CPU 00
Write Memory - CA 0000000000, CL 0000020000, CPU 00
Read Memory - CA 0000000000, CL 0000020000, CPU 00
CPU Test Mode Off - CPU 00
```

Test Mode: Automatic | Manual | [✓ Compose]
Error Mode: [✓ Stop] | Continue
Scope Mode: [✓ Disabled] | Enabled
Step Mode: [✓ Disabled] | Enabled

Passes: 2
Errors: 0

Inactive – Reset Channel On Error Enabled
Offline Environment ENV0

Loopback: [✓ Off] On
Function: 030000
Write Memory
CPU NL: 00
CA: 0000000000
CL: 0000020000
Expected Ful Addr: 0000000000

NOTE: If the instruction you chose to run was a compare instruction, an `MME Compare` window will appear instead of an `MME Function` window. Refer to pages 3-55 through 3-59 for descriptions of both the `MME Function` window and the `MME Compare` window.

5. In the `MME Function` window or the `MME Compare` window, double click on the field you want to change.

6. Enter the new field value and press the return (↵) key.

7. Click on (Apply). The instruction in the `MME` base window changes when you click on (Apply).

To change the instruction type of an existing function, perform the following procedure:

8. In the `MME Function` window, select the function to which you want to change from `Function:` ▽, as shown in Figure 3-8.

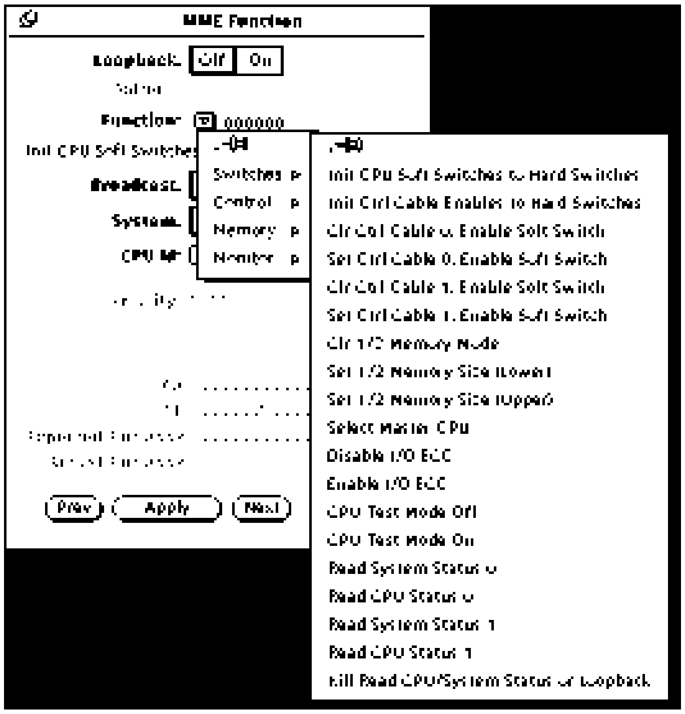
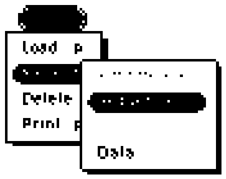

MME Function
Loopback
Off
On
Function
000000
Init CPU Soft Switches
Broadcast
System
CPU Nr
Switches
Control
Memory
Monitor
Init CPU Soft Switches to Hard Switches
Init Ctrl Cable Enables to Hard Switches
Clr Ctrl Cable 0. Enable Soft Switch
Set Ctrl Cable 0. Enable Soft Switch
Clr Ctrl Cable 1. Enable Soft Switch
Set Ctrl Cable 1. Enable Soft Switch
Clr 1/2 Memory Mode
Set 1/2 Memory Size (Lower)
Set 1/2 Memory Size (Upper)
Select Master CPU
Disable I/O ECC
Enable I/O ECC
CPU Test Mode Off
CPU Test Mode On
Read System Status 0
Read CPU Status 0
Read System Status 1
Read CPU Status 1
Kill Read CPU/System Status or Loopback
Prev
Apply
Next

Figure 3-8. Changing the Instruction

If you want to rerun a sequence in the future, you must save the sequence and the data used in the sequence. To save the sequence, perform the following procedure:



9. Choose **`File --> Save --> Sequence`**, as shown at the left.

10. In the `MME Save Sequence` window, enter a directory in the `Dir` field. In the `File` field, enter a name for the file, as shown in Figure 3-9.

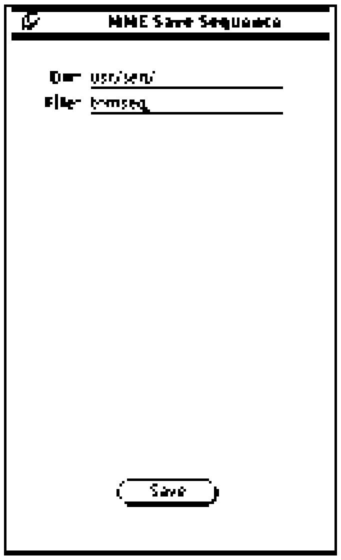
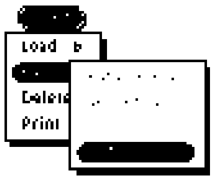


Figure 3-9. Saving a Sequence

11. Click on Save.

In order to use the data later, you must save the data. To save data, perform the following procedure:



12. Choose **`File --> Save --> Data`**, as shown at the left.

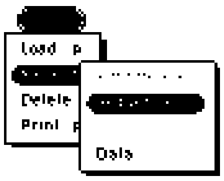
13. Double click on the `File` field, and enter a name for the file.

14. Double click on the `Start Address` field, and enter a starting buffer memory address.

15. Double click on the `Length` field, and enter a length.

NOTE: When you enter data in two of the fields (`Start Address`, `Length`, or `End Address`), MME automatically updates the third field.

To run the modified sequence, perform the following procedure:

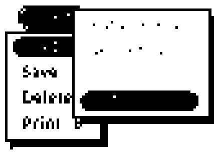


16. Choose **`File --> Load --> Sequence`**, as shown at the left.

17. From the `MME Load Sequence` window, click on the file you want to load.

18. Click on Load.

To load data, perform the following procedure:



19. Select **`File --> Load --> Data`**, as shown at the left.

20. In the `Files:` scroll box in the `MME Load Data to Buffer` window, click on the file you want to load.

NOTE: You can change the starting address and length of the data block by entering new values in the corresponding fields found in the `MME Load Data to Buffer` window. Press the return (↵) key.

21. Click on Load .

22. Click on Go .

4 MME ENVIRONMENTS 1 AND 2

Environments 1 and 2 are similar in operation. The point of testing is moved from the MWS-E to the mainframe. Control points, a generic name used to reference diagnostic programs, utilities, or loops, are loaded into mainframe memory. Memory allocation and control point configuration are controlled by MME through user settings and information available from the maintenance channel. CPUs are assigned to the control points, and MME provides users control for starting and stopping the CPU execution of control points. Various displays show information you can use to analyze hardware failures.

MME environments 1 and 2 can be run in two modes: concurrent and offline. Concurrent mode enables troubleshooting while the OS is running in some CPUs. Offline mode provides full access to the mainframe but requires that the OS is stopped in all CPUs.

In concurrent mode, the OS is running and has control of the mainframe. MME runs in restricted mode, using the upper 256 Kwords of memory and the CPU(s) you specify as usable in the MCE `Soft Switches` window. These CPUs must be downed by the OS before you run diagnostics and utilities with MME. Concurrent mode is useful for troubleshooting bad CPU(s) while the OS runs in the other CPUs.

CAUTION

Do not configure a usable CPU in 256K mode. UNICOS will crash when you apply the configuration. Currently, UNICOS does not support concurrent maintenance.

In offline mode, you will need to stop the OS in the mainframe or the OS will crash. Once you have stopped the OS, MME has total control of the mainframe and has access to all of memory. This enables you to perform extensive troubleshooting of the mainframe.

Environments 1 and 2 start in concurrent mode by default to ensure that you do not accidentally crash the OS in the mainframe. You can force MME environments 1 and 2 into offline mode with the `–offline` option; refer to the descriptions of how to start MME environments 1 and 2 later in this section for more information. You can also use MCE to switch from concurrent to offline mode; refer to the description of MCE in Section 2 for more information.

Environment 1 enables one diagnostic and/or one or more loops to be loaded into memory. All diagnostics are loaded into memory address 0; loops are loaded at an origin. Generally, because only one control point resides in the mainframe at a given time, it has access to all the resources such as memory, I/O channels, and shared registers.

Environment 1 replaces the MM and MI monitors used in the previous CRAY Y-MP computer systems. Because the maintenance channel functions are available, such as individual CPU control and direct memory access, monitors are no longer required to perform these functions. In environment 1, all control points (except loops) contain a segment of code referred to as the interrupt router. The interrupt router provides one method of handling interrupts among all the control points.

Environment 2 enables one or more control points to be loaded into memory. For example, you can load multiple copies of the same diagnostic or utility, or you can load different diagnostics and utilities. Several memory-allocation schemes are available that cause the control points to be loaded starting from the lowest memory location (bottom up), starting from the highest memory location (top down), randomly, or equally (partitioned).

A small code segment referred to as the controller resides in the lower 040000_8 words of memory. Because there may be more than one control point in memory, the controller negotiates the sharing of resources such as I/O channels and shared registers. Environment 2 controls run system operation. The run system automatically rotates the CPUs among various control points. Environment 2 replaces the M8 and Run System monitors used in the previous CRAY Y-MP computer systems.

This section provides information about starting MME environments 1 and 2, starting MME environments 1 and 2 with the simulator, the MME environments 1 and 2 interfaces, and performing tasks in MME environments 1 and 2.

Starting MME Environment 1 or Environment 2

CAUTION

Do not start MME in offline mode when UNICOS is running. Several functions that MME performs in offline mode will crash UNICOS if it is running.

MME environments 1 and 2 can be started from the OpenWindows workspace menu or from a UNIX command prompt.

For information about how to start MME from a service center through a hub, refer to the appendix of this manual.

From the OpenWindows Desktop Workspace Menu

To start MME environments 1 or 2 from the OpenWindows desktop workspace menu, perform one of the following procedures.

Environment 1

To start MME environment 1 from the OpenWindows workspace menu, perform one of the following actions:

- Choose **`Maintenance Tools --> MME --> MME env 1 --> Copy#`** to start MME environment 1 with the copy number specified by the `Copy#` selection.

  The copy number option enables you to differentiate between multiple independent MME sessions that are supported from the same MWS-E. Copy numbers 0, 1, 2, and 3 are available from the workspace menu. The copy number does not affect performance; it serves as an identifier only.

- Choose **`Maintenance Tools Simulator --> MME --> MME env 1 --> Simulator`** to start MME environment 1 with the simulator.

- Choose **`Maintenance Tools (simulated) --> MME --> MME env 1 --> Simulator with Debugger`** to start MME environment 1 with the simulator and debugger.

Environment 2

To start MME environment 2 from the OpenWindows workspace menu, perform one of the following actions:

- Choose **`Maintenance Tools --> MME --> MME env 2 --> Copy#`** to start MME environment 2 with the copy number specified by the `Copy#` selection.

  The copy number option enables you to differentiate between multiple independent MME sessions that are supported from the same MWS-E. Copy numbers 0, 1, 2, and 3 are available from the workspace menu. The copy number does not affect performance; it serves as an identifier only.

- Choose **`Maintenance Tools --> MME Simulator --> MME env 2 --> Simulator`** to start MME environment 2 with the simulator.

- Choose **`Maintenance Tools --> MME Simulator --> MME env 2  --> Simulator with Debugger`** to start MME environment 2 with the simulator and debugger.

From a UNIX Command Prompt

To start MME environment 1 or 2 from a UNIX prompt, type **`mme -1`** or **`mme`** (environment 2 is the default) followed by any appropriate command line options, and press the return (↵) key. Parameters enclosed in brackets [ ] are optional, parameters enclosed in angle brackets < > are required, and a vertical bar | indicates an either/or choice. Table 4-1 describes the available command line options.

```
mme [–1 | -2]
    [–copy <num>]
    [–remote <host> | –client | –server]
    [–kill] [–io<num>]
    [–config <file>] [–l <file>]
    [–concurrent | –offline]
    [–chn<num> | -sim | -debug]
```

Table 4-1. Command Line Options

Option	Description
`–chn<`*num*`>`	Use FEI channel specified by <*num*>, which can range from 0 to 7. The default channel number is 1.
`–client`	Start the client only
`–concurrent`	Use concurrent mode
`–config <`*file*`>`	Configure MCE with the configuration data stored in the file specified by <*file*>
`–copy <`*num*`>`	Connect to maintenance software assigned to the copy number specified by <*num*>. This option allows you to differentiate which system is being supported by this session of the software.
`–debug`	Use the simulator and bugger/debugger
`–io<`*num*`>`	Use the CPU specified by <*num*>
`–kill`	Kill all MME processes
`–l <`*file*`>`	Load a layout file
`–offline`	Use offline mode
`–remote <`*host*`>`	Start client only, connect to remote host
`–server`	Start the server only
`–sim`	Use the simulator
`–1`	Start MME environment 1
`–2`	Start MME environment 2 (default)

For example, enter **`mme –1 –offline –io1 –chn2 –l myfile`** ↵ to start MME environment 1 in offline mode using CPU 1 as the I/O CPU, using FEI channel 2, and using the layout saved in `usr/myfile`. Enter **`mme –offline –io1 –sim –l myfile`**↵ to start MME environment 2 in offline mode using CPU 1 as the I/O CPU, using the simulator, and using the layout saved in `usr/myfile`.

Environments 1 and 2 Common Interface

Environments 1 and 2 share a common interface, which changes slightly for each environment. The components of the interface, shown in Figure 4-1, are described following the figure. The text describes the components as they appear in Figure 4-1, clockwise from the upper-left corner.

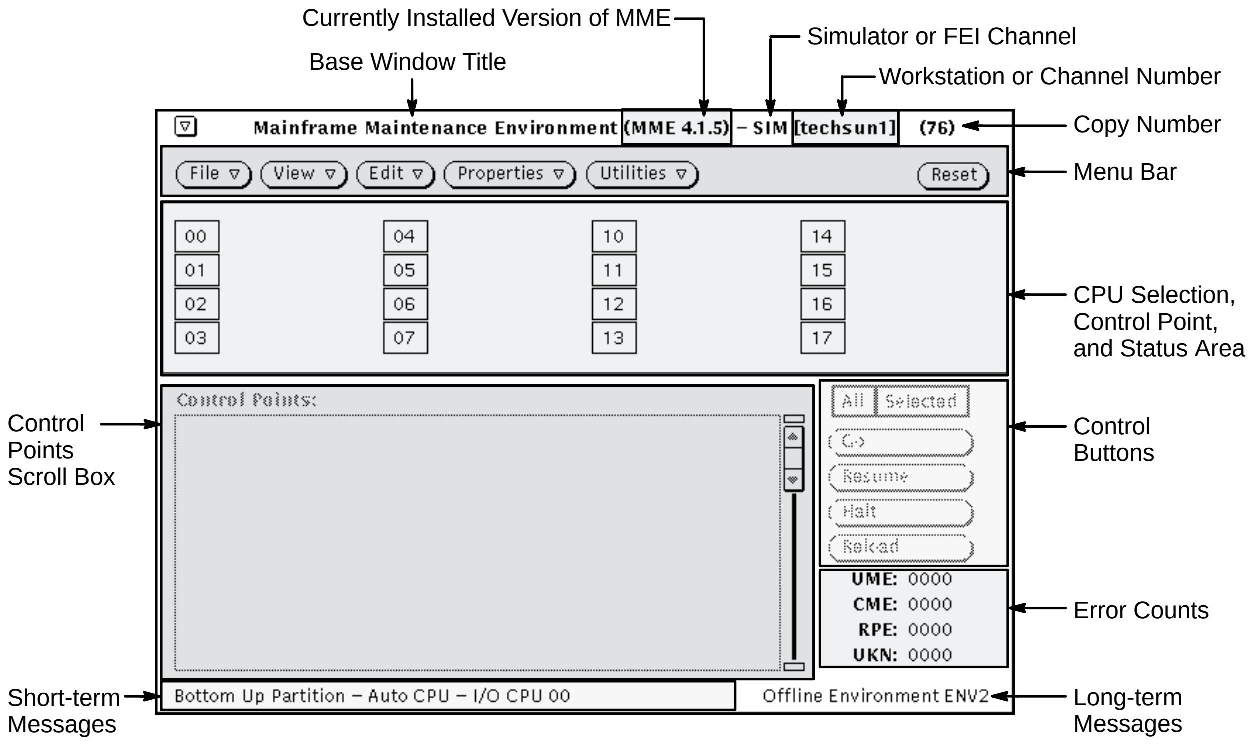


Figure 4-1. Components of the MME Environments 1 and 2 Common Interface

Base Window Title

The base window title displays the name of the program: `Mainframe Maintenance Environment`.

Currently Installed Version of MME

The currently installed version of MME is a number in parentheses that indicates which version of MME you are running.

Simulator or FEI Channel

The simulator or FEI channel indicator shows that you are running MME with the simulator (indicated by `SIM`) or an FEI channel (indicated by `FEI CHN 0` for channel 0, `FEI CHN 1` for channel 1, or `FEI CHN 2` for channel 2).

Workstation or Channel Number

The workstation or channel number indicator indicates the name of the workstation or the channel number that MME is running on.

Copy Number

The copy number identifies the copy of MME you are using. Because you may run more than one session of MME at a time from a single MWS-E, the copy number differentiates the sessions. To set the copy number, start MME with the `–copy` option. If you start MME with the default copy number of 0, the MME base window does not display a copy number. The copy number is used for identification only and will not affect performance. For more information about starting MME Environment 1 or 2 with the `–copy` option, refer to "Starting MME Environment 1 or Environment 2" earlier in this section.

Menu Bar

The menu bar contains the menu buttons for controlling many functions of MME environments 1 and 2. There are five menu buttons: (File ▽), (View ▽), (Properties ▽), (Utilities ▽), and (Reset). For descriptions of the tasks you can perform with the commands contained in these menu buttons, refer to "Performing Tasks with MME Environment 1 or 2" later in this section.

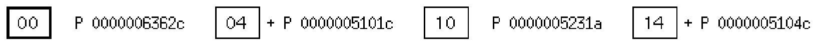
CPU Selection, Control Point, and Status Area

The CPU selection and error information area is where you assign the CPU(s) to control points and where MME displays the status of running control points. You can click on the CPUs ([00] through [17]) to assign them to the current control point.

By default, MME displays the control point name next to the CPU number:

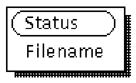
```
[00]  00 aab.c     [04] + 00 mem3.c     [10]  04 aab.c     [14] + 06 mem3.c
```

MME can show the P-register value for running control points also:



```
00   P 0000006362c   04 + P 0000005101c   10   P 0000005231a   14 + P 0000005104c
```

To toggle between the two displays, press the MENU mouse button in the status area and choose **`Status`** (for the P-register values) or **`Filename`** (for the control point filename):



The number shown to the left of the control point name indicates the number of the control point copy when you have several copies loaded (for example, `00` indicates the first copy loaded, `01` indicates the second copy loaded, and `02` indicates the third copy loaded).

A plus (+) next to the copy number indicates the master CPU for a group of CPUs assigned to one control point. The master CPU is the first CPU assigned to the control point.

NOTE: This master CPU has nothing to do with the master CPU set by the hardware and software switches.

Control point execution status information is displayed to the right of the control point name or P-register value for each CPU. This status information is either an interrupt flag or controller error. Table 4-2 identifies the interrupt flags and their meanings. For a detailed explanation of each interrupt flag, refer to Section 3 of the *CRAY Y-MP C90 System Programmer Reference Manual*, CSM-0500-000. Table 4-3 identifies the controller errors and the conditions in which they are issued.

Table 4-2. Interrupt Flags

Interrupt Flag	Meaning
`BPI`	Breakpoint interrupt
`DL`	Deadlock
`EEX`	Error exit
`FPE`	Floating-point error interrupt
`ICP`	Interprocessor interrupt

Table 4-2. Interrupt Flags (continued)

Interrupt Flag	Meaning
`IOI`	Input/output (I/O) interrupt
`MCU`	MCU interrupt
`MEC`	Memory error correctable interrupt
`MEU`	Memory error uncorrectable interrupt
`MII`	Monitor mode interrupt
`NEX`	Normal exit
`ORE`	Operand range-error interrupt
`PCI`	Programmable-clock interrupt
`PRE`	Program-range-error interrupt
`RPE`	Register-parity-error interrupt
`RTI`	Real-time interrupt

Table 4-3. Environment 2 Abbreviated Status Messages

Message	Meaning
`CIB`	The control point attempted to clear the CLN, but the CLN was not in the IBA space.
`CNR`	The control point attempted to clear a CLN that was not reserved.
`CRE`	A channel reservation error occurred.
`DBA`	The DBA in the working exchange package in the dump area was less than the DBA assigned by MME.
`DLA`	The DLA in the working exchange package in the dump area was greater than the DLA assigned by MME.
`DMP`	The CPU dumped registers and is in the controller’s idle loop.
`ERR`	The CPU did not respond to a run system to dump and idle or restart request.
`HTM`	The diagnostic has stopped itself and requested all CPUs to hang.
`HTS`	The diagnostic has stopped at the request of another CPU.

Table 4-3. Environment 2 Abbreviated Status Messages (continued)

Message	Meaning
`IBA`	The IBA in the working exchange package in the dump area was less than the IBA assigned by MME.
`ILA`	The ILA in the working exchange package in the dump area was greater than the ILA assigned by MME.
`INF`	The CPU exchanged to the controller with no interrupt flags.
`IUC`	An interrupt occurred on an unreserved channel.
`MEI`	Execution stopped on an invalid memory error.
`MEM`	The CPU exchanged on a memory error but the status register did not indicate an error, or the standard location MRSTOP was set to stop on error.
`MES`	The diagnostic stopped on a memory error (MRSTOP).
`MWS`	Bad request was sent to or from MME.
`PAR`	The CPU exchanged on a register parity error but the status register did not indicate an error, or the standard location MRSTOP was set to stop on error.
`PEI`	Execution stopped on a parity error.
`PEM`	The diagnostic stopped on a parity error (MRSTOP).
`SRE`	A cluster reservation error occurred.
`TRP`	The CPU exchanged to the trap exchange package.
`???`	No flag bit is set in the working exchange package (WEXP) or no status bits are set in the controller.

Control Buttons

The control buttons perform the following actions:

Button	Action
Go	Start testing
Halt	Stop testing
Resume	Return testing control to a control point after a user response; some control points that require user-supplied data set the sign bit of error; this button clears the sign bit, allowing the control points to resume testing.
Reload	Reload the selected (or all) control point(s); this provides a quick way to remove all your changes and/or data.

Error Counts

There are four error counts displayed on the interface. The `UME` field displays the number of uncorrectable memory errors detected. The `CME` field displays the number of correctable memory errors detected. The `RPE` field displays the number of register parity errors detected. The `UKN` field displays the number of unknown errors detected.

Long-term Messages

The long-term message area displays which environment you are working in for the current base window. The following long-term messages are displayed:

Message	Description
`Offline Environment ENV1`	Environment 1 in offline mode
`Concurrent Environment ENV1`	Environment 1 in concurrent mode
`Offline Environment ENV2`	Environment 2 in offline mode
`Concurrent Environment ENV2`	Environment 2 in concurrent mode

Short-term Messages

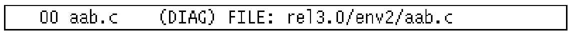
The short-term message area on the interface displays messages about the current state of the MME program. The following short-term messages are displayed:

Message	Description
`Bottom Up`	Memory allocation is bottom up.
`Top Down`	Memory allocation is top down.
`Random`	Memory allocation is random.
`Partition`	Memory is partitioned.
`Auto CPU`	CPU allocation mode is auto.
`I/O ##`	CPU ## is the I/O CPU. `I/O Disabled` is displayed if I/O is disabled.

Control Points Scroll Box

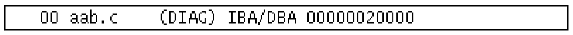
The `Control Points` scroll box contains the control points you currently have loaded in MME. Click on a control point to select it. You can assign a CPU to the selected control point by clicking on the desired CPU number (00 through 17). The control point appears next to the the CPU number.

By default, the `Control Points` scroll box displays the filename for any loaded control points:



```
  00 aab.c    (DIAG) FILE: rel3.0/env2/aab.c
```

You may want to know the location of the control points in memory. MME can display this in the `Control Points` scroll box also:



```
  00 aab.c    (DIAG) IBA/DBA 00000020000
```

You can toggle between the two displays by pressing the MENU button in the `Control Points` scroll box and choosing **`Filename`** or **`Location`** from the popup menu that appears:

The number shown to the left of the control point name indicates which copy the control point is when you have several copies loaded (for example, `00` indicates the first copy loaded, `01` indicates the second copy loaded, and `02` indicates the third copy loaded).

A plus (`+`) next to the copy number indicates the control point is a multiple-CPU control point to which more than one CPU can be assigned. The first CPU assigned to the control point is the master CPU.

NOTE: This master CPU has nothing to do with the master CPU that is set by the hardware and software switches.

The information shown in parentheses indicates that the control point is a diagnostic [when `(DIAG)` is shown], utility [when `(UTIL)` is shown], or loop [when `(LOOP)` is shown].

Performing Tasks with MME Environment 1 or 2

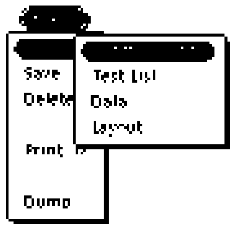
This subsection provides procedures that describe how to perform the necessary troubleshooting tasks with MME environments 1 and 2. Not all tasks can be performed in environments 1 and 2; tasks that can be performed in only one of the environments are noted as such.

Loading Control Points in Environment 1 or 2

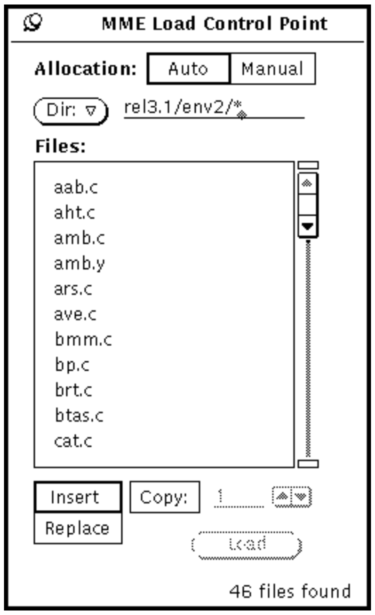
Within environment 1 or 2, you can select one or more (environment 2 only) diagnostic programs and load each one into memory as a control point to test for hardware problems. For a list of diagnostic programs or utilities, refer to Section 7, "Diagnostic Tests and Utilities," in this manual.

NOTE: Before you load a control point, you may want to allocate resources differently. To allocate resources differently, refer to "Allocating Resources in Environment 1 or 2" later in this section. Any changes you make affect only the control points that you load after making the changes.

To load a diagnostic program or utility as a control point, perform the following procedure:



1. Choose **`File --> Load --> Control Point`**, as shown at the left. MME displays the `MME Load Control Point` window:



rel3.1/env2/*

aab.c
aht.c
amb.c
amb.y
ars.c
ave.c
bmm.c
bp.c
brt.c
btas.c
cat.c

1

46 files found

This window lists the diagnostic program files you can click on to load into mainframe memory as control points. The directory path displayed to the right of the (Dir: ▽) button shows where each diagnostic program file resides. The `Files` list shows the diagnostic program files stored in the directory identified in the (Dir: ▽) field.

NOTE: A `.y` extension means the diagnostic program was assembled in Y-MP mode, and a `.c` extension means that the diagnostic program was assembled in C90 mode.

2. Change the directory, if necessary, by performing one of the following actions.

   - Choose the directory from the (Dir: ▽) button. You can choose from the following directories:

     `Release` – Either environment 1 or 2 diagnostic program files (depending on the environment you are using) from the current release

     `User` – Diagnostic program files that the user has changed and saved

     `Alpha` – Pre-release diagnostic program files that are being tested and have not been released

     `Utility --> Release` – Utility files from the current release

     `Utility --> Alpha` – Pre-release utility files that are being tested and have not been released

   - Triple click on the (Dir: ▽) field, type the name of the directory you want to use, and press the return (↵) key.

NOTE: Files for environment 1 are in the `rel/env1/*` directory, and files for environment 2 are in the `rel/env2/*` directory. In this example, `rel` indicates the files are for the current offline diagnostic release, `env1` specifies environment 1, `env2` specifies environment 2, and `*` specifies all the files.

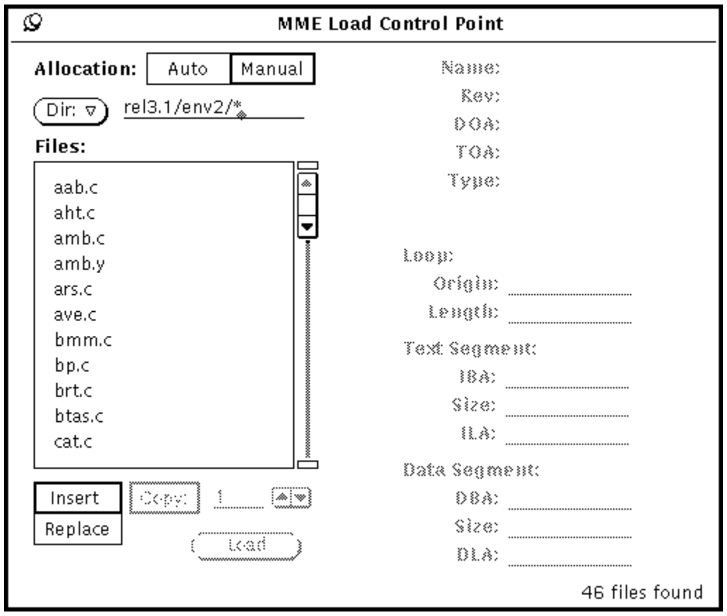
3. Click on the file you want to use.

4. Click on [Insert] to insert a new diagnostic or utility in memory as a control point or [Replace] to replace the selected control point in the `Control Points` scroll box of the MME base window with the file you have selected.

NOTE: If you want to insert more than one copy, click on Copy, and click on either ▲ or ▼ until the number of copies you want appears in the `Copy` field, or click on the `Copy` field and type the value you want.

5. Click on Auto (automatic) to enable MME to load the control point in the default area of mainframe memory [in environment 1, that area is at address 0; and in environment 2, the location of the area varies, depending on the memory mode (bottom up, top down, or random)], or click on Manual to specify where you want to load the control point in mainframe memory.

   If you click on Manual, MME displays an expanded `MME Load Control Point` window.



The expanded `MME Load Control Point` window displays the following headings, which identify the diagnostic program or utility and provide information about where it is loaded into memory as a control point:

`Name`	Name of the diagnostic program or utility you want to load as a control point
`Rev`	Revision level of the diagnostic program or utility you want to load as a control point
`DOA`	Date of assembly (creation date)
`TOA`	Time of assembly (creation time)

`Type`	Diagnostic program or utility
`Origin`	Starting address for the loop when loaded in memory
`Length`	Octal length of the loop when loaded in memory
`IBA`	Instruction base address for the text segment when loaded in memory
`Size`	Octal size of the text segment when loaded in memory
`ILA`	Instruction limit address (last address) for the text segment when loaded in memory
`DBA`	Data base address for the data segment when loaded in memory
`Size`	Octal size of the data segment when loaded in memory
`DLA`	Data limit address (last address) for the data segment when loaded in memory

6. Perform one of the following steps:

   - If you clicked on Auto, go to Step 11.
   - If you clicked on Manual and this is a loop control point, go to Step 7.
   - If you clicked on Manual and you want to change where the text segment is loaded, go to Step 9.
   - If you clicked on Manual and you want to change where the data segment is loaded, go to Step 10.

7. Double click on the `Origin` field. Type the origin location.

8. Double click on the `Length` field. Type the length of the control point. Go to Step 11.

9. After you perform any two of the three following steps, MME updates the third field. Go to Step 11.

   - Double click on the `IBA` (instruction base address) field. Type the octal address you want to use.

- Double click on the `Size` field. Type the octal address you want to use.

- Double click on the `ILA` (instruction length address) field. Type the instruction length address in octal.

10. After you perform any two of the three following steps, MME updates the third field. Go to Step 11:

    - Double click on the `DBA` (data base address) field. Type the octal data base address you want to use.

    - Double click on the `Size` field. Type the size in octal you want to use.

    - Double click on the `DLA` (data base length address) field. Type the data base length address in octal you want to use.

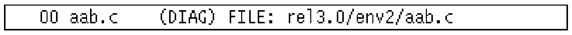
11. Click on Load ; the control point appears in the list of `Control Points` in the `Mainframe Maintenance Environment` window. The control point you loaded becomes the current control point.

    When you click on Load , the copies of the control point you specified appear in the `Control Points` list of the `Mainframe Maintenance Environment` window. If you specified more than 9 copies, only the first 9 copies are shown in the `Control Points` list of the `Mainframe Maintenance Environment` window. You can scroll the list to see the other control points.

12. Click on a CPU. MME loads the specified control point.

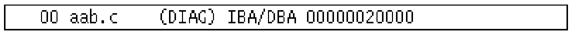
13. Click on Go to start running your control point.

    After you have loaded a control point, the filename and path are shown in the `Control Points` scroll box of the `Mainframe Maintenance Environment` window:



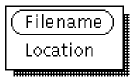
```
00 aab.c    (DIAG) FILE: rel3.0/env2/aab.c
```

If you want to know where the control points are in memory, MME can also display the location of the control point in memory:



```
00 aab.c    (DIAG) IBA/DBA 0000020000
```

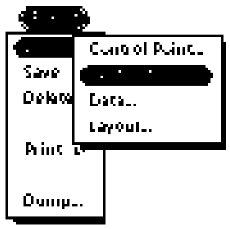
You can toggle between the two displays by pressing the MENU button in the `Control Points` scroll box and choosing **`Filename`** or **`Location`** from the pop-up menu that appears:



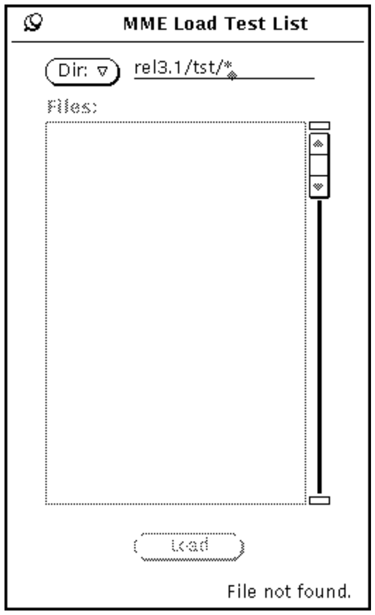
Loading a Test List in Environment 1 or 2

Within environment 1 or 2, you can load a test list that contains control points used to test for hardware problems.

To load a test list you have previously saved (refer to "Saving a Test List in Environment 1 or 2" later in this section), perform the following procedure:



1. Choose **`File --> Load --> Test List`**, as shown at the left. MME displays the `MME Load Test List` window:



2. Change the directory, if necessary, by performing one of the following actions:

   - Choose the directory from the (Dir: ▽) button. You can choose from the following directories:

     `Release` – Test lists of environment 1 or 2 diagnostic program files from the current release

     `User` – Test lists that the user has changed and saved

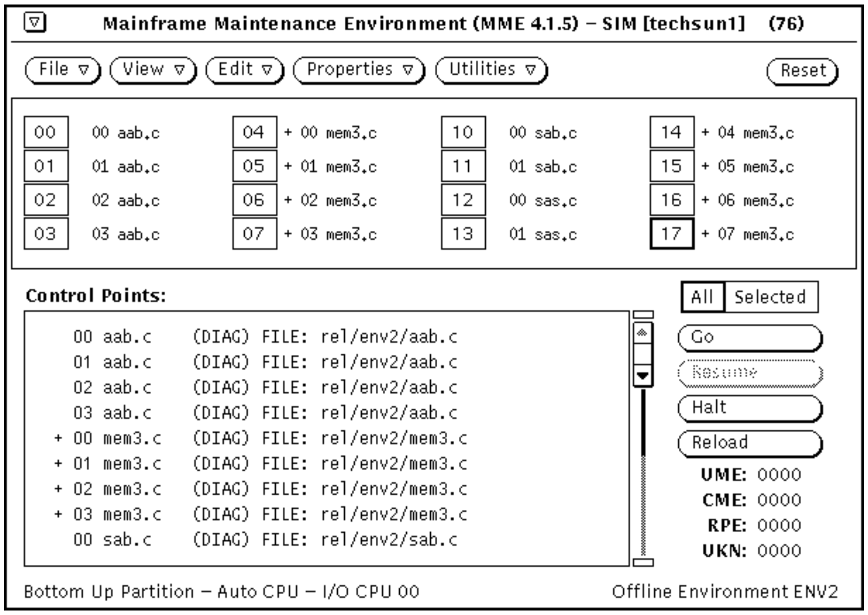
     `Alpha` – Pre-release test lists that are being tested and have not been released

   - Triple click on the (Dir: ▽) field, type the name of the directory you want to use, and press the return (↵) key.

3. Click on a test list file.

4. Click on (Load). MME loads the test list.

When you load a test list, the control points appear in the `Control Point` scroll box, and the specified CPUs are assigned to the control points:



```
00  00 aab.c      04  + 00 mem3.c    10  00 sab.c    14  + 04 mem3.c
01  01 aab.c      05  + 01 mem3.c    11  01 sab.c    15  + 05 mem3.c
02  02 aab.c      06  + 02 mem3.c    12  00 sas.c    16  + 06 mem3.c
03  03 aab.c      07  + 03 mem3.c    13  01 sas.c    17  + 07 mem3.c
```

Control Points:

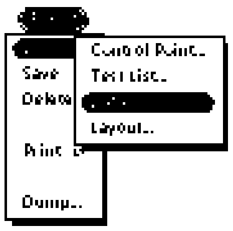
```
   00 aab.c    (DIAG) FILE: rel/env2/aab.c
   01 aab.c    (DIAG) FILE: rel/env2/aab.c
   02 aab.c    (DIAG) FILE: rel/env2/aab.c
   03 aab.c    (DIAG) FILE: rel/env2/aab.c
 + 00 mem3.c   (DIAG) FILE: rel/env2/mem3.c
 + 01 mem3.c   (DIAG) FILE: rel/env2/mem3.c
 + 02 mem3.c   (DIAG) FILE: rel/env2/mem3.c
 + 03 mem3.c   (DIAG) FILE: rel/env2/mem3.c
   00 sab.c    (DIAG) FILE: rel/env2/sab.c
```

UME: 0000
CME: 0000
RPE: 0000
UKN: 0000

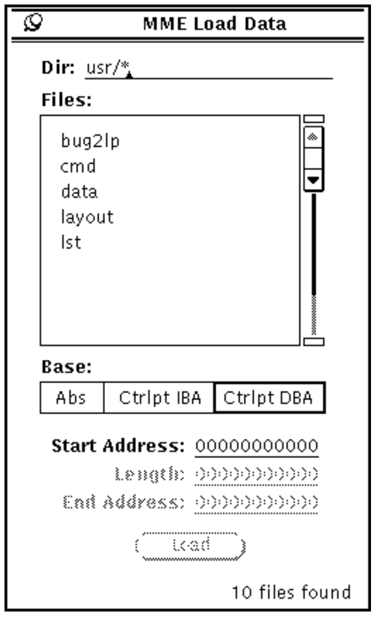
Bottom Up Partition – Auto CPU – I/O CPU 00 Offline Environment ENV2

Loading Data in Environment 1 or 2

To load data you have saved (refer to “Saving Data in Environment 1 or 2” later in this section), perform the following procedure:



1. Choose **`File --> Load --> Data`**, as shown at the left. MME displays the `MME Load Data` window:



MME Load Data

Dir: usr/*

Files:

bug2lp
cmd
data
layout
lst

Base: Abs | Ctrlpt IBA | Ctrlpt DBA

Start Address: 00000000000
Length: 00000000000
End Address: 00000000000

Load

10 files found

2. Change the directory, if necessary, by triple clicking on the `Dir` field, typing the name of the directory you want, and pressing the return (↵) key.

3. Click on the file you want to load.

4. Specify the memory `Base` by clicking on Abs (absolute) to choose a fixed location in mainframe memory, Ctrlpt IBA (control point instruction base address) to choose a relative control point location within memory, or Ctrlpt DBA (control point data base address) to choose a relative control point location within memory.

5. Enter the starting address, length, and ending address:

   NOTE: After you perform any two of the three following steps, MME updates the third field.

   - Double click on the source data `Start` field where MME will start copying or moving the data. Type the starting address of the source data you want to copy or move, and press the return (↵) key.

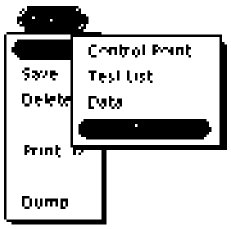
   - Double click on the `Length` field. Type the block length of the source data you want to copy or move, and press the return (↵) key.

   - Double click on the `End` field where MME will start copying or moving the data. Type the ending address of the source data you want to copy or move, and press the return (↵) key.

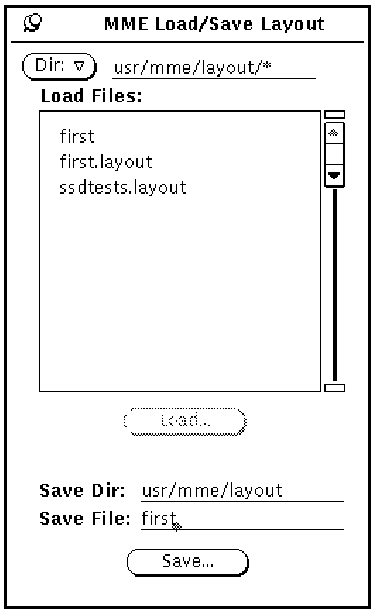
6. Click on [Load]. MME loads the data into memory.

Loading a Screen Layout in Environment 1 or 2

After you have saved a screen layout containing the data in one or more windows (refer to "Saving a Screen Layout in Environment 1 or 2"), you can load a screen layout to display those windows again. To load a layout, perform the following procedure:



1. Choose **`File --> Load --> Layout`**, as shown at the left. MME displays the `MME Load/Save Layout` window:



2. Change the directory, if necessary, by performing one of the following actions:

   - Choose the directory from the (Dir: ▽) button. You can choose from the following directories:

     `Release` – Layout files from the current release

     `User` – Layout files that the user has changed and saved

     `Alpha` – Pre-release layout files that are being tested and have not been released

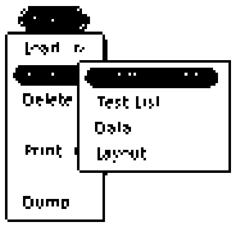
   - Triple click on the (Dir: ▽) field, type the name of the directory you want to use, and press the return (↵) key.

3. Click on the file you want to load.

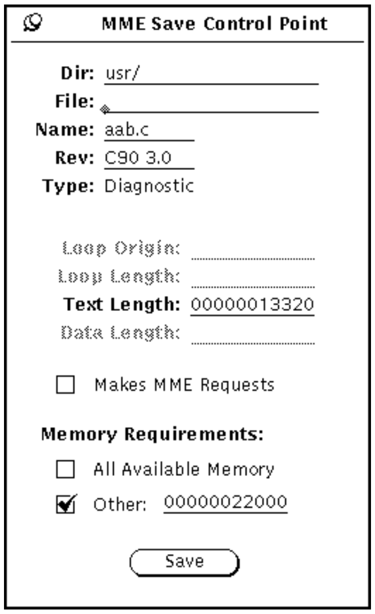
4. Click on (Load). The windows that were saved in the specified file now appear on your screen.

Saving a Control Point in Environment 1 or 2

Perform the following procedure to save a control point so you do not need to reset diagnostic program or utility parameters after loading the diagnostic program or utility as a control point from the general list:



1. Choose **`File --> Save --> Control Point`**, as shown at the left. MME displays the `MME Save Control Point` window:



Dir: usr/
File:
Name: aab.c
Rev: C90 3.0
Type: Diagnostic

Loop Origin:
Loop Length:
Text Length: 00000013320
Data Length:

☐ Makes MME Requests

Memory Requirements:

☐ All Available Memory
☑ Other: 00000022000

NOTE: A `.y` extension means the diagnostic program was assembled in Y-MP mode, and a `.c` extension means that the diagnostic program was assembled in C90 mode.

2. Double click on the `Dir` field and type the subdirectory name you want to use within the `usr` directory in the `Dir` field.

3. Double click on the `File` field and type the filename you want to use in the `File` field.

4. Double click on the `Name` field and type the control point name you want to save. MME displays the default name.

The `MME Save Control Point` window has the following fields, which include information about saving the control point:

`Name`	Name of the control point
`Rev`	Revision level of the control point
`Type`	Diagnostic control point or utility control point
`Origin`	Starting address in mainframe memory for the loop
`Length`	Octal length of the loop
`Text Length`	Octal length of the text segment
`Data Length`	Octal length of the data segment

NOTE: MME displays the control point type as `Diagnostic`.

5. Perform one of the three following steps:

   - To modify the starting and ending addresses of the loop to be saved, go to Step 6.
   - To modify the size of the text segment to be saved, go to Step 8.
   - To modify the size of the data segment to be saved, go to Step 9.

6. Enter the starting address for the loop (origin) by double clicking on the `Loop Origin` field, typing the octal starting address, and pressing the return (↵) key.

7. Enter the last octal address to be used by the loop by double clicking the `Loop Length` field, typing the octal length, and pressing the return (↵) key. Go to Step 10.

8. Enter the last octal address available in the text segment by double clicking on the `Text Length` field, typing the octal length, and pressing the return (↵) key. Go to Step 10.

9. Enter the last octal address available in the data segment by double clicking on the `Data Length` field, typing the octal length, and pressing the return (↵) key. Go to Step 10.

NOTE: In environment 1, all of memory is used and only one loaded control point runs at a time. In environment 2, you can specify the amount of memory used, and you can load and run more than one control point at a time.

10. Perform one of the three following steps:

    - Use the default amount of mainframe memory.

    - Double click on the `Other` field and type the amount of memory you want to use when the control point is reloaded.

    - Click on the `All Available Memory` field to use all of mainframe memory to run this control point. You may want to use all available memory when you save memory control points. Then when you reload and run one or more memory control points, MME uses all of memory.

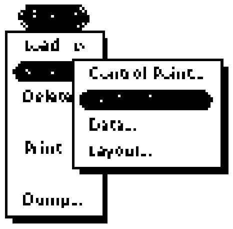
11. Click on Save. MME saves the `aab.c` control point in the `usr` directory with the filename you specified.

Saving a Test List in Environment 1 or 2

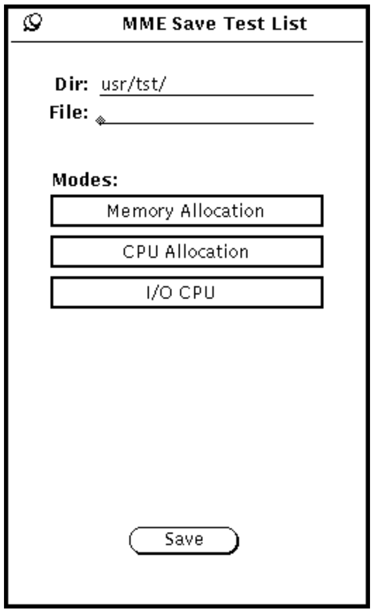
To avoid having to recreate a list of customized control points you loaded into memory, you can save the control points as a test list. Before beginning the following procedure, ensure that you have loaded the control points you want to save in a test list.

To create the sample test list used in this manual, load four copies of `aab.c`, four copies of `mem3.c`, two copies of `sab.c`, two copies of `sas.c`, and four copies of `mem3.c` in that order. You can later use this test list in the “Using the Run System in Environment 2” procedure in this section and in the “Troubleshooting a Mainframe Simulator Bug” procedure in Section 9. If you do not remember how to load a control point, refer to “Loading Control Points in Environment 1 or 2” earlier in this section.

To save the control points in a test list, perform the following procedure:



1. Choose **`File --> Save --> Test List`**, as shown at the left. MME displays the `MME Save Test List` window:



2. Change the directory, if necessary, by triple clicking on the `Dir` field, typing the name of the directory you want to use, and pressing the return (↵) key.

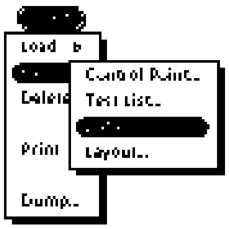
3. Double click on the `File` field and type the filename you want to use.

4. Click on the `Mode` choices (Memory Allocation, CPU Allocation, and/or I/O CPU) you want to save with the test list.

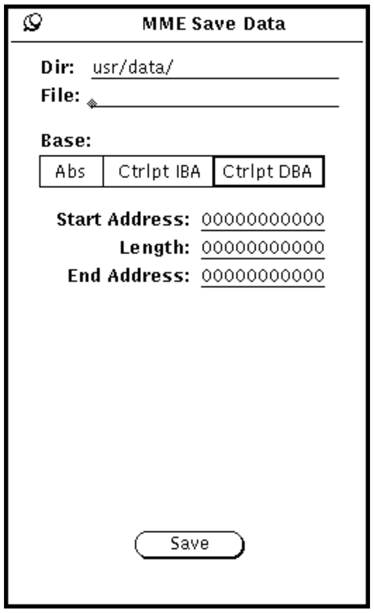
5. Click on Save. MME saves the test list in the specified directory with the specified filename.

Saving Data in Environment 1 or 2

To save data in environment 1 or 2 for later use, perform the following procedure:



1. Choose **`File --> Save --> Data`**, as shown at the left. MME displays the `MME Save Data` window:


MME Save Data

Dir: usr/data/
File:

Base:
Abs | Ctrlpt IBA | Ctrlpt DBA

Start Address: 00000000000
Length: 00000000000
End Address: 00000000000

2. Change the directory, if necessary, by triple clicking on the `Dir` field, typing the name of the directory you want to use, and pressing the return (↵) key.

3. Enter the filename to save. Double click on the `File` field and type the name of the file you want to use.

4. Perform one of the following three actions:

   - Click on Abs to indicate that addresses are absolute (fixed locations in memory).

   - Click on Ctrlpt IBA (control point instruction base address) to specify addresses are relative to the control point IBA.

   - Click on Ctrlpt DBA (control point data base address) to specify addresses are relative to the control point DBA.

5. Enter the starting address, length, and ending address:

   NOTE: After you perform any two of the three following steps, MME updates the third field.

   - Specify the address where you want MME to start saving data from by triple clicking on the `Start Address` field, typing the address you want, and pressing the return (↵) key.

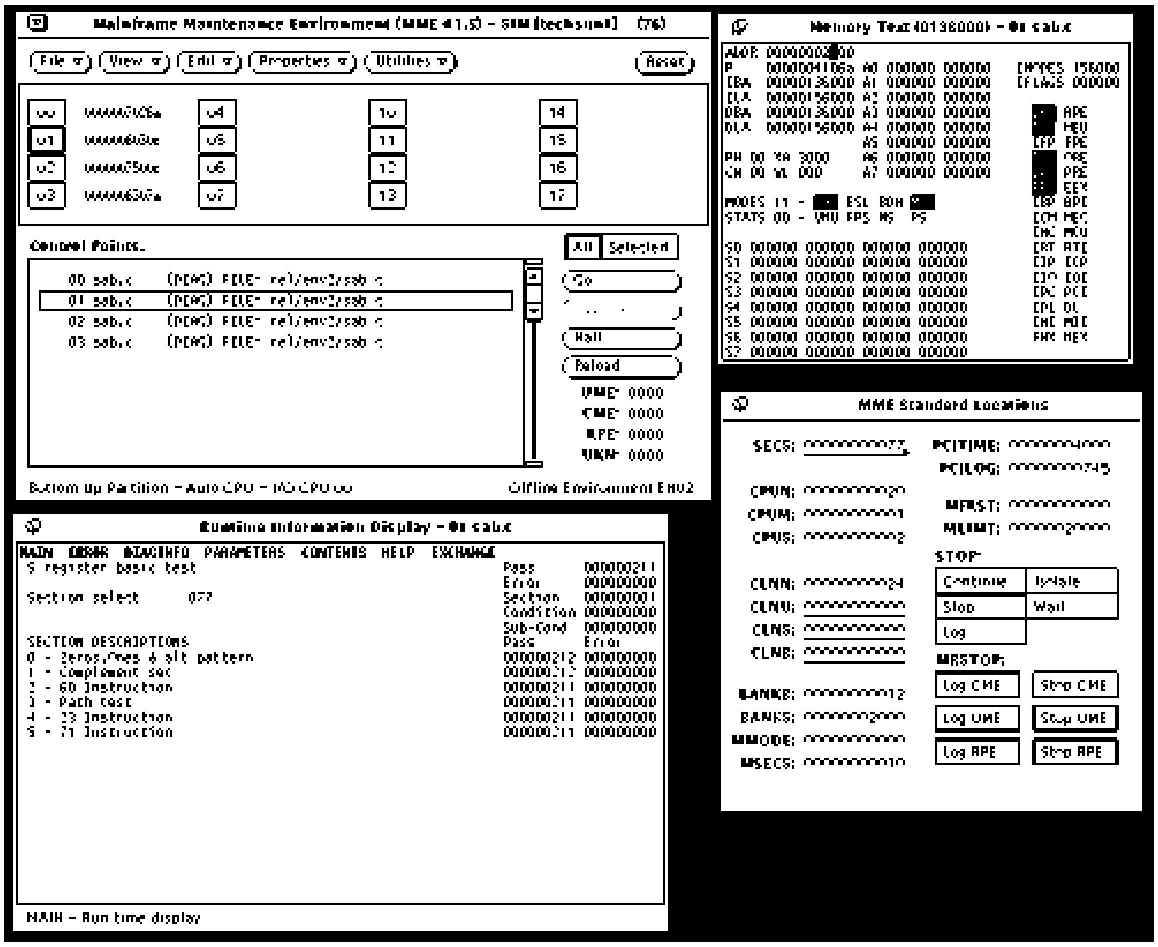
   - Specify the length of the data you want to save by triple clicking on the `Length` field, typing the address you want, and pressing the return (↵) key.

   - Specify the ending address at which you want MME to stop saving the data by clicking on the `End Address` field, typing the address you want, and pressing the return (↵) key.

6. Click on Save.

Saving a Screen Layout in Environment 1 or 2

You can save a screen layout that preserves the organization of the windows you have opened and that enables you to display them later by loading the layout. Figure 4-2 shows a sample layout.


```
Mainframe Maintenance Environment (MME 4.1.5) - SIM [techsun1]   (76)

File ▽   View ▽   Edit ▽   Properties ▽   Utilities ▽          Reset

00  00000?06a   04    10    14
01  0000060e    05    11    15
02  0000050e    06    12    16
03  000063?a    07    13    17

Control Points.                                   All  Selected
    00 sab.c    (PEAS) FILE- rel/env2/sab.c       Go
    01 sab.c    (PEAS) FILE- rel/env2/sab.c
    02 sab.c    (PEAS) FILE- rel/env2/sab.c       Halt
    03 sab.c    (PEAS) FILE- rel/env2/sab.c       Reload
                                                  VME- 0000
                                                  CME- 0000
                                                  APE- 0000
                                                  UKN- 0000
Bottom Up Partition - Auto CPU - I/O CPU 00      Offline Environment ENV2
```

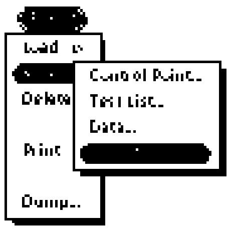
```
Memory Test (01360000) - 01 sab.c
ADDR 0000000200
P    00000041065  A0 000000 000000   INTYES 156000
IBA  00000136000  A1 000000 000000   IFLAGS 000000
ILA  00000156000  A2 000000 000000
DBA  00000136000  A3 000000 000000          APE
DLA  00000156000  A4 000000 000000          MEU
                  A5 000000 000000      IFP FPE
PN 00 XA 3000     A6 000000 000000          ORE
CN 00 VL 000      A7 000000 000000          PRE
                                            EEX
MODES 11 -     ESL BDM                  IBP BPE
STATS 00 - VNU FPS MS  PS               ICM MEC
                                        IMC MCU
S0 000000 000000 000000 000000          IRT RTE
S1 000000 000000 000000 000000          IIP ICP
S2 000000 000000 000000 000000          II0 IOE
S3 000000 000000 000000 000000          IPC PCE
S4 000000 000000 000000 000000          IPL DL
S5 000000 000000 000000 000000          IMC MOE
S6 000000 000000 000000 000000          FNX HEX
S7 000000 000000 000000 000000
```

```
MME Standard Locations

 SECS: 00000000077    PCITIME: 00000004000
                       PCILOG: 00000000745
 CPUN: 00000000020
 CPUM: 00000000001      MFRST: 00000000000
 CPUS: 00000000002      MLIMT: 00000020000
                       STOP:
 CLNN: 00000000024     Continue  Isolate
 CLNU: 00000000000     Stop      Wait
 CLNS: 00000000000     Log
 CLNB: 00000000000     MRSTOP:
BANKB: 00000000012     Log CME   Stop CME
BANKS: 00000002000     Log UME   Stop UME
MMODE: 00000000000     Log APE   Stop APE
MSECS: 00000000010
```

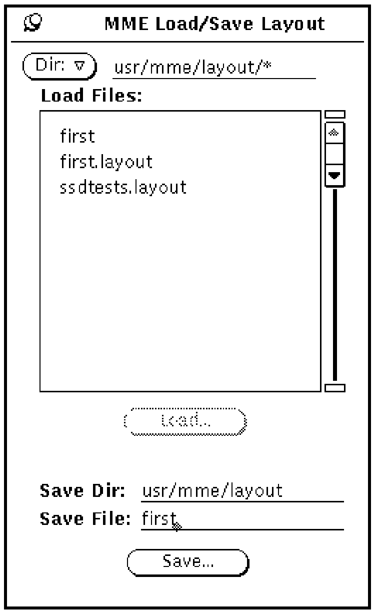
```
Runtime Information Display - 01 sab.c
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
 S register basic test                               Pass       00000211
                                                     Error      00000000
 Section select      077                             Section    00000001
                                                     Condition  00000000
                                                     Sub-Cond   00000000
 SECTION DESCRIPTIONS                                Pass       Error
 0 - Zeros,Ones & alt pattern                        00000212   00000000
 1 - Compliment sec                                  00000212   00000000
 2 - 60 Instruction                                  00000211   00000000
 3 - Path test                                       00000211   00000000
 4 - 23 Instruction                                  00000211   00000000
 5 - 71 Instruction                                  00000211   00000000

 MAIN - Run time display
```

Figure 4-2. MME Sample Layout

To save a screen layout, perform the following procedure:



1. Choose **`File --> Save --> Layout`**, as shown at the left. MME displays the `MME Load/Save Layout` window.



2. Change the directory, if necessary, by performing one of the following actions:

   - Choose the directory from the (Dir: ▽) button. You can choose from the following directories:

     `Release` – Layout files from the current release

     `User` – Layout files that the user has changed and saved

     `Alpha` – Pre-release layout files that are being tested and have not been released

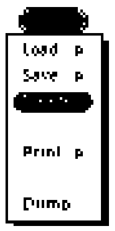
   - Triple click on the (Dir: ▽) field, type the name of the directory you want to use, and press the return (↵) key.

3. Enter the name of the file to save by clicking on the `Save File` field (triple click on the field if a filename is already there) and typing the name of the file. In this example, enter **`mylayout`** as the name of the file.

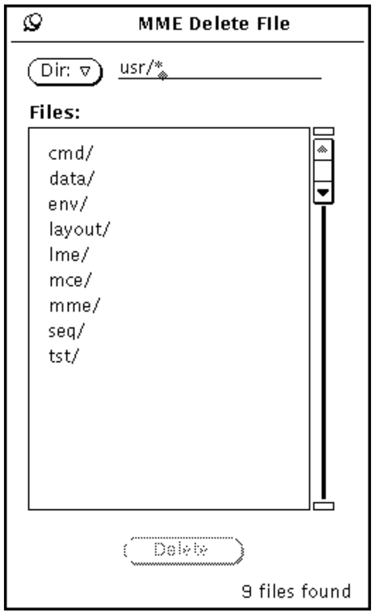
4. Click on (Save). MME saves the specified layout file, and the filename (`mylayout` in this example) appears in the `Load File` scroll box.

Deleting a File in Environment 1 or 2

To delete a file that you no longer need, perform the following procedure.



1. Choose **`File --> Delete`**, as shown at the left. MME displays the `MME Delete File` window:



```
cmd/
data/
env/
layout/
lme/
mce/
mme/
seq/
tst/
```

9 files found

2. Change the directory, if necessary, by performing one of the following actions:

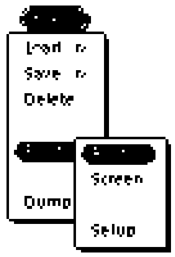
   - Choose the directory from the Dir: ▽ button. Choose from these directories: `usr/*` (all user directories), `usr/cmd` (user command buffers), `usr/tst` (user test lists), `usr/lst` (user listings), and `usr/seq` (user sequences).

   - Triple click on the Dir: ▽ field, type the name of the directory you want to use, and press the return (↵) key.

3. Click on the file you want to delete.

4. Click on Delete . MME deletes the file.

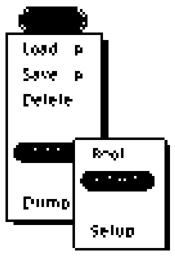
Printing the Root Window in Environment 1 or 2



Before performing this procedure, you must first set up printing. Refer to "Setting Up or Changing Where MME Data is Printed in Environment 1 or Environment 2."

To print an image of everything contained in the root window, choose **`File --> Print --> Root`**, as shown at the left. Use this command to print the MME base window.

Printing a Screen or Panel in Environment 1 or 2



Before performing this procedure, you must first set up printing. Refer to "Setting Up or Changing Where MME Data is Printed in Environment 1 and Environment 2."

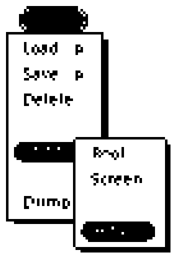
To print a window or an icon, choose **`File --> Print --> Screen`**, as shown at the left. When you choose this command, the cursor becomes a plus symbol. Move the cursor to the window or icon you want to print an image of and click on a mouse button.

NOTE: You cannot print the MME base window with this command. To print the MME base window, choose **`File --> Print --> Root`**.

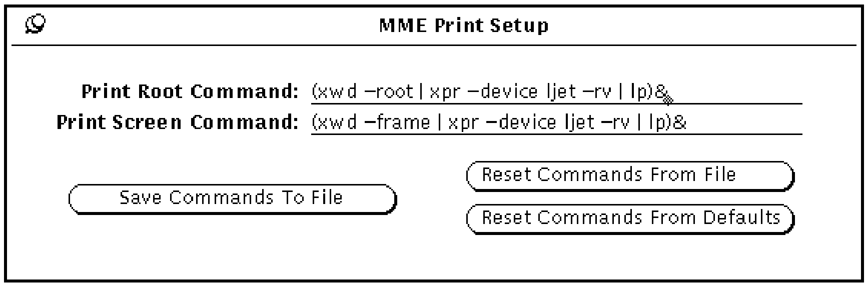
Setting Up or Changing Where MME Data is Printed in Environment 1 or 2

Before you print a root or screen, you must set up the printer options by entering the appropriate UNIX commands in the `Print Root Command` and `Print Screen Command` fields of the `MME Print Setup` window. MME uses the specified commands for the print functions to ensure that output goes to the proper printer.

NOTE: This process uses the `xwd`, `xpr`, and `lp` UNIX commands. The `xwd` command dumps an image of an X window. The `xpr` command prints an image of the X window dump. The `lp` command sends a request to the printer. For detailed information about these commands, refer to the UNIX man pages (enter **`man xwd`**, **`man xpr`**, or **`man lp`** at a UNIX prompt).



To set up where you want data printed, choose **`File --> Print --> Setup`**, as shown at the left. MME displays the `MME Print Setup` window.


MME Print Setup

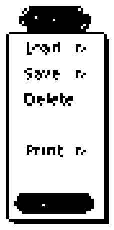
Print Root Command: (xwd –root | xpr –device ljet –rv | lp)&
Print Screen Command: (xwd –frame | xpr –device ljet –rv | lp)&

The buttons in this window are described in the following list:

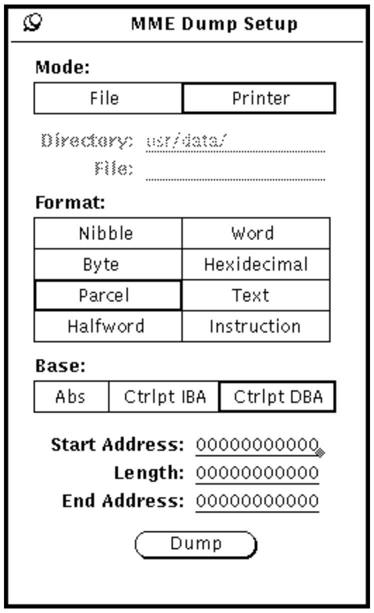
- Click on Save Commands To File to save your current printer setup commands for later use.
- Click on Reset Commands From File to load the printer setup commands you saved previously.
- Click on Reset Commands From Defaults to load the default printer setup commands provided with the MME program.

Dumping Data to a Printer or a File in Environment 1 or 2

To dump data to a printer or a file, perform the following procedure:



1. Choose **`File --> Dump`**, as shown at the left. MME displays the `MME Dump Setup` window:


MME Dump Setup

Mode: File [✓ Printer]

Directory: usr/data/
File:

Format: Nibble, Word, Byte, Hexidecimal, [✓ Parcel], Text, Halfword, Instruction

Base: Abs, Ctrlpt IBA, [✓ Ctrlpt DBA]

Start Address: 00000000000
Length: 00000000000
End Address: 00000000000

2. Specify where you want the dump to go by performing one of the following actions:

   - Click on Printer to dump the data to the default printer you specified in the `MME Printer Setup` window.

   - Click on File to dump the data to a file. Triple click on the `Directory` field, type the name of the directory you want to use, and press the return (↵) key. Triple click on the `File` field, type the name of the file you want, and press the return (↵) key.

3. Specify the `Format` of the dump by clicking on one of the following: Nibble, Byte, Parcel (default), Halfword, Word, Hexidecimal, Text, or Instruction.

4. Specify the `Base` for addressing by clicking on one of the following: Abs (absolute), Ctrlpt IBA (control point instruction base address), or Ctrlpt DBA (control point data base address).

5. Specify the address where you want MME to start dumping data by triple clicking on the `Start Address` field, typing the address you want, and pressing the return (↵) key.

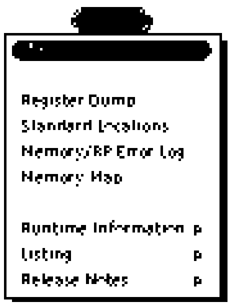
6. Specify the length of the data you want to dump by triple clicking on the `Length` field, typing the address you want, and pressing the return (↵) key.

7. Specify the ending address at which you want MME to stop dumping the data by clicking on the `End Address` field, typing the address you want, and pressing the return (↵) key.

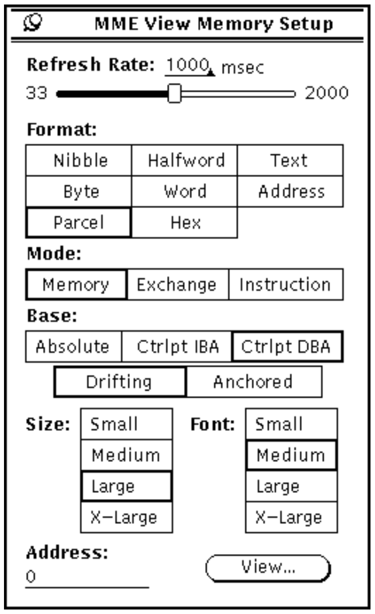
8. Click on Dump . MME dumps the data to the printer or file.

Viewing Memory in Environment 1 or 2

You can view memory to track the status of a control point that you are running in environment 1 or 2. To view memory, perform the following procedure:



1. Choose **`View --> Memory`**, as shown at the left. MME displays the `MME View Memory Setup` window in environment 1 or 2:



MME View Memory Setup

Refresh Rate: 1000 msec
33 2000

Format: Nibble Halfword Text Byte Word Address [✓ Parcel] Hex

Mode: [✓ Memory] Exchange Instruction

Base: Absolute Ctrlpt IBA [✓ Ctrlpt DBA]
[✓ Drifting] Anchored

Size: Small Medium [✓ Large] X–Large

Font: Small [✓ Medium] Large X–Large

Address: 0

View...

NOTE: You can set the interval at which memory windows are updated by moving the `Refresh Rate` slider or by double clicking on the `Refresh Rate` field, typing a new value, and pressing the return (↵) key. Setting this value too low can monopolize the workstation CPU.

2. Click on a `Format` [Nibble, Halfword, Text, Byte, Word, Address, Parcel, or Hex (hexadecimal)] to specify the format in which you want the memory displayed.

3. Specify the `Mode` by clicking on Memory, Exchange, or Instruction.

   Memory mode displays normal memory, exchange mode displays exchange information, and instruction mode decodes the memory into instructions.

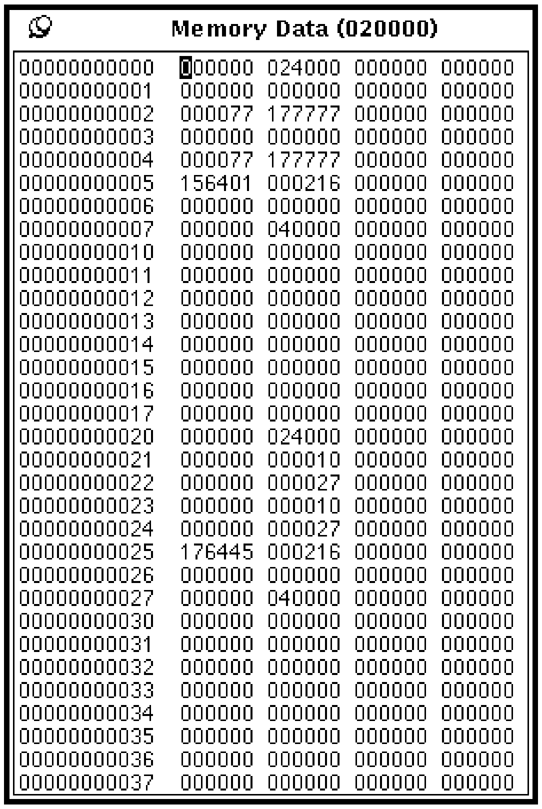
4. Specify the memory `Base` by clicking on Abs to choose a fixed location in mainframe memory, Ctrlpt IBA (control point instruction base address) to choose a relative control point location within memory, or Ctrlpt DBA (control point data base address) to choose a relative control point location within memory.

5. Click on Drifting or Anchored.

   NOTE: If you use Anchored to look at a fixed location in mainframe memory, then the base address does not change. If you use Drifting to look at a location in mainframe memory, then the base address for the control point is relative and will change.

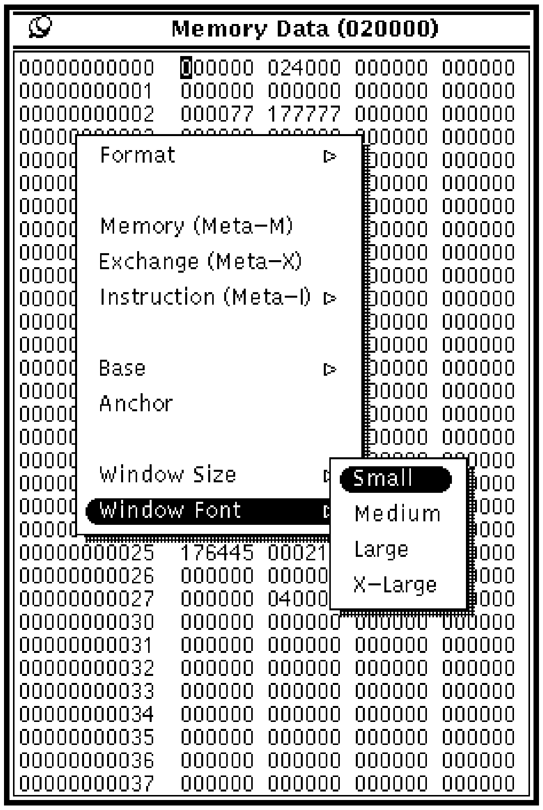
6. Specify the memory to view, if necessary, by double clicking on the `Address` field, typing an octal starting address, and pressing the return (↵) key.

7. Click on View... . MME displays the specified memory window:



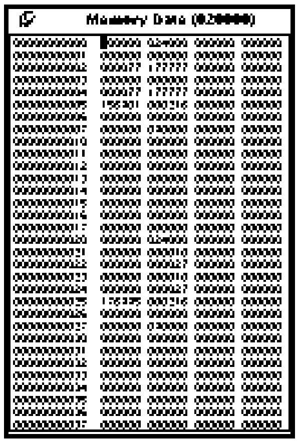
```
Memory Data (020000)
00000000000  000000 024000 000000 000000
00000000001  000000 000000 000000 000000
00000000002  000077 177777 000000 000000
00000000003  000000 000000 000000 000000
00000000004  000077 177777 000000 000000
00000000005  156401 000216 000000 000000
00000000006  000000 000000 000000 000000
00000000007  000000 040000 000000 000000
00000000010  000000 000000 000000 000000
00000000011  000000 000000 000000 000000
00000000012  000000 000000 000000 000000
00000000013  000000 000000 000000 000000
00000000014  000000 000000 000000 000000
00000000015  000000 000000 000000 000000
00000000016  000000 000000 000000 000000
00000000017  000000 000000 000000 000000
00000000020  000000 024000 000000 000000
00000000021  000000 000010 000000 000000
00000000022  000000 000027 000000 000000
00000000023  000000 000010 000000 000000
00000000024  000000 000027 000000 000000
00000000025  176445 000216 000000 000000
00000000026  000000 000000 000000 000000
00000000027  000000 040000 000000 000000
00000000030  000000 000000 000000 000000
00000000031  000000 000000 000000 000000
00000000032  000000 000000 000000 000000
00000000033  000000 000000 000000 000000
00000000034  000000 000000 000000 000000
00000000035  000000 000000 000000 000000
00000000036  000000 000000 000000 000000
00000000037  000000 000000 000000 000000
```

If you want to change the `Format`, `Window Size`, or `Window Font` from the `Memory Data` window, press the MENU mouse button and choose the menu item you want:



```
00000000000   000000  024000  000000  000000
00000000001   000000  000000  000000  000000
00000000002   000077  177777  000000  000000
00000000025   176445  00021
00000000026   000000  00000
00000000027   000000  04000
00000000030   000000  000000  000000  000000
00000000031   000000  000000  000000  000000
00000000032   000000  000000  000000  000000
00000000033   000000  000000  000000  000000
00000000034   000000  000000  000000  000000
00000000035   000000  000000  000000  000000
00000000036   000000  000000  000000  000000
00000000037   000000  000000  000000  000000
```

For example, the following `Memory Data` window appears when you choose a small font:



Memory Data (020000)

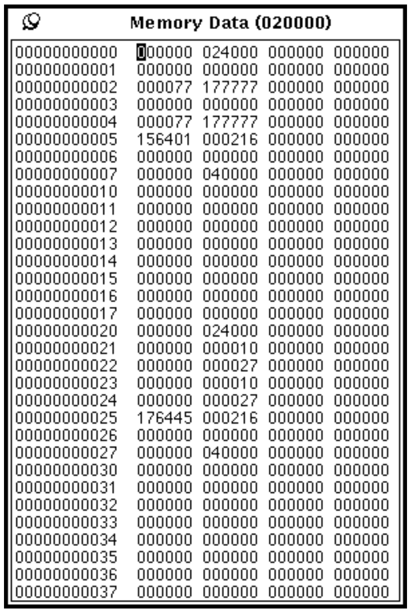
```
0000000000  000000 024000 000000 000000
0000000001  000000 000000 000000 000000
0000000002  000077 177777 000000 000000
0000000003  000000 000000 000000 000000
0000000004  000077 177777 000000 000000
0000000005  156401 000216 000000 000000
0000000006  000000 000000 000000 000000
0000000007  000000 030000 000000 000000
0000000010  000000 000000 000000 000000
0000000011  000000 000000 000000 000000
0000000012  000000 000000 000000 000000
0000000013  000000 000000 000000 000000
0000000014  000000 000000 000000 000000
0000000015  000000 000000 000000 000000
0000000016  000000 000000 000000 000000
0000000017  000000 000000 000000 000000
0000000020  000000 024000 000000 000000
0000000021  000000 000010 000000 000000
0000000022  000000 000027 000000 000000
0000000023  000000 000010 000000 000000
0000000024  000000 000027 000000 000000
0000000025  176325 000216 000000 000000
0000000026  000000 000000 000000 000000
0000000027  000000 030000 000000 000000
0000000030  000000 000000 000000 000000
0000000031  000000 000000 000000 000000
0000000032  000000 000000 000000 000000
0000000033  000000 000000 000000 000000
0000000034  000000 000000 000000 000000
0000000035  000000 000000 000000 000000
0000000036  000000 000000 000000 000000
0000000037  000000 000000 000000 000000
```

You can change the `Format`, mode, and `Window Size` in a manner similar to changing a window font.

Changing Memory in Environment 1 or 2

After you have displayed memory (refer to “Viewing Memory in Environment 1 or 2”), you may want to change the value at a memory address for a loop or a control point you want to run in environment 1 or 2. To change a value at a memory address, perform the following procedure.

1. Refer to “Viewing Memory in Environment 1 or 2” to view the following `Memory Data` window (use all the default values on the `MME View Memory` window).



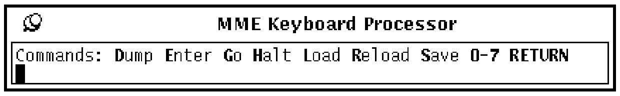
```
Memory Data (020000)
00000000000  000000 024000 000000 000000
00000000001  000000 000000 000000 000000
00000000002  000077 177777 000000 000000
00000000003  000000 000000 000000 000000
00000000004  000077 177777 000000 000000
00000000005  156401 000216 000000 000000
00000000006  000000 000000 000000 000000
00000000007  000000 040000 000000 000000
00000000010  000000 000000 000000 000000
00000000011  000000 000000 000000 000000
00000000012  000000 000000 000000 000000
00000000013  000000 000000 000000 000000
00000000014  000000 000000 000000 000000
00000000015  000000 000000 000000 000000
00000000016  000000 000000 000000 000000
00000000017  000000 000000 000000 000000
00000000020  000000 024000 000000 000000
00000000021  000000 000010 000000 000000
00000000022  000000 000027 000000 000000
00000000023  000000 000010 000000 000000
00000000024  000000 000027 000000 000000
00000000025  176445 000216 000000 000000
00000000026  000000 000000 000000 000000
00000000027  000000 040000 000000 000000
00000000030  000000 000000 000000 000000
00000000031  000000 000000 000000 000000
00000000032  000000 000000 000000 000000
00000000033  000000 000000 000000 000000
00000000034  000000 000000 000000 000000
00000000035  000000 000000 000000 000000
00000000036  000000 000000 000000 000000
00000000037  000000 000000 000000 000000
```

Notice that the word at memory address 0, parcel a, appears highlighted in the `Memory Data` window.

NOTE: To change the format, mode, window size, font size, or address, you can use the MENU mouse button or you can use the diamond-shaped (◇) meta key to the left of the space bar and one of the following alphabetical keys: `a` for address format, `n` for nibble format, `b` for byte format, `p` for parcel format, `h` for halfword format, `w` for word format, `e` for hexadecimal format, `t` for text format, `i` for instruction mode, `x` for exchange mode, `d` for diagnostic monitor mode, and `m` for memory mode.

2. Perform one of the following actions:

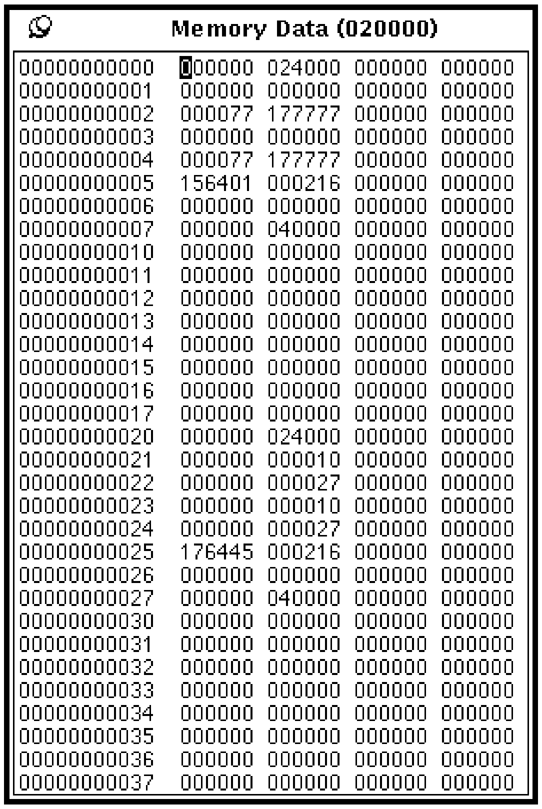
   - Press and release the space bar; the `MME Keyboard Processor` window appears:



```
MME Keyboard Processor
Commands: Dump Enter Go Halt Load Reload Save 0-7 RETURN
```

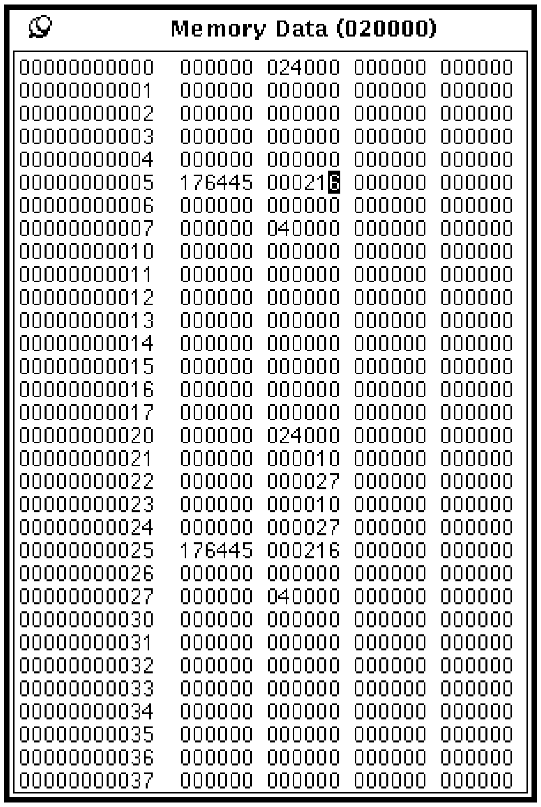
Using this window, you can change (convert) the format; dump memory data to the printer; enter data in mainframe memory; start or halt control points; load, reload, or save control points; or view mainframe memory at a specific address. When you type the letter or number shown in bold, the command runs. The next window prompts you for any additional commands or information you must enter.

- Move the mouse pointer to the word in mainframe memory you want to change using the PgUp (page up) or PgDn (page down) keys and the cursor movement keys. Click on the word you want so that the word appears in reverse video. For example by default, the first 0 of the word at memory address 000000, parcel a, appears highlighted:



```
Memory Data (020000)
00000000000   000000  024000  000000  000000
00000000001   000000  000000  000000  000000
00000000002   000077  177777  000000  000000
00000000003   000000  000000  000000  000000
00000000004   000077  177777  000000  000000
00000000005   156401  000216  000000  000000
00000000006   000000  000000  000000  000000
00000000007   000000  040000  000000  000000
00000000010   000000  000000  000000  000000
00000000011   000000  000000  000000  000000
00000000012   000000  000000  000000  000000
00000000013   000000  000000  000000  000000
00000000014   000000  000000  000000  000000
00000000015   000000  000000  000000  000000
00000000016   000000  000000  000000  000000
00000000017   000000  000000  000000  000000
00000000020   000000  024000  000000  000000
00000000021   000000  000010  000000  000000
00000000022   000000  000027  000000  000000
00000000023   000000  000010  000000  000000
00000000024   000000  000027  000000  000000
00000000025   176445  000216  000000  000000
00000000026   000000  000000  000000  000000
00000000027   000000  040000  000000  000000
00000000030   000000  000000  000000  000000
00000000031   000000  000000  000000  000000
00000000032   000000  000000  000000  000000
00000000033   000000  000000  000000  000000
00000000034   000000  000000  000000  000000
00000000035   000000  000000  000000  000000
00000000036   000000  000000  000000  000000
00000000037   000000  000000  000000  000000
```

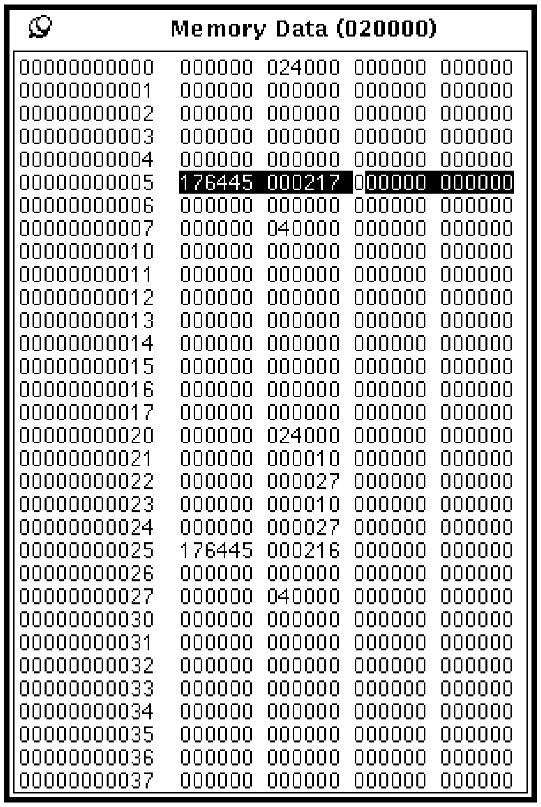
3. Move the cursor to the parcel you want to change. In this example, it is parcel 000005b:



```
Memory Data (020000)
00000000000   000000 024000 000000 000000
00000000001   000000 000000 000000 000000
00000000002   000000 000000 000000 000000
00000000003   000000 000000 000000 000000
00000000004   000000 000000 000000 000000
00000000005   176445 000216 000000 000000
00000000006   000000 000000 000000 000000
00000000007   000000 040000 000000 000000
00000000010   000000 000000 000000 000000
00000000011   000000 000000 000000 000000
00000000012   000000 000000 000000 000000
00000000013   000000 000000 000000 000000
00000000014   000000 000000 000000 000000
00000000015   000000 000000 000000 000000
00000000016   000000 000000 000000 000000
00000000017   000000 000000 000000 000000
00000000020   000000 024000 000000 000000
00000000021   000000 000010 000000 000000
00000000022   000000 000027 000000 000000
00000000023   000000 000010 000000 000000
00000000024   000000 000027 000000 000000
00000000025   176445 000216 000000 000000
00000000026   000000 000000 000000 000000
00000000027   000000 040000 000000 000000
00000000030   000000 000000 000000 000000
00000000031   000000 000000 000000 000000
00000000032   000000 000000 000000 000000
00000000033   000000 000000 000000 000000
00000000034   000000 000000 000000 000000
00000000035   000000 000000 000000 000000
00000000036   000000 000000 000000 000000
00000000037   000000 000000 000000 000000
```

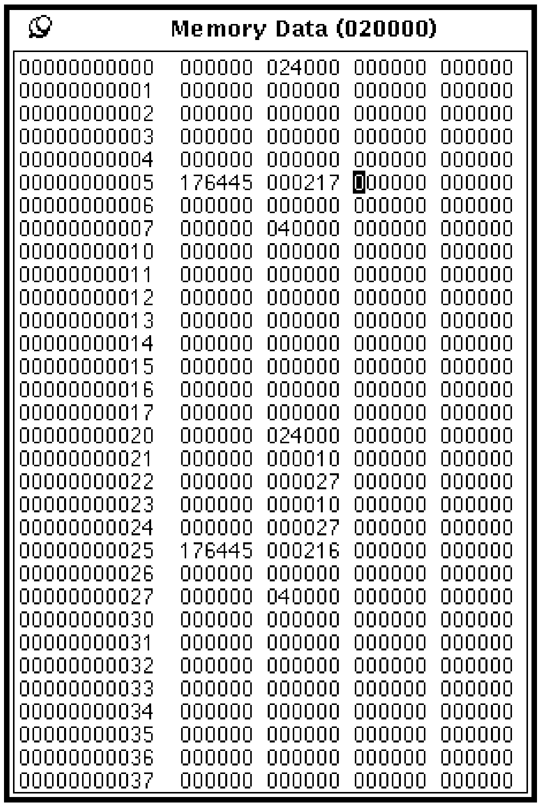
4. Press and release the Esc key. Type the new word value. The entire word is highlighted, which allows you to change the entire word.

   When you type a new word value, it automatically changes the value in mainframe memory. In this example, the `Memory Data` window shows the word value changed from 000216 to 000217 at memory address 000005, parcel b:



```
Memory Data (020000)
00000000000   000000 024000 000000 000000
00000000001   000000 000000 000000 000000
00000000002   000000 000000 000000 000000
00000000003   000000 000000 000000 000000
00000000004   000000 000000 000000 000000
00000000005   176445 000217 000000 000000
00000000006   000000 000000 000000 000000
00000000007   000000 040000 000000 000000
00000000010   000000 000000 000000 000000
00000000011   000000 000000 000000 000000
00000000012   000000 000000 000000 000000
00000000013   000000 000000 000000 000000
00000000014   000000 000000 000000 000000
00000000015   000000 000000 000000 000000
00000000016   000000 000000 000000 000000
00000000017   000000 000000 000000 000000
00000000020   000000 024000 000000 000000
00000000021   000000 000010 000000 000000
00000000022   000000 000027 000000 000000
00000000023   000000 000010 000000 000000
00000000024   000000 000027 000000 000000
00000000025   176445 000216 000000 000000
00000000026   000000 000000 000000 000000
00000000027   000000 040000 000000 000000
00000000030   000000 000000 000000 000000
00000000031   000000 000000 000000 000000
00000000032   000000 000000 000000 000000
00000000033   000000 000000 000000 000000
00000000034   000000 000000 000000 000000
00000000035   000000 000000 000000 000000
00000000036   000000 000000 000000 000000
00000000037   000000 000000 000000 000000
```

5. Press and release the return (↵) key; the memory is updated, as shown in the following `Memory Data` window:



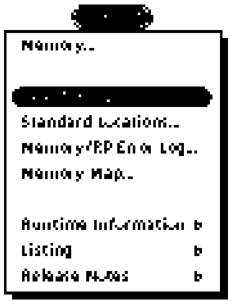
```
Memory Data (020000)
00000000000   000000 024000 000000 000000
00000000001   000000 000000 000000 000000
00000000002   000000 000000 000000 000000
00000000003   000000 000000 000000 000000
00000000004   000000 000000 000000 000000
00000000005   176445 000217 000000 000000
00000000006   000000 000000 000000 000000
00000000007   000000 040000 000000 000000
00000000010   000000 000000 000000 000000
00000000011   000000 000000 000000 000000
00000000012   000000 000000 000000 000000
00000000013   000000 000000 000000 000000
00000000014   000000 000000 000000 000000
00000000015   000000 000000 000000 000000
00000000016   000000 000000 000000 000000
00000000017   000000 000000 000000 000000
00000000020   000000 024000 000000 000000
00000000021   000000 000010 000000 000000
00000000022   000000 000027 000000 000000
00000000023   000000 000010 000000 000000
00000000024   000000 000027 000000 000000
00000000025   176445 000216 000000 000000
00000000026   000000 000000 000000 000000
00000000027   000000 040000 000000 000000
00000000030   000000 000000 000000 000000
00000000031   000000 000000 000000 000000
00000000032   000000 000000 000000 000000
00000000033   000000 000000 000000 000000
00000000034   000000 000000 000000 000000
00000000035   000000 000000 000000 000000
00000000036   000000 000000 000000 000000
00000000037   000000 000000 000000 000000
```

6. Repeat Steps 1 through 5 until you finish changing mainframe memory.

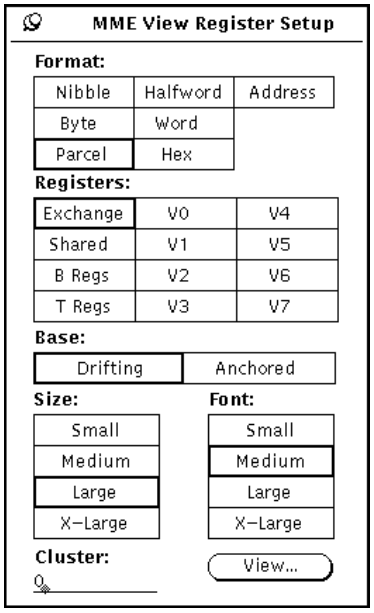
Viewing the Contents of Registers in Environment 1 or 2

To view the contents of exchange, shared, A, B, V, or T registers; perform the following procedure:

NOTE: This data is available (valid) only if a register dump was performed by the interrupt handler for the control point, by the diagnostic controller program, or by the dump-on-halt option (Refer to “Dumping Registers When Halt is Selected in Environment 1” later in this section.).



1. Choose **`View --> Register Dump`**, as shown at the left. The `MME View Register Setup` window appears:



MME View Register Setup

Format: Nibble, Halfword, Address, Byte, Word, [✓ Parcel], Hex

Registers: [✓ Exchange], Shared, B Regs, T Regs, V0, V1, V2, V3, V4, V5, V6, V7

Base: [✓ Drifting], Anchored

Size: Small, Medium, [✓ Large], X-Large

Font: Small, [✓ Medium], Large, X-Large

Cluster: 0

View...

2. Click on the `Format` [Nibble, Halfword, Address, Byte, Word, Parcel, or Hex (hexadecimal)] you want the contents to have.

3. Click on the register you want to view: Exchange, Shared, B Regs, T Regs, V0, V1, V2, V3, V4, V5, V6, or V7.

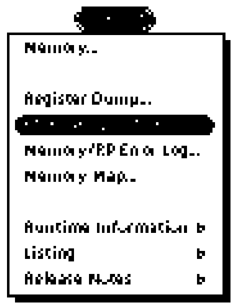
4. Click on Drifting to look at a fixed location in memory (the base address does not change), or click on Anchored to look at a relative location in memory (the base address will change).

5. Click on a `Size` [Small, Medium, Large, or X-Large (extra large)] to indicate the size of the display window.

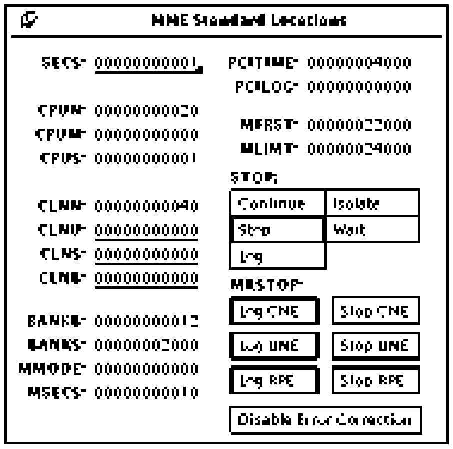
6. Click on a `Font` [Small, Medium, Large, or X-Large (extra large)] to indicate the font type size.

7. Click on View... to dump the contents of the registers to memory so you can view the data.

Viewing and Changing Control Point Standard Locations in Environment 1 or 2



You may want to change the way a control point runs in mainframe memory. To do so, you must view and change the standard locations for a loaded control point. You can set the standard location values to be equal to or less than the physical configuration of your computer system. For example, you can set the number of CPUs to 5. However, with 20 CPUs (in octal) physically present, you cannot set the configuration to more than 20 CPUs in octal. To view and change a standard location for a control point, choose **`View --> Standard Locations`**, as shown at the left. MME shows the `MME Standard Locations` window:



```
MME Standard Locations

  SECS- 00000000001    PCTIME- 00000004000
                        PCLOG- 00000000000
  CPUN- 00000000020
  CPUM- 00000000000     MFRST- 00000022000
  CPUS- 00000000001     MLIMT- 00000024000
                       STOP:
  CLNN- 00000000040    Continue  Isolate
  CLNU- 00000000000    Stop      Wait
  CLNS- 00000000000    Log
  CLNB- 00000000000    MBSTOP-
                       Log CNE   Stop CNE
 BANKB- 00000000012    Log UNE   Stop UNE
 BANKS- 00000002000    Log RPE   Stop RPE
 MMODE- 00000000000
 MSECS- 00000000010    Disable Error Correction
```

The `MME Standard Locations` window contains the following information:

The `SECS` field contains a bit mask representing the sections of the control point that will run. Change this value to control which sections run when you click on Go. Table 4-4 shows the octal values you should enter to select the different test sections. Use the table values to enter a single section you want to run. However, if you want to run more than one section but not all the sections, add the octal values for the sections you want to run. For example, if you want to run sections 1 (00000000002), 2 (00000000004), and 7 (00000000200), add the octal numbers together and enter the value in the `SECS` field. In this example, you would enter 00000000206 in the `SECS` field.

NOTE: Refer to the Section 7, “Diagnostic Tests and Utilities,” in this manual to determine which sections are available for the control points.

Table 4-4. Control Point Sections Run

Section Selection Bit Mask Octal Value	Octal Sections Run
00000000001	0
00000000002	1
00000000004	2
00000000010	3
00000000020	4
00000000040	5
00000000100	6
00000000200	7
00000000400	10
00000001000	11
00000002000	12
00000004000	13

The `CPUN` (CPU number) read-only field shows the number of CPUs you can use. You must use the MCE application to change the number of CPUs.

The `CPUM` (master CPU) read-only field shows the specified master CPU. This location is set by MME for multi-CPU control points

The `CPUS` (CPUs) read-only field shows the bit mask of the CPUs you want to test. This location is set by the CPUs that have been assigned multi-CPU control points. Table 4-5 lists the bit mask values and CPUs tested.

Table 4-5. CPUs Tested

CPU Bit Mask Octal Value	CPUs (in Octal) Tested
00000000001	0
00000000002	1
00000000004	2
00000000010	3
00000000020	4
00000000040	5
00000000100	6
00000000200	7
00000000400	10
00000001000	11
00000002000	12
00000004000	13
00000010000	14
00000020000	15
00000040000	16
00000100000	17

The `CLNN` read-only field specifies the number of clusters you want to use. Valid values are 4 through 40. You must use the MCE application to change this value.

The `CLNU` field contains the cluster number to use. Change this value to specify a different cluster number to use.

The CLNS field contains a bit mask that specifies the clusters you want to test. Change this value to specify different clusters to test. Table 4-6 shows you the octal value to enter to test a single cluster. Use the table values to enter a single cluster you want to test. However, if you want to test more than one cluster but not all the clusters, add the octal values for the clusters you want to test. For example, if you want to test clusters 1 (00000000002), 2 (00000000004), and 7 (00000000200), add the octal numbers together and enter the value in the CLNS field. In this example, you would enter 00000000206 in the CLNS field.

Table 4-6. Clusters Tested

Cluster Number Bit Mask Octal Value	Clusters (in Octal) Tested
00000000001	0
00000000002	1
00000000004	2
00000000010	3
00000000020	4
00000000040	5
00000000100	6
00000000200	7
00000000400	10
00000001000	11
00000002000	12
00000004000	13
00000010000	14
00000020000	15
00000040000	16
00000100000	17

The CLNB field specifies the background cluster number. Change this field, if desired. Enter a 0 to avoid using semaphores.

The BANKB read-only field specifies the number of bank bits you want to test.

The BANKS read-only field specifies the number of banks you want to test. You must use the MCE application to change the number of banks you want to test.

The `MMODE` read-only field specifies the monitor mode. Monitor mode instructions, such as channel control, real-time clock, programmable clock interrupts, and so on, perform special functions that are useful to the operating system. These instructions run only when the CPU is operating in monitor mode. If a monitor mode instruction issues while the CPU is not in monitor mode, the instruction is treated as a no-operation instruction.

The `MSECS` read-only field specifies the number of memory sections you want to test. You must use the MCE application to change the number of memory sections.

The `PCITIME` read-only field shows the programmable-clock interrupt time interval. You must use the `MME Resource Allocation` window to change the programmable-clock interrupt time interval value.

The `PCILOG` read-only field shows the programmable-clock interrupt counter or log.

The `MFIRST` read-only field shows the address of the first memory word you want to test. Use the `MME Load Control Point` window to change the address of the first memory word you want to test.

The `MLMT` read-only field specifies the octal memory limit (last memory address) you want to test. Use the `MME Load Control Point` window to change the octal memory limit.

The `STOP` parameter specifies what MME should do on error. Perform one of the following actions to change this parameter:

- Click on Continue to update CPUINFO and continue processing.
- Click on Stop (default) to update CPUINFO and stop processing.
- Click on Log to write data to the error information block (EIBK) or error log.
- Click on Isolate to start a loop or a diagnostic test when the control point finds an error.

- Click on Wait to enable the wait/resume function. Some control points use this function to stop control point execution at a programmer-defined breakpoint. When this happens, the Resume button activates in the MME base window. Click on Resume to continue control point execution.

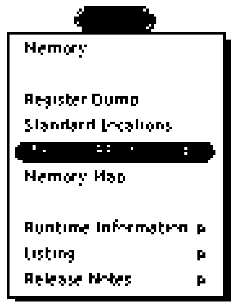
  This function enables you to review the current error information before testing is resumed. This function also enables you to respond to messages displayed by the control point in the MME `Runtime Information Display` window. These messages prompt you to perform a necessary action before control point execution can continue (for example, a message may prompt you to set specific switches).

The `MRSTOP` parameter is a memory and register bit mask. Perform one or more of the following actions to specify which errors to stop for or log:

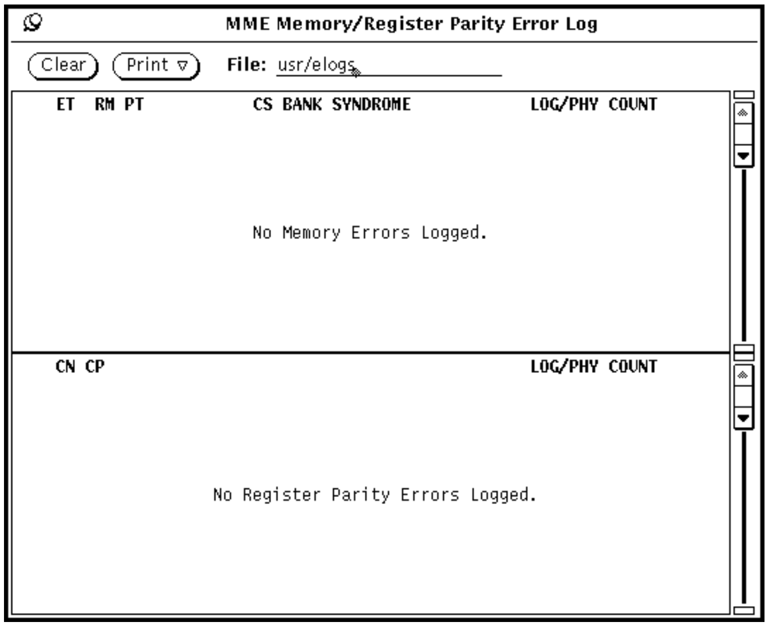
- Click on Log CME to log correctable memory errors.
- Click on Log UME to log uncorrectable memory errors.
- Click on Log RPE to log register parity errors.
- Click on Stop CME to stop logging correctable memory errors.
- Click on Stop UME to stop logging uncorrectable memory errors.
- Click on Stop RPE to stop logging register parity errors.
- Click on Disable Error Correction to disable the error-correction code.

Viewing the Memory/Register Parity Error Log in Environment 1 or 2

To view the `Memory/Register Parity Error Log` for memory or register parity errors, perform the following procedure:



1. Choose **`View --> Memory Error/RP Error Log`**, as shown at the left. MME shows the `Error Log Display`:



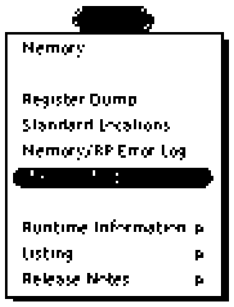
```
ET  RM PT          CS BANK SYNDROME          LOG/PHY COUNT

                 No Memory Errors Logged.

CN CP                                        LOG/PHY COUNT

              No Register Parity Errors Logged.
```

2. Perform one of the following actions:

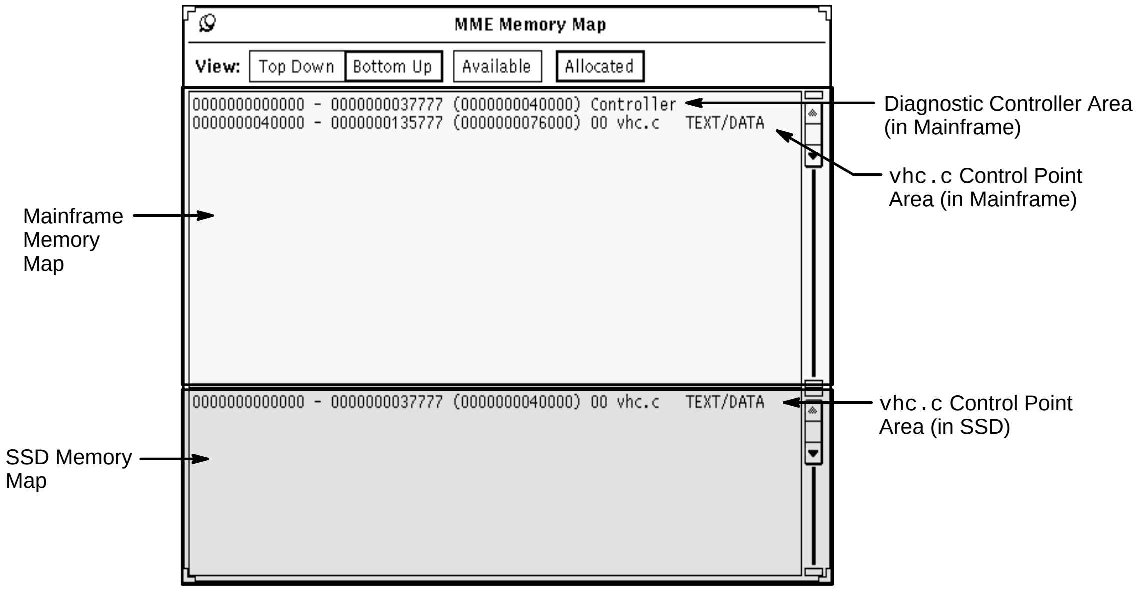
   - Click on (Clear) to erase the data in the `MME Memory/Register Parity Error Log` window.

   - Double click on the `File` field, type the name of the file in which you want to place the data, and press the return (↵) key. To print the data in the file, choose **`Print --> xxxx`**, where ***`xxxx`*** is the printer you want to send the data to.

Viewing the MME Memory Map in Environment 1 or 2



The memory map enables you to display an overview of mainframe memory. Using the memory map, you can view the location of a control point in memory, a control point's size in memory, and available memory. To display the MME memory map, choose **`View -->`** **`Memory Map`**, as shown at the left. MME displays the `MME Memory Map`:


MME Memory Map
View:
Top Down
Bottom Up
Available
Allocated
Diagnostic Controller Area (in Mainframe)
vhc.c Control Point Area (in Mainframe)
Mainframe Memory Map
vhc.c Control Point Area (in SSD)
SSD Memory Map

```
0000000000000 - 0000000037777 (0000000040000) Controller
0000000040000 - 0000000135777 (0000000076000) 00 vhc.c   TEXT/DATA
```

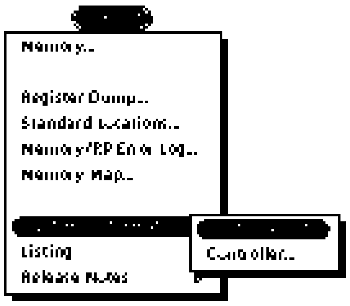
```
0000000000000 - 0000000037777 (0000000040000) 00 vhc.c   TEXT/DATA
```

The top portion of the memory map shows the mainframe memory, and the bottom portion of the memory map shows the SSD memory.

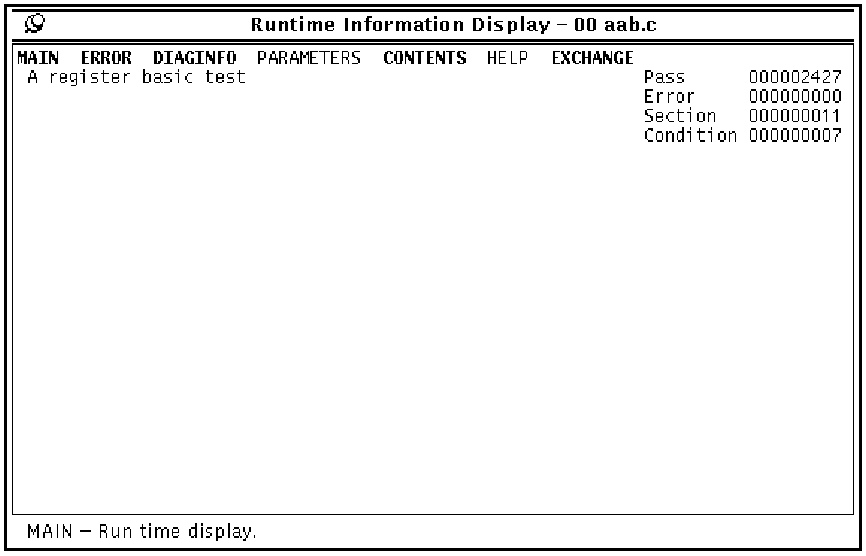
You can manipulate the memory map by performing the following actions:

- Click on Top Down to view memory from the highest memory location to the lowest memory location.
- Click on Bottom Up to view memory from the lowest memory location to the highest memory location.
- Click on Available to view memory that is not currently used.
- Click on Allocated to view memory that is currently used.

Viewing the Control Point Runtime Information Display in Environment 1 or 2



To view the `Runtime Information Display` for the current control point, choose **`View --> Runtime Information --> Control Point`**, as shown at the left. MME displays the `MAIN` runtime display:

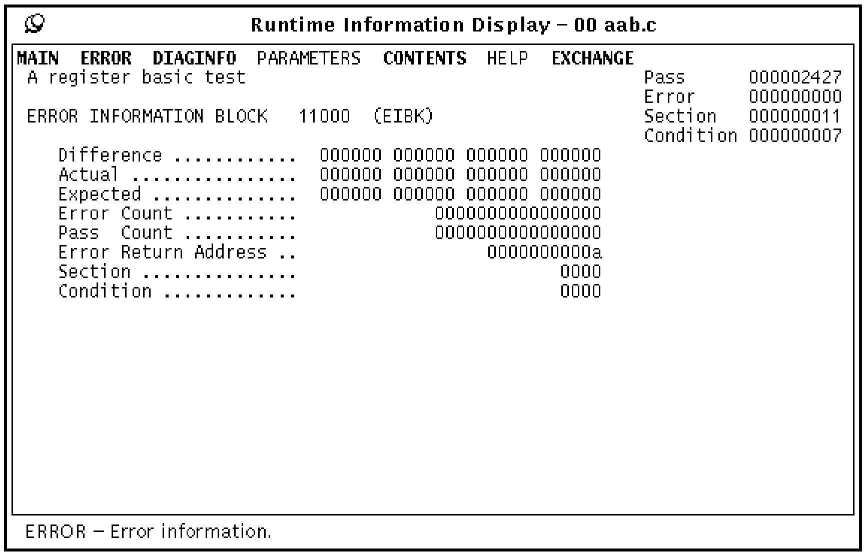


```
Runtime Information Display – 00 aab.c
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
 A register basic test                                          Pass      000002427
                                                                Error     000000000
                                                                Section   000000011
                                                                Condition 000000007

MAIN – Run time display.
```

The `MAIN` runtime display shows the following information for the control point: the number of passes (`Pass`), the number of errors (`Error`), the current section that is running (`Section`), and the current condition that is running (`Condition`).

You can switch runtime information displays by clicking on the bold headings on the current display. Perform the following actions to view the different displays.

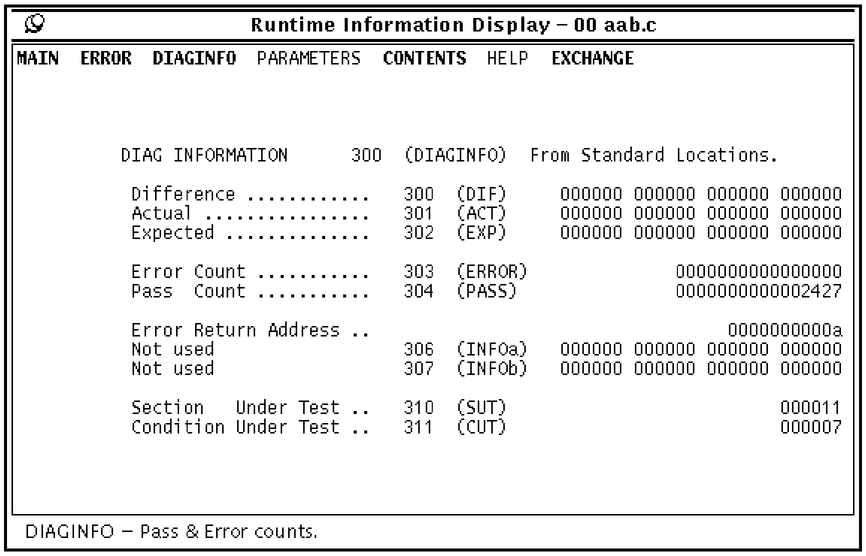
- Click on **ERROR**. MME displays the error information on the `MME Runtime Information Display`:



```
Runtime Information Display – 00 aab.c
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
 A register basic test                                         Pass      000002427
                                                               Error     000000000
 ERROR INFORMATION BLOCK   11000  (EIBK)                       Section   000000011
                                                               Condition 000000007
    Difference ............  000000 000000 000000 000000
    Actual ................  000000 000000 000000 000000
    Expected ..............  000000 000000 000000 000000
    Error Count ...........             0000000000000000
    Pass  Count ...........             0000000000000000
    Error Return Address ..                  0000000000a
    Section ...............                         0000
    Condition .............                         0000

 ERROR – Error information.
```

The `ERROR` display shows the information contained in the error information block stored at address 11000.

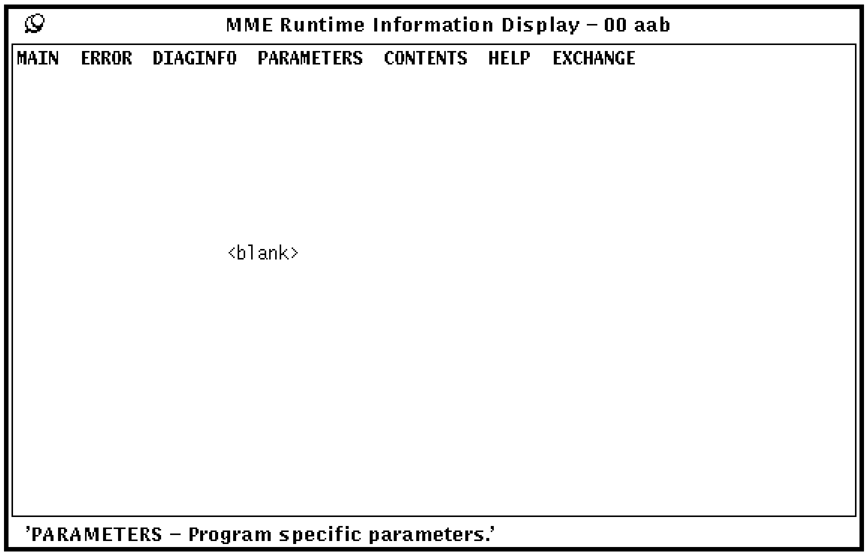
- Click on **DIAGINFO**. MME displays the CPU information on the `MME Runtime Information Display`:



```
Runtime Information Display – 00 aab.c
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE


          DIAG INFORMATION      300  (DIAGINFO)  From Standard Locations.

           Difference ...........  300  (DIF)     000000 000000 000000 000000
           Actual ...............  301  (ACT)     000000 000000 000000 000000
           Expected .............  302  (EXP)     000000 000000 000000 000000

           Error Count ..........  303  (ERROR)              0000000000000000
           Pass  Count ..........  304  (PASS)               0000000000002427

           Error Return Address ..                                0000000000a
           Not used                306  (INFOa)   000000 000000 000000 000000
           Not used                307  (INFOb)   000000 000000 000000 000000

           Section   Under Test ..  310  (SUT)                          000011
           Condition Under Test ..  311  (CUT)                          000007

 DIAGINFO – Pass & Error counts.
```

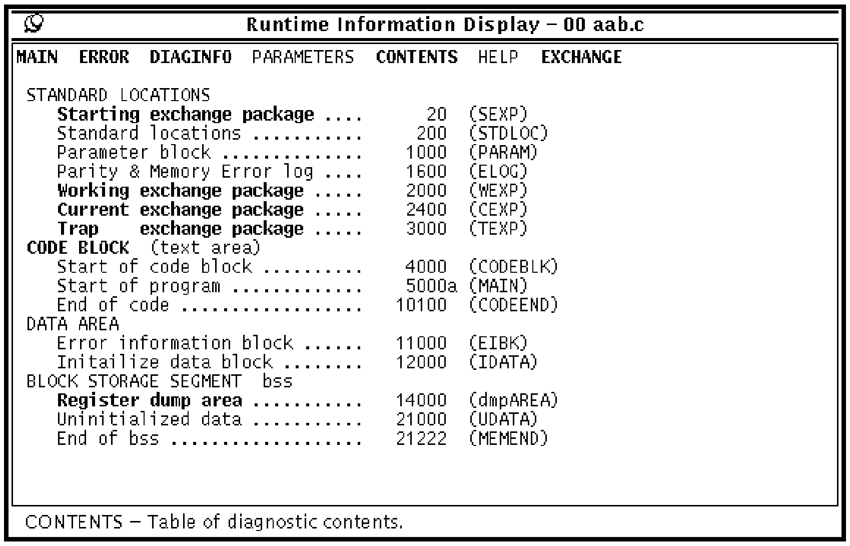
The `DIAGINFO` display shows information about the diagnostic taken from the standard locations.

- Click on **PARAMETERS**. MME displays the parameters on the `MME Runtime Information Display`:


MME Runtime Information Display – 00 aab
MAIN ERROR DIAGINFO PARAMETERS CONTENTS HELP EXCHANGE
<blank>
'PARAMETERS – Program specific parameters.'

The PARAMETERS display shows information about the parameters if any is available.

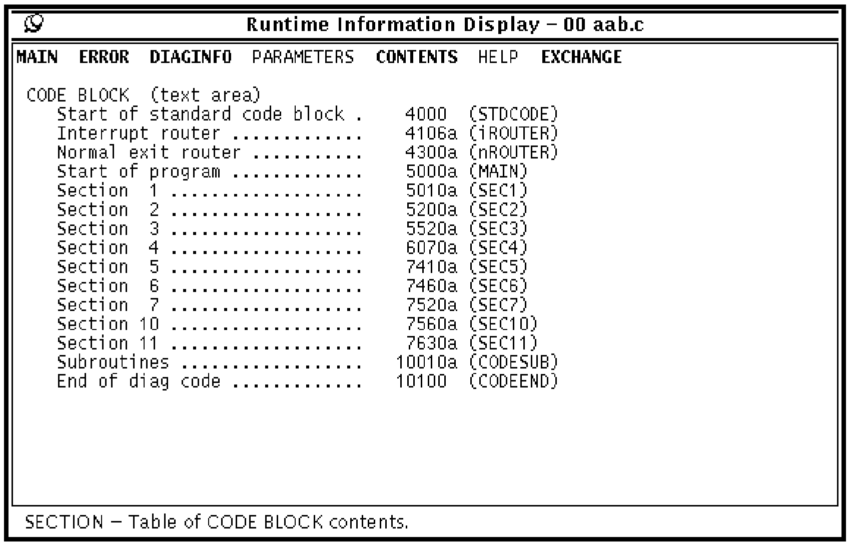
- Click on **CONTENTS**. MME displays the contents on the `MME Runtime Information Display`.


Runtime Information Display – 00 aab.c
MAIN ERROR DIAGINFO PARAMETERS CONTENTS HELP EXCHANGE

```
STANDARD LOCATIONS
   Starting exchange package ....      20  (SEXP)
   Standard locations ..........      200  (STDLOC)
   Parameter block .............     1000  (PARAM)
   Parity & Memory Error log ....    1600  (ELOG)
   Working exchange package .....    2000  (WEXP)
   Current exchange package .....    2400  (CEXP)
   Trap    exchange package .....    3000  (TEXP)
CODE BLOCK  (text area)
   Start of code block .........     4000  (CODEBLK)
   Start of program ............     5000a (MAIN)
   End of code .................    10100  (CODEEND)
DATA AREA
   Error information block ......   11000  (EIBK)
   Initailize data block ........   12000  (IDATA)
BLOCK STORAGE SEGMENT  bss
   Register dump area ..........    14000  (dmpAREA)
   Uninitialized data ..........    21000  (UDATA)
   End of bss ..................    21222  (MEMEND)

CONTENTS – Table of diagnostic contents.
```

The `CONTENTS` display shows a table of diagnostic contents. You can click on the bold entries to switch to different displays. The starting exchange package, working exchange package, current exchange package, and trap exchange package are described as the `SEXP`, `WEXP` `CEXP`, and `TEXP` displays, respectively; the `EXCHANGE` display is described later in this subsection. Perform the following actions to see the `CODE BLOCK` and `Register dump area` displays:

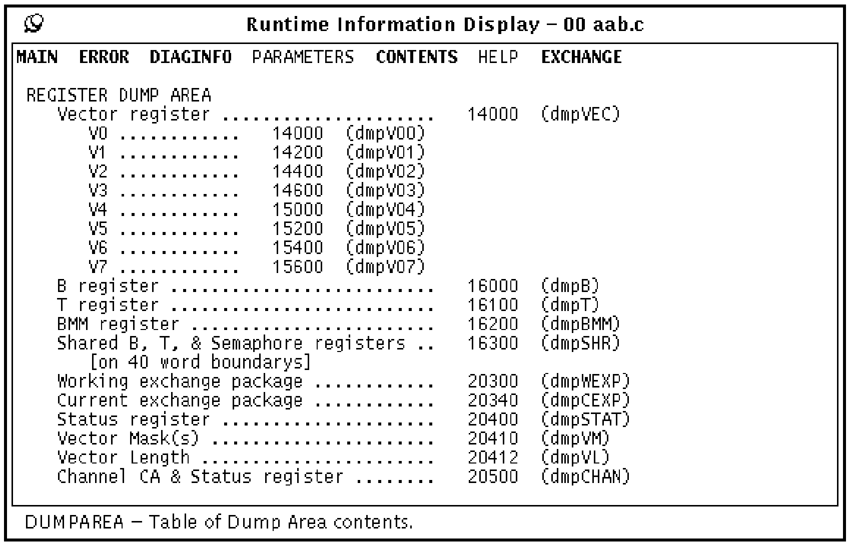
- Click on **`CODE BLOCK`**. MME displays the code block information for the `Runtime Information Display`:



```
Runtime Information Display – 00 aab.c
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE

 CODE BLOCK  (text area)
    Start of standard code block .    4000  (STDCODE)
    Interrupt router ............    4106a (iROUTER)
    Normal exit router ...........    4300a (nROUTER)
    Start of program ............    5000a (MAIN)
    Section  1 ..................    5010a (SEC1)
    Section  2 ..................    5200a (SEC2)
    Section  3 ..................    5520a (SEC3)
    Section  4 ..................    6070a (SEC4)
    Section  5 ..................    7410a (SEC5)
    Section  6 ..................    7460a (SEC6)
    Section  7 ..................    7520a (SEC7)
    Section 10 ..................    7560a (SEC10)
    Section 11 ..................    7630a (SEC11)
    Subroutines .................   10010a (CODESUB)
    End of diag code ............   10100  (CODEEND)

 SECTION – Table of CODE BLOCK contents.
```

The `SECTION` display shows a table of the code block contents. The locations are shown for each part of the code.

- Click on **`Register dump area`**. MME displays the register dump information for the `Runtime Information Display`.

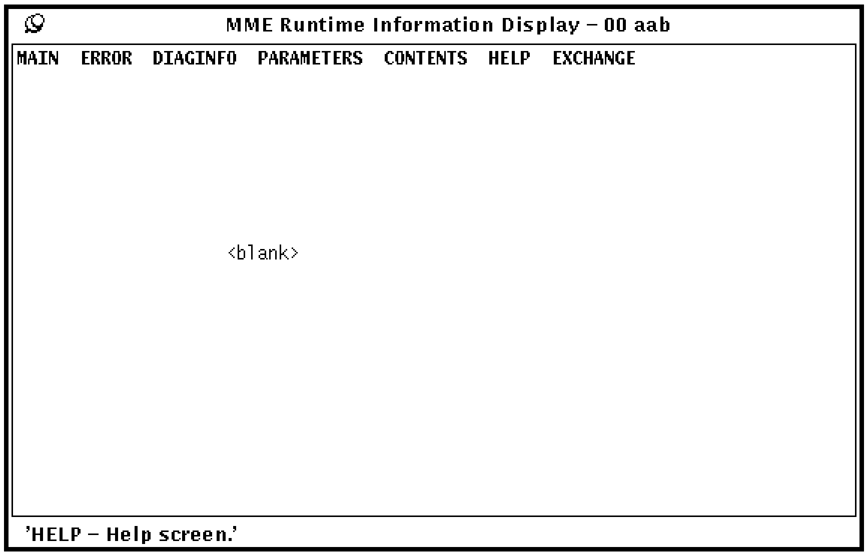

Runtime Information Display – 00 aab.c
MAIN ERROR DIAGINFO PARAMETERS CONTENTS HELP EXCHANGE

```
REGISTER DUMP AREA
   Vector register .....................   14000  (dmpVEC)
      V0 ............   14000  (dmpV00)
      V1 ............   14200  (dmpV01)
      V2 ............   14400  (dmpV02)
      V3 ............   14600  (dmpV03)
      V4 ............   15000  (dmpV04)
      V5 ............   15200  (dmpV05)
      V6 ............   15400  (dmpV06)
      V7 ............   15600  (dmpV07)
   B register ..........................   16000  (dmpB)
   T register ..........................   16100  (dmpT)
   BMM register ........................   16200  (dmpBMM)
   Shared B, T, & Semaphore registers ..   16300  (dmpSHR)
      [on 40 word boundarys]
   Working exchange package ............   20300  (dmpWEXP)
   Current exchange package ............   20340  (dmpCEXP)
   Status register .....................   20400  (dmpSTAT)
   Vector Mask(s) ......................   20410  (dmpVM)
   Vector Length .......................   20412  (dmpVL)
   Channel CA & Status register ........   20500  (dmpCHAN)
```

DUMPAREA – Table of Dump Area contents.

The DUMPAREA display shows a table of the dump area contents. The memory locations of the dumps are shown.

- Click on **HELP**. MME displays the help information for the `MME Runtime Information Display`:

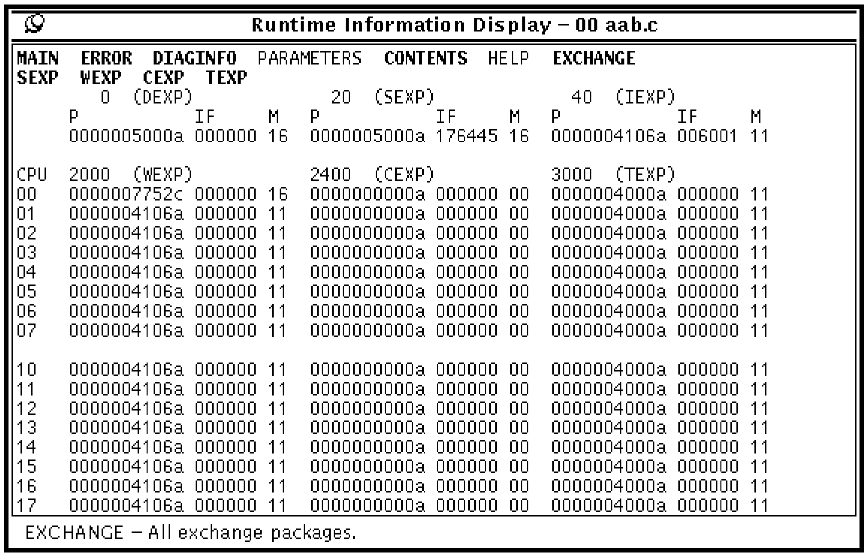

MME Runtime Information Display – 00 aab
MAIN ERROR DIAGINFO PARAMETERS CONTENTS HELP EXCHANGE

```
<blank>
```

'HELP – Help screen.'

The `HELP` display shows help information if any is available.

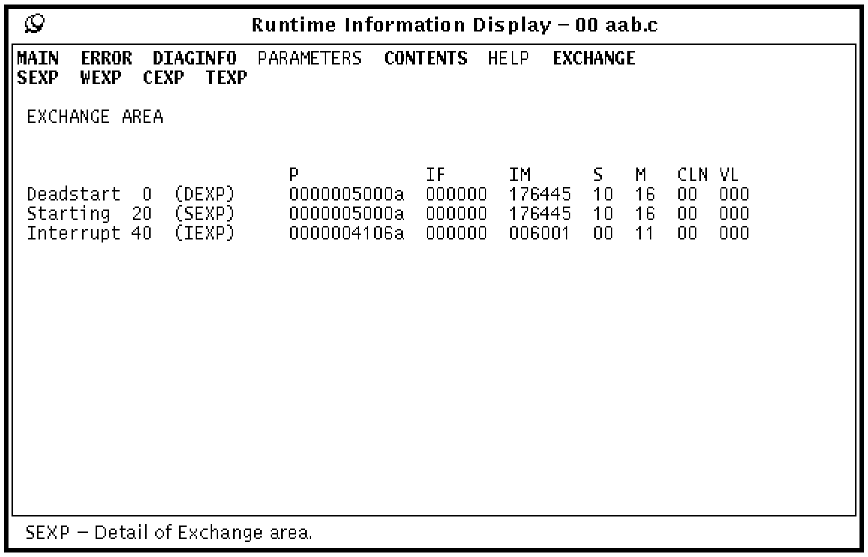
- Click on **EXCHANGE**. MME displays the exchange packages available for the `MME Runtime Information Display`.



```
Runtime Information Display – 00 aab.c
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
SEXP  WEXP  CEXP  TEXP
        0  (DEXP)            20  (SEXP)            40  (IEXP)
     P            IF     M   P            IF     M   P            IF     M
     0000005000a 000000 16  0000005000a 176445 16  0000004106a 006001 11

CPU  2000  (WEXP)           2400  (CEXP)           3000  (TEXP)
00   0000007752c 000000 16  0000000000a 000000 00  0000004000a 000000 11
01   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
02   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
03   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
04   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
05   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
06   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
07   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11

10   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
11   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
12   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
13   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
14   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
15   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
16   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
17   0000004106a 000000 11  0000000000a 000000 00  0000004000a 000000 11
EXCHANGE – All exchange packages.
```

The EXCHANGE display gives an overview of the information contained in exchange packages. Click on the new set of bold headings (**SEXP**, **WEXP**, **CEXP**, **TEXP**) to get more information about the individual exchange packages. The following information is displayed when you perform these actions:

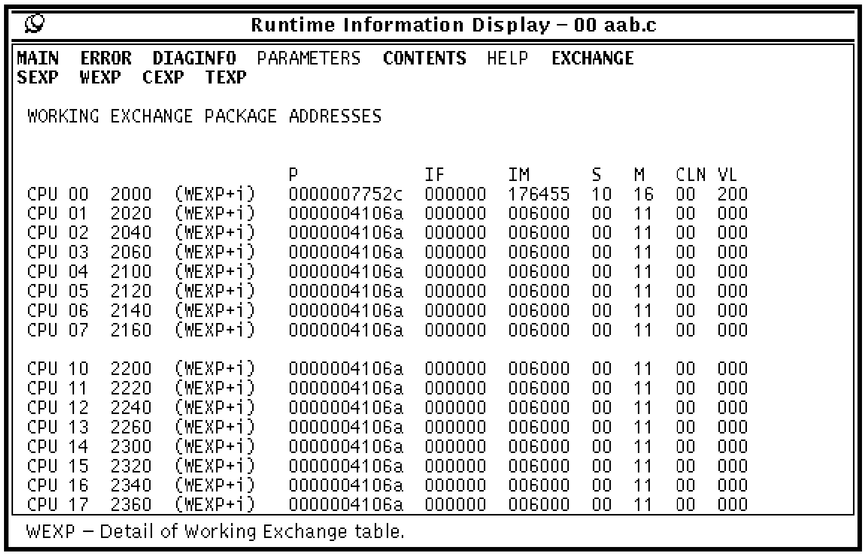
- Click on **SEXP**. MME displays the starting exchange package for the MME Runtime Information Display:



```
Runtime Information Display – 00 aab.c
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
SEXP  WEXP  CEXP  TEXP

 EXCHANGE AREA


                            P            IF      IM      S    M    CLN  VL
 Deadstart  0   (DEXP)      0000005000a  000000  176445  10   16   00   000
 Starting  20   (SEXP)      0000005000a  000000  176445  10   16   00   000
 Interrupt 40   (IEXP)      0000004106a  000000  006001  00   11   00   000

SEXP – Detail of Exchange area.
```

The SEXP display provides a detailed description of the exchange area.

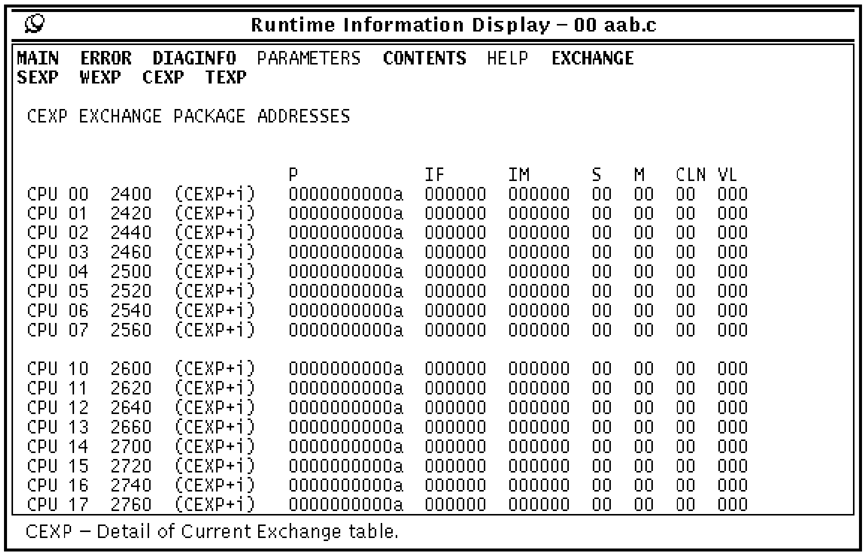
- Click on **WEXP**. MME displays the working exchange package for the `MME Runtime Information Display`:



```
Runtime Information Display – 00 aab.c

MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
SEXP  WEXP  CEXP  TEXP

 WORKING EXCHANGE PACKAGE ADDRESSES


                           P            IF      IM      S   M   CLN  VL
 CPU 00  2000  (WEXP+i)   0000007752c  000000  176455  10  16  00  200
 CPU 01  2020  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 02  2040  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 03  2060  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 04  2100  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 05  2120  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 06  2140  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 07  2160  (WEXP+i)   0000004106a  000000  006000  00  11  00  000

 CPU 10  2200  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 11  2220  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 12  2240  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 13  2260  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 14  2300  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 15  2320  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 16  2340  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 CPU 17  2360  (WEXP+i)   0000004106a  000000  006000  00  11  00  000
 WEXP – Detail of Working Exchange table.
```

The `WEXP` display provides a detailed description of the working exchange table.

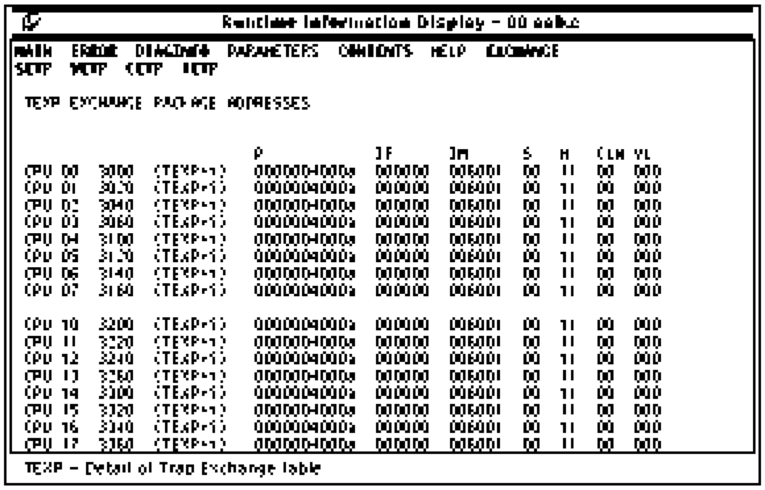
- Click on **CEXP**. MME displays the current exchange package for the `MME Runtime Information Display`:



```
Runtime Information Display – 00 aab.c

MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
SEXP  WEXP  CEXP  TEXP

 CEXP EXCHANGE PACKAGE ADDRESSES


                           P            IF      IM      S   M   CLN  VL
 CPU 00  2400  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 01  2420  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 02  2440  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 03  2460  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 04  2500  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 05  2520  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 06  2540  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 07  2560  (CEXP+i)   0000000000a  000000  000000  00  00  00  000

 CPU 10  2600  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 11  2620  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 12  2640  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 13  2660  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 14  2700  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 15  2720  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 16  2740  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CPU 17  2760  (CEXP+i)   0000000000a  000000  000000  00  00  00  000
 CEXP – Detail of Current Exchange table.
```

The CEXP display provides a detailed description of the current exchange table.

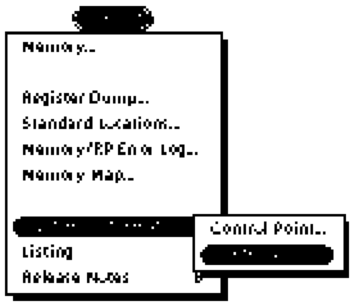
- Click on **TEXP**. MME displays the trap exchange package for the `MME Runtime Information Display`:



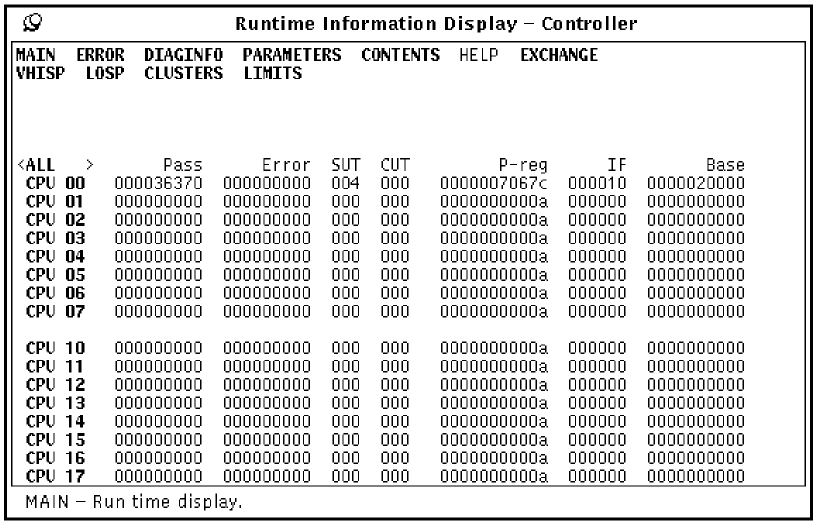
```
Runtime Information Display - 00 aabc
MAIN   ERROR   DIAGINFO   PARAMETERS   COMMENTS   HELP   EXCHANGE
SEXP   MEXP   CEXP   TEXP

 TEXP EXCHANGE PACKAGE ADDRESSES


                          P            IF       JM       S    M    CLN VL
 CPU 00   3000  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 01   3020  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 02   3040  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 03   3060  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 04   3100  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 05   3120  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 06   3140  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 07   3160  (TEXP+1)  0000004000a  000000   006001   00   11   00  000

 CPU 10   3200  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 11   3220  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 12   3240  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 13   3260  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 14   3300  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 15   3320  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 16   3340  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 CPU 17   3360  (TEXP+1)  0000004000a  000000   006001   00   11   00  000
 TEXP - Detail of Trap Exchange Table
```

The TEXP table provides a detailed description of the trap exchange table.

Viewing the Controller Runtime Information Display in Environment 2



To view the control point `Runtime Information Display`, choose **`View --> Runtime Information --> Control Point`**, as shown at the left. MME displays the `MAIN` runtime display.

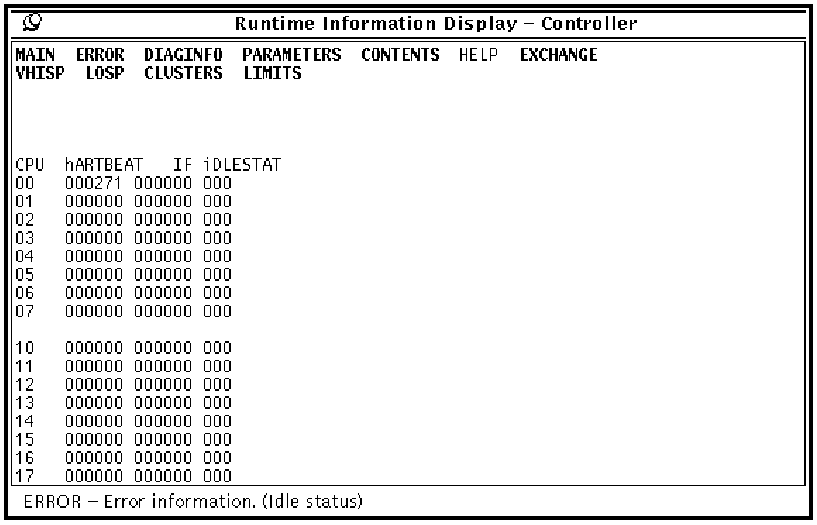


```
Runtime Information Display – Controller
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
VHISP  LOSP  CLUSTERS  LIMITS

<ALL   >       Pass      Error  SUT  CUT       P-reg      IF       Base
 CPU 00  000036370  000000000  004  000  0000007067c  000010  0000020000
 CPU 01  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 02  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 03  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 04  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 05  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 06  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 07  000000000  000000000  000  000  0000000000a  000000  0000000000

 CPU 10  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 11  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 12  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 13  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 14  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 15  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 16  000000000  000000000  000  000  0000000000a  000000  0000000000
 CPU 17  000000000  000000000  000  000  0000000000a  000000  0000000000
 MAIN – Run time display.
```

The MAIN runtime display shows the runtime information for all CPUs.

You can switch runtime information displays by clicking on the bold headings on the current display. Perform the following actions to view the different displays:

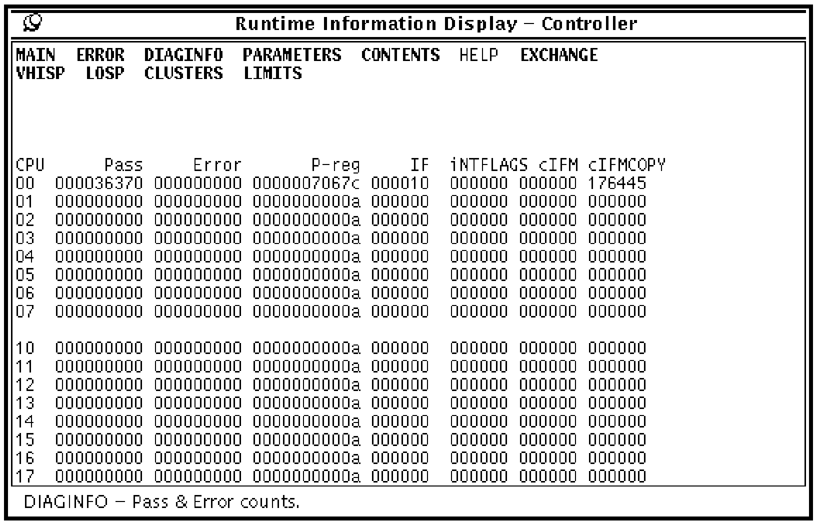
- Click on **ERROR**. MME displays the error information on the `MME Runtime Information Display`:



```
Runtime Information Display – Controller
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
VHISP  LOSP  CLUSTERS  LIMITS

CPU  hARTBEAT   IF iDLESTAT
00   000271 000000 000
01   000000 000000 000
02   000000 000000 000
03   000000 000000 000
04   000000 000000 000
05   000000 000000 000
06   000000 000000 000
07   000000 000000 000

10   000000 000000 000
11   000000 000000 000
12   000000 000000 000
13   000000 000000 000
14   000000 000000 000
15   000000 000000 000
16   000000 000000 000
17   000000 000000 000
 ERROR – Error information. (Idle status)
```

The ERROR display shows information for each CPU.

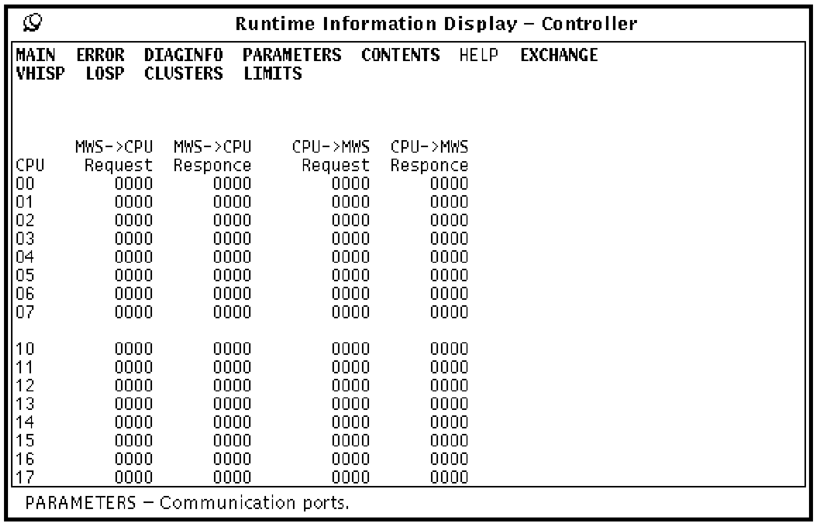
- Click on **DIAGINFO**. MME displays the diagnostic error information on the `MME Runtime Information Display`:



```
Runtime Information Display – Controller
MAIN   ERROR   DIAGINFO   PARAMETERS   CONTENTS   HELP   EXCHANGE
VHISP  LOSP    CLUSTERS   LIMITS

CPU      Pass      Error       P-reg        IF    iNTFLAGS cIFM  cIFMCOPY
00   000036370 000000000 0000007067c 000010   000000 000000 176445
01   000000000 000000000 0000000000a 000000   000000 000000 000000
02   000000000 000000000 0000000000a 000000   000000 000000 000000
03   000000000 000000000 0000000000a 000000   000000 000000 000000
04   000000000 000000000 0000000000a 000000   000000 000000 000000
05   000000000 000000000 0000000000a 000000   000000 000000 000000
06   000000000 000000000 0000000000a 000000   000000 000000 000000
07   000000000 000000000 0000000000a 000000   000000 000000 000000

10   000000000 000000000 0000000000a 000000   000000 000000 000000
11   000000000 000000000 0000000000a 000000   000000 000000 000000
12   000000000 000000000 0000000000a 000000   000000 000000 000000
13   000000000 000000000 0000000000a 000000   000000 000000 000000
14   000000000 000000000 0000000000a 000000   000000 000000 000000
15   000000000 000000000 0000000000a 000000   000000 000000 000000
16   000000000 000000000 0000000000a 000000   000000 000000 000000
17   000000000 000000000 0000000000a 000000   000000 000000 000000
DIAGINFO – Pass & Error counts.
```

The `DIAGINFO` display shows pass and error counts for each CPU.

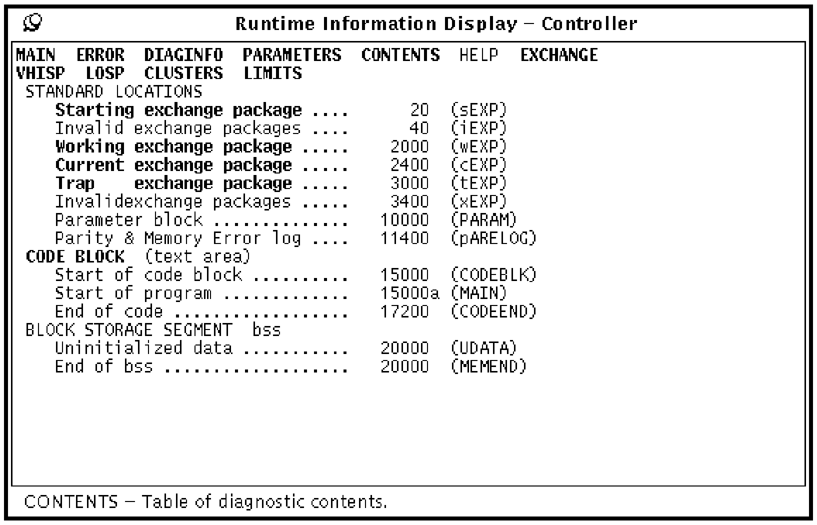
- Click on **PARAMETERS**. MME displays the parameter information on the `MME Runtime Information Display`:



```
Runtime Information Display – Controller
MAIN   ERROR   DIAGINFO   PARAMETERS   CONTENTS   HELP   EXCHANGE
VHISP  LOSP    CLUSTERS   LIMITS

      MWS->CPU  MWS->CPU    CPU->MWS  CPU->MWS
CPU    Request  Responce     Request  Responce
00        0000      0000        0000      0000
01        0000      0000        0000      0000
02        0000      0000        0000      0000
03        0000      0000        0000      0000
04        0000      0000        0000      0000
05        0000      0000        0000      0000
06        0000      0000        0000      0000
07        0000      0000        0000      0000

10        0000      0000        0000      0000
11        0000      0000        0000      0000
12        0000      0000        0000      0000
13        0000      0000        0000      0000
14        0000      0000        0000      0000
15        0000      0000        0000      0000
16        0000      0000        0000      0000
17        0000      0000        0000      0000
PARAMETERS – Communication ports.
```

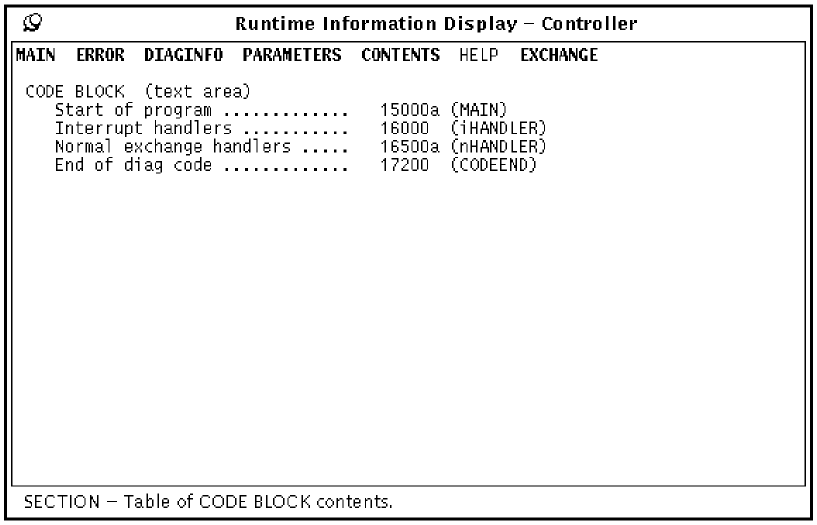
The `PARAMETERS` display shows the parameters for the communication ports.

- Click on **CONTENTS**. MME displays the diagnostic contents information on the `MME Runtime Information Display`:


```
Runtime Information Display – Controller
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
VHISP  LOSP  CLUSTERS  LIMITS
 STANDARD LOCATIONS
    Starting exchange package ....      20  (sEXP)
    Invalid exchange packages ....      40  (iEXP)
    Working exchange package .....    2000  (wEXP)
    Current exchange package .....    2400  (cEXP)
    Trap    exchange package .....    3000  (tEXP)
    Invalidexchange packages .....    3400  (xEXP)
    Parameter block .............   10000  (PARAM)
    Parity & Memory Error log ....   11400  (pARELOG)
 CODE BLOCK  (text area)
    Start of code block ..........   15000  (CODEBLK)
    Start of program .............   15000a (MAIN)
    End of code ..................   17200  (CODEEND)
 BLOCK STORAGE SEGMENT  bss
    Uninitialized data ...........   20000  (UDATA)
    End of bss ...................   20000  (MEMEND)

 CONTENTS – Table of diagnostic contents.
```

The `CONTENTS` display shows a table of diagnostic contents. You can click on the bold entries to switch to different displays. The starting exchange package, working exchange package, current exchange package, and trap exchange package are described as the `sEXP`, `wEXP` `cEXP`, and `tEXP` displays, respectively, with the `EXCHANGE` display later in this subsection. To see the `CODE BLOCK`:

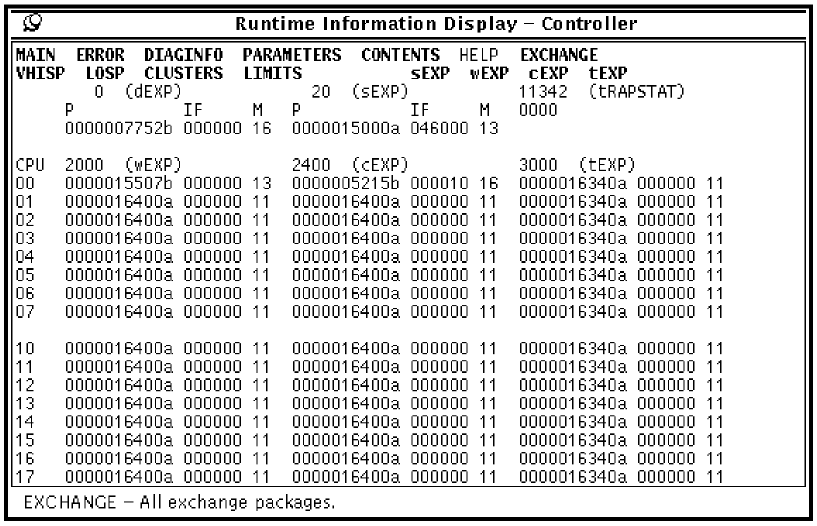
- Click on **CODE BLOCK**. MME displays the code block information for the `Runtime Information Display`.



```
Runtime Information Display – Controller
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE

 CODE BLOCK  (text area)
    Start of program .............   15000a (MAIN)
    Interrupt handlers ...........   16000  (iHANDLER)
    Normal exchange handlers .....   16500a (nHANDLER)
    End of diag code .............   17200  (CODEEND)

 SECTION – Table of CODE BLOCK contents.
```

The `SECTION` display shows a table of the code block contents. The locations are shown for each part of the code.

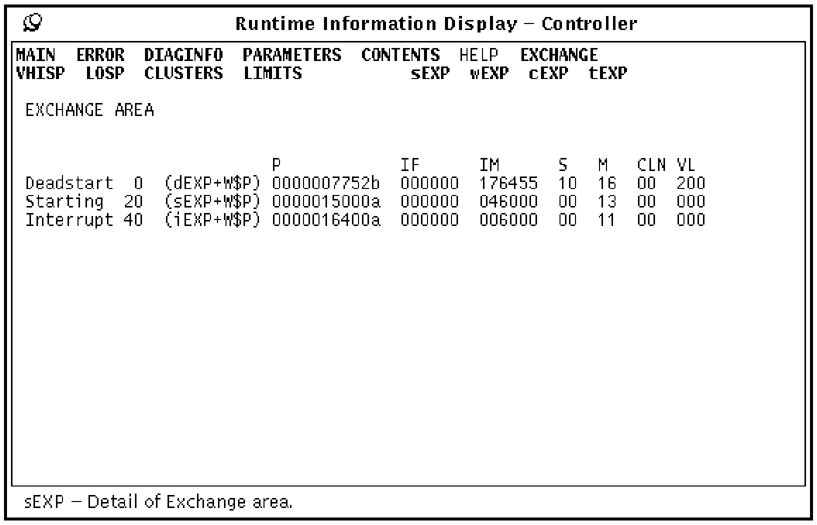
- Click on **`EXCHANGE`**. MME displays the exchange package information on the `MME Runtime Information Display`:



```
Runtime Information Display – Controller
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
VHISP  LOSP  CLUSTERS  LIMITS            sEXP  wEXP  cEXP  tEXP
        0  (dEXP)               20  (sEXP)            11342  (tRAPSTAT)
     P          IF     M   P          IF     M    0000
     0000007752b 000000 16  0000015000a 046000 13

CPU  2000  (wEXP)           2400  (cEXP)            3000  (tEXP)
00   0000015507b 000000 13  0000005215b 000010 16   0000016340a 000000 11
01   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
02   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
03   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
04   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
05   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
06   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
07   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11

10   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
11   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
12   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
13   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
14   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
15   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
16   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
17   0000016400a 000000 11  0000016400a 000000 11   0000016340a 000000 11
 EXCHANGE – All exchange packages.
```

The `EXCHANGE` display gives an overview of the information contained in exchange packages. Click on the new set of bold headings (**`sEXP`**, **`wEXP`**, **`cEXP`**, **`tEXP`**) to get more information about the individual exchange packages. The following information is displayed when you perform these actions.

- Click on **sEXP**. MME displays the starting exchange package for the `MME Runtime Information Display`:

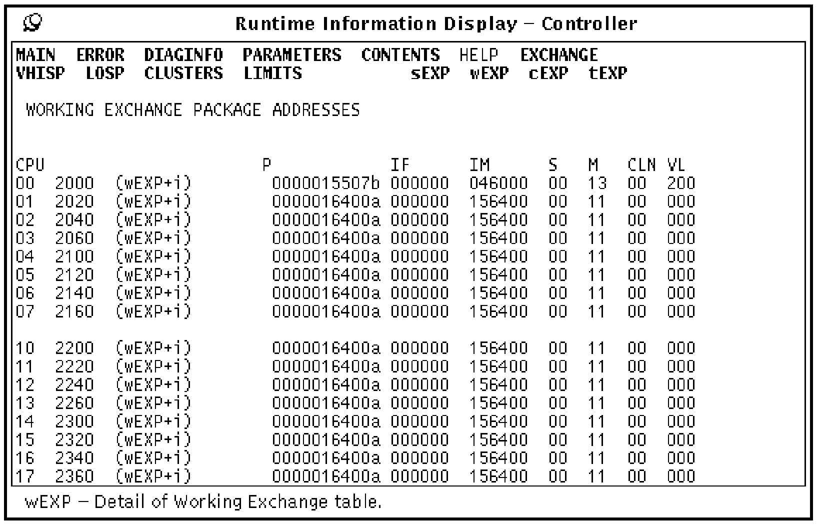

Runtime Information Display – Controller
MAIN ERROR DIAGINFO PARAMETERS CONTENTS HELP EXCHANGE
VHISP LOSP CLUSTERS LIMITS sEXP wEXP cEXP tEXP

```
 EXCHANGE AREA


                          P            IF      IM      S    M    CLN  VL
 Deadstart  0  (dEXP+W$P) 0000007752b  000000  176455  10   16   00   200
 Starting  20  (sEXP+W$P) 0000015000a  000000  046000  00   13   00   000
 Interrupt 40  (iEXP+W$P) 0000016400a  000000  006000  00   11   00   000
```

sEXP – Detail of Exchange area.

The sEXP display provides a detailed description of the exchange area.

- Click on **wEXP**. MME displays the working exchange package for the `MME Runtime Information Display`:

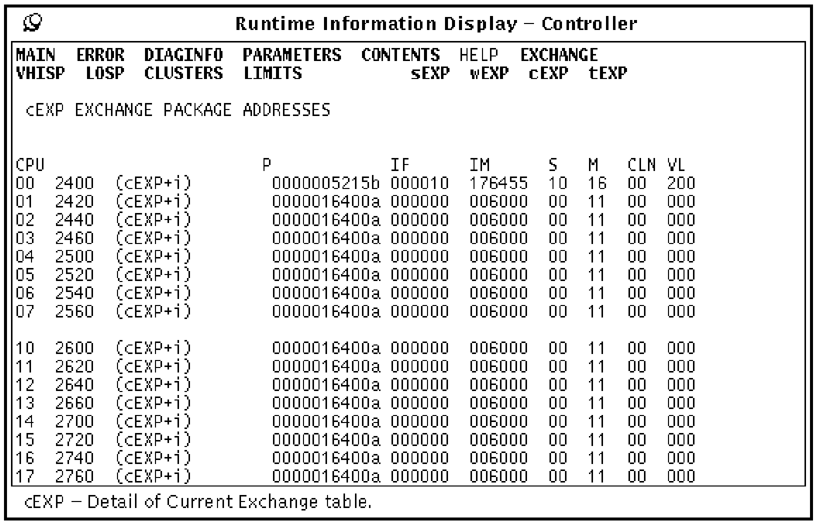

Runtime Information Display – Controller
MAIN ERROR DIAGINFO PARAMETERS CONTENTS HELP EXCHANGE
VHISP LOSP CLUSTERS LIMITS sEXP wEXP cEXP tEXP

```
 WORKING EXCHANGE PACKAGE ADDRESSES


CPU                      P            IF      IM      S    M    CLN  VL
00  2000  (wEXP+i)        0000015507b 000000  046000  00   13   00   200
01  2020  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
02  2040  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
03  2060  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
04  2100  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
05  2120  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
06  2140  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
07  2160  (wEXP+i)        0000016400a 000000  156400  00   11   00   000

10  2200  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
11  2220  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
12  2240  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
13  2260  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
14  2300  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
15  2320  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
16  2340  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
17  2360  (wEXP+i)        0000016400a 000000  156400  00   11   00   000
```

wEXP – Detail of Working Exchange table.

The wEXP display provides a detailed description of the working exchange table.

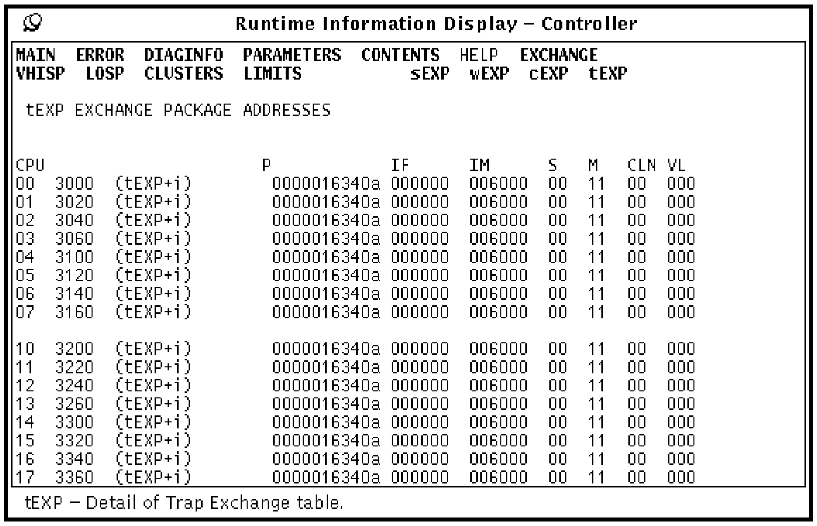
- Click on **cEXP**. MME displays the current exchange package for the MME Runtime Information Display:



```
Runtime Information Display – Controller
MAIN   ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
VHISP  LOSP   CLUSTERS  LIMITS           sEXP  wEXP  cEXP  tEXP

 cEXP EXCHANGE PACKAGE ADDRESSES


CPU                         P            IF      IM      S   M   CLN VL
00  2400  (cEXP+i)         0000005215b 000010  176455  10  16  00  200
01  2420  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
02  2440  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
03  2460  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
04  2500  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
05  2520  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
06  2540  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
07  2560  (cEXP+i)         0000016400a 000000  006000  00  11  00  000

10  2600  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
11  2620  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
12  2640  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
13  2660  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
14  2700  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
15  2720  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
16  2740  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
17  2760  (cEXP+i)         0000016400a 000000  006000  00  11  00  000
 cEXP – Detail of Current Exchange table.
```

The cEXP display provides a detailed description of the current exchange table.

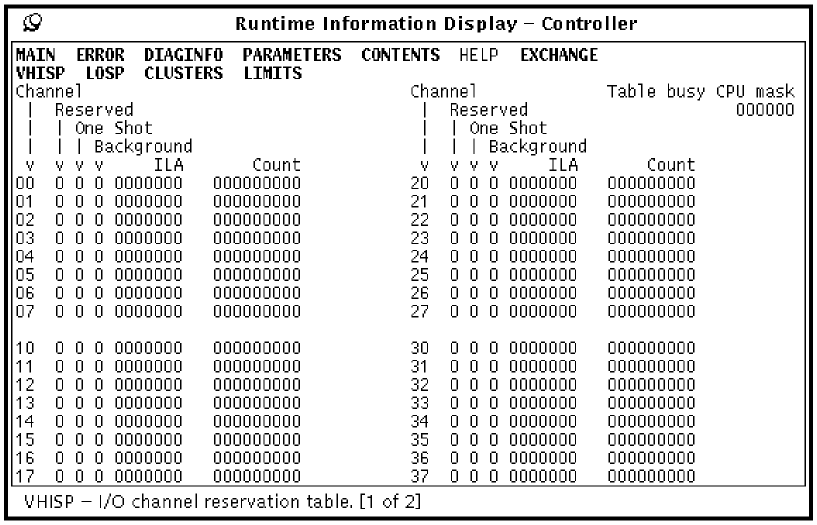
- Click on **tEXP**. MME displays the trap exchange package for the MME Runtime Information Display:



```
Runtime Information Display – Controller
MAIN   ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
VHISP  LOSP   CLUSTERS  LIMITS           sEXP  wEXP  cEXP  tEXP

 tEXP EXCHANGE PACKAGE ADDRESSES


CPU                         P            IF      IM      S   M   CLN VL
00  3000  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
01  3020  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
02  3040  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
03  3060  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
04  3100  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
05  3120  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
06  3140  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
07  3160  (tEXP+i)         0000016340a 000000  006000  00  11  00  000

10  3200  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
11  3220  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
12  3240  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
13  3260  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
14  3300  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
15  3320  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
16  3340  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
17  3360  (tEXP+i)         0000016340a 000000  006000  00  11  00  000
 tEXP – Detail of Trap Exchange table.
```

The tEXP table provides a detailed description of the trap exchange table.

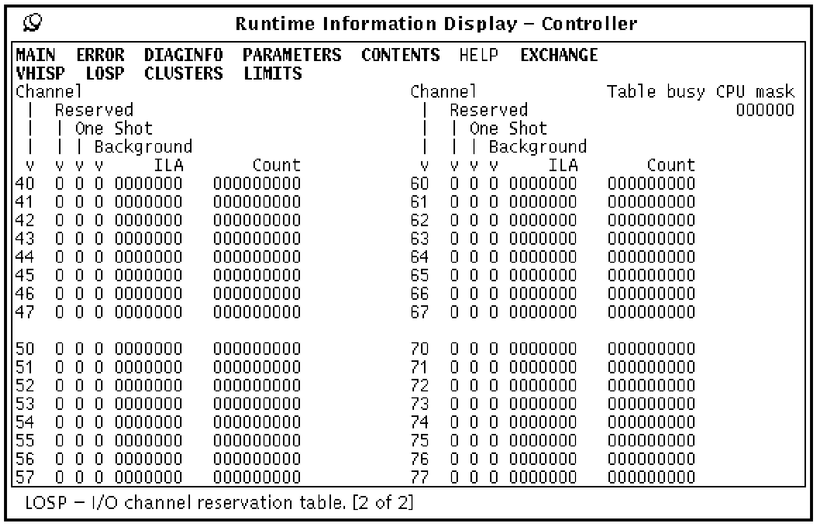
- Click on **`VHISP`**. MME displays the VHISP information on the `MME Runtime Information Display`:


Runtime Information Display – Controller
MAIN ERROR DIAGINFO PARAMETERS CONTENTS HELP EXCHANGE
VHISP LOSP CLUSTERS LIMITS

```
Channel                                           Channel             Table busy CPU mask
 |  Reserved                                       |  Reserved                   000000
 |  | One Shot                                     |  | One Shot
 |  | | Background                                 |  | | Background
 v  v v v      ILA        Count                    v  v v v      ILA        Count
00  0 0 0 0000000    000000000                    20  0 0 0 0000000    000000000
01  0 0 0 0000000    000000000                    21  0 0 0 0000000    000000000
02  0 0 0 0000000    000000000                    22  0 0 0 0000000    000000000
03  0 0 0 0000000    000000000                    23  0 0 0 0000000    000000000
04  0 0 0 0000000    000000000                    24  0 0 0 0000000    000000000
05  0 0 0 0000000    000000000                    25  0 0 0 0000000    000000000
06  0 0 0 0000000    000000000                    26  0 0 0 0000000    000000000
07  0 0 0 0000000    000000000                    27  0 0 0 0000000    000000000

10  0 0 0 0000000    000000000                    30  0 0 0 0000000    000000000
11  0 0 0 0000000    000000000                    31  0 0 0 0000000    000000000
12  0 0 0 0000000    000000000                    32  0 0 0 0000000    000000000
13  0 0 0 0000000    000000000                    33  0 0 0 0000000    000000000
14  0 0 0 0000000    000000000                    34  0 0 0 0000000    000000000
15  0 0 0 0000000    000000000                    35  0 0 0 0000000    000000000
16  0 0 0 0000000    000000000                    36  0 0 0 0000000    000000000
17  0 0 0 0000000    000000000                    37  0 0 0 0000000    000000000
 VHISP – I/O channel reservation table. [1 of 2]
```

The `VHISP` display shows the I/O channel reservation table for the VHISP channels.

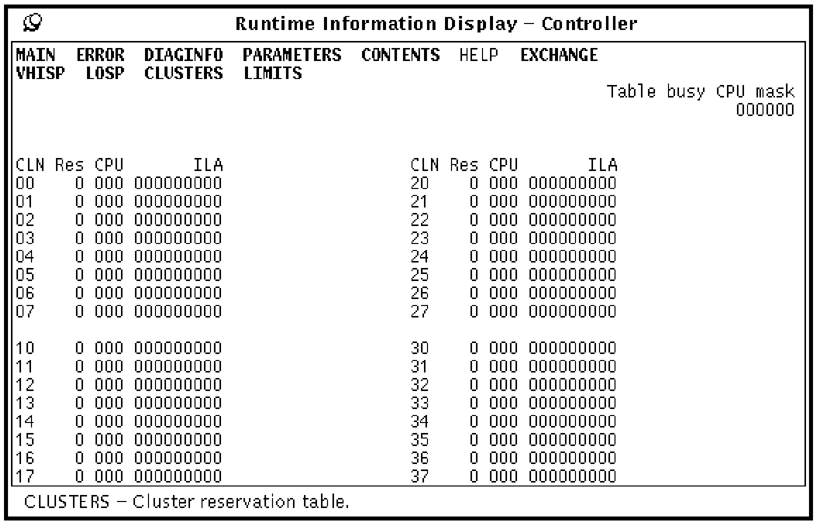
- Click on **`LOSP`**. MME displays the LOSP information on the `MME Runtime Information Display`:


Runtime Information Display – Controller
MAIN ERROR DIAGINFO PARAMETERS CONTENTS HELP EXCHANGE
VHISP LOSP CLUSTERS LIMITS

```
Channel                                           Channel             Table busy CPU mask
 |  Reserved                                       |  Reserved                   000000
 |  | One Shot                                     |  | One Shot
 |  | | Background                                 |  | | Background
 v  v v v      ILA        Count                    v  v v v      ILA        Count
40  0 0 0 0000000    000000000                    60  0 0 0 0000000    000000000
41  0 0 0 0000000    000000000                    61  0 0 0 0000000    000000000
42  0 0 0 0000000    000000000                    62  0 0 0 0000000    000000000
43  0 0 0 0000000    000000000                    63  0 0 0 0000000    000000000
44  0 0 0 0000000    000000000                    64  0 0 0 0000000    000000000
45  0 0 0 0000000    000000000                    65  0 0 0 0000000    000000000
46  0 0 0 0000000    000000000                    66  0 0 0 0000000    000000000
47  0 0 0 0000000    000000000                    67  0 0 0 0000000    000000000

50  0 0 0 0000000    000000000                    70  0 0 0 0000000    000000000
51  0 0 0 0000000    000000000                    71  0 0 0 0000000    000000000
52  0 0 0 0000000    000000000                    72  0 0 0 0000000    000000000
53  0 0 0 0000000    000000000                    73  0 0 0 0000000    000000000
54  0 0 0 0000000    000000000                    74  0 0 0 0000000    000000000
55  0 0 0 0000000    000000000                    75  0 0 0 0000000    000000000
56  0 0 0 0000000    000000000                    76  0 0 0 0000000    000000000
57  0 0 0 0000000    000000000                    77  0 0 0 0000000    000000000
 LOSP – I/O channel reservation table. [2 of 2]
```

The `LOSP` display shows the I/O channel reservation table for the LOSP channels.

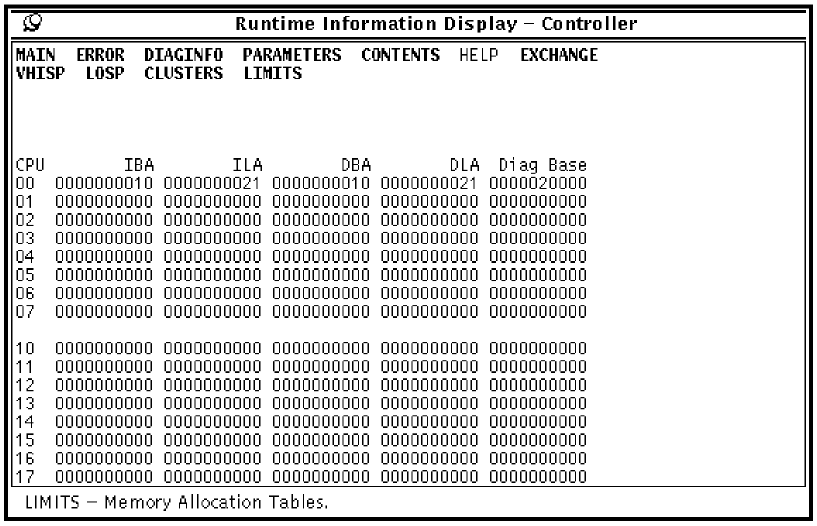
- Click on **`CLUSTERS`**. MME displays the cluster information on the `MME Runtime Information Display`:



```
Runtime Information Display – Controller
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
VHISP  LOSP  CLUSTERS  LIMITS
                                                          Table busy CPU mask
                                                                       000000


CLN Res CPU       ILA                    CLN Res CPU       ILA
00    0 000 000000000                    20    0 000 000000000
01    0 000 000000000                    21    0 000 000000000
02    0 000 000000000                    22    0 000 000000000
03    0 000 000000000                    23    0 000 000000000
04    0 000 000000000                    24    0 000 000000000
05    0 000 000000000                    25    0 000 000000000
06    0 000 000000000                    26    0 000 000000000
07    0 000 000000000                    27    0 000 000000000

10    0 000 000000000                    30    0 000 000000000
11    0 000 000000000                    31    0 000 000000000
12    0 000 000000000                    32    0 000 000000000
13    0 000 000000000                    33    0 000 000000000
14    0 000 000000000                    34    0 000 000000000
15    0 000 000000000                    35    0 000 000000000
16    0 000 000000000                    36    0 000 000000000
17    0 000 000000000                    37    0 000 000000000
CLUSTERS – Cluster reservation table.
```

The `CLUSTERS` display shows the cluster reservation table.

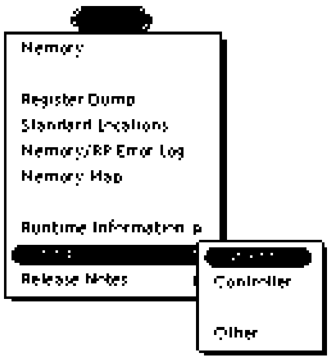
- Click on **`LIMITS`**. MME displays the base limit information on the `MME Runtime Information Display`:



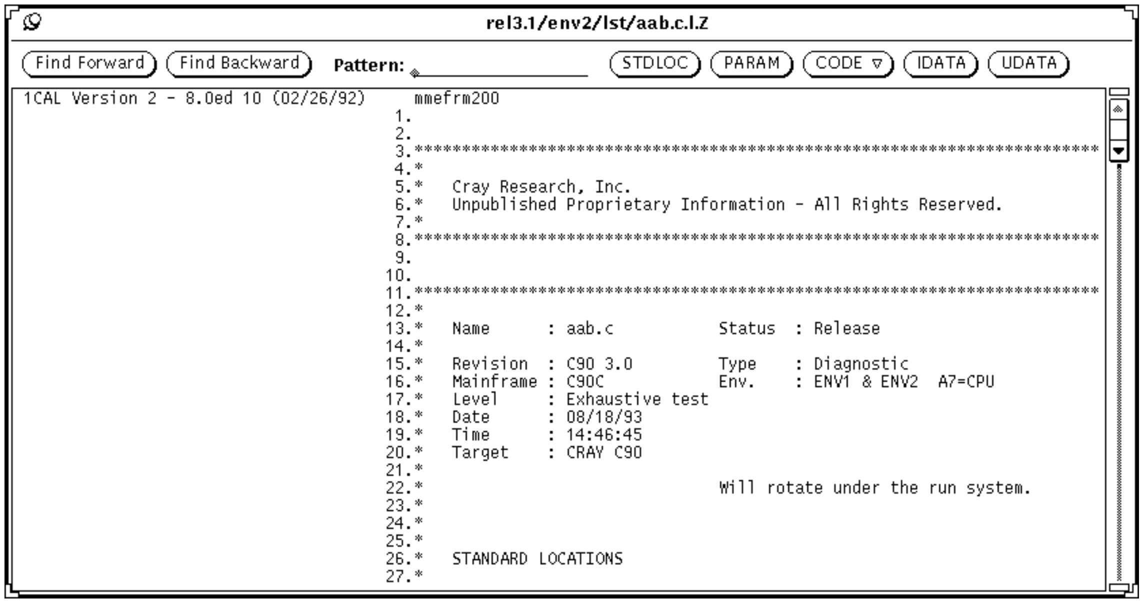
```
Runtime Information Display – Controller
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
VHISP  LOSP  CLUSTERS  LIMITS



CPU        IBA        ILA        DBA        DLA  Diag Base
00  0000000010 0000000021 0000000010 0000000021 0000020000
01  0000000000 0000000000 0000000000 0000000000 0000000000
02  0000000000 0000000000 0000000000 0000000000 0000000000
03  0000000000 0000000000 0000000000 0000000000 0000000000
04  0000000000 0000000000 0000000000 0000000000 0000000000
05  0000000000 0000000000 0000000000 0000000000 0000000000
06  0000000000 0000000000 0000000000 0000000000 0000000000
07  0000000000 0000000000 0000000000 0000000000 0000000000

10  0000000000 0000000000 0000000000 0000000000 0000000000
11  0000000000 0000000000 0000000000 0000000000 0000000000
12  0000000000 0000000000 0000000000 0000000000 0000000000
13  0000000000 0000000000 0000000000 0000000000 0000000000
14  0000000000 0000000000 0000000000 0000000000 0000000000
15  0000000000 0000000000 0000000000 0000000000 0000000000
16  0000000000 0000000000 0000000000 0000000000 0000000000
17  0000000000 0000000000 0000000000 0000000000 0000000000
LIMITS – Memory Allocation Tables.
```

The `LIMITS` display shows the memory allocation tables for the CPUs.

Viewing the Current Control Point Listing in Environment 1 or 2



From the `Mainframe Maintenance Environment`, choose **`View --> Listing --> Current`**, as shown at the left. MME displays the current control point listing:



```
1CAL Version 2 - 8.0ed 10 (02/26/92)     mmefrm200


 **************************************************************************
 *
 *   Cray Research, Inc.
 *   Unpublished Proprietary Information - All Rights Reserved.
 *
 **************************************************************************


 **************************************************************************
 *
 *   Name      : aab.c          Status  : Release
 *
 *   Revision  : C90 3.0        Type    : Diagnostic
 *   Mainframe : C90C           Env.    : ENV1 & ENV2  A7=CPU
 *   Level     : Exhaustive test
 *   Date      : 08/18/93
 *   Time      : 14:46:45
 *   Target    : CRAY C90
 *
 *                              Will rotate under the run system.
 *
 *
 *
 *   STANDARD LOCATIONS
 *
```

You can manipulate the listing by performing the following actions:

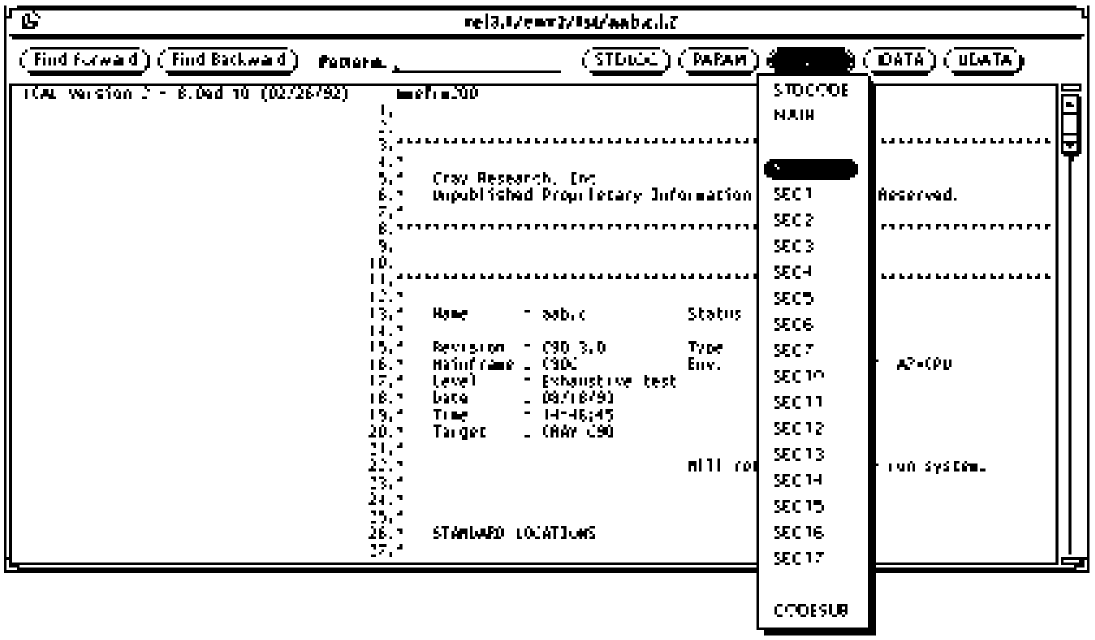
- Type the data you want to find in the `Pattern` field. Click on Find Forward to search from the current cursor location to the end of the current control point listing. MME displays the data you want to find.

- Type the data you want to find in the `Pattern` field. Click on Find Backward to search from the current cursor location to the front of the current control point listing. MME displays the data you want to find.

- Click on the scrollbar and drag the scrollbar to display the data you want.

- Click on (STDLOC) (standard code), (PARAM) (parameters), (IDATA) (initialized data), or (UDATA) (uninitialized data). MME displays the data you want.

- Choose **CODE** —> **STDLOC** (standard locations), **CODE** —> **MAIN** (location 5000 in octal), **CODE** —> **SEC*x*** (section, where *x* is the section number; for example, choose **CODE --> SEC0** to view section 0, as shown below), or **CODE --> CODESUB** (subroutine code) to view the corresponding information in the listing.

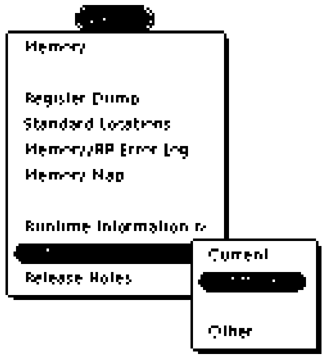


```
CAL Version 2 - 8.0ad 10 (02/26/92)      mmefile200
 *****************************************                ******************
 *
 *   Cray Research, Inc
 *   Unpublished Proprietary Information                  Reserved.
 *
 *****************************************                ******************

 *****************************************                ******************
 *
 *   Name       - aabc          Status
 *
 *   Revision   - C90 3.0       Type
 *   Mainframe  - C90C          Env.                      A2*CPU
 *   Level      - Exhaustive test
 *   Date       - 08/18/93
 *   Time       - 14:46:45
 *   Target     - CRAY C90
 *
 *                              Will ru                   run system.
 *
 *
 *
 *   STANDARD LOCATIONS
 *
```

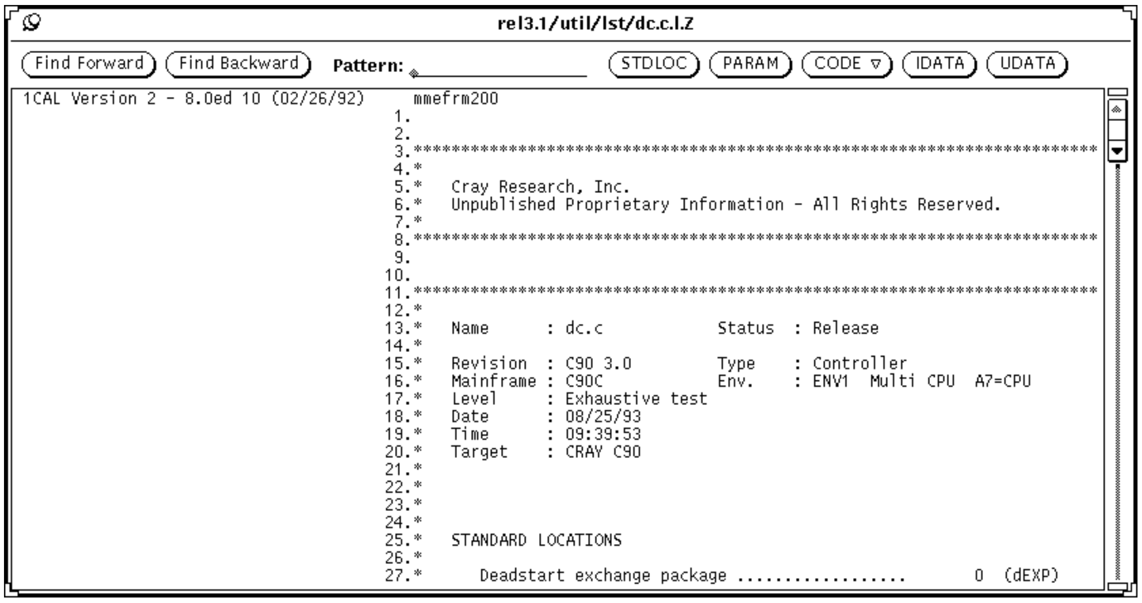
- Press the MENU mouse button in the scroll bar and choose **Split View**. This allows you to view more than one area of the listing at one time. Choose **Join Views** to return the listing to normal.

- Double click in the window header to make the window full-screen size. This allows you to view a larger portion of the listing. Double click in the window header to return the window to normal size.

Viewing the Controller Listing in Environment 1 or 2



To view the controller listing in environment 1 or 2, choose **`View --> Listing --> Controller`**, as shown at the left. The following window appears:



rel3.1/util/lst/dc.c.l.Z

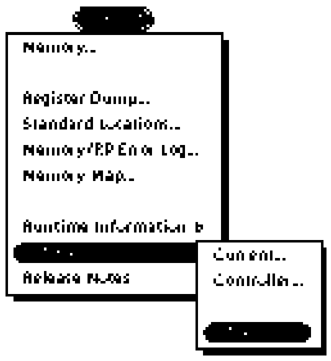
Find Forward | Find Backward | Pattern: | STDLOC | PARAM | CODE ▽ | IDATA | UDATA

```
1CAL Version 2 - 8.0ed 10 (02/26/92)     mmefrm200
                                      1.
                                      2.
                                      3.************************************************************************
                                      4.*
                                      5.*   Cray Research, Inc.
                                      6.*   Unpublished Proprietary Information - All Rights Reserved.
                                      7.*
                                      8.************************************************************************
                                      9.
                                     10.
                                     11.************************************************************************
                                     12.*
                                     13.*   Name      : dc.c            Status  : Release
                                     14.*
                                     15.*   Revision  : C90 3.0         Type    : Controller
                                     16.*   Mainframe : C90C            Env.    : ENV1  Multi CPU  A7=CPU
                                     17.*   Level     : Exhaustive test
                                     18.*   Date      : 08/25/93
                                     19.*   Time      : 09:39:53
                                     20.*   Target    : CRAY C90
                                     21.*
                                     22.*
                                     23.*
                                     24.*
                                     25.*   STANDARD LOCATIONS
                                     26.*
                                     27.*      Deadstart exchange package .................      0  (dEXP)
```

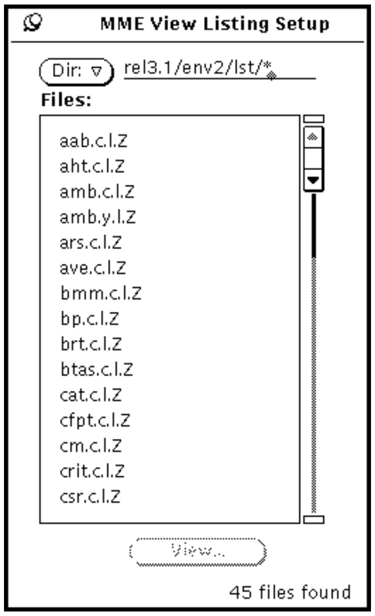
Refer to “Viewing the Current Control Point Listing in Environment 1 or 2” earlier in this section for more information about manipulating the listing.

Viewing Another Control Point Listing in Environment 1 or 2

From the `Mainframe Maintenance Environment` window, perform the following procedure to view another control point listing:



1. Choose **`View --> Listing --> Other`**, as shown at the left. MME displays the `MME View Listing Setup` window:



MME View Listing Setup

Dir: rel3.1/env2/lst/*

Files:

aab.c.l.Z
aht.c.l.Z
amb.c.l.Z
amb.y.l.Z
ars.c.l.Z
ave.c.l.Z
bmm.c.l.Z
bp.c.l.Z
brt.c.l.Z
btas.c.l.Z
cat.c.l.Z
cfpt.c.l.Z
cm.c.l.Z
crit.c.l.Z
csr.c.l.Z

View...

45 files found

NOTE: A `.y` extension means the diagnostic program was assembled in Y-MP mode, and a `.c` extension means that the diagnostic program was assembled in C90 mode. The `.Z` extension means the listing is in a compressed format.

2. Change the directory, if necessary, by performing one of the following actions:

   - Triple click on the Dir: ▽ field, type the name of the directory you want to use, and press the return (↵) key.

   - Choose the directory from the Dir: ▽ button. You can choose from the following directories:

     `Release` – Either environment 1 or 2 diagnostic program files (depending on the environment you are using) from the current release

     `User` – Diagnostic program files that the user has changed and saved

     `Alpha` – Pre-release diagnostic program files that are being tested and have not been released

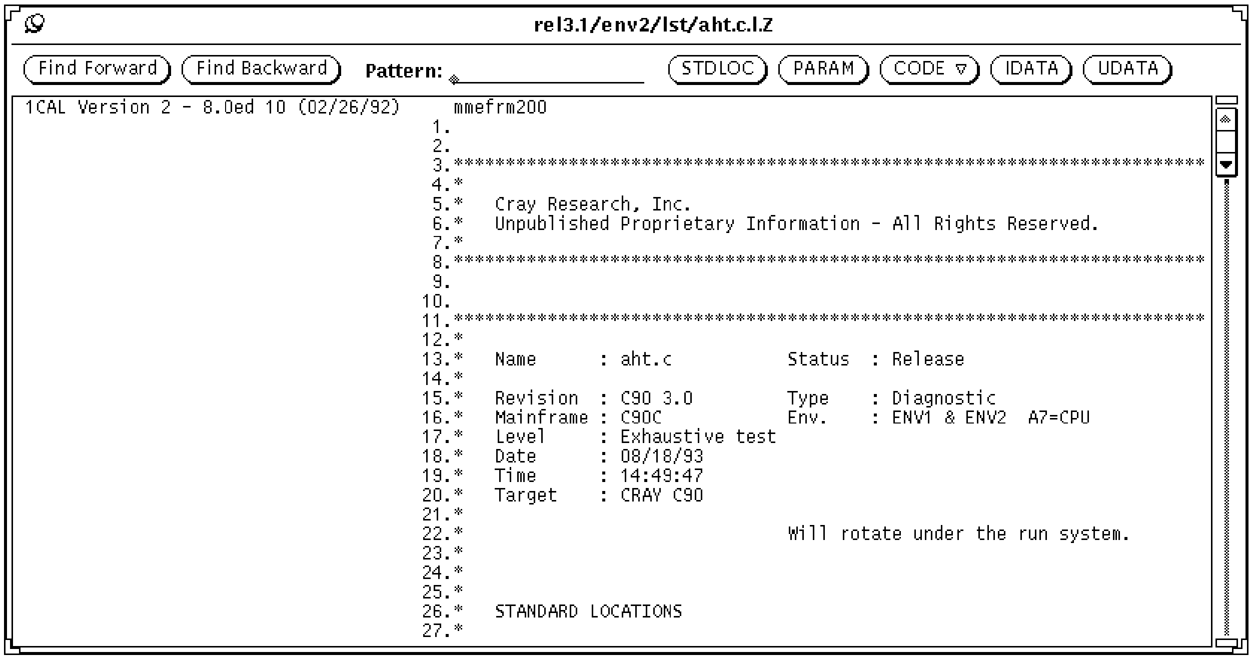
     `Utility --> Release` – Utility files from the current release

     `Utility --> Alpha` – Pre-release utility files that are being tested and have not been released

   NOTE: Files for environment 1 are in the `rel/env1/*` directory, and files for environment 2 are in the `rel/env2/*` directory. In this example, `rel` indicates that the files are from the current offline diagnostic release, `env1` specifies environment 1, `env2` specifies environment 2, and `*` specifies all the files.

3. Click on the listing you want to select from the available listings. For this example, click on the `aht.c` listing.

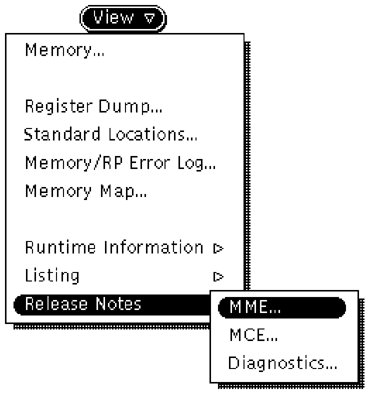
4. Click on View... . The listing appears in a window.



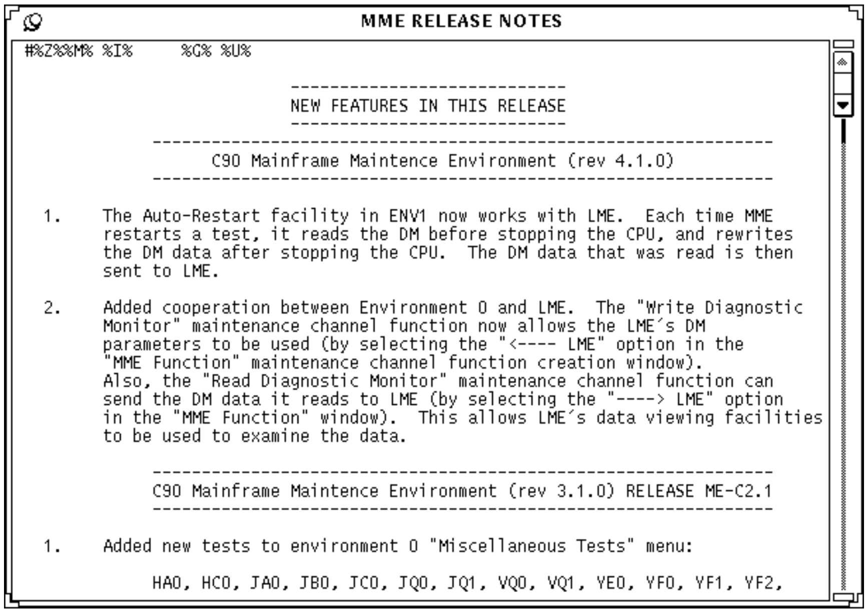
```
1CAL Version 2 - 8.0ed 10 (02/26/92)     mmefrm200
                                          1.
                                          2.
                                          3.***********************************************************************
                                          4.*
                                          5.*   Cray Research, Inc.
                                          6.*   Unpublished Proprietary Information - All Rights Reserved.
                                          7.*
                                          8.***********************************************************************
                                          9.
                                         10.
                                         11.***********************************************************************
                                         12.*
                                         13.*   Name      : aht.c            Status  : Release
                                         14.*
                                         15.*   Revision  : C90 3.0          Type    : Diagnostic
                                         16.*   Mainframe : C90C             Env.    : ENV1 & ENV2  A7=CPU
                                         17.*   Level     : Exhaustive test
                                         18.*   Date      : 08/18/93
                                         19.*   Time      : 14:49:47
                                         20.*   Target    : CRAY C90
                                         21.*
                                         22.*                                Will rotate under the run system.
                                         23.*
                                         24.*
                                         25.*
                                         26.*   STANDARD LOCATIONS
                                         27.*
```

Refer to “Viewing the Current Control Point Listing in Environment 1 or 2” earlier in this section for more information about manipulating the listing.

Viewing MME Release Notes in Environment 1 or 2

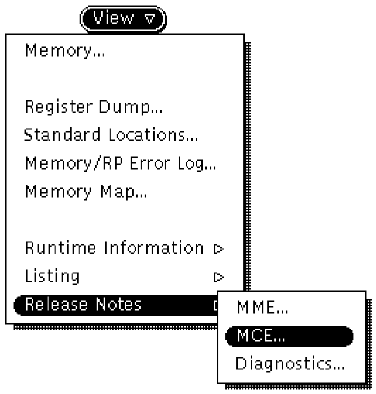


To view information about the most recently installed release of MME, choose **`View --> Release Notes --> MME`**, as shown at the left. MME displays the `MME RELEASE NOTES` window.

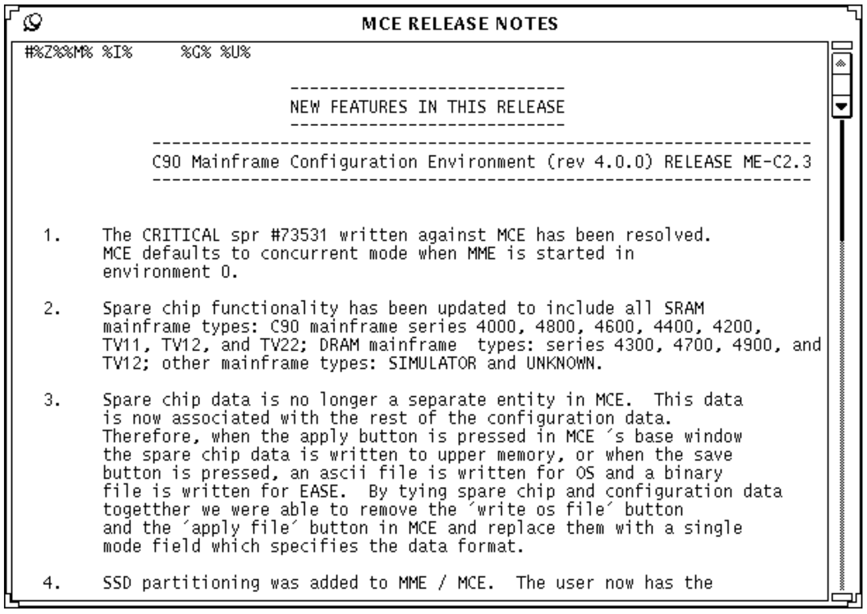


```
#%Z%%M% %I%     %G% %U%

                         ----------------------------
                         NEW FEATURES IN THIS RELEASE
                         ----------------------------
            ---------------------------------------------------------------
                  C90 Mainframe Maintence Environment (rev 4.1.0)
            ---------------------------------------------------------------

 1.     The Auto-Restart facility in ENV1 now works with LME.  Each time MME
        restarts a test, it reads the DM before stopping the CPU, and rewrites
        the DM data after stopping the CPU.  The DM data that was read is then
        sent to LME.

 2.     Added cooperation between Environment 0 and LME.  The "Write Diagnostic
        Monitor" maintenance channel function now allows the LME's DM
        parameters to be used (by selecting the "<---- LME" option in the
        "MME Function" maintenance channel function creation window).
        Also, the "Read Diagnostic Monitor" maintenance channel function can
        send the DM data it reads to LME (by selecting the "----> LME" option
        in the "MME Function" window).  This allows LME's data viewing facilities
        to be used to examine the data.

            ---------------------------------------------------------------
            C90 Mainframe Maintence Environment (rev 3.1.0) RELEASE ME-C2.1
            ---------------------------------------------------------------

 1.     Added new tests to environment 0 "Miscellaneous Tests" menu:

            HA0, HC0, JA0, JB0, JC0, JQ0, JQ1, VQ0, VQ1, YE0, YF0, YF1, YF2,
```

Viewing MCE Release Notes in Environment 1 or 2

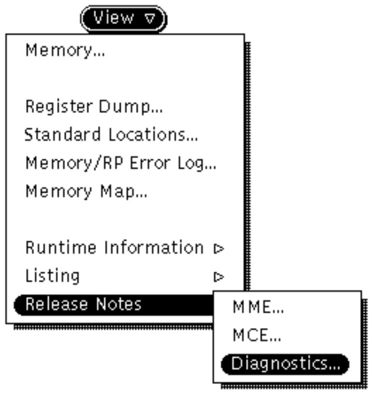


To view information about the most recently installed release of MCE, choose **`View --> Release Notes --> MCE`**, as shown at the left. MME displays the `MCE RELEASE NOTES` window.

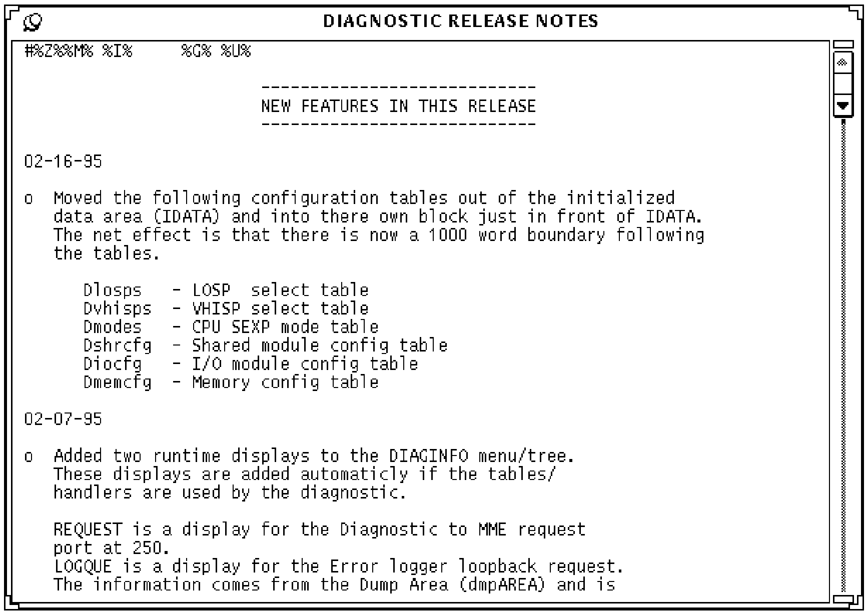


```
#%Z%%M% %I%     %G% %U%

                             ----------------------------
                             NEW FEATURES IN THIS RELEASE
                             ----------------------------
              -------------------------------------------------------------------
              C90 Mainframe Configuration Environment (rev 4.0.0) RELEASE ME-C2.3
              -------------------------------------------------------------------


  1.    The CRITICAL spr #73531 written against MCE has been resolved.
        MCE defaults to concurrent mode when MME is started in
        environment 0.

  2.    Spare chip functionality has been updated to include all SRAM
        mainframe types: C90 mainframe series 4000, 4800, 4600, 4400, 4200,
        TV11, TV12, and TV22; DRAM mainframe  types: series 4300, 4700, 4900, and
        TV12; other mainframe types: SIMULATOR and UNKNOWN.

  3.    Spare chip data is no longer a separate entity in MCE.  This data
        is now associated with the rest of the configuration data.
        Therefore, when the apply button is pressed in MCE 's base window
        the spare chip data is written to upper memory, or when the save
        button is pressed, an ascii file is written for OS and a binary
        file is written for EASE.  By tying spare chip and configuration data
        togetther we were able to remove the 'write os file' button
        and the 'apply file' button in MCE and replace them with a single
        mode field which specifies the data format.

  4.    SSD partitioning was added to MME / MCE.  The user now has the
```

Viewing Diagnostic Program Release Notes in Environment 1 or 2

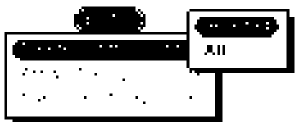


To view information about the most recently installed release of diagnostic programs, choose **`View --> Release Notes --> Diagnostics`**, as shown at the left. MME displays the `DIAGNOSTIC RELEASE NOTES` window:



```
#%Z%%M% %I%     %G% %U%

                          ----------------------------
                          NEW FEATURES IN THIS RELEASE
                          ----------------------------

02-16-95

o  Moved the following configuration tables out of the initialized
   data area (IDATA) and into there own block just in front of IDATA.
   The net effect is that there is now a 1000 word boundary following
   the tables.

      Dlosps   - LOSP  select table
      Dvhisps  - VHISP select table
      Dmodes   - CPU SEXP mode table
      Dshrcfg  - Shared module config table
      Diocfg   - I/O module config table
      Dmemcfg  - Memory config table

02-07-95

o  Added two runtime displays to the DIAGINFO menu/tree.
   These displays are added automaticly if the tables/
   handlers are used by the diagnostic.

   REQUEST is a display for the Diagnostic to MME request
   port at 250.
   LOGQUE is a display for the Error logger loopback request.
   The information comes from the Dump Area (dmpAREA) and is
```

Deleting a Control Point in Environment 1 or 2

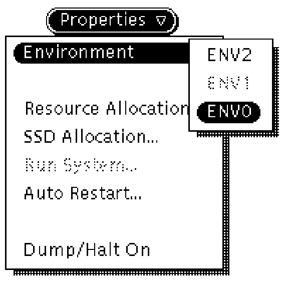


After loading and running a control point in mainframe memory, you may want to delete a control point in order to keep only the ones you want to run in the future. To delete a control point in environment 1 or 2 from the `Mainframe Maintenance Environment` window, choose **`Edit --> Delete Control Point --> Selected`**, as shown at the left.

Deleting All Control Points in Environment 1 or 2

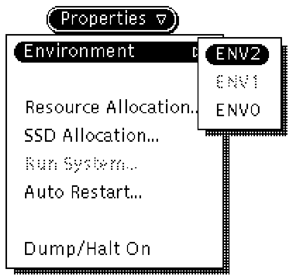
After loading and running a control point in mainframe memory, you may want to delete all control points if you no longer need them for testing. To delete all control points, choose **`Edit --> Delete Control Point --> All`**, as shown at the left.

Changing Environments in Environment 1 or 2

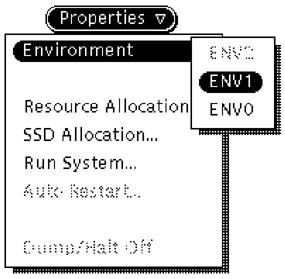


To change from environment 1 or 2 to environment 0 in offline mode, choose **`Properties --> Environment --> ENV0`**, as shown at the left.

The `ENV0` option is grayed-out in concurrent mode because environment 0 cannot be used in concurrent mode.



To change from environment 1 to environment 2, choose **`Properties --> Environment --> ENV2`**, as shown at the left.



To change from environment 2 to environment 1, choose **`Properties --> Environment --> ENV1`**, as shown at the left.

Allocating Resources in Environment 1 or 2

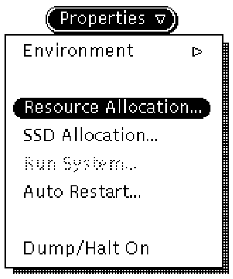
The three subgroups of resource allocation are memory allocation in environment 2 only, CPU allocation in environment 1 or 2, and execution mode allocation in environment 1 or 2.

Memory Allocation in Environment 2 Only

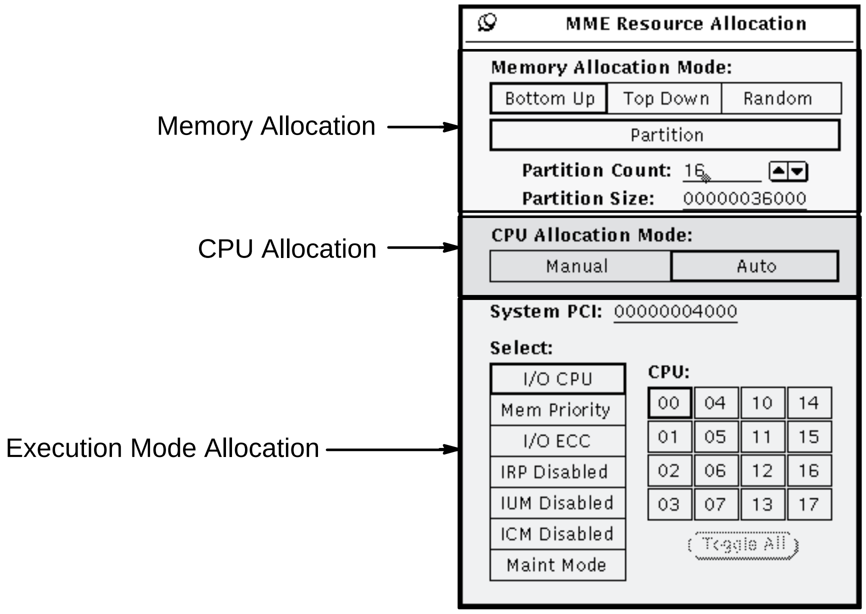
Before loading a control point, you can change the mainframe memory resources. For example, you can allocate resources in order to provide more memory for running a control point in environment 2. You can change memory resources in order to

- Load control points from the bottom up, top down, or randomly in mainframe memory
- Partition or divide memory for the number of control points you want to run, usually the number of CPUs in the system

To allocate memory resources, perform the following procedure:



1. Choose **`Properties --> Resource Allocation`**, as shown at the left. MME displays the `MME Resource Allocation` window.



Partition Count: 16
Partition Size: 00000036000
System PCI: 00000004000

2. Specify where you want the control point to load in memory by performing one of the following actions:

   - Click on Bottom Up to load the control point from the first memory partition to the last.
   - Click on Top Down to load the control point starting from the last memory partition to the first.
   - Click on Random to load the control point into random memory partitions.

   NOTE: Mainframe memory reserves addresses 0 through 40000 (octal) for the diagnostic controller. When you partition or divide mainframe memory, control points are loaded in a top-down or bottom-up manner, depending on which memory allocation mode you specified.

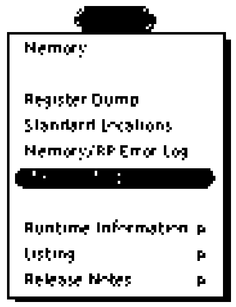
3. To divide mainframe memory into one or more smaller portions for testing, perform one of the following actions:

   NOTE: A partition count equal to the number of CPUs is the default.

   - Double click on the `Partition Count` field. Type the number of partitions you want. Press the return (↵) key.
   - Click on ▲ to increase the value or ▼ to decrease the value in the `Partition Count` field.

   You can run as many control points as there are partitions. As you increase the number of partitions, the partition size decreases. If you change the partition size, the number of partitions changes. Normally, you will set the number of partitions, and MME calculates the partition size.

   You can also change the size of each partition instead of changing the number of partitions. You do this by double clicking on the `Partition Size` field, typing the desired partition size, and pressing the return (↵) key. MME updates the number of partitions.



The amount of available mainframe memory is the MLAST value minus 40000_8 (because the diagnostic controller resides in mainframe memory addresses 0_8 through 40000_8). To see where the control points are loaded in memory, choose **`View --> Memory Map`**. Refer to “Viewing the MME Memory Map in Environment 1 or 2” earlier in this section for more information. Proceed with “CPU Allocation in Environment 1 or 2.”

CPU Allocation in Environment 1 or 2

After specifying the memory allocation (environment 2 only), perform the following steps:

1. Specify the `CPU Allocation Mode` you want to use by performing one of the following actions:

   - Click on Manual to assign a loaded control point to a specific CPU.

   - Click on Auto to assign the control point automatically to a system-selected CPU. The system-selected default is CPU 0.

   NOTE: Auto (Auto) mode is the default mode.

2. Proceed with “Execution Mode Allocation in Environment 1 or 2.”

Execution Mode Allocation in Environment 1 or 2

To allocate execution modes after specifying the memory allocation (environment 2 only) and the CPU allocation (in environment 1 or 2), perform the following steps:

1. If you do not want to use the default system programmable-clock interrupt (PCI) value, specify the number of clock periods that should complete before the system updates the exchange package for a control point. You can do this by clicking on the `System PCI` field, typing the value you want, and pressing the return (↵) key. If you specify 0, no exchange package updates occur.

NOTE: For information about displaying an exchange package, refer to "Viewing Memory in Environment 1 or 2" earlier in this section.

2. Click on I/O CPU (input/output central processing unit) to specify the I/O CPU (MME uses `CPU 00` as the default). Click on a `CPU`.

3. Click on Mem Priority to specify which CPU has the highest priority to mainframe memory. Click on a `CPU`.

4. Click on I/O ECC (I/O error-correction code) to enable the error-correction code for a specific CPU. Click on a `CPU`. If you want to select all the CPUs, click on Toggle All .

5. Click on IRP Disabled (interrupt for register parity errors) to disable interrupts for register parity errors. Click on a `CPU`. If you want to select all the CPUs, click on Toggle All .

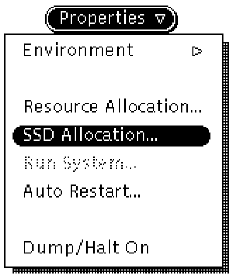
6. Click on IUM Disabled (interrupt for uncorrectable memory errors) to disable interrupts for uncorrectable memory errors. Click on a `CPU`. If you want to select all the CPUs, click on Toggle All .

7. Click on ICM Disabled (interrupt for correctable memory errors) to disable interrupts for correctable memory errors. Click on a `CPU`. If you want to select all the CPUs, click on Toggle All .

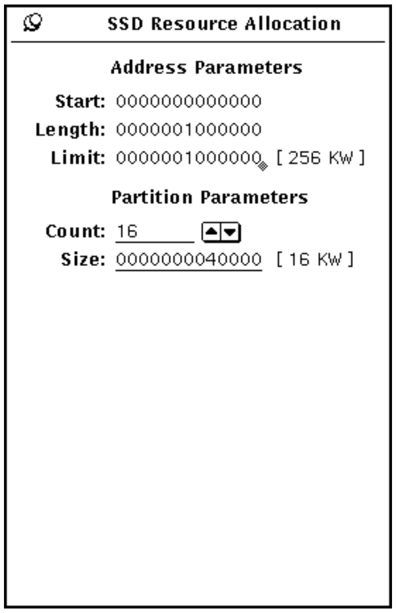
8. Click on Maint Mode to enable maintenance mode. Click on a `CPU`. If you want to select all the CPUs, click on Toggle All .

Allocating SSD Resources in Environment 1 or 2

Before loading any control points that run in the attached SSD(s), you may want to allocate the SSD resources. This enables you to set the addresses used and the partition count and size (partition count and size are available in environment 2 only) to specify where control points are loaded in SSD memory. To allocate SSD resources, perform the following procedure:



1. Choose **`Properties --> SSD Allocation`**, as shown at the left. MME displays the `SSD Resource Allocation` window:



SSD Resource Allocation

Address Parameters

Start: 0000000000000
Length: 0000001000000
Limit: 0000001000000 [256 KW]

Partition Parameters

Count: 16
Size: 0000000040000 [16 KW]

The `Address Parameters` fields indicate where in SSD memory control points can be loaded. The `Partition Parameters` fields, available in environment 2 only, define the number and size of the partitions.

2. Specify the addresses to use in the SSD by entering the starting address, length, and limit address (after you perform any two of the three following steps, MME updates the third field).

   NOTE: If SSD mapping is enabled in the `MCE – Configuration` window; the `Start`, `Length`, and `Limit` fields default to the values specified by the `Start Addr` and `Limit Addr` fields in the `MCE – Configuration` window. For more information about SSD mapping, refer to Section 2, "Mainframe Configuration Environment," of this manual.

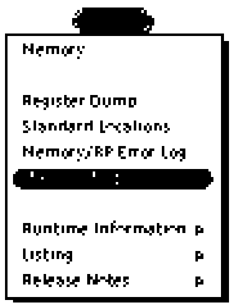
   - Double click on the `Start` field, type the starting address in the SSD you want to use, and press the return (↵) key.
   - Double click on the `Length` field, type the number of 64-bit words of memory in the SSD you want to use, and press the return (↵) key.
   - Double click on the `Limit` field, type the last address in the SSD you want to use, and press the return (↵) key. The amount of SSD memory being used is displayed in brackets next to the `Limit` field.

3. To divide the SSD memory into smaller portions for testing (partitions), perform one of the following actions:

   NOTE: A partition count equal to the number of CPUs is the default.

   - Double click on the `Count` field, type the number of partitions you want, and press the return (↵) key.
   - Click on ▲ to increase the value or ▼ to decrease the value in the `Count` field.

   You can also change the size of each partition instead of changing the number of partitions. You do this by double clicking on the `Size` field, typing the desired partition size, and pressing the return (↵) key. MME updates the number of partitions. Normally, you will set the number of partitions and let MME calculate the partition size. The size of each partition is displayed in brackets next to the `Size` field.



   To see where the control points are loaded in SSD memory, choose **`View --> Memory Map`**, as shown at the left. Refer to "Viewing the MME Memory Map in Environment 1 or 2" earlier in this section for more information.

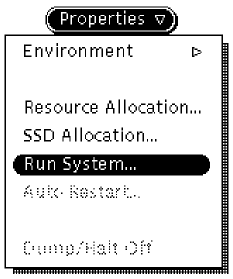
Using the Run System in Environment 2

You can use the `MME Run System` window to ensure that the major resources of the system are functioning properly. It enables you to find the error or failure quickly because you can automatically run many control points at once. When field engineers use the run system function, they can use control points loaded from a test list to detect a problem. The run system enables you to perform confidence testing on a CRAY C90 series computer system with an operating system-like environment for hardware evaluation. The operating system-like environment is simulated by swapping jobs (loaded control points) among active CPUs. The number of active CPUs is determined by your CPU assignment selections.

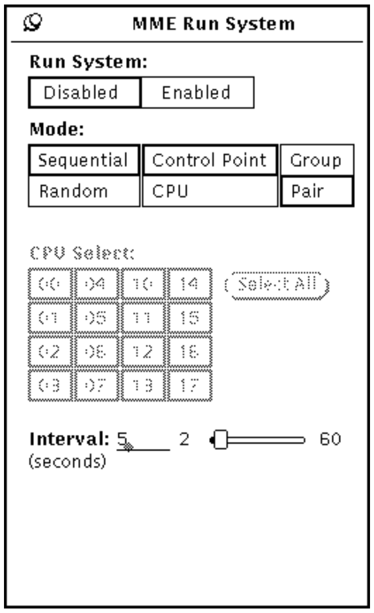
The following control points rotate under the run system:

aab.c	cat.c	jpt.c	pave.c	sfr.c
aht.c	clrmem.c	jpt.y	pave.y	sr2.c
amb.c	clrreg.c	jpt256.c	pmt.c	sr2.y
ars.c	csr.c	jpt256.y	sab.c	sr3.c
ave.c	diag.c	lsc.c	sas.c	sr3.y
bat.c	find.c	mem3.c	scan.c	vbt.c
bp.c	fpt.c	mem4.c	scl.c	vhc.c
brt.c	ibba.y	patt00.c	scs.c	vpt.c
btas.c	ibbc.y	patt01.c	sfa.c	vsg.c

To use the `MME Run System` window, perform the following procedure:



1. Choose **`Properties --> Run System`**, as shown at the left. MME displays the `MME Run System` window:



MME Run System

Run System: Disabled | Enabled

Mode: Sequential | Control Point | Group
Random | CPU | Pair

CPU Select:
00 04 10 14 Select All
01 05 11 15
02 06 12 16
03 07 13 17

Interval: 5 (seconds) 2 60

2. Click on Sequential. Loaded control points (or jobs) swap among CPUs one after the other, by CPU number from the lowest to the highest CPU number available.

   NOTE: Random mode is not available.

3. Click on Control Point. Loaded control points swap among CPUs in each of the available CPUs.

   NOTE: CPU mode is not available.

4. Specify how many control points/CPUs swap during each swap interval by performing one of the following actions:

   - Click on Group to swap all control points or all CPUs during each swap interval.

   - Click on Pair to swap two control points or two CPUs during each swap interval.

   NOTE: In Group or Pair mode, MME swaps the control points among CPUs during a swap interval. When all the control points have been swapped, the interval is complete.

5. Specify how often you want the run system to swap (in decimal seconds) by performing one of the following actions:

   - Click and drag the `Interval` slider to the value you want.

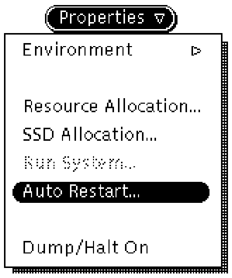
   - Double click on the `Interval` field, type the value you want, and press the return (↵) key.

6. Enable or disable the run system by performing one of the following actions:

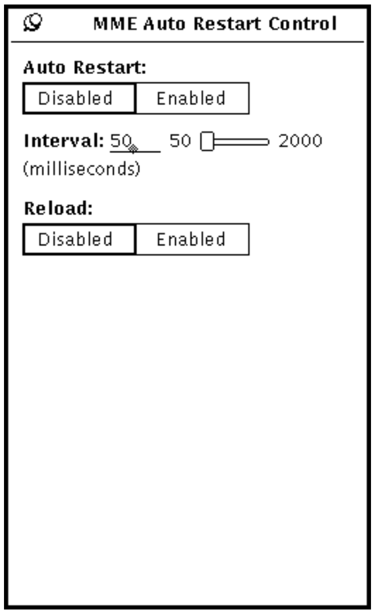
   - Click on Enabled to enable the run system. Click on Go in the `Mainframe Maintenance Environment` window to start the control points.

   - Click on Disabled to prevent the run system from starting now.

Using the Auto Restart Function in Environment 1



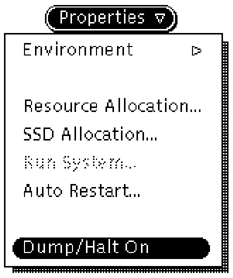
You can use the auto restart function to halt, reload, and restart a control point at a specified interval. This function is available in environment 1 only. To enable/disable the auto restart function, change the interval, and enable/disable the reload control point option; choose **`Properties --> Auto Restart`**, as shown at the left. MME displays the `Auto Restart Control` window:



You can perform the following actions using this window:

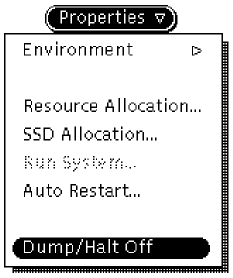
- Click on `Auto Restart` Disabled to disable the auto restart function.
- Click on `Auto Restart` Enabled to enable the auto restart function.
- Edit the `Interval` field or move the slider to change the interval at which the restart occurs.
- Click on `Reload` Disabled to disable reloading a control point at the specified interval.
- Click on `Reload` Enabled to enable reloading a control point at the specified interval.

Dumping Registers When Halt is Selected in Environment 1



To dump the register data to memory when you click on Halt/Dump in environment 1, choose **`Properties --> Dump/Halt On`**, as shown at the left. The Halt button changes to Halt/Dump. Refer to “Viewing the Contents of Registers in Environment 1 or 2” earlier in this section for more information about how to view a register dump.

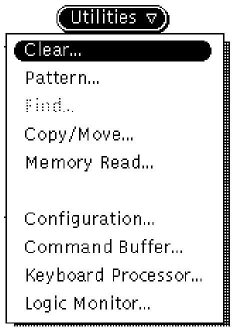
When you run one or more control points and you click on Halt/Dump, MME dumps the data either to a printer or a file, depending on what you specified in the `MME Dump Setup` window



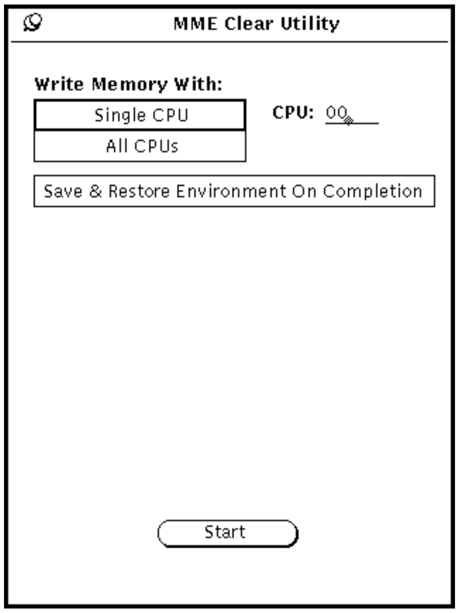
NOTE: You can disable the dump data to memory when the Halt button is clicked by choosing **`Properties --> Dump/Halt Off`**, as shown at the left. The button changes to Halt.

Using the Clear Utility in Environment 1 or 2

While using MME, you can clear (set to zero) all the interrupts, registers, and memory within a mainframe. To use the clear utility, perform the following procedure:



1. Choose **`Utilities --> Clear`**, as shown at the left. MME displays the `MME Clear Utility` window:



2. Click on Single CPU (to run `clrmem.c` in CPU 00) or All CPUs (to run `clr.c` in all CPUs).

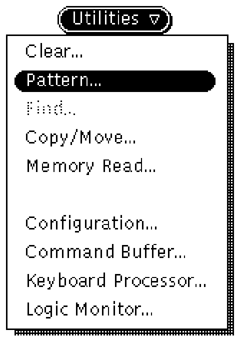
   NOTE: If you clicked on the Single CPU, you must specify which CPU to write to memory with [click on the `CPU` field, type a CPU number, and press the return (↵) key].

3. Click on Save & Restore Environment On Completion to save any control points you loaded or any settings you specified, if desired. If you do not click on Save & Restore Environment On Completion, all control points and settings will be lost.

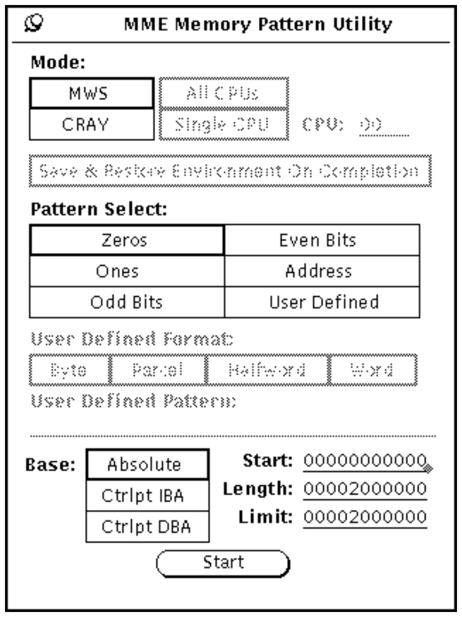
4. Click on Start. The clear utility runs the appropriate control point utility to clear all the interrupts, registers, and memory within the mainframe.

Using the Pattern Utility in Environment 1 or 2

The pattern utility writes a specified pattern of data into mainframe memory. To enter a zeros, ones, odd bits, even bits, random, addresses, or user-defined data pattern into mainframe memory for testing purposes; perform the following procedure:



1. Choose **`Utilities --> Pattern`**, as shown at the left. MME displays the `MME Pattern Utility` window:



2. Specify how to enter the pattern into mainframe memory by performing one of the following actions:

   - Click on MWS to use the MWS-E to enter the pattern into mainframe memory. Memory (from the starting address to the ending address) is written with the pattern transferred over the maintenance channel. No code is executed in any CPU. If this option is run from environment 2, the utility writes over the diagnostic controller program code so no program may been run until the diagnostic controller code is reloaded.

   - Click on CRAY to use mainframe program execution to enter the pattern into mainframe memory.

3. Specify how many CPUs to use by performing one of the following actions:

   - Click on **All CPUs** to use all CPUs. In environment 1, `patt+.c` is loaded into memory and executed in all CPUs. In environment 2, MME switches to environment 1 to load `patt+.c` into memory and executes it in all CPUs. When `patt+.c` has completed, MME switches back to environment 2 and reloads the diagnostic controller back into memory, making it appear that memory containing the diagnostic controller was not patterned.

   - Click on **Single CPU** to use one CPU, and then click on the `CPU` field, type the number of the CPU (in octal) you want to use, and press the return (↵) key. In environment 1, `patt.c` is loaded into memory and executed in all CPUs. In environment 2, MME switches to environment 1 to load `patt.c` into memory and executes it in all CPUs. When `patt.c` has completed, MME switches back to environment 2 and reloads the diagnostic controller back into memory, making it appear that memory containing the diagnostic controller was not patterned.

4. Click on **Save & Restore Environment On Completion** to save any control points you loaded or any settings you specified, if desired. If you do not click on **Save & Restore Environment On Completion**, any control points or settings will be lost.

5. Click on one or more of the following patterns: **Zeros**, Ones, **Ones**, **Even Bits**, **Address**, and/or **User Defined**.

   NOTE: If you are using a user-defined pattern (if you clicked on **User Defined**), click on the `User Defined Format` you want (**Byte**, **Parcel**, **Halfword**, or **Word**), and enter the data pattern you want to use [triple click on the `User Defined Format` field, type the data pattern you want to use, and press the return (↵) key.

6. Enter the starting address, block length, and ending address. After you perform any two of the three following steps, MME updates the third field.

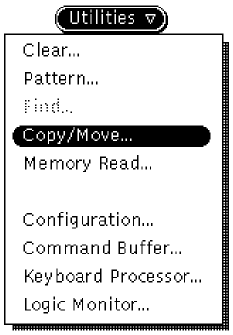
   NOTE: By default, the starting address is 0 and the ending address is the value stored in MLAST. The sum of the starting address and length values you specify must not be greater than the MLAST value.

   - Double click on the `Start Address` field, type the starting address in octal, and press the return (↵) key.
   - Double click on the `Block Length` field, type the block length in octal, and press the return (↵) key.
   - Double click on the `End Address` field, type the ending address in octal, and press the return (↵) key.

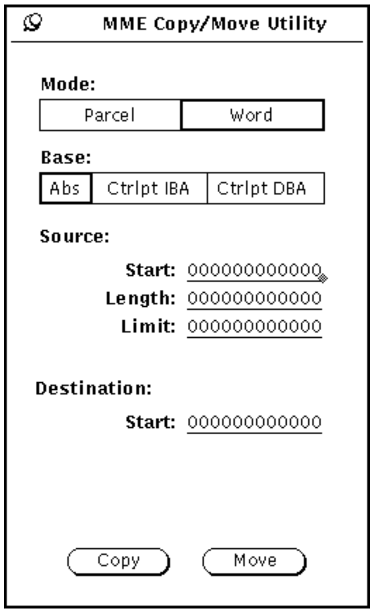
7. Click on Start. MME runs the pattern utility.

Using the Copy or Move Data Utility in Environment 1 or 2

When you have loaded data in memory, you can either copy or move the data to another location in memory. To copy or move data in memory, perform the following procedure:



1. Choose **`Utilities --> Copy/Move`**, as shown at the left. MME displays the `MME Copy/Move` window:



MME Copy/Move Utility

Mode: Parcel | Word

Base: Abs | Ctrlpt IBA | Ctrlpt DBA

Source:

Start: 000000000000
Length: 000000000000
Limit: 000000000000

Destination:

Start: 000000000000

2. Click on Parcel or Word for the `Mode` to specify the format you want for the memory to be moved/copied.

3. Click on a `Base` [Abs (absolute memory), Ctrlpt IBA (control point instruction base address), or Ctrlpt DBA (control point data base address)] to indicate the base for memory addressing.

4. Enter the starting address, length, and ending address:

   NOTE: After you perform any two of the three following steps, MME updates the third field.

   - Double click on the source data `Start` field where MME will start copying or moving the data. Type the starting address of the source data you want to copy or move, and press the return (↵) key.

   - Double click on the `Length` field. Type the block length of the source data you want to copy or move, and press the return (↵) key.

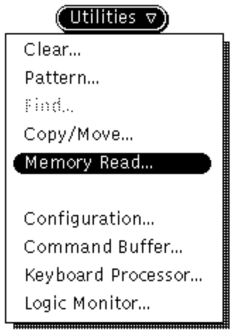
- Double click on the `End` field where MME will start copying or moving the data. Type the ending address of the source data you want to copy or move, and press the return (↵) key.

5. Double click on the data destination `Start` field. Type the starting destination address of the source data you want to copy or move, and press the return (↵) key.

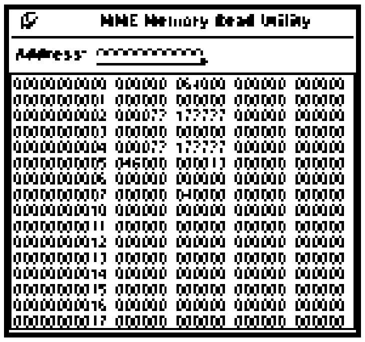
6. Click on Copy to copy the specified data or Move to move the specified data.

Using the MME Memory Read Utility in Environment 1 or 2

The memory read utility displays the results of a series of 1-word reads of mainframe memory that are used to debug maintenance function release failures. To display the MME memory read utility, perform the following procedure:



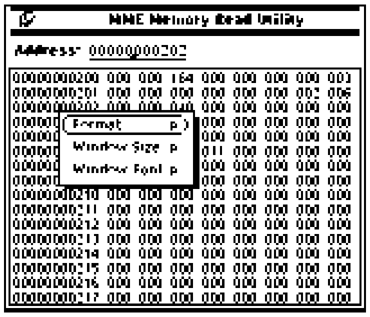
1. Choose **`Utility --> Memory Read`**, as shown at the left. MME displays the `MME Memory Read Utility` window:



```
Address: 00000000000
0000000000 000000 064000 000000 000000
0000000001 000000 000000 000000 000000
0000000002 000077 177777 000000 000000
0000000003 000000 000000 000000 000000
0000000004 000077 177777 000000 000000
0000000005 046000 000013 000000 000000
0000000006 000000 000000 000000 000000
0000000007 000000 040000 000000 000000
0000000010 000000 000000 000000 000000
0000000011 000000 000000 000000 000000
0000000012 000000 000000 000000 000000
0000000013 000000 000000 000000 000000
0000000014 000000 000000 000000 000000
0000000015 000000 000000 000000 000000
0000000016 000000 000000 000000 000000
0000000017 000000 000000 000000 000000
```

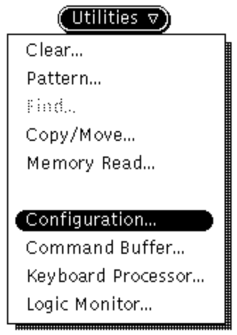
2. Double click on the `Address` field. Type the address you want to view. Press the return (↵) key, and MME displays memory starting at the specified address.

NOTE: You can change the format of the memory displayed in the window, the size of the window, and the font size of the data displayed in the window by choosing the corresponding options from the menu that appears when you press the MENU mouse button on the data area of this window.



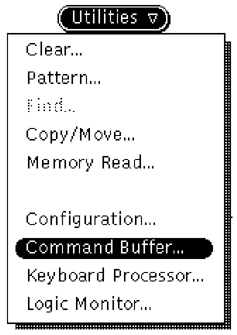
```
Address: 0000000000202
0000000200 000 000 164 000 000 000 000 003
0000000201 000 000 000 000 000 000 002 006
0000000202 000 000 040 000 000 000 000 000
00000                  000 000 000 000 000
00000                  000 000 000 000 000
00000                  011 000 000 000 000
00000                  000 000 000 000 000
00000                  000 000 000 000 000
0000000210 000 000 000 000 000 000 000 000
0000000211 000 000 000 000 000 000 000 000
0000000212 000 000 000 000 000 000 000 000
0000000213 000 000 000 000 000 000 000 000
0000000214 000 000 000 000 000 000 000 000
0000000215 000 000 000 000 000 000 000 000
0000000216 000 000 000 000 000 000 000 000
0000000217 000 000 000 000 000 000 000 000
```

Configuring MME in Environment 1 or 2



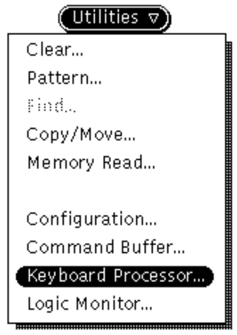
Choose **`Utilities --> Configuration`**, as shown at the left, to use the mainframe configuration environment (MCE) to configure the MME program. For more information about how to use MCE, refer to Section 2, “Mainframe Configuration Environment,” in this manual.

Starting the Command Buffer Parser Utility in Environment 1 or 2



To start the command buffer parser utility, choose **`Utilities --> Command Buffer`**, as shown at the left. This utility provides the ability to automate the testing process. For more information about the command buffer parser utility, refer to Section 6, “CRAY C90 CBP Runtime Module,” in this manual.

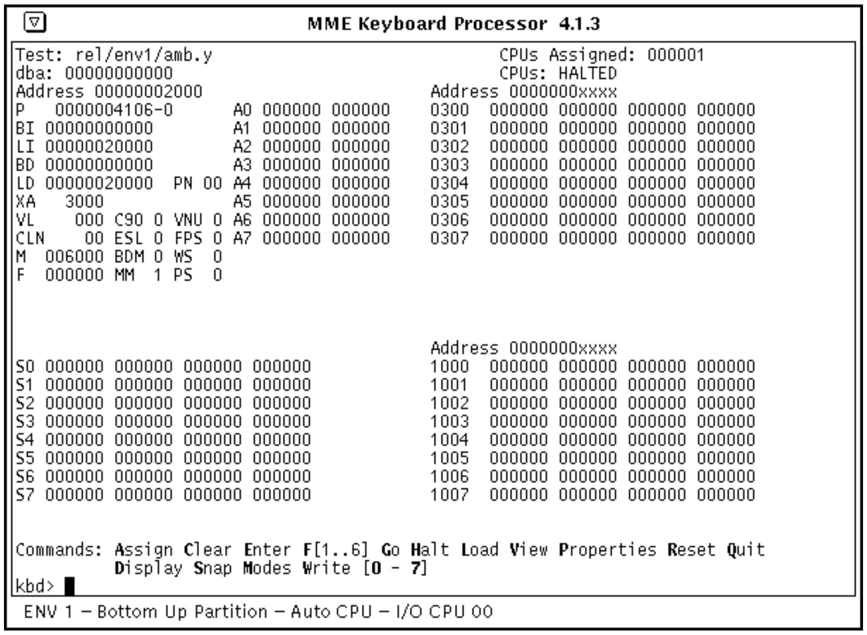
Using the Keyboard Processor in Environment 1 or 2



The MME keyboard processor provides a Cray maintenance system (CMS)-style interface for loading and controlling one control point. The keyboard processor works in a window that is separate from the MME base window. It provides a command line interface for manipulating the execution of a control point.

NOTE: The keyboard processor utility works with environments 1 and 2 only.

To start the keyboard processor utility, choose **`Utilities --> Keyboard Processor`**, as shown at the left. MME displays the `MME Keyboard Processor` window.



MME Keyboard Processor 4.1.3

```
Test: rel/env1/amb.y                             CPUs Assigned: 000001
dba: 00000000000                                 CPUs: HALTED
Address 00000002000                       Address 0000000xxxx
P   0000004106-0      A0 000000 000000    0300  000000 000000 000000 000000
BI 00000000000        A1 000000 000000    0301  000000 000000 000000 000000
LI 00000020000        A2 000000 000000    0302  000000 000000 000000 000000
BD 00000000000        A3 000000 000000    0303  000000 000000 000000 000000
LD 00000020000  PN 00 A4 000000 000000    0304  000000 000000 000000 000000
XA   3000             A5 000000 000000    0305  000000 000000 000000 000000
VL     000 C90 0 VNU 0 A6 000000 000000   0306  000000 000000 000000 000000
CLN     00 ESL 0 FPS 0 A7 000000 000000   0307  000000 000000 000000 000000
M  006000 BDM 0 WS  0
F  000000 MM  1 PS  0


                                          Address 0000000xxxx
S0 000000 000000 000000 000000            1000  000000 000000 000000 000000
S1 000000 000000 000000 000000            1001  000000 000000 000000 000000
S2 000000 000000 000000 000000            1002  000000 000000 000000 000000
S3 000000 000000 000000 000000            1003  000000 000000 000000 000000
S4 000000 000000 000000 000000            1004  000000 000000 000000 000000
S5 000000 000000 000000 000000            1005  000000 000000 000000 000000
S6 000000 000000 000000 000000            1006  000000 000000 000000 000000
S7 000000 000000 000000 000000            1007  000000 000000 000000 000000

Commands: Assign Clear Enter F[1..6] Go Halt Load View Properties Reset Quit
          Display Snap Modes Write [0 - 7]
kbd> █
 ENV 1 – Bottom Up Partition – Auto CPU – I/O CPU 00
```

The default keyboard processor window displays the following information:

- `Test` field - displays the loaded control point
- `CPUs Assigned` field - displays a bit mask indicating the CPUs assigned to the control point
- `CPUs` field - displays `RUNNING` or `HALTED` CPU status
- Exchange package fields (Refer to “Exchange Package Descriptions” in Volume 2, Section 2 of the *CRAY C90 Series Hardware Maintenance Manual*, publication number CMM-0502-0A0, for detailed descriptions of the exchange package fields.)
- S0 through S7 registers
- Address locations 300 through 307
- Address locations 1000 through 1007
- Status information (displayed on the last line) - indicates the current MME settings for the environment, memory setup, CPU allocation, and I/O CPU

Commands

Table 4-7 provides an overview of the commands available from the `Keyboard Processor` window. These menu-driven commands are displayed at the bottom of the `Keyboard Processor` display. You enter these commands by typing the bold letter. When you do this, the keyboard processor executes the command or advances to the next level of menu commands. Press the Esc key to cancel the current menu option and return to the main menu.

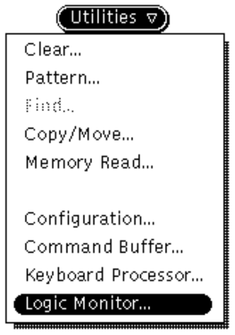
Table 4-7. Keyboard Processor Commands

Command	Description
Assign	Assigns or deassigns the control point to CPUs; you can use a **C**PU Mask to assign or deassign to specific CPUs or you can assign or deassign to **A**ll CPUs.
Clear	Loads and runs `clr.c` in all CPUs.
Display	Enables you to specify what you want to be displayed and on which portion of the screen you want it displayed. You can display **T**ext or e**X**change packages on the **L**eft or **R**ight portion of the display. You can also scroll memory **B**ackward or **F**orward.
Enter	Enables you to enter data into the e**X**change package or select **A**uto to implement an automatic incrementing feature to enter data into consecutive memory locations.
F[1..6]	Displays an additional menu that enables you to view the MME runtime information main display (**1**), view the runtime information error display (**2**), cancel all runtime information and error log displays (**3**), view the keyboard processor default display (**4**), view the MME memory error log display (**5**), and view the standard locations for the loaded control point (**6**).
Go	Starts the loaded control point in **A**ll or **S**elected CPUs.
Halt	Halts the loaded control point in **A**ll or **S**elected CPUs.
Load	Loads a **C**ontrol point from the current release directory or **D**ata from the `usr/data` directory.
Modes	Changes the mode format of displayed data in the `MME Keyboard Processor` window. You can set the display to **B**yte, **P**arcel, **H**alfword, **W**ord, he**X** (hexidecimal), or **T**ext to the **L**eft or **R**ight portion of the display.
Properties	Changes the properties of the MME **E**nvironment, **R**esource allocation, **A**uto_restart function, or Run **S**ystem.
Reset	Resets MME.
Snap	Snaps memory to the **P**rinter or to a **F**ile in the `usr` directory; the snap can be saved in **B**yte, **P**arcel, **H**alf (halfword), **W**ord, h**E**x (hexadecimal), **T**ext, or **A**sm (assembly language) formats.
Quit	Quits the keyboard processor utility.

Table 4-7. Keyboard Processor Commands (continued)

Command	Description
View	Enables you to view **I**nstruction data, **D**irectory contents, **E**rrorlogs, **L**isting displays (The file you enter must have the following form: cat.c.l.Z. Append .l.Z to the control point name.), **M**emory, **R**untime information, e**X**change packages, Resource **A**llocation information, **S**tandard locations, and backwards (**B**ack) or forwards (**F**orward) in the display.
Write	Saves specified **C**ontrol point or **D**ata to the usr directory.
0 – 7	Displays the specified memory on the left portion of the display; enter **F4** (default screen) to return the display to normal.

Starting the Logic Monitor Environment in Environment 1 or 2



Choose **Utilities –-> Logic Monitor**, as shown at the left, to start the logic monitor environment (LME). LME provides an X Window System based interface to the diagnostic monitor (DM). For more information about how to use LME, refer to Section 5, “Logic Monitor Environment,” in this manual.

Resetting the Software in Environment 1 or 2

Click on the Reset button to reset the MME software. This causes a system reset, which initializes the driver, initializes the channel, reasserts the configuration, reloads the controller (in environment 2 only), and reloads the control points.

5 LOGIC MONITOR ENVIRONMENT

The logic monitor environment (LME) is an X Window System based application that provides an interface to a CRAY C90 series mainframe diagnostic monitor (DM), located on the HR10 and HF0 options; for more information about the DM, refer to Volume 2 of the *CRAY C90 Series Hardware Maintenance Manual*, publication number CMM-0502-0A0. You can use LME to monitor the CPU(s) as instructions are executing. LME causes the DM to start and stop recording on specific events that occur while instructions are executing. This enables you to monitor CPU(s) as instructions execute without affecting any CPUs. You can also use LME to compare the actual results with expected results that you provide. This section describes how to start LME, how to perform tasks in LME, and how to use LME.

LME is designed to interact with MME; for example, when MME halts a CPU, all LME activity to that CPU is also stopped. LME and MME environment 0 are incompatible. LME will run with MME environment 0, but this is not recommended.

LME runs in a concurrent or offline environment (mode). In the concurrent environment (mode), LME can perform DM functions and read memory through any CPUs you specify as usable. LME can perform DM functions, read memory, and write memory through any CPUs that you specify as usable and for which you set 256K mode. Use the MCE soft-switch settings to specify CPUs that are usable and in 256K mode.

In the offline environment (mode), LME can perform all DM functions, read mainframe memory through any CPU, and write mainframe memory through any CPU; the OS cannot be running in the mainframe.

Starting LME

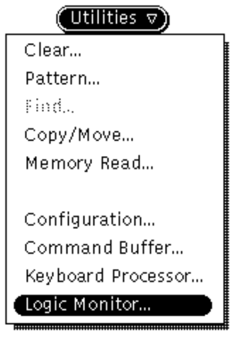
LME can be started from MME environment 1 or 2, the OpenWindows workspace menu, or a UNIX prompt. Multiple sessions of LME can be run from a single MWS-E.

For information about how to start LME from a service center through a hub, refer to “Appendix: CRAY C90 Remote Support.”

CAUTION

Do not start LME in offline mode when UNICOS is running. Several functions LME performs in offline mode will crash UNICOS if it is running.

From MME Environment 1 or 2



To start LME from MME environment 1 or 2, choose **`Utilities --> Logic Monitor`**, as shown at the left, in the MME base window.

From the OpenWindows Desktop Workspace Menu

Perform one of the following actions to start LME from the OpenWindows workspace menu:

- Choose **`Maintenance Tools --> MME --> LME --> Copy#`** to start LME with the copy number specified by the `Copy#` selection.

  The copy number option enables you to differentiate between multiple independent LME sessions that are supported from the same MWS-E. Copy numbers 0, 1, 2, and 3 are available from the workspace menu. The copy number does not affect performance; it serves as an identifier only.

- Choose **`Maintenance Tools Simulator --> MME --> LME`** to start LME with the simulator.

From a UNIX Command Prompt

To start MME environment 1 or 2 from a UNIX prompt, type **lme** followed by any appropriate command line options and press the return (↵) key. In the following list of commands, parameters in angle brackets < > are required, and parameters in brackets [ ] are optional. A bar | indicates a choice. Table 5-1 describes the available command line options.

```
lme     [–copy <num>]
        [–kill]
        [–config <file>] [–l <file>]
        [–concurrent | –offline]
        [–chn<num> | –sim]
        [–remote <host> | –client | –server]
        [–m] [–mg] [–mh [num]] [–mm]
```

NOTE: All command line options are passed to the LME server.

Table 5-1. Command Line Options

Option	Description
`–chn<num>`	Use the front-end interface (FEI) channel specified by <*num*>, which ranges from 0 to 7. The default channel number is 1.
`–client`	Start the client only
`–concurrent`	Use concurrent mode
`–config <file>`	Configure MCE with the configuration data stored in the file specified by <*file*>
`–copy <num>`	Connect to maintenance software assigned to the copy number specified by <*num*>. This option allows you to differentiate which system is being supported by this session of the software.
`–kill`	Kill all MME, MCE, and LME processes
`–l <file>`	Load a layout file
`–m`	Open the System Monitor utility window; do not start the System Monitor utility
`–mg`	Open System Monitor utility window; start the System Monitor utility
`–mh [num]`	Open System Monitor utility window, enable the hung CPU check feaure, and set it to [*num*] seconds; do not start the System Monitor utility
`–mm`	Enable SMON to perform dumps for trigger instructions issued in user mode **NOTE:** By default, SMON does not perform dumps for trigger instruction sequences that issue in user mode.
`–offline`	Use offline mode
`–remote <host>`	Start client only, connect to remote host

Table 5-1. Command Line Options (continued)

Option	Description
`–server`	Start server only
`–sim`	Use the simulator

For example, enter **`lme –offline –chn2 –l myfile`**↵ to start LME in offline mode using FEI channel 2 and the layout saved in `usr/myfile`.

LME Interface

The LME interface used to control the DMs is located in the LME base window. The components of this interface are shown in Figure 5-1. The text following Figure 5-1 describes the components as they appear in the figure, clockwise from the upper-left corner.

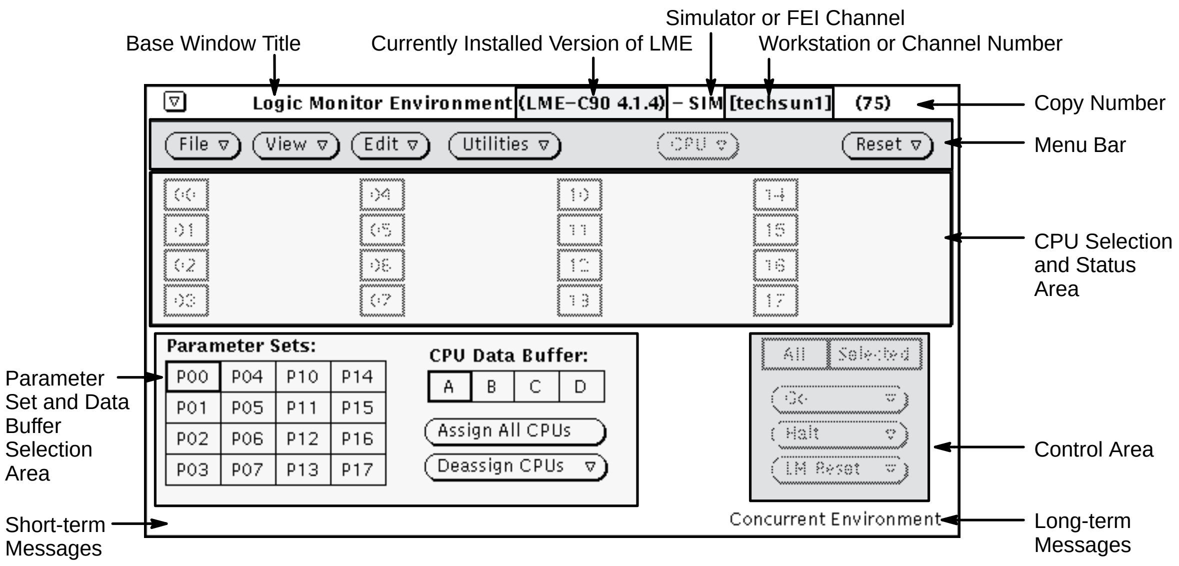


Figure 5-1. LME Interface Components

Base Window Title

The base window title displays the name of the program: `Logic Monitor Environment`.

Currently Installed Version of LME

The currently installed version of LME is a number in parentheses that indicates which version of LME you are running.

Simulator or FEI Channel

The simulator or FEI channel indicator shows you are running LME with the simulator (indicated by `SIM`) or an FEI channel (indicated by `FEI CHN 0` for channel 0, `FEI CHN 1` for channel 1, or `FEI CHN 2` for channel 2).

Workstation or Channel Number

The workstation or channel number indicator lists the name of the workstation or the channel number that LME is running on.

Copy Number

The copy number identifies the copy of LME you are using. Because you may run more than one session of LME at a time from a single MWS-E, the copy number differentiates the sessions. To set the copy number, start LME with the `–copy` option. If you start LME with the default copy number of 0, the LME base does not display a copy number. The copy number is used for identification only and will not affect performance. For more information about starting LME with the `–copy` option, refer to “Starting LME” earlier in this section.

Menu Bar

The menu bar contains the menu buttons for controlling many functions of LME. There are six menu buttons: File ▿, View ▿, Edit ▿, Utilities ▿, CPU ▿ and Reset ▿. For descriptions of the tasks you can perform with these menu buttons, refer to “Performing Tasks in LME” later in this section.

NOTE: To make the CPU ▿ menu button active, you must click on a parameter set displayed next to a CPU in the CPU selection and status area.

CPU Selection and Status Area

The CPU selection and status area is where you select which CPU(s) are assigned to a parameter set and where status information about the current DM run(s) is displayed. To assign a CPU to the current parameter set, click on the CPU number; the parameter set is displayed next to the CPU.

Control Area

The control area contains the buttons and settings used to start and halt the DM's execution of the specified parameter sets. This area contains the following buttons and settings:

Button/Setting	Description
All	Performs the go and halt functions on all CPUs
Selected	Performs the go and halt functions on the selected CPUs only
Go ▽	Starts the DM execution with the parameter set(s) for the CPU(s); choose **`Go --> One-shot`** to run the parameter set(s) once or choose **`Go --> Continuous`** to run the parameter set(s) continuously
Halt ▽	Halts the executing DM runs; choose **`Halt --> Hold Issue`** to hold the next instruction in the current instruction parcel register or choose **`Halt --> No Hold Issue`** to release the next instruction from the current instruction parcel
LM Reset ▽	Resets the time stamp feature

Long-term Messages

The long-term message area displays which environment you are working in for the current base window. The following messages are displayed:

Message	Description
`Concurrent Environment`	LME is in the concurrent environment (mode).
`Offline Environment`	LME is in the offline environment (mode).

Short-term Messages

The short-term message area displays `*** WARNING:  MME Environment 0 Running! ***` when environment 0 and LME are running at the same time. This warning is displayed because LME and environment 0 are incompatible.

Parameter Set and Expected Data Buffer Selection Area

The parameter set and expected data buffer selection area is where you select the parameter set and expected data buffer to assign CPUs to. You can select any one of the 16 parameter sets (`P00` – `P17`); you can select any one of the four CPU data buffers (`A`, `B`, `C`, or `D`). This area contains the following buttons:

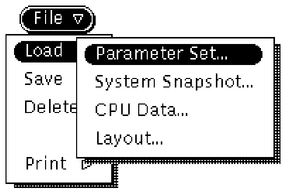
Button	Description
Assign All CPUs	Assigns all CPUs to the selected parameter set and CPU data buffer
Deassign CPUs ▽	Deassigns CPUs from the parameter set(s) and CPU data buffer(s) they are assigned to; choose **`Deassign CPUs --> All`** to deassign all CPUs or choose **`Deassign CPUs --> Selected`** to deassign the CPUs you have selected by clicking on them in CPU selection and status area

Performing Tasks in LME

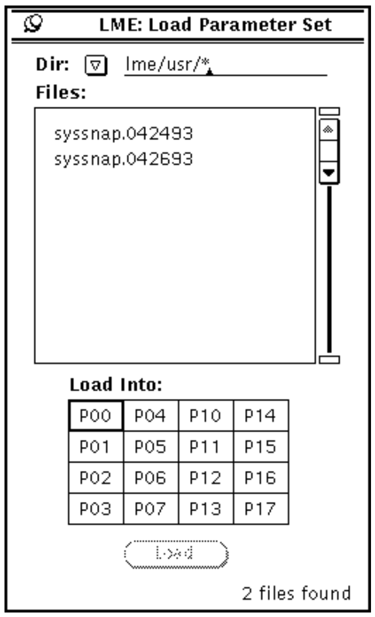
There are several tasks you need to perform to use the LME application. This subsection provides procedures that describe how to perform these tasks by manipulating the LME interface.

Loading a Parameter Set

A parameter set contains a 12-parcel block of DM data, a 64-word block of expected data, and a word-long mask used to determine which bits in each word of actual data from a DM run are compared to the corresponding expected data. You can load a parameter set that you previously saved (refer to "Saving a Parameter Set") by performing the following procedure:



1. Choose **`File --> Load --> Parameter Set`**, as shown at the left. LME displays the `LME: Load Parameter Set` window.



Dir: lme/usr/*

syssnap.042493
syssnap.042693

P00	P04	P10	P14
P01	P05	P11	P15
P02	P06	P12	P16
P03	P07	P13	P17

2 files found

2. Change the directory, if necessary, by performing one of the following actions:

   - Choose the directory from the `Dir:` ▽. You can choose from the following directories:

     `Release` – Parameter sets included in the current release

     `User` – User parameter sets

     `Alpha` – Pre-release files that are being tested and have not been released

   - Triple click on the `Dir` field, type the name of the directory you want to use, and press the return (↵) key.

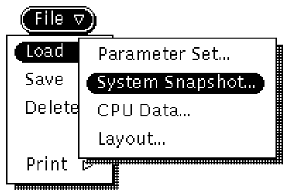
3. Click on the file you want to load.

4. Click on the parameter set (P00 through P17) that you want to load the data into.

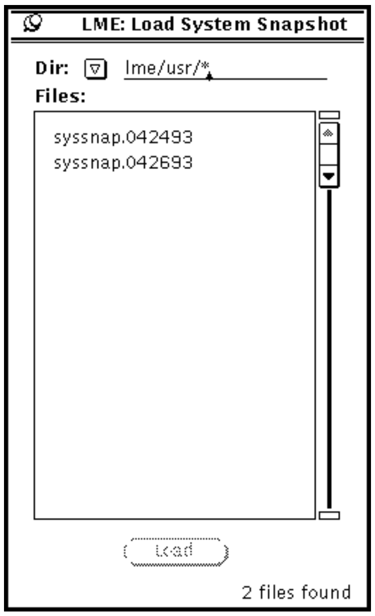
5. Click on Load. LME loads the parameter data into the specified parameter set.

Loading a System Snapshot

A system snapshot contains all CPU assignments, parameter sets, expected data, and CPU data used by LME. You can load a system snapshot that you previously saved (refer to “Saving a System Snapshot”) by performing the following procedure:



1. Choose **`File --> Load --> System Snapshot`**, as shown at the left. LME displays the `LME: Load System Snapshot` window:



syssnap.042493
syssnap.042693

2 files found

2. Change the directory, if necessary, by performing one of the following actions:

   - Choose the directory from the `Dir:` ▿. You can choose from the following directories:

     `Release` – Parameter sets included in the current release

     `User` – User parameter sets

     `Alpha` – Pre-release files that are being tested and have not been released

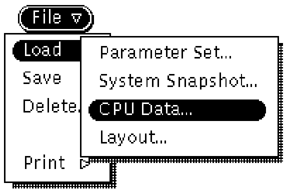
   - Triple click on the `Dir` field, type the name of the directory you want to use, and press the return (↵) key.

3. Click on the file you want to load.

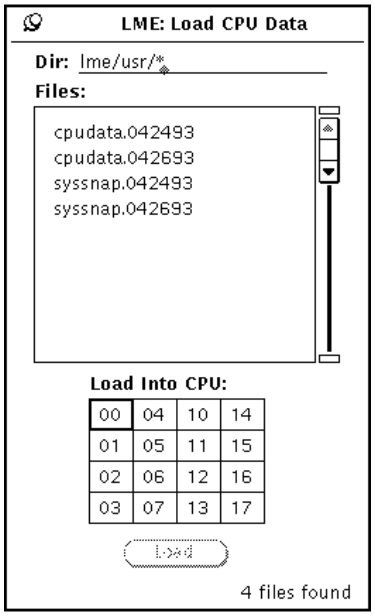
4. Click on Load. LME loads the specified system snapshot.

Loading CPU Data

You can load CPU data that you previously saved (refer to “Saving CPU Data”) by performing the following procedure:



1. Choose **`File --> Load --> CPU Data`**, as shown at the left. LME displays the `LME: Load CPU Data` window:



LME: Load CPU Data

Dir: lme/usr/*

Files:

cpudata.042493
cpudata.042693
syssnap.042493
syssnap.042693

Load Into CPU:

00	04	10	14
01	05	11	15
02	06	12	16
03	07	13	17

Load

4 files found

2. Change the directory, if necessary, by triple clicking on the `Dir` field, typing the name of the directory you want to use, and pressing the return (↵) key.

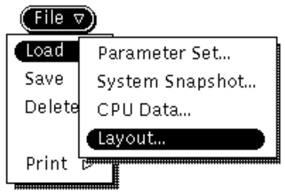
3. Click on the file you want to load.

4. Click on the CPU you want to load the CPU data into.

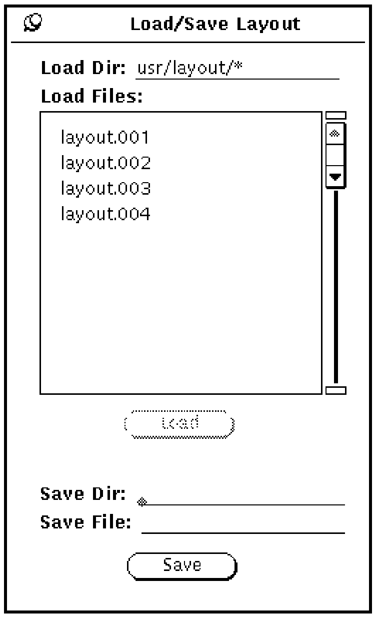
5. Click on Load. LME loads the specified CPU data.

Loading a Screen Layout

A screen layout contains the window size and position for all open windows and the data type stored in the window for all windows that have data. After you have saved a screen layout (refer to "Saving a Screen Layout"), you can load the screen layout to display those windows again. To load a screen layout, perform the following procedure:



1. Choose **`File --> Load --> Layout`**, as shown at the left. LME displays the `Load/Save Layout` window:



Load Dir: usr/layout/*
Load Files:
layout.001
layout.002
layout.003
layout.004
Save Dir:
Save File:

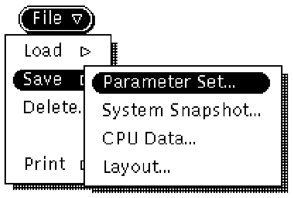
2. Change the directory, if necessary, by triple clicking on the `Load Dir` field, typing the name of the directory you want to use, and pressing the return (↵) key.

3. Click on the file you want to load.

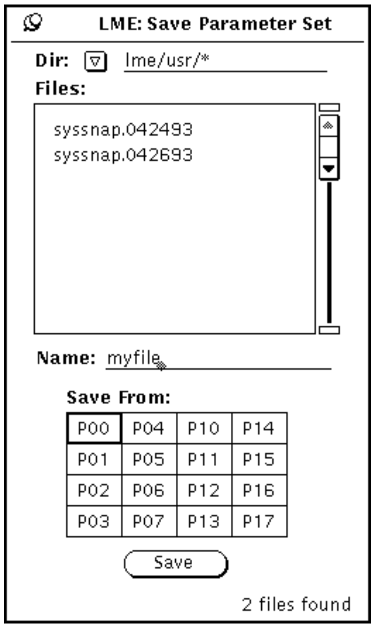
4. Click on . The windows that were saved in the specified file now appear on your screen.

Saving a Parameter Set

You can save a parameter set so you can use it later (refer to “Loading a Parameter Set”) by performing the following procedure:



1. Choose **`File --> Save --> Parameter Set`**, as shown at the left. LME displays the `LME: Save Parameter Set` window:



2. Change the directory, if necessary, by performing one of the following actions:

   - Choose the directory from the `Dir:` ▽. You can choose from the following directories:

     `Release` – Parameter sets included in the current release

     `User` – User parameter sets

     `Alpha` – Pre-release files that are being tested and have not been released

   - Triple click on the `Dir` field, type the name of the directory you want to use, and press the return (↵) key.

3. Specify the name of the file you want to save by performing one of the following actions:

   - Double click on the `Name` field and type the name of the file you want to use.

   - Click on the file in the `Files` scroll box if you want to save parameter set data to a file that already exists.

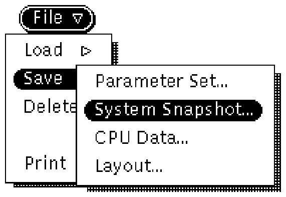
4. Click on the parameter set (`P00` through `P17`) that you want to save the data from.

5. Click on Save. LME saves the parameter set data in the specified file.

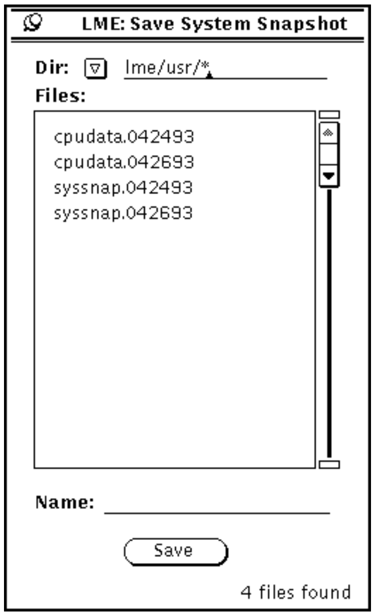
Saving a System Snapshot

You can save a system snapshot to preserve data for later use (refer to "Loading a System Snapshot") by performing the following procedure:

NOTE: If you save a system snapshot and the screen layout, you can restore LME to the current state by loading both the snapshot and layout.



1. Choose **`File --> Save --> System Snapshot`**, as shown at the left. LME displays the `LME:  Save System Snapshot` window:


LME: Save System Snapshot

Dir: ▽ lme/usr/*

Files:

cpudata.042493
cpudata.042693
syssnap.042493
syssnap.042693

Name:

Save

4 files found

2. Change the directory, if necessary, by performing one of the following actions:

   - Choose the directory from the `Dir:` ▽. You can choose from the following directories:

     `Release` – Parameter sets included in the current release

     `User` – User parameter sets

     `Alpha` – Pre-release files that are being tested and have not been released

   - Triple click on the `Dir` field, type the name of the directory you want to use, and press the return (↵) key.

3. Specify the name of the file you want to save by performing one of the following actions:

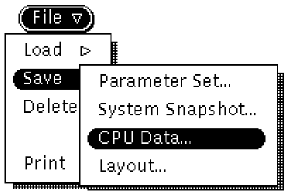
   - Double click on the `Name` field and type the name of the file you want to use.

   - Click on the file in the `Files` scroll box if you want to save parameter set data to a file that already exists.

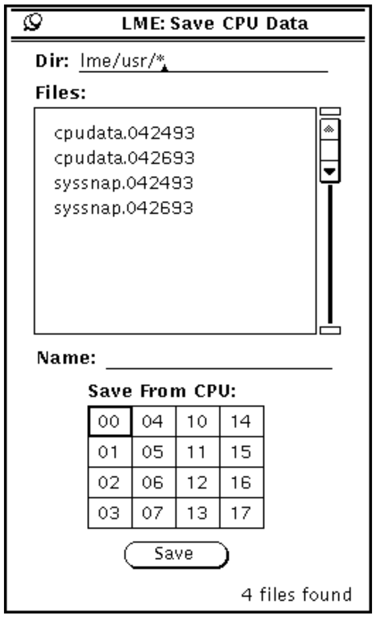
4. Click on Save. LME saves the specified system snapshot.

Saving CPU Data

You can save CPU data for future use (refer to “Loading CPU Data”) by performing the following procedure:



1. Choose **`File --> Save --> CPU Data`**, as shown at the left. LME displays the `LME: Save CPU Data` window:



2. Change the directory, if necessary, by triple clicking on the `Dir` field, typing the name of the directory you want to use, and pressing the return (↵) key.

3. Specify the name of the file you want to save by performing one of the following actions:

   - Double click on the `Name` field and type the name of the file you want to use.

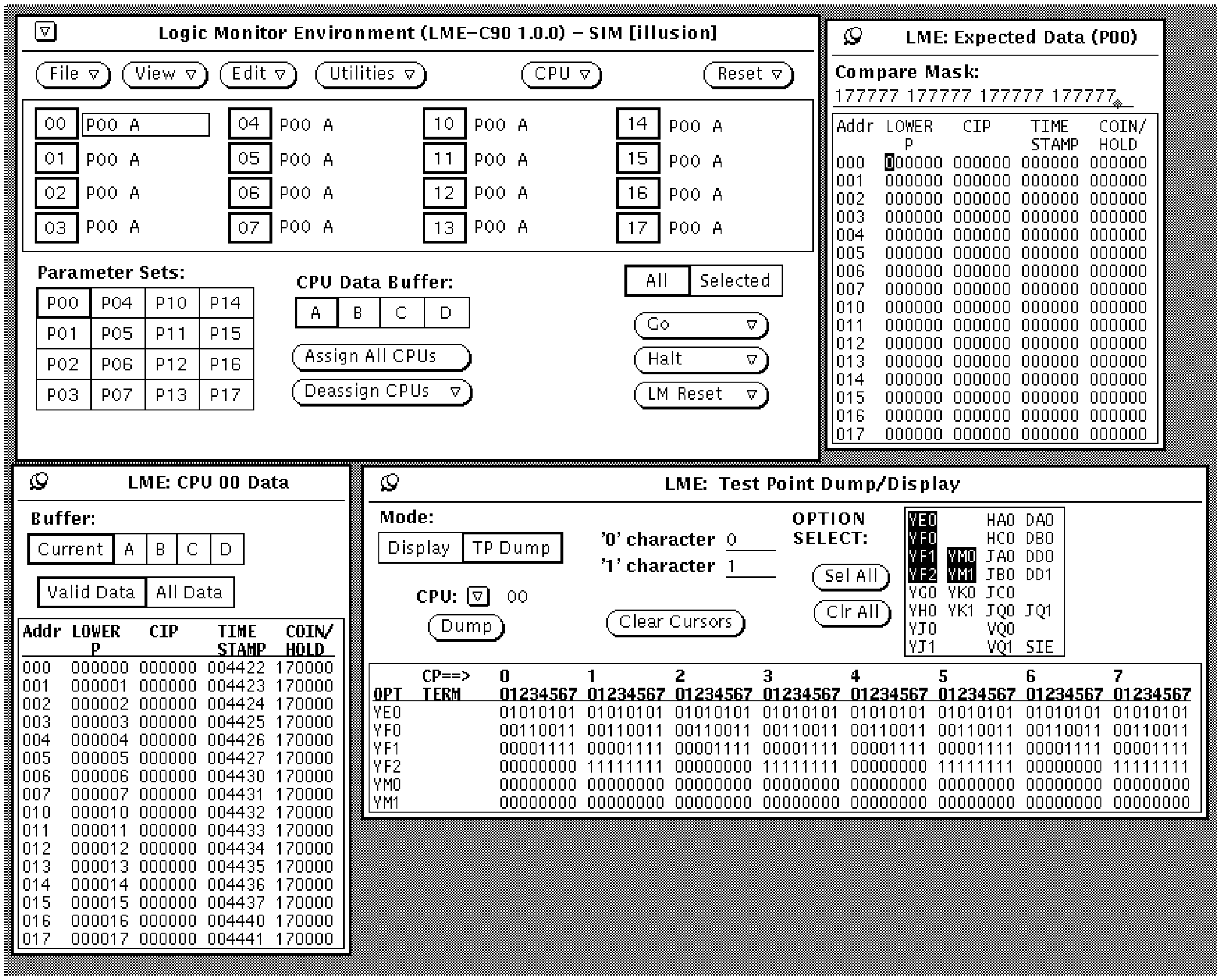
   - Click on the file in the `Files` scroll box if you want to save parameter set data to a file that already exists.

4. Click on the CPU (00 through 17) that you want to save the data from.

5. Click on (Save). LME saves the CPU data in the specified file.

Saving a Screen Layout

You can save a screen layout that preserves the organization of the windows you have opened and enables you to display them later by loading the layout. Figure 5-2 shows a sample layout.



```
Logic Monitor Environment (LME-C90 1.0.0) - SIM [illusion]

File   View   Edit   Utilities          CPU          Reset

00  P00 A      04  P00 A      10  P00 A      14  P00 A
01  P00 A      05  P00 A      11  P00 A      15  P00 A
02  P00 A      06  P00 A      12  P00 A      16  P00 A
03  P00 A      07  P00 A      13  P00 A      17  P00 A

Parameter Sets:           CPU Data Buffer:        All  Selected
P00  P04  P10  P14        A  B  C  D              Go
P01  P05  P11  P15        Assign All CPUs         Halt
P02  P06  P12  P16        Deassign CPUs           LM Reset
P03  P07  P13  P17
```

```
LME: Expected Data (P00)

Compare Mask:
177777 177777 177777 177777

Addr LOWER   CIP    TIME   COIN/
       P            STAMP  HOLD
000  000000 000000 000000 000000
001  000000 000000 000000 000000
002  000000 000000 000000 000000
003  000000 000000 000000 000000
004  000000 000000 000000 000000
005  000000 000000 000000 000000
006  000000 000000 000000 000000
007  000000 000000 000000 000000
010  000000 000000 000000 000000
011  000000 000000 000000 000000
012  000000 000000 000000 000000
013  000000 000000 000000 000000
014  000000 000000 000000 000000
015  000000 000000 000000 000000
016  000000 000000 000000 000000
017  000000 000000 000000 000000
```

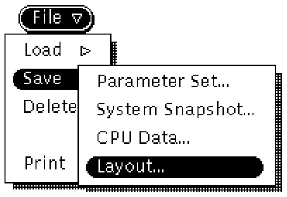
```
LME: CPU 00 Data

Buffer:
Current  A  B  C  D

Valid Data  All Data

Addr LOWER   CIP    TIME   COIN/
       P            STAMP  HOLD
000  000000 000000 004422 170000
001  000001 000000 004423 170000
002  000002 000000 004424 170000
003  000003 000000 004425 170000
004  000004 000000 004426 170000
005  000005 000000 004427 170000
006  000006 000000 004430 170000
007  000007 000000 004431 170000
010  000010 000000 004432 170000
011  000011 000000 004433 170000
012  000012 000000 004434 170000
013  000013 000000 004435 170000
014  000014 000000 004436 170000
015  000015 000000 004437 170000
016  000016 000000 004440 170000
017  000017 000000 004441 170000
```

```
LME:  Test Point Dump/Display

Mode:                                           OPTION      YE0      HA0  DA0
Display  TP Dump       '0' character  0         SELECT:     YF0      HC0  DB0
                       '1' character  1                     YF1 YM0  JA0  DD0
                                                Sel All     YF2 YM1  JB0  DD1
CPU:  00                                                    YG0 YK0  JC0
Dump                   Clear Cursors            Clr All     YH0 YK1  JQ0  JQ1
                                                            YJ0      VQ0
                                                            YJ1      VQ1  SIE

       CP==>  0        1        2        3        4        5        6        7
OPT    TERM   01234567 01234567 01234567 01234567 01234567 01234567 01234567 01234567
YE0           01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101
YF0           00110011 00110011 00110011 00110011 00110011 00110011 00110011 00110011
YF1           00001111 00001111 00001111 00001111 00001111 00001111 00001111 00001111
YF2           00000000 11111111 00000000 11111111 00000000 11111111 00000000 11111111
YM0           00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
YM1           00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

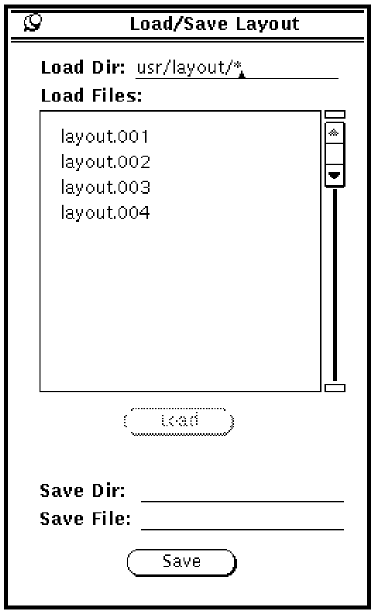
Figure 5-2. LME Sample Layout

To save a layout, perform the following procedure:

NOTE: If you save a system snapshot and the screen layout, you can restore LME to the current state by loading both the snapshot and layout.



1. Choose **`File --> Save --> Layout`**, as shown at the left. LME displays the `Load/Save Layout` window.



2. Change the directory, if necessary, by triple clicking on the `Save Dir` field, typing the name of the directory you want to use, and pressing the return (↵) key.

3. Specify the name of the file you want to save by performing one of the following actions:

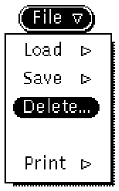
   - Double click on the `Name` field and type the name of the file you want to use.

   - Click on the file in the `Files` scroll box if you want to save parameter set data to a file that already exists.

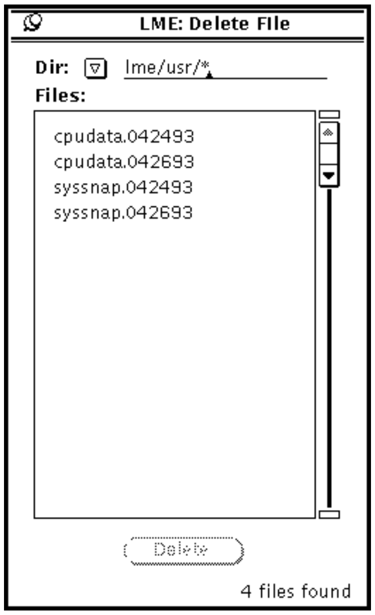
4. Click on Save. LME saves the specified layout file, and the filename appears in the `Load Files` scroll box.

Deleting a File

To delete files in the `lme/usr` that you no longer need, perform the following procedure:



1. Choose **`File --> Delete`** as shown at the left. LME displays the `LME: Delete File` window:



cpudata.042493
cpudata.042693
syssnap.042493
syssnap.042693

2. Change the directory, if necessary, by performing one of the following actions:

   - Choose the directory from the `Dir:` ▽. You can choose from the following directories:

     `Release` – Parameter sets included in the current release

     `User` – User parameter sets

     `Alpha` – Pre-release files that are being tested and have not been released

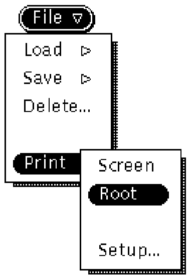
   - Triple click on the `Dir` field, type the name of the directory you want to use, and press the return (↵) key.

3. Click on the file(s) you want to delete.

4. Click on (Delete). LME deletes the file(s).

Printing the Root Window

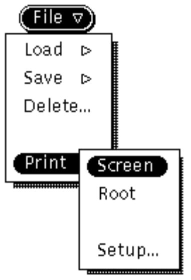
Before performing this procedure, you must first set up printing. Refer to “Setting Up or Changing Where LME Data is Printed.”



To print an image of everything contained in the root window, choose **`File --> Print --> Root`**, as shown at the left. Use this command to print the MME base window.

Printing a Screen or Panel

Before performing this procedure, you must first set up printing. Refer to “Setting Up or Changing Where LME Data is Printed.”



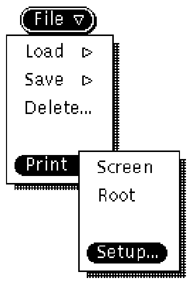
To print a window or an icon, choose **`File --> Print --> Screen`**, as shown at the left. When you choose this command, the cursor becomes a plus symbol. Move the cursor to the window or icon you want to print an image of and click on a mouse button.

NOTE: You cannot print the LME base window with this command. To print the LME base window, choose **`File --> Print --> Root`**.

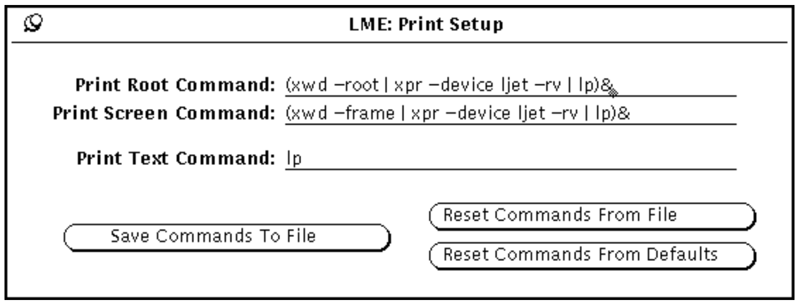
Setting Up or Changing Where LME Data is Printed

Before you print a root or screen, you must set up the printer options by entering the appropriate UNIX commands in the `Print Root Command` and `Print Screen Command` fields of the `LME: Print Setup` window. LME uses the specified commands for the print functions to ensure that output goes to the proper printer.

NOTE: This process uses the `xwd`, `xpr`, and `lp` UNIX commands. The `xwd` command dumps an image of an X window. The `xpr` command prints an image of the X window dump. The `lp` command sends a request to the printer. For detailed information about these commands, refer to the UNIX man pages (enter **`man xwd`**, **`man xpr`**, or **`man lp`** at a UNIX prompt).



To set up where you want data printed, choose **`File --> Print --> Setup`**, as shown at the left. LME displays the `LME: Print Setup` window:


LME: Print Setup

Print Root Command: (xwd –root | xpr –device ljet –rv | lp)&

Print Screen Command: (xwd –frame | xpr –device ljet –rv | lp)&

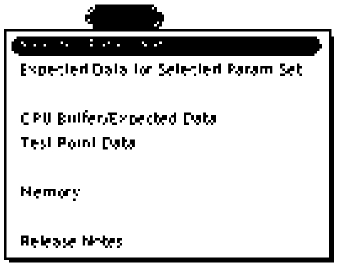
Print Text Command: lp

The buttons in this window are described in the following list:

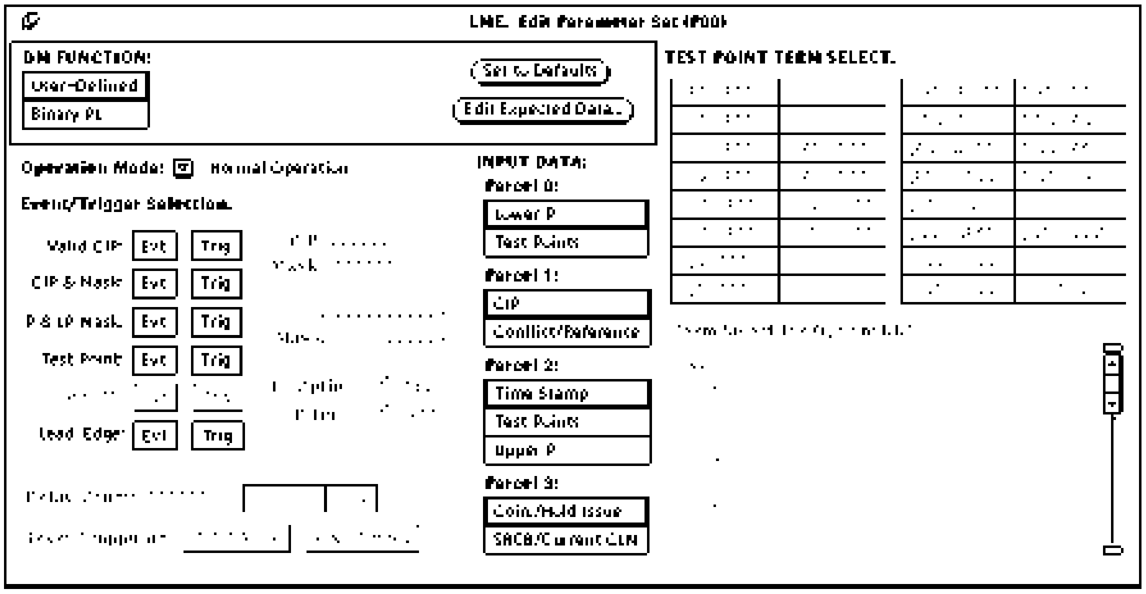
- Click on Save Commands To File to save your current printer setup commands for later use.
- Click on Reset Commands From File to load the printer setup commands you saved previously.
- Click on Reset Commands From Defaults to load the default printer setup commands provided with the MME program.

MME and LME share the print commands defined in their `Print Setup` windows. If you change the values in LME, they also change for MME.

Viewing the Selected Parameter Set

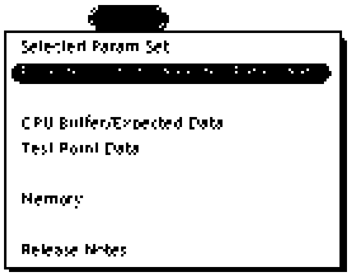


You can modify the parameter set that controls the DM run. To view the selected parameter set, choose **`View --> Selected Param Set`**, as shown at the left. LME displays the `LME: Edit Parameter Set` window:



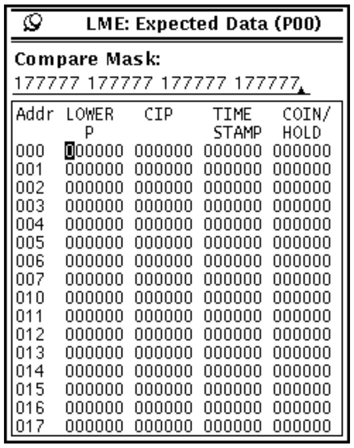
For more information about the parameters displayed in this window, refer to the “Setting Up the Parameters and Expected Data” discussion in the “Using LME” subsection of this section.

Viewing the Expected Data for the Selected Parameter Set



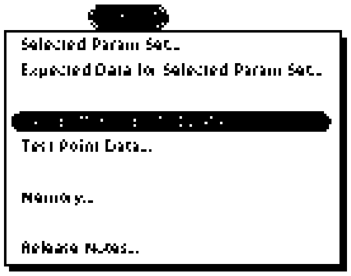
To view the expected data for the selected parameter set, choose **`View --> Expected Data for Selected Param Set`**, as shown at the left. LME displays the `LME: Expected Data` window for the selected parameter set:

NOTE: You can change the `Compare Mask` value to control which bits are compared.

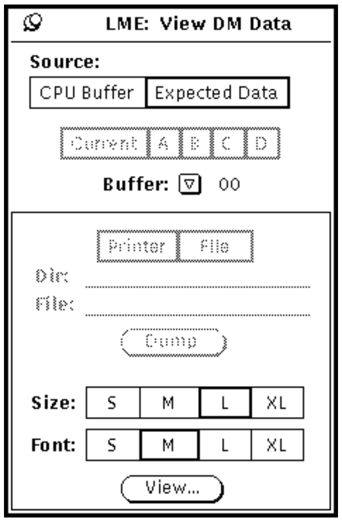


```
LME: Expected Data (P00)
Compare Mask:
177777 177777 177777 177777
Addr  LOWER    CIP     TIME    COIN/
        P             STAMP    HOLD
000   000000  000000  000000  000000
001   000000  000000  000000  000000
002   000000  000000  000000  000000
003   000000  000000  000000  000000
004   000000  000000  000000  000000
005   000000  000000  000000  000000
006   000000  000000  000000  000000
007   000000  000000  000000  000000
010   000000  000000  000000  000000
011   000000  000000  000000  000000
012   000000  000000  000000  000000
013   000000  000000  000000  000000
014   000000  000000  000000  000000
015   000000  000000  000000  000000
016   000000  000000  000000  000000
017   000000  000000  000000  000000
```

Viewing the CPU Buffer and Expected Data



To view the CPU buffer and expected data, choose **`View --> CPU Buffer/Expected Data`**, as shown at the left. LME displays the following `LME:  View DM Data` window.



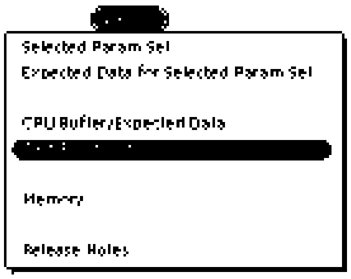
You can perform the following actions from this window:

- Click on `Source:` CPU Buffer to view the CPU buffer(s). If you do this, choose the CPU you want from the `CPU:` ▽ and click on the buffer you want to view (Current, A, B, C, or D).

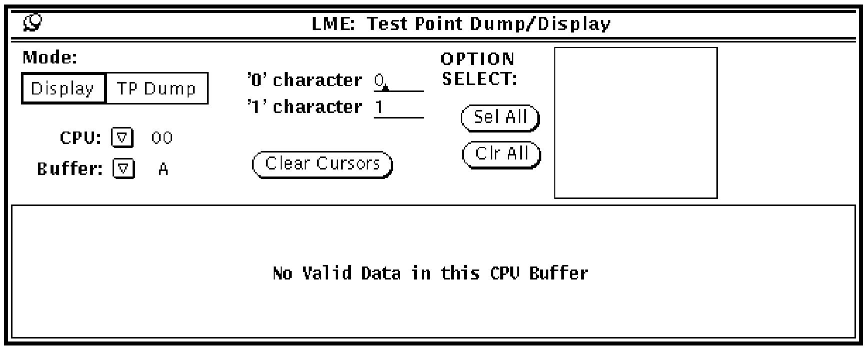
- Click on `Source:` Expected Data to view the expected data buffer(s). If you do this, choose the expected data buffer you want to view from the `Buffer:` ▽.

- Specify that you want the data to be displayed in a window; to do this, indicate the window `Size` (click on S, M, L, or XL) and `Font` size (click on `S`, `M`, `L`, or `XL`) and click on View .

Viewing the Test Point Data



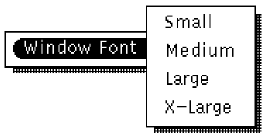
To view the test point data, choose **`View --> Test Point Data`**, as shown at the left. LME displays the `LME: Test Point Dump/Display` window:



No Valid Data in this CPU Buffer

You can perform the following actions from this window:

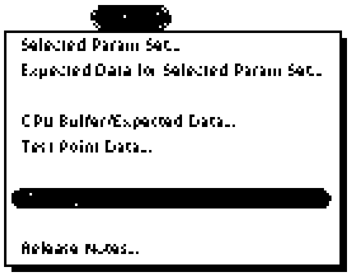
- Toggle between test point display mode and test point dump mode
- Specify the CPU (choose from `CPU:` ▽) and buffer (choose from `Buffer:` ▽) you want to view
- Change the 0 and 1 characters on the display [enter the new character(s) in the corresponding field(s)]
- Clear the cursors (you set cursors to highlight lines within the display by clicking on the lines you want to highlight)
- Select all options to be displayed
- Clear all options from being displayed
- Click on the MENU mouse button in the text area to access a menu to change the `Window Font` size:



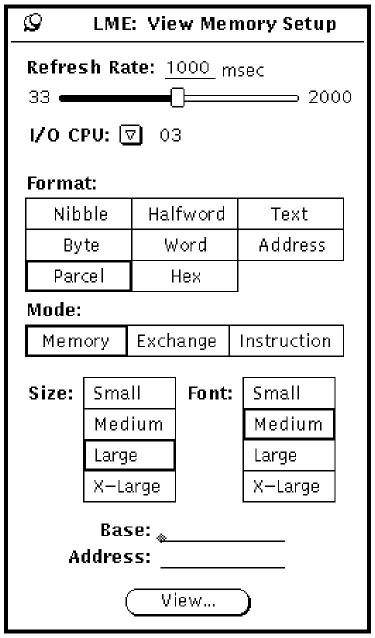
NOTE: You can select/deselect individual options to display by clicking on the options in the `OPTION SELECT` box.

Viewing Mainframe Memory

You can view and edit mainframe memory on the screen using the view memory option. (In concurrent mode, you cannot write memory.) To view memory, perform the following procedure:


Selected Param Set..
Expected Data for Selected Param Set..
CPU Buffer/Expected Data..
Test Point Data..
Release Notes..

1. Choose **`View --> Memory`**, as shown at the left. LME displays the `LME View Memory Setup` window:



LME: View Memory Setup

Refresh Rate: 1000 msec
33 2000
I/O CPU: ▽ 03

Format:

Nibble	Halfword	Text
Byte	Word	Address
Parcel	Hex	

Mode:

Memory	Exchange	Instruction

Size: Small / Medium / Large / X-Large

Font: Small / Medium / Large / X-Large

Base:
Address:

View...

NOTE: You can set the interval at which memory windows are updated by moving the `Refresh Rate` slider or by double clicking on the `Refresh Rate` field, typing a new value, and pressing the return (↵) key. Setting this value too low can monopolize the workstation CPU.

2. Select the CPU you want to use to read and write memory from the `I/O CPU:` ▽ menu. (If you choose a CPU that is not usable or that is not configured with I/O, LME will use the lowest-numbered CPU that is usable and configured with I/O.)

3. Click on a `Format` [Nibble, Halfword, Text, Byte, Word, Address, Parcel, or Hex (hexadecimal)] to specify the format in which you want the memory displayed.

4. Click on Memory, Exchange, or Instruction to specify the `Mode` you want.

   Memory mode displays normal memory, exchange mode displays exchange information, and instruction mode decodes the memory into instructions.

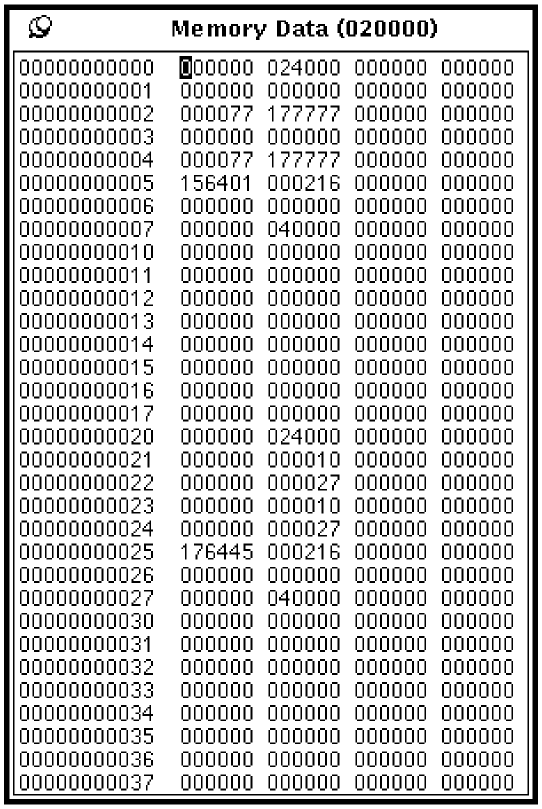
5. Click on `Size:` Small, Medium, Large, or X-Large to select the display window size.

6. Click on `Font:` Small, Medium, Large, or X-Large to specify the font type size.

7. Enter a base number in the `Base` field.

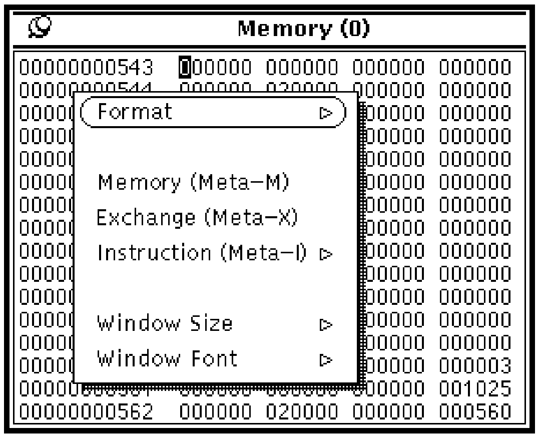
8. Enter the memory address to be viewed in the `Address` field.

9. Click on View... . LME displays the specified memory window:


Memory Data (020000)

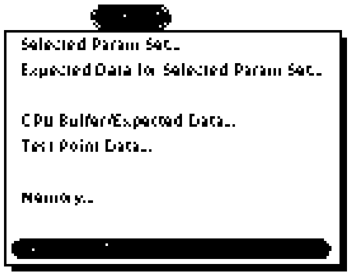
```
00000000000   000000  024000  000000  000000
00000000001   000000  000000  000000  000000
00000000002   000077  177777  000000  000000
00000000003   000000  000000  000000  000000
00000000004   000077  177777  000000  000000
00000000005   156401  000216  000000  000000
00000000006   000000  000000  000000  000000
00000000007   000000  040000  000000  000000
00000000010   000000  000000  000000  000000
00000000011   000000  000000  000000  000000
00000000012   000000  000000  000000  000000
00000000013   000000  000000  000000  000000
00000000014   000000  000000  000000  000000
00000000015   000000  000000  000000  000000
00000000016   000000  000000  000000  000000
00000000017   000000  000000  000000  000000
00000000020   000000  024000  000000  000000
00000000021   000000  000010  000000  000000
00000000022   000000  000027  000000  000000
00000000023   000000  000010  000000  000000
00000000024   000000  000027  000000  000000
00000000025   176445  000216  000000  000000
00000000026   000000  000000  000000  000000
00000000027   000000  040000  000000  000000
00000000030   000000  000000  000000  000000
00000000031   000000  000000  000000  000000
00000000032   000000  000000  000000  000000
00000000033   000000  000000  000000  000000
00000000034   000000  000000  000000  000000
00000000035   000000  000000  000000  000000
00000000036   000000  000000  000000  000000
00000000037   000000  000000  000000  000000
```

If you want to change the `Format`, `Window Size`, or `Window Font` from the `Memory Data` window, press the MENU mouse button and choose the menu item you want:

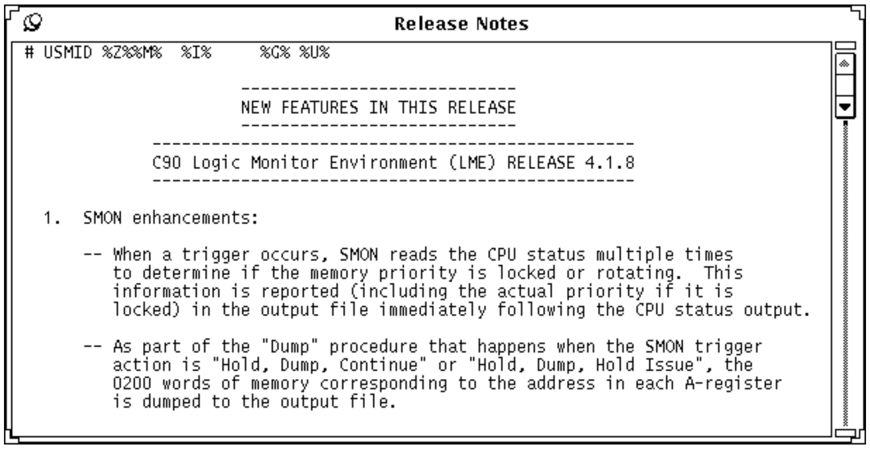


```
00000000543  000000 000000 000000 000000
00000000544  000000 020000 000000 000000
00000                          00000 000000
00000                          00000 000000
00000                          00000 000000
00000                          00000 000000
00000                          00000 000000
00000                          00000 000000
00000                          00000 000000
00000                          00000 000000
00000                          00000 000000
00000                          00000 000000
00000                          00000 000000
00000                          00000 000003
00000                          00000 001025
00000000562  000000 020000 000000 000560
```

Viewing the Release Notes


Selected Param Set...
Expected Data for Selected Param Set...
CPU Buffer/Expected Data...
Test Point Data...
Memory...

To see the current LME release notes, choose **`View --> Release Notes`**, as shown at the left. LME displays the `Release Notes` window:



```
# USMID %Z%%M%  %I%     %G% %U%

                    ----------------------------
                    NEW FEATURES IN THIS RELEASE
                    ----------------------------
            --------------------------------------------------
            C90 Logic Monitor Environment (LME) RELEASE 4.1.8
            --------------------------------------------------

   1.  SMON enhancements:

       -- When a trigger occurs, SMON reads the CPU status multiple times
          to determine if the memory priority is locked or rotating.  This
          information is reported (including the actual priority if it is
          locked) in the output file immediately following the CPU status output.

       -- As part of the "Dump" procedure that happens when the SMON trigger
          action is "Hold, Dump, Continue" or "Hold, Dump, Hold Issue", the
          0200 words of memory corresponding to the address in each A-register
          is dumped to the output file.
```

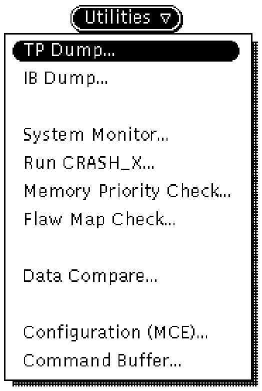
Copying the Parameter Set

This feature has not been implemented yet.

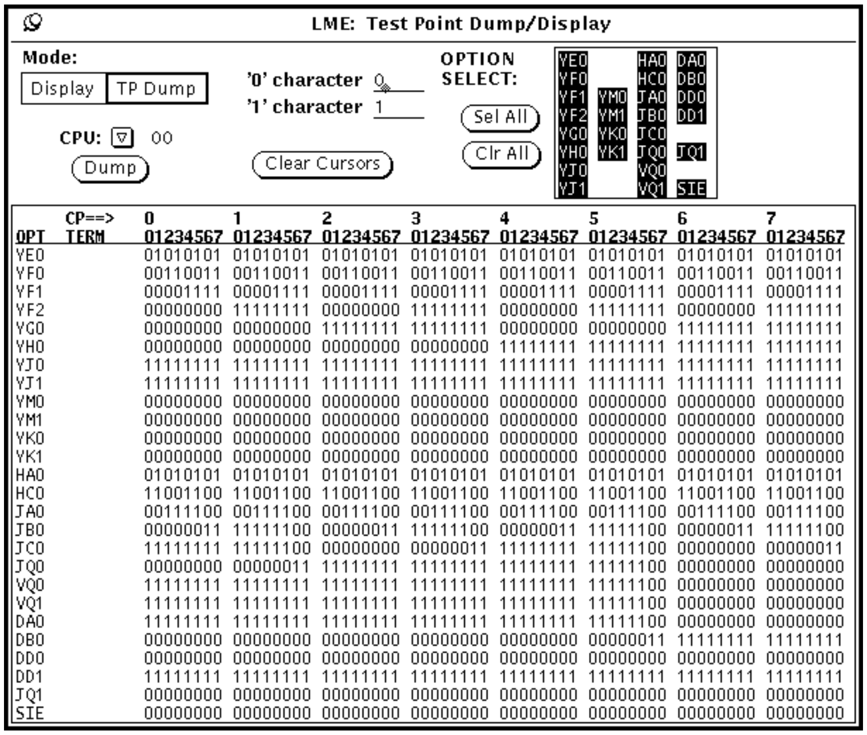
Copying the LM Data

This feature has not been implemented yet.

Using the Test Point Dump Utility



The TP dump utility uses the DM to dump all of the test points in all chips on a CPU. To use the test point dump utility, choose **`Utilities --> TP Dump`**, as shown at the left. LME displays the `LME: Test Point Dump/Display` window:



'0' character 0
'1' character 1
CPU: 00

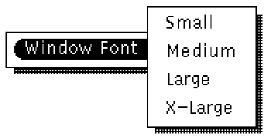
```
YE0        HA0  DA0
YF0        HC0  DB0
YF1  YM0   JA0  DD0
YF2  YM1   JB0  DD1
YG0  YK0   JC0
YH0  YK1   JQ0  JQ1
YJ0        VQ0
YJ1        VQ1  SIE
```

```
     CP==>  0        1        2        3        4        5        6        7
OPT  TERM   01234567 01234567 01234567 01234567 01234567 01234567 01234567 01234567
YE0         01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101
YF0         00110011 00110011 00110011 00110011 00110011 00110011 00110011 00110011
YF1         00001111 00001111 00001111 00001111 00001111 00001111 00001111 00001111
YF2         00000000 11111111 00000000 11111111 00000000 11111111 00000000 11111111
YG0         00000000 00000000 11111111 11111111 00000000 00000000 11111111 11111111
YH0         00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111
YJ0         11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
YJ1         11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
YM0         00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
YM1         00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
YK0         00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
YK1         00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
HA0         01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101
HC0         11001100 11001100 11001100 11001100 11001100 11001100 11001100 11001100
JA0         00111100 00111100 00111100 00111100 00111100 00111100 00111100 00111100
JB0         00000011 11111100 00000011 11111100 00000011 11111100 00000011 11111100
JC0         11111111 11111100 00000000 00000011 11111111 11111100 00000000 00000011
JQ0         00000000 00000011 11111111 11111111 11111111 11111100 00000000 00000000
VQ0         11111111 11111111 11111111 11111111 11111111 11111100 00000000 00000000
VQ1         11111111 11111111 11111111 11111111 11111111 11111100 00000000 00000000
DA0         11111111 11111111 11111111 11111111 11111111 11111100 00000000 00000000
DB0         00000000 00000000 00000000 00000000 00000000 00000011 11111111 11111111
DD0         00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
DD1         11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
JQ1         00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
SIE         00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

You can perform the following actions from this window:

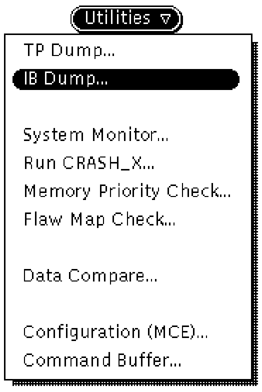
- Toggle between test point display mode and test point dump mode
- Specify the CPU (choose from `CPU:` ▽) you want to view
- Create a dump of the DM information
- Change the 0 and 1 characters on the display [enter the new character(s) in the corresponding field(s)]
- Clear the cursors (you set cursors to highlight lines within the display by clicking on the lines you want to highlight)

- Select all options to be displayed (click on Sel All)
- Clear all options from being displayed (click on Clr All)
- Select/deselect individual options to display by clicking on the options in the `OPTION SELECT` box
- Click on the MENU mouse button in the text area to access a menu to change the `Window Font` size:

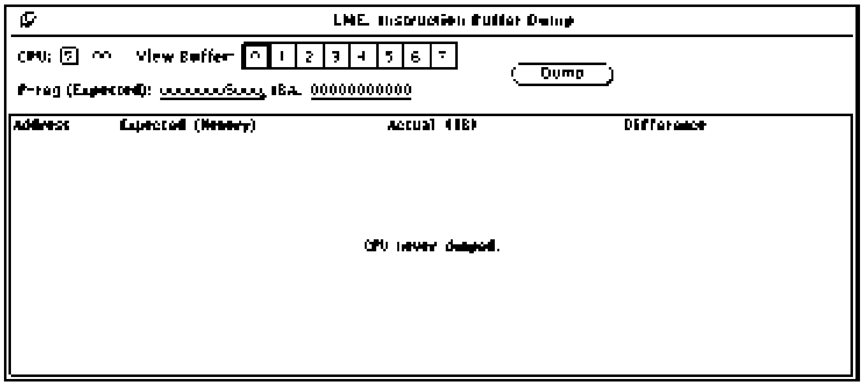


Using the Instruction Buffer Dump Utility

To dump the contents of a CPU's instruction buffers, perform the following procedure:



1. Choose **`Utilities --> IB Dump`**, as shown at the left. LME displays the `LME: Instruction Buffer Dump` window:



CPU: ▽ 00

P-reg (Expected): [illegible] IBA: 00000000000

Address Expected (Memory) Actual (IB) Difference

CPU never dumped.

2. Choose the CPU from the `CPU:` ▽ that corresponds to the instruction buffers you want to dump.

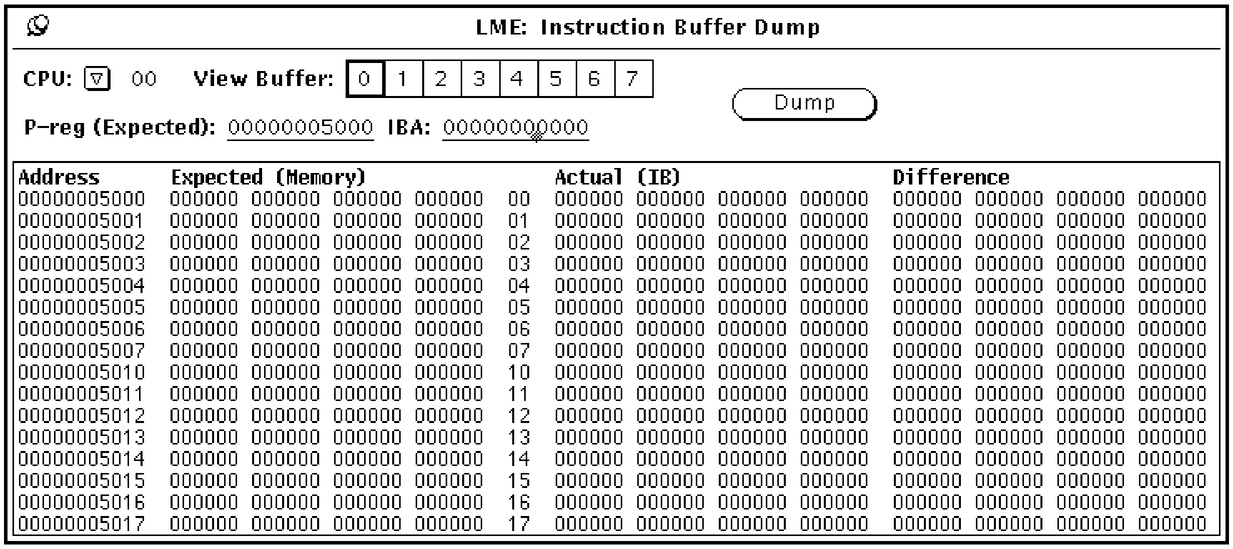
3. Double click on the `P–reg (Expected)` field. Type the memory address of the data with which you want the contents of the instruction buffers compared, and press the return (↵) key.

4. Double click in the `IBA` field, enter a base value for relative addressing, and press the return (↵) key.

   The instruction buffer dump displays mainframe memory relative to the value you enter in the `IBA` field. For example, if the `IBA:` field contains the value 1000 and the `P–reg (Expected)` field contains the value 200, the address column shows address 200 relative to the IBA. This address is actually mainframe memory address 1200.

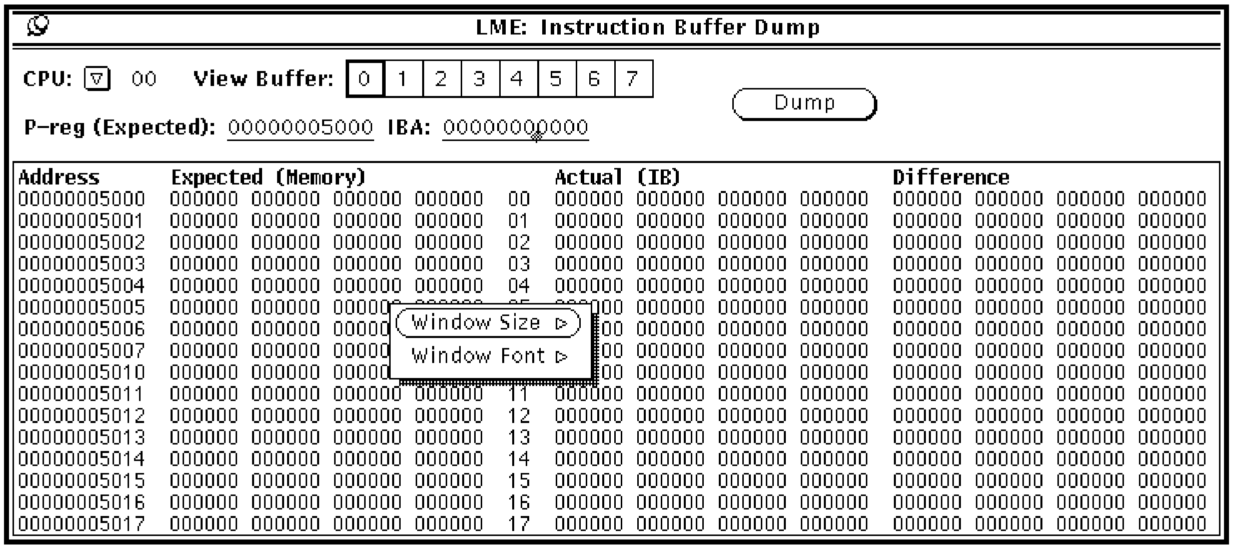
5. Click on Dump. LME updates the `LME: Instruction Buffer Dump` window.

Once the instruction buffers are dumped, you can view any of the instruction buffers for the CPU. In the `View Buffer` field, click on the instruction buffer (numbered 0 through 7) you want to view. LME displays the instruction buffer contents. For example, the following snap displays the contents of instruction buffer 0 for CPU 0.



```
CPU: 00     View Buffer: [0] 1 2 3 4 5 6 7
P-reg (Expected): 00000005000   IBA: 00000000000

Address      Expected (Memory)                       Actual (IB)                         Difference
00000005000  000000 000000 000000 000000   00  000000 000000 000000 000000   000000 000000 000000 000000
00000005001  000000 000000 000000 000000   01  000000 000000 000000 000000   000000 000000 000000 000000
00000005002  000000 000000 000000 000000   02  000000 000000 000000 000000   000000 000000 000000 000000
00000005003  000000 000000 000000 000000   03  000000 000000 000000 000000   000000 000000 000000 000000
00000005004  000000 000000 000000 000000   04  000000 000000 000000 000000   000000 000000 000000 000000
00000005005  000000 000000 000000 000000   05  000000 000000 000000 000000   000000 000000 000000 000000
00000005006  000000 000000 000000 000000   06  000000 000000 000000 000000   000000 000000 000000 000000
00000005007  000000 000000 000000 000000   07  000000 000000 000000 000000   000000 000000 000000 000000
00000005010  000000 000000 000000 000000   10  000000 000000 000000 000000   000000 000000 000000 000000
00000005011  000000 000000 000000 000000   11  000000 000000 000000 000000   000000 000000 000000 000000
00000005012  000000 000000 000000 000000   12  000000 000000 000000 000000   000000 000000 000000 000000
00000005013  000000 000000 000000 000000   13  000000 000000 000000 000000   000000 000000 000000 000000
00000005014  000000 000000 000000 000000   14  000000 000000 000000 000000   000000 000000 000000 000000
00000005015  000000 000000 000000 000000   15  000000 000000 000000 000000   000000 000000 000000 000000
00000005016  000000 000000 000000 000000   16  000000 000000 000000 000000   000000 000000 000000 000000
00000005017  000000 000000 000000 000000   17  000000 000000 000000 000000   000000 000000 000000 000000
```

You can press the MENU mouse button in the memory data tables in the `LME Instruction Buffer Dump` window to change window size or font type size:

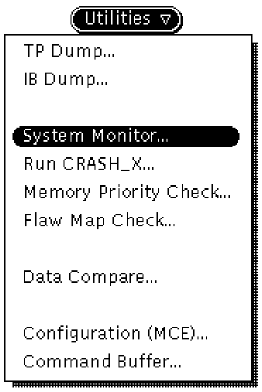


```
CPU: 00     View Buffer: [0] 1 2 3 4 5 6 7
P-reg (Expected): 00000005000   IBA: 00000000000

Address      Expected (Memory)                       Actual (IB)                         Difference
00000005000  000000 000000 000000 000000   00  000000 000000 000000 000000   000000 000000 000000 000000
00000005001  000000 000000 000000 000000   01  000000 000000 000000 000000   000000 000000 000000 000000
00000005002  000000 000000 000000 000000   02  000000 000000 000000 000000   000000 000000 000000 000000
00000005003  000000 000000 000000 000000   03  000000 000000 000000 000000   000000 000000 000000 000000
00000005004  000000 000000 000000 000000   04  000000 000000 000000 000000   000000 000000 000000 000000
00000005005  000000 000000 000000                    000000 000000 000000   000000 000000 000000 000000
00000005006  000000 000000 00000                     000000 000000 000000   000000 000000 000000 000000
00000005007  000000 000000 00000                     000000 000000 000000   000000 000000 000000 000000
00000005010  000000 000000 00000                     000000 000000 000000   000000 000000 000000 000000
00000005011  000000 000000 000000 000000   11  000000 000000 000000 000000   000000 000000 000000 000000
00000005012  000000 000000 000000 000000   12  000000 000000 000000 000000   000000 000000 000000 000000
00000005013  000000 000000 000000 000000   13  000000 000000 000000 000000   000000 000000 000000 000000
00000005014  000000 000000 000000 000000   14  000000 000000 000000 000000   000000 000000 000000 000000
00000005015  000000 000000 000000 000000   15  000000 000000 000000 000000   000000 000000 000000 000000
00000005016  000000 000000 000000 000000   16  000000 000000 000000 000000   000000 000000 000000 000000
00000005017  000000 000000 000000 000000   17  000000 000000 000000 000000   000000 000000 000000 000000
```

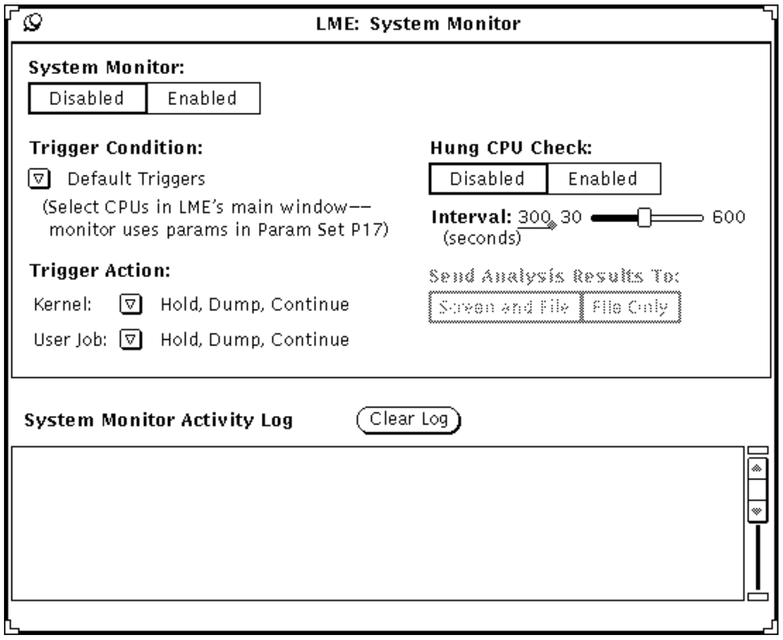
You can use either the `File --> Save --> CPU Data` or `File --> Save --> System Snapshot` command to save an instruction buffer dump.

You can use either the `File --> Load --> CPU Data` or `File --> Load --> System Snapshot` command to load a saved instruction buffer dump. Then, you should view the `LME: Instruction Buffer Dump` window, and select the appropriate `CPU` and `View Buffer` settings.

Using the System Monitor Utility

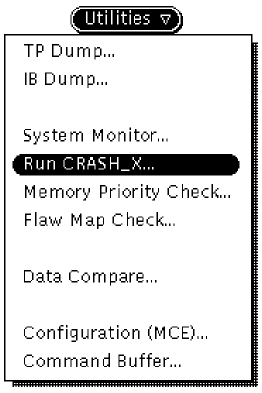


Choose **`Utilities --> System Monitor`**, as shown at the left. LME displays the `LME: System Monitor` window:



The System Monitor (SMON) utility automatically acquires CPU information during hardware and software failures by using the diagnostic monitor hardware. For more information about SMON, refer to the *CRAY C90 Series LME System Monitor Utility* document, publication number HDM-120-0.

Running the CRASH_X Utility



Choose **`Utilities --> Run CRASH_X`**, as shown at the left to run the `CRASH_X` utility. The `CRASH_X` utility collects data from multiple DM dumps to report the following information:

- Hold issues on A registers
- Hold issues on S registers
- Hold issues on V registers
- Hold issues on B/T registers
- Hold issues on functional units
- Hold issues on shared registers
- Hold issues on memory
- Hold issues on JA option (issue logic)
- Hold issues on JB option (issue logic)
- Solid port conflicts on ports A, A', B, B', C, C', D, and D'
- Active exchanges (any CPUs that are exchanging)
- Loop sequences of 1 to 15 instructions
- Activity from the last 1000 clock periods
- Test point dump data

If a CPU is holding issue, the utility checks that CPU's YM Release Enable signals and YJ Busy signals for activity. If there is a discrepancy between these signals, the utility determines which CPU arbitrates access to the memory subsection being accessed by the CPU that is holding issue. The utility then prints a message indicating that one of these two CPUs or an interconnecting wire is causing the problem.

If a shared register is holding issue, the current cluster number is reported for the CPU that is holding issue. If more than one CPU is holding issue in the same cluster, the message `ANY CPU IN CLUSTER # COULD BE THE FAILING CPU` is reported.

The `CRASH_X` utility performs the following functions:

1. `CRASH_X` puts the DM data into `/cri/cme/c90/usr/crash.cpu_`*`x`* files, where *x* indicates the number of the CPU.

2. `CRASH_X` calls the `/cri/cme/c90/bin/crash_check` executable file, which formats the collected data and writes it to the `/cri/cme/c90/usr/crash/crash_x_data` file.

3. The `/cri/cme/c90/bin/crash_check` file analyzes the data and writes an analysis to the `/cri/cme/c90/usr/crash/crash_x_out` file.

LME displays the `crash_x_out` and `crash_x_data` files in a separate window on the screen.

Using the Memory Priority Check Utility

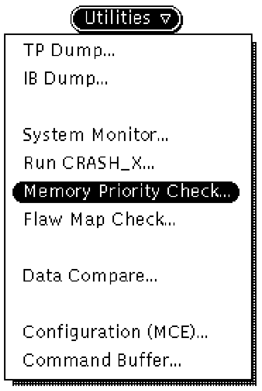
CAUTION

The memory priority check utility will Master Clear all CPUs in the mainframe. Use this utility after a mainframe crash only. This utility will crash UNICOS if it is running.

The memory priority check utility uses the maintenance channel and diagnostic monitor to determine whether the dynamic memory priority between two modules in a CRAY C90 series system is out of synchronization.

Run this utility in concurrent mode after a mainframe crash, but before you toggle I/O Master Clear, which resets memory priority.

Do not switch to offline mode before you run this utility; switching to offlline mode causes MCE to toggle I/O Master Clear.



Choose `Utilities --> Memory Priority Check`, as shown at the left, to run the memory priority check utility.

NOTE: For detailed information about the following procedure, view the online listing in the `/cri/cme/c90/rel/env0/lst/prichk.c.l` file.

This utility performs the following procedure to check memory priority:

1. The utility writes a pattern to each subsection. For the rest of this procedure, this pattern is called *pattern 1*.

2. The utility writes a different pattern to each subsection at a different area in memory. For the rest of this procedure, this pattern is called *pattern 2*.

3. The utility causes each CPU to preload its subsection read-out registers on the memory module. This is done by having each CPU sequentially read pattern 2 into its subsection read-out registers.

4. The utility uses the diagnostic monitors to have the CPUs synchronously read pattern 1, forcing all CPUs to conflict with each other. This requires the memory module to use the CPU priority table to arbitrate the references.

> If memory priority is out of synchronization, one-half of the CPUs will not be able to correctly read from the affected subsection(s). These CPUs will read *stale* data (pattern 2) instead of the expected data (pattern 1).

The utility performs this procedure 16 times. For each pass through the procedure, the priority is set to the next CPU to ensure that the priority is locked to each CPU in the system one time. The utility then repeats the procedure without locking the priority to a specific CPU.

What Happens If the Memory Priority Check Utility Does Not Detect Errors?

If the utility does not detect errors, LME displays a notice that indicates no errors were detected.

What Happens If the Memory Priority Check Utility Detects Errors?

If the utility detects errors, the utility generates an output file containing a table that summarizes which CPUs failed and the subsections with which they failed. LME displays the output file in a window on the screen.

Figure 5-3 shows example excerpts from the output file this utility generates. The bit failure summary for each CPU appears at the end of the output file. A 32-bit difference of 177777 004000 (octal) is shown in the summary if one of the memory modules fails with a priority-synchronization problem. Use this summary to determine which of the two memory modules failed (each memory module handles 32 data bits). In this example, the module handling the upper bits (bits 2^{32} to 2^{63}) is the failing module.

<text deleted>

```
FAILING SUBSECTION TABLES
Each table contains failure information for 8 subsections.
The first column shows priority setting being tested.
Columns 2–9 are bit masks indicating which cpus failed.

PRIORITY   SS=000 SS=010 SS=020 SS=030 SS=040 SS=050 SS=060 SS=070
 cpu 0     000000 000000 000125 000125 000000 000000 000125 000125
 cpu 1     000000 000000 000063 000063 000000 000000 000063 000063
 cpu 2     000000 000000 000125 000125 000000 000000 000125 000125
 cpu 3     000000 000000 000017 000017 000000 000000 000017 000017
 cpu 4     000000 000000 000125 000125 000000 000000 000125 000125
 cpu 5     000000 000000 000063 000063 000000 000000 000063 000063
 cpu 6     000000 000000 000125 000125 000000 000000 000125 000125
 cpu 7     000000 000000 000340 000340 000000 000000 000340 000340
 cpu10     000000 000000 000125 000125 000000 000000 000125 000125
 cpu11     000000 000000 000063 000063 000000 000000 000063 000063
 cpu12     000000 000000 000125 000125 000000 000000 000125 000125
 cpu13     000000 000000 000017 000017 000000 000000 000017 000017
 cpu14     000000 000000 000125 000125 000000 000000 000125 000125
 cpu15     000000 000000 000063 000063 000000 000000 000063 000063
 cpu16     000000 000000 000125 000125 000000 000000 000125 000125
 cpu17     000000 000000 000340 000340 000000 000000 000340 000340
unlocked   000000 000000 000105 000011 000000 000000 000301 000123
```

<text deleted>

```
SUMMARY OF FAILING BITMASKS

SS 0020
-------
  CPU         FAILING BITMASK
cpu  0      177777 004000 000000 000000
cpu  1      177777 004000 000000 000000
cpu  2      177777 004000 000000 000000
cpu  3      177777 004000 000000 000000
cpu  4      177777 004000 000000 000000
cpu  5      177777 004000 000000 000000
cpu  6      177777 004000 000000 000000
cpu  7      177777 004000 000000 000000

SS 0030
-------
  CPU         FAILING BITMASK
cpu  0      177777 004000 000000 000000
cpu  1      177777 004000 000000 000000
cpu  2      177777 004000 000000 000000
cpu  3      177777 004000 000000 000000
cpu  4      177777 004000 000000 000000
cpu  5      177777 004000 000000 000000
cpu  6      177777 004000 000000 000000
cpu  7      177777 004000 000000 000000
```

<text deleted>

Figure 5-3. Excerpts from Memory Priority Check Output File

Using the Flaw Map Check Utility

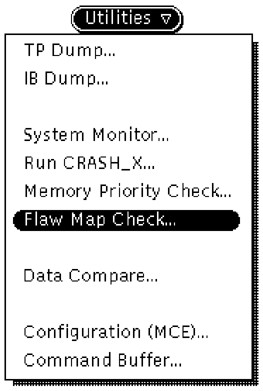
The flaw map check utility uses the maintenance channel, the diagnostic monitor, and the error-correction code (ECC) maintenance features of the CPUs to determine the current flaw-map configuration. Then, the utility compares this flaw map to the expected MCE flaw map.

Run this utility in concurrent mode after a mainframe crash, but before you toggle I/O Master Clear, which resets memory priority.

Do not switch to offline mode before you run this utility; switching to offlline mode causes MCE to toggle I/O Master Clear.

CAUTION

The flaw map check utility will Master Clear all CPUs in the mainframe. Use this utility after a mainframe crash only.



Choose **`Utilities --> Flaw Map Check`**, as shown at the left, to run the flaw map check utility.

This utility performs the following procedure to check the flaw map:

1. This utility writes 32 data patterns to each memory bank.

   The data patterns place a unique value in each nibble. This enables the utility to determine the chip configuration by determining where the read data begins to shift to the left.

2. The utility reconfigures the flaw map to the default settings.

3. The utility reads in the data and check bits, which reads in the contents of chips 0 through 9.

4. The utility reconfigures the flaw map with all zeros, which configures the spare chip in.

5. The utility reads the check bits, which reads in the contents of the spare chip.

6. The utility compares the data written to the data stored in the memory chips. The utility uses this information to determine the current flaw-map configuration.

The utility compares the current flaw map with the expected flaw map. If MCE is in concurrent mode, the expected flaw map is the UNICOS flaw map. If MCE is in offline mode, the expected flaw map is the last flaw map that MCE wrote to memory.

What Happens If the Flaw Map Check Utility Does Not Detect Errors?

If the flaw maps are the same, LME displays a notice that indicates the flaw maps match.

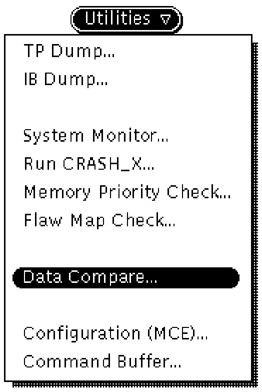
What Happens If the Flaw Map Check Utility Detects Errors?

If the flaw maps are different, LME displays the output file from this utility. Figure 5-4 shows an excerpt from an example output file.

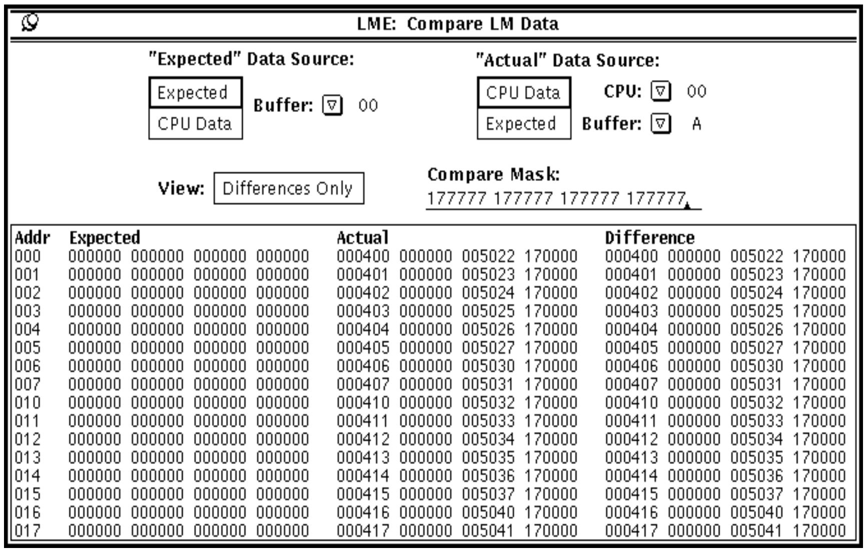
```
*******************************************************************************
                 LME–C90 4.1.8    Flaw Map Check Summary
       Check performed on S/N 4030 at:  Tue Feb  7 16:08:54 1995
*******************************************************************************


Flaw Map Check INFO:     CPU 00 selected as I/O and test CPU

Flaw Map Check INFO:     Determining flawmap for comparison...
    ––MCE in offline mode, using last flawmap written to the hardware


TABLE OF DIFFERENCES DETECTED

NOTE:  * a value of EEE in either half of a bank in the MCE FLAW MAP
         means that the disable code was written
       * a value of ABC in either half of a bank in the HARDWARE FLAW MAP
         means that the utility could not determine the flaw map value

       Either of these values will result in ”????” appearing in the
       DIFFERENCE field for that half of the bank

BANK       MCE FLAW MAP       HARDWARE FLAW MAP         DIFFERENCE
-----------------------------------------------------------------------
0000    0xAAAAAAAAAAAAAAAA    0xAAAAAAAAAAAAAAAA    0x0000000000000000
0001    0xAAAAAAAAAAAAAAAA    0xAAAAAAAAAAAAAAAA    0x0000000000000000
0002    0xAAAAAAAAAAAAAAAA    0xAAAAAAAAAAAAAAAA    0x0000000000000000
0003    0xAAAAAAAAAAAAAAAA    0xAAAAAAAAAAAAAAAA    0x0000000000000000
0004    0x0000055500000444    0x0000000000000000    0x0000055500000444
0005    0x0000044400000555    0x0000000000000000    0x0000044400000555
0006    0x0000033300000666    0x0000000000000000    0x0000033300000666
0007    0x0000022200000777    0x0000000000000000    0x0000022200000777
0010    0xAAAAAAAAAAAAAAAA    0xAAAAAAAAAAAAAAAA    0x0000000000000000
0011    0xAAAAAAAAAAAAAAAA    0xAAAAAAAAAAAAAAAA    0x0000000000000000
0012    0xAAAAAAAAAAAAAAAA    0xAAAAAAAAAAAAAAAA    0x0000000000000000
0013    0xAAAAAAAAAAAAAAAA    0xAAAAAAAAAAAAAAAA    0x0000000000000000
```

Figure 5-4. Excerpt from Flaw Map Check Utility Output File

Using the Data Compare Utility



Choose **`Utilities --> Data Compare`**, as shown at the left to use LME to make comparisons between the expected data and all CPU data. LME displays the `LME: Compare LME Data` window:


LME: Compare LM Data

"Expected" Data Source: Expected / CPU Data Buffer: ▽ 00

"Actual" Data Source: CPU Data / Expected CPU: ▽ 00 Buffer: ▽ A

View: Differences Only

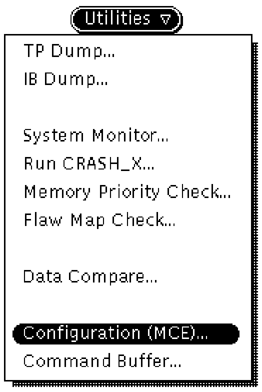
Compare Mask: 177777 177777 177777 177777

```
Addr  Expected                       Actual                         Difference
000   000000 000000 000000 000000   000400 000000 005022 170000   000400 000000 005022 170000
001   000000 000000 000000 000000   000401 000000 005023 170000   000401 000000 005023 170000
002   000000 000000 000000 000000   000402 000000 005024 170000   000402 000000 005024 170000
003   000000 000000 000000 000000   000403 000000 005025 170000   000403 000000 005025 170000
004   000000 000000 000000 000000   000404 000000 005026 170000   000404 000000 005026 170000
005   000000 000000 000000 000000   000405 000000 005027 170000   000405 000000 005027 170000
006   000000 000000 000000 000000   000406 000000 005030 170000   000406 000000 005030 170000
007   000000 000000 000000 000000   000407 000000 005031 170000   000407 000000 005031 170000
010   000000 000000 000000 000000   000410 000000 005032 170000   000410 000000 005032 170000
011   000000 000000 000000 000000   000411 000000 005033 170000   000411 000000 005033 170000
012   000000 000000 000000 000000   000412 000000 005034 170000   000412 000000 005034 170000
013   000000 000000 000000 000000   000413 000000 005035 170000   000413 000000 005035 170000
014   000000 000000 000000 000000   000414 000000 005036 170000   000414 000000 005036 170000
015   000000 000000 000000 000000   000415 000000 005037 170000   000415 000000 005037 170000
016   000000 000000 000000 000000   000416 000000 005040 170000   000416 000000 005040 170000
017   000000 000000 000000 000000   000417 000000 005041 170000   000417 000000 005041 170000
```

You can perform the following actions from this window:

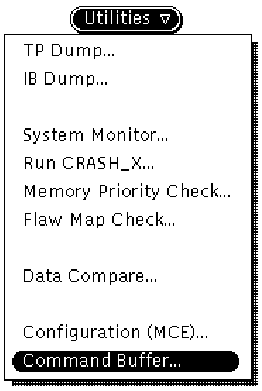
- Select the expected data source (select either expected or CPU data and choose the buffer and/or CPU from the corresponding ▽)
- Select the actual data source (select either CPU data or expected data and choose the buffer and/or CPU from the corresponding ▽)
- View only the addresses that have differences
- Select which bits will be compared by changing the `Compare Mask` value

Using the Configuration Utility



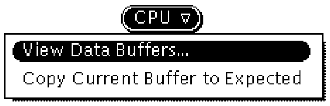
To start the MCE configuration application, choose **`Utilities -->`** **`Configuration (MCE)`**, as shown at the left. For more information about using MCE, refer to Section 2, “Mainframe Configuration Environment.”

Starting the Command Buffer Parser Utility

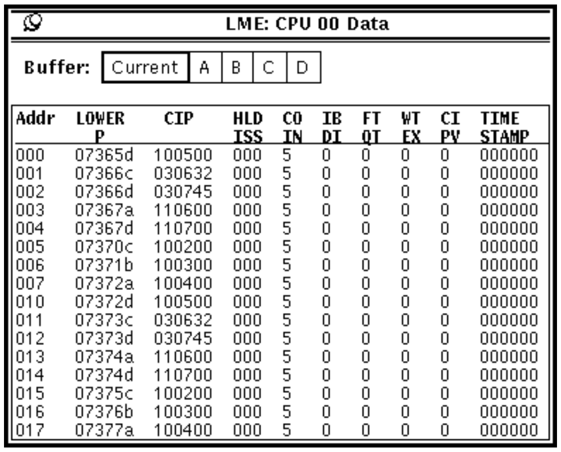


To start the command buffer parser utility, choose **`Utilities -->`** **`Command Buffer`**, as shown at the left. This utility allows you to automate functions of LME. For more information about the command buffer parser (CBP) utility, refer to Section 6, “CRAY C90 CBP Runtime Module.”

Viewing the Data Buffers for a CPU



Choose **`CPU --> View Data Buffers`**, as shown at the left, to see the data buffers for the selected CPU. A data window appears for the selected CPU:



LME: CPU 00 Data

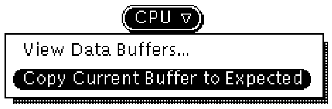
Buffer: Current | A | B | C | D

Addr	LOWER P	CIP	HLD ISS	CO IN	IB DI	FT QT	WT EX	CI PV	TIME STAMP
000	07365d	100500	000	5	0	0	0	0	000000
001	07366c	030632	000	5	0	0	0	0	000000
002	07366d	030745	000	5	0	0	0	0	000000
003	07367a	110600	000	5	0	0	0	0	000000
004	07367d	110700	000	5	0	0	0	0	000000
005	07370c	100200	000	5	0	0	0	0	000000
006	07371b	100300	000	5	0	0	0	0	000000
007	07372a	100400	000	5	0	0	0	0	000000
010	07372d	100500	000	5	0	0	0	0	000000
011	07373c	030632	000	5	0	0	0	0	000000
012	07373d	030745	000	5	0	0	0	0	000000
013	07374a	110600	000	5	0	0	0	0	000000
014	07374d	110700	000	5	0	0	0	0	000000
015	07375c	100200	000	5	0	0	0	0	000000
016	07376b	100300	000	5	0	0	0	0	000000
017	07377a	100400	000	5	0	0	0	0	000000

Click on the buffer of DM data that you want to view.

NOTE: The CPU menu button activates when you click on an assigned parameter set displayed next to a CPU.

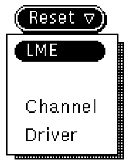
Copying the Current Buffer Data to the Expected Data



To copy the current buffer data to the expected data for a parameter set, choose **`CPU --> Copy Current Buffer to Expected`**, as shown at the left. This provides a way to use data from a DM run for comparison with a future DM run.

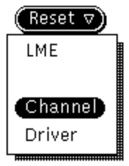
NOTE: The CPU menu button activates when you click on an assigned parameter set displayed next to a CPU.

Resetting LME



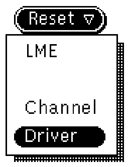
To initialize the driver and the maintenance channel, choose **`Reset --> LME`**, as shown at the left.

Resetting the Channel



To initialize the maintenance channel, choose **`Reset --> Channel`**, as shown at the left.

Resetting the Driver



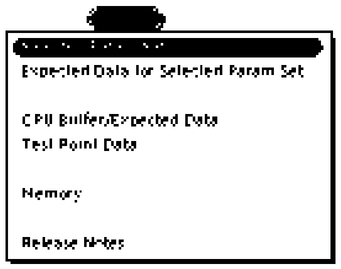
To initialize the driver (the software that controls the FEI-3 or FEI-4 hardware that is connected to the maintenance channel), choose **`Reset --> Driver`**, as shown at the left.

Using LME

There are three steps to using the LME:

1. Setting up the parameters and expected data
2. Selecting the CPUs to run
3. Viewing the DM data

Setting Up the Parameters and Expected Data



The parameter sets (`P00` through `P17`) contain the parameters that tell the DM how to run. Each parameter set is separate: this enables you to have each CPU's DM run a different parameter set or to have multiple CPUs run the same parameter set. You can edit the parameter sets by choosing **`View --> Selected Parameter Set`**, as shown at the left, and manipulating the information that appears in the corresponding `LME: Edit Parameter Set` window, shown in Figure 5-5.

There are two ways to view and manipulate the parameter information. You can view the `User-Defined` information, contained in the `LME: Edit Parameter Set` window, shown in Figure 5-5, or you can view the `Binary PL` information, shown in Figure 5-7, by clicking on Binary PL.

The `User-Defined` window enables you to select parameter information with your mouse. The `Binary PL` window enables you to modify parameters by changing the values at a bit level.

Using the Edit Parameter Set Window to Set Parameters

The `Edit Parameter Set` window shows the settings for the parameter set that was selected (clicked on) in the LME base window; this parameter set is displayed in parentheses in the window title [for example, Figure 5-5 shows that parameter set 00 is being edited, indicated by `(P00)`]. To reset the parameters in this window to the starting values, click on (Set to Defaults). To edit the expected data, click on (Edit Expected Data...) .

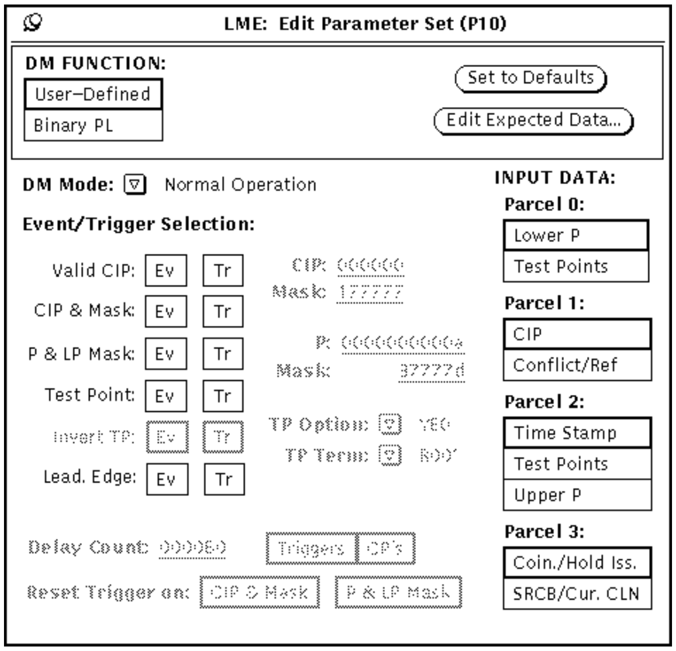
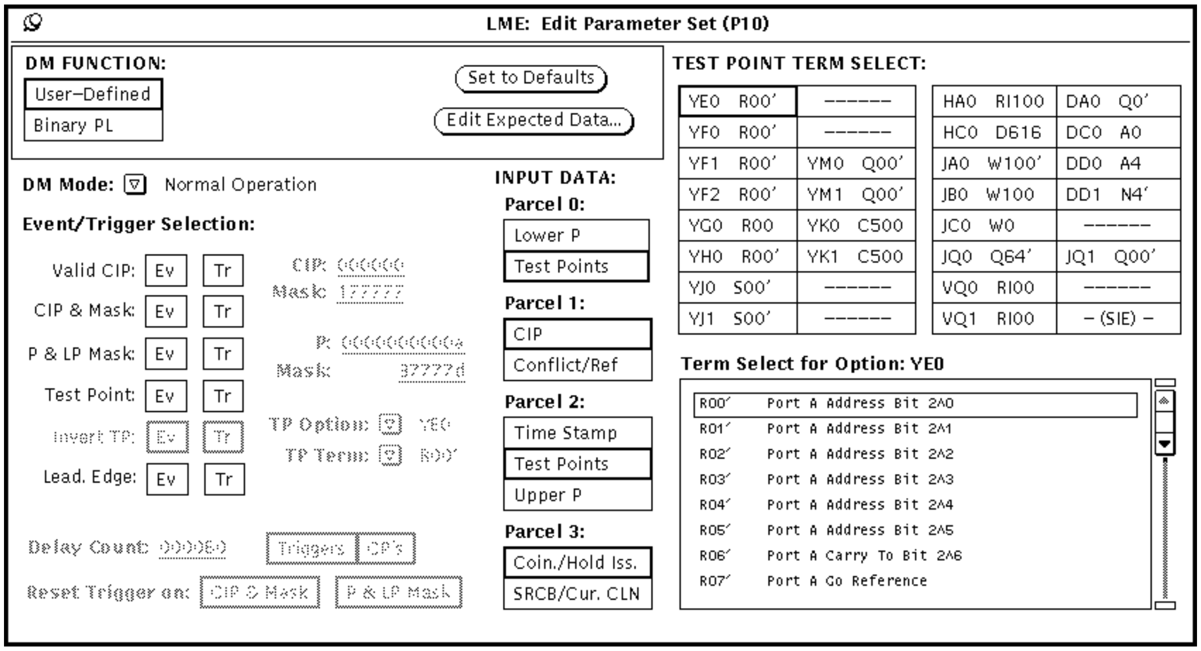


Figure 5-5. LME Edit Parameter Set Window

The parameters available in this window are divided into the following areas: `Operation Mode`, `Event/Trigger Selection`, and `Input Data`.

NOTE: The `TEST POINT TERM SELECT`: options appear if you click on [Test Points] in either the parcel 0 or the parcel 2 input data selections (refer to Figure 5-6).



LME: Edit Parameter Set (P10)

DM FUNCTION:
User–Defined
Binary PL

Set to Defaults
Edit Expected Data...

DM Mode: ▽ Normal Operation

Event/Trigger Selection:

Valid CIP:	Ev	Tr
CIP & Mask:	Ev	Tr
P & LP Mask:	Ev	Tr
Test Point:	Ev	Tr
Invert TP:	Ev	Tr
Lead. Edge:	Ev	Tr

CIP: 000000
Mask: 177777
P: 00000000004
Mask: 37777d
TP Option: ▽ YE0
TP Term: ▽ R00'

Delay Count: 000060 Triggers CP's
Reset Trigger on: CIP & Mask P & LP Mask

INPUT DATA:
Parcel 0: Lower P / Test Points
Parcel 1: CIP / Conflict/Ref
Parcel 2: Time Stamp / Test Points / Upper P
Parcel 3: Coin./Hold Iss. / SRCB/Cur. CLN

TEST POINT TERM SELECT:

YE0 R00′	-------	HA0 RI100	DA0 Q0′
YF0 R00′	-------	HC0 D616	DC0 A0
YF1 R00′	YM0 Q00′	JA0 W100′	DD0 A4
YF2 R00′	YM1 Q00′	JB0 W100	DD1 N4′
YG0 R00	YK0 C500	JC0 W0	-------
YH0 R00′	YK1 C500	JQ0 Q64′	JQ1 Q00′
YJ0 S00′	-------	VQ0 RI00	-------
YJ1 S00′	-------	VQ1 RI00	– (SIE) –

Term Select for Option: YE0

```
R00′   Port A Address Bit 2^0
R01′   Port A Address Bit 2^1
R02′   Port A Address Bit 2^2
R03′   Port A Address Bit 2^3
R04′   Port A Address Bit 2^4
R05′   Port A Address Bit 2^5
R06′   Port A Carry To Bit 2^6
R07′   Port A Go Reference
```

Figure 5-6. LME Edit Parameter Set Window with Test Point Term Select Options

Operation Mode

The following operation modes are available (choose them from the `Operation Mode:` ▽):

- Normal operation
- Hold issue after delayed trigger
- Hold issue after undelayed trigger
- TP dump/IB test
- IB dump/IB test

Event/Trigger Selection

Events and triggers indicate when the DM should start and stop recording data. You can use the `Event/Trigger Selection` section of this window to set when the DM should start recording (an event) and when the DM should stop recording (a trigger). There are several items that you can select as events or triggers. To select an item as an event, click on Evt ; to select an item as a trigger, click on Trig . You can set the following items as events or triggers.

Item	Description
`Valid CIP`	Starts/stops on a valid current instruction parcel (CIP)
`CIP & Mask`	Starts/stops on the CIP indicated in the `CIP` field (controlled by mask indicated in `Mask` field)
`P & LP Mask`	Starts/stops on the P value indicated in the `P` field (controlled by mask indicated in the `Mask` field below the `P` field)
`Test Point`	Starts/stops on the test point indicated by the `TP Option` and `TP Term` values (select these values from the abbreviated menu button – test point bit is 1)
`Invert TP`	Starts/stops when a test point bit is 0
`Lead. Edge`	Records an event or trigger once when it occurs (if this option is selected) or records an event or trigger every clock period after it occurs (if this option is not selected)

When you set triggers, you can set the delay count to create a window or frame around a certain event. This enables you to look at what was happening before and after the trigger occurred because recording is adjusted to the `Delay Count` specified.

Input Data

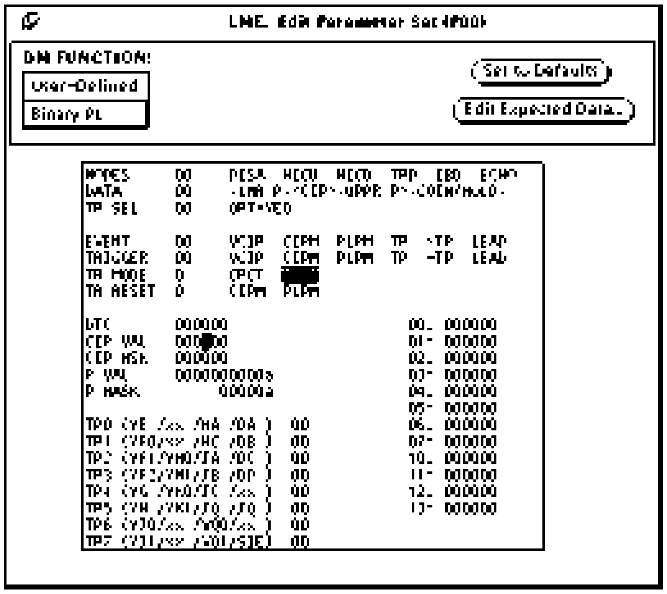
The `INPUT DATA` area is where you specify what will be placed in the parcels of input to the DM. Parcel 0 can contain the lower P-register data or tests points (select the test points in the `TEST POINT TERM SELECT` area). Parcel 1 can contain the current instruction parcel (CIP) or conflict/reference data. Parcel 2 can contain a time stamp, test points (select the test points in the `TEST POINT TERM SELECTION` area), or the upper P-register data. Parcel 3 can contain the coincidence/hold issue data or the shared resource control byte (SRCB) and current cluster number (CLN).

Test Point Term Select

The `TEST POINT TERM SELECT` area is where you select the test point terms that are used for the input data to the DM if you clicked on Test Points for `Parcel 0` or `Parcel 2` in the `INPUT DATA` area. The first block of test points can be set for `Parcel 0`, and the second block of test points can be set for `Parcel 2`. Click on the test point terms you want to select. When a test point term is selected, it is selected for an entire row (4 options); when the term for one option in a row changes, the other terms also change.

Using the Binary PL Parameter Window to Set Parameters

You can access the `Binary PL` version of the `LME Edit Parameter Set` window, shown in Figure 5-7, by clicking on [Binary PL]. This version of the window enables you to change the parameters by editing the octal parameters shown in the window. The parameters in this window correspond to the parameters in the `User-Defined` window (for the bit-masked items, reverse video indicates the selected items).



```
MODES      00     PESA  HECU  HECD  TPP   CBD   ECHO
DATA       00     ·LMA P·/CCP\·UPPR P\·COEN/HOLD·
TP SEL     00     OPT=SED

EVENT      00     VCJP  CCPM  PLPM  TP    -TP   LEAD
TRIGGER    00     VCJP  CCPM  PLPM  TP    -TP   LEAD
TR MODE    0      CPCT  [illegible]
TR RESET   0      CCPM  PLPM

LTC        000000                          00.  000000
CCP VAL    000000                          01-  000000
CCP MSK    000000                          02.  000000
P VAL      0000000000a                     03-  000000
P MASK          00000a                     04.  000000
                                           05-  000000
TP0 (YE  /xx /HA /DA )  00                 06.  000000
TP1 (YF0/xx /HC /DB )  00                  07-  000000
TP2 (YF1/YH0/JA /DC )  00                  10.  000000
TP3 (YF2/YH1/JB /DP )  00                  11-  000000
TP4 (YG /YK0/JC /xx )  00                  12.  000000
TP5 (YH /YK1/JQ /JQ )  00                  13-  000000
TP6 (YJ0/xx /Y00/xx )  00
TP7 (YJ1/xx /Y01/SJE)  00
```

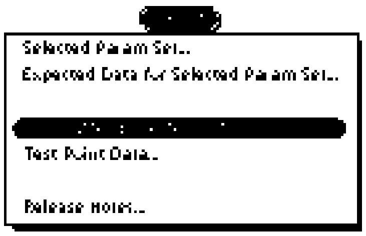
Figure 5-7. LME Binary PL Parameter Window

Item	Description
`MODES`	Bit mask representing the modes that are available
`DATA`	Bit mask representing the input data for parcels 0, 1, 2, and 3
`TP SEL`	Test point term selection
`EVENT`	Bit mask representing the items that have been selected as events

Item	Description
`TRIGGER`	Bit mask representing the items that have been selected as triggers
`TR MODE`	Bit mask representing trigger mode (triggers or CPs)
`TR RESET`	Bit mask representing what to reset trigger on (CIP & Mask or P & LP Mask)
`DTC`	Delay count
`CIP VAL`	Current instruction parcel value
`CIP MASK`	Current instruction parcel mask
`P VAL`	P-register value (corresponds to `P` field on `User–Defined` display)
`P MASK`	P-register mask
`TP0 – TP7`	Test point term selections
`00 – 13`	Raw 12-parcel DM parameter list (duplicate of binary PL information in different form)

Setting up the Expected Data

You will need to set up the expected data that will be used for comparisons with the results of running the parameters. To set up the expected data, perform the following procedure:



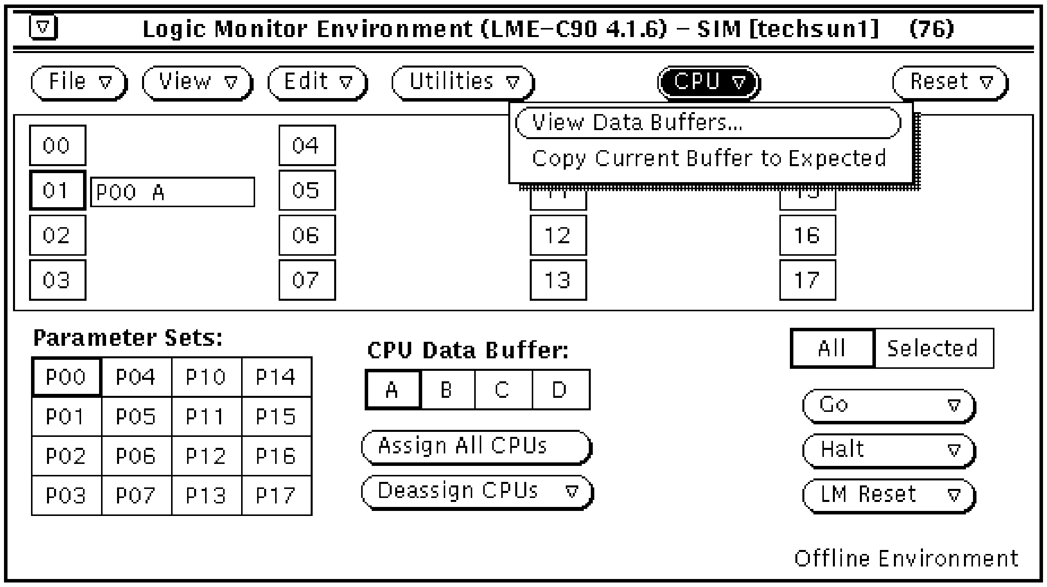
1. You can manually enter the expected data by choosing **`View --> CPU Buffer/Expected Data`**, as shown at the left.

2. Click on `Expected Data` and `View...`. The `LME: Expected Data` window for the buffer specified by your choice in the `Buffer:` ▽ appears.

To copy the results of an execution, perform the following steps:

1. Click on the parameter set number and buffer letter next to a CPU (these values will be surrounded by a box and the CPU menu button will become active)

2. Choose **`CPU --> Copy Current Buffer to Expected`**, as shown in the following window illustration.



Selecting the CPUs to Run

Selecting the CPUs you want to run requires assigning parameter sets to the CPUs you want to monitor. To do this, perform the following steps in the LME base window:

1. Click on the parameter set you want to use.

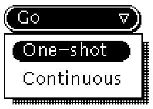
2. Click on the CPU data buffer you want to use. The data is returned into the buffers. There are four buffers (A, B, C, and D) for each CPU, so you can make comparisons between the results of several DM runs.

3. Click on the CPU(s) you want to assign to the parameter set; more than one CPU can be assigned to a parameter set.

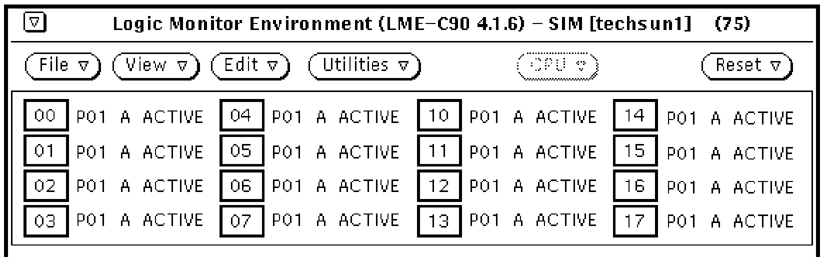
NOTE: You can use the Assign All CPUs button to assign all the CPUs to the current parameter set.

4. Repeat Steps 1 through 3 to select all the CPUs you want to run.

5. Perform one of the following commands:

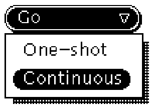


- Choose **`Go --> One-shot`**, as shown at the left, to run the parameter set(s) one time. `ACTIVE` appears in the CPU status display to indicate active mode (refer to Figure 5-8). The parameter set will run until the DM triggers or until you click on Halt ▽.

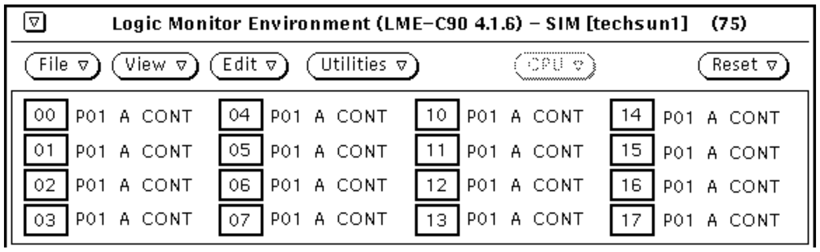


```
Logic Monitor Environment (LME–C90 4.1.6) – SIM [techsun1]   (75)

00 P01 A ACTIVE   04 P01 A ACTIVE   10 P01 A ACTIVE   14 P01 A ACTIVE
01 P01 A ACTIVE   05 P01 A ACTIVE   11 P01 A ACTIVE   15 P01 A ACTIVE
02 P01 A ACTIVE   06 P01 A ACTIVE   12 P01 A ACTIVE   16 P01 A ACTIVE
03 P01 A ACTIVE   07 P01 A ACTIVE   13 P01 A ACTIVE   17 P01 A ACTIVE
```

Figure 5-8. Active CPUs in One-shot Mode



- Choose **`Go --> Continuous`**, as shown at the left, to run the parameter set(s) continuously. `CONTINUOUS` appears in the CPU status display to indicate continuous mode (refer to Figure 5-9). The parameter set will run until you click on Halt ▽.



```
Logic Monitor Environment (LME–C90 4.1.6) – SIM [techsun1]   (75)

00 P01 A CONT   04 P01 A CONT   10 P01 A CONT   14 P01 A CONT
01 P01 A CONT   05 P01 A CONT   11 P01 A CONT   15 P01 A CONT
02 P01 A CONT   06 P01 A CONT   12 P01 A CONT   16 P01 A CONT
03 P01 A CONT   07 P01 A CONT   13 P01 A CONT   17 P01 A CONT
```

Figure 5-9. Active CPUs in Continuous Mode

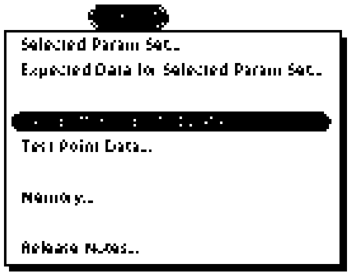
NOTE: You can halt the DM run with the commands in the `Halt` menu button. You can reset the time stamp feature with the command in the LM Reset ▽ menu button.

6. If **`Go --> One-shot`** is selected, the `ACTIVE` status will clear when the DM triggers.

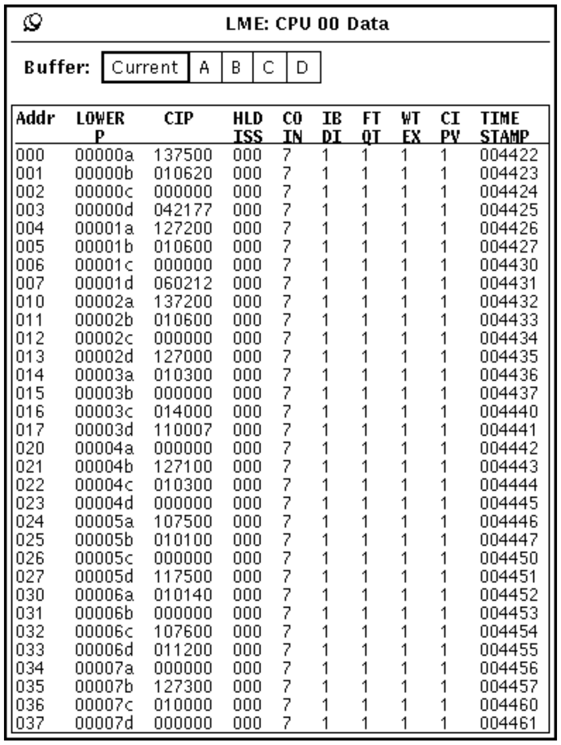
7. When the DM triggers or is halted, LME automatically reads the data back from the DM.

Viewing the DM Data

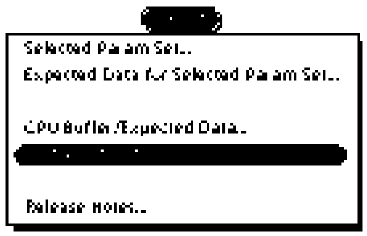
Several different displays are available for you to use to view the DM data and analyze the results. To see the different displays, perform the following actions:



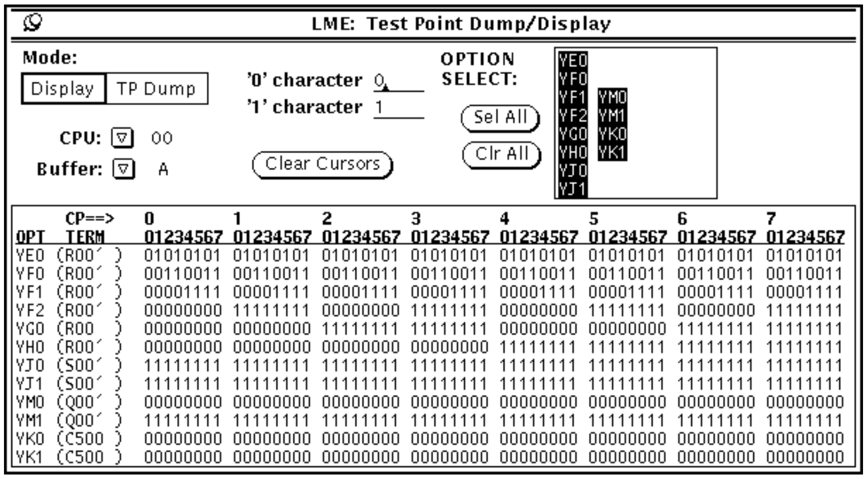
- Choose **`View --> CPU Buffer/Expected Data`**, as shown at the left, to view the DM data returned for any CPU:



Addr	LOWER P	CIP	HLD ISS	CO IN	IB DI	FT QT	WT EX	CI PV	TIME STAMP
000	00000a	137500	000	7	1	1	1	1	004422
001	00000b	010620	000	7	1	1	1	1	004423
002	00000c	000000	000	7	1	1	1	1	004424
003	00000d	042177	000	7	1	1	1	1	004425
004	00001a	127200	000	7	1	1	1	1	004426
005	00001b	010600	000	7	1	1	1	1	004427
006	00001c	000000	000	7	1	1	1	1	004430
007	00001d	060212	000	7	1	1	1	1	004431
010	00002a	137200	000	7	1	1	1	1	004432
011	00002b	010600	000	7	1	1	1	1	004433
012	00002c	000000	000	7	1	1	1	1	004434
013	00002d	127000	000	7	1	1	1	1	004435
014	00003a	010300	000	7	1	1	1	1	004436
015	00003b	000000	000	7	1	1	1	1	004437
016	00003c	014000	000	7	1	1	1	1	004440
017	00003d	110007	000	7	1	1	1	1	004441
020	00004a	000000	000	7	1	1	1	1	004442
021	00004b	127100	000	7	1	1	1	1	004443
022	00004c	010300	000	7	1	1	1	1	004444
023	00004d	000000	000	7	1	1	1	1	004445
024	00005a	107500	000	7	1	1	1	1	004446
025	00005b	010100	000	7	1	1	1	1	004447
026	00005c	000000	000	7	1	1	1	1	004450
027	00005d	117500	000	7	1	1	1	1	004451
030	00006a	010140	000	7	1	1	1	1	004452
031	00006b	000000	000	7	1	1	1	1	004453
032	00006c	107600	000	7	1	1	1	1	004454
033	00006d	011200	000	7	1	1	1	1	004455
034	00007a	000000	000	7	1	1	1	1	004456
035	00007b	127300	000	7	1	1	1	1	004457
036	00007c	010000	000	7	1	1	1	1	004460
037	00007d	000000	000	7	1	1	1	1	004461


Selected Param Set..
Expected Data for Selected Param Set..
CPU Buffer/Expected Data..
Release Notes..

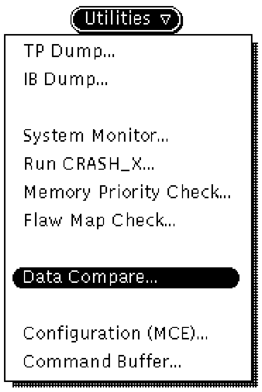
- Choose **`View --> Test Point Data`**, as shown at the left, to view information about any test points you selected to run:



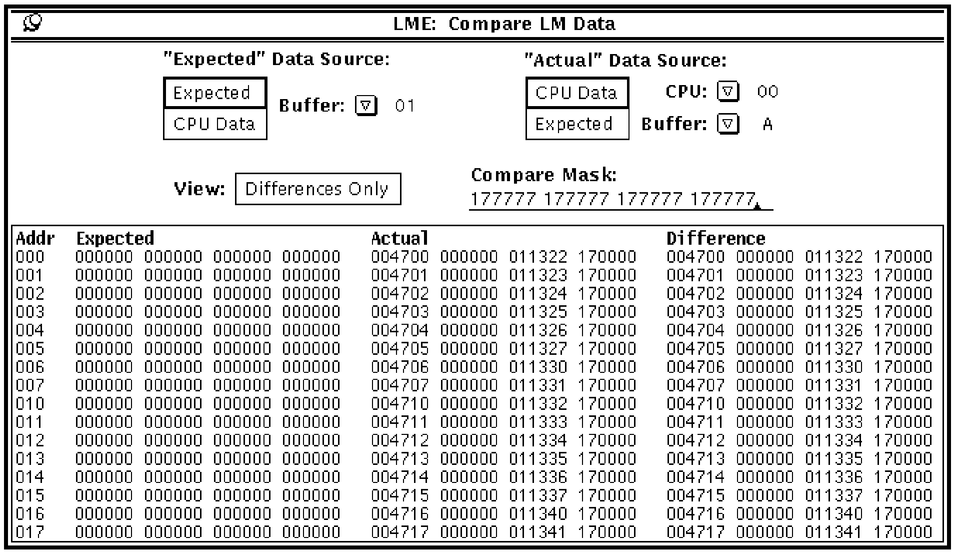
LME: Test Point Dump/Display

Mode: Display | TP Dump
'0' character 0
'1' character 1
CPU: 00
Buffer: A
OPTION SELECT: YE0 YF0 YF1 YF2 YG0 YH0 YJ0 YJ1 YM0 YM1 YK0 YK1

```
        CP==>   0        1        2        3        4        5        6        7
OPT   TERM      01234567 01234567 01234567 01234567 01234567 01234567 01234567 01234567
YE0  (R00´ )    01010101 01010101 01010101 01010101 01010101 01010101 01010101 01010101
YF0  (R00´ )    00110011 00110011 00110011 00110011 00110011 00110011 00110011 00110011
YF1  (R00´ )    00001111 00001111 00001111 00001111 00001111 00001111 00001111 00001111
YF2  (R00´ )    00000000 11111111 00000000 11111111 00000000 11111111 00000000 11111111
YG0  (R00  )    00000000 00000000 11111111 11111111 00000000 00000000 11111111 11111111
YH0  (R00´ )    00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111111
YJ0  (S00´ )    11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
YJ1  (S00´ )    11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
YM0  (Q00´ )    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
YM1  (Q00´ )    11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111
YK0  (C500 )    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
YK1  (C500 )    00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```



- Choose **`Utilities --> Data Compare`**, as shown at the left, to compare data in the CPU buffers and expected buffers:


LME: Compare LM Data

"Expected" Data Source: Expected | CPU Data   Buffer: 01
"Actual" Data Source: CPU Data | Expected   CPU: 00   Buffer: A
View: Differences Only
Compare Mask: 177777 177777 177777 177777

```
Addr  Expected                       Actual                         Difference
000   000000 000000 000000 000000    004700 000000 011322 170000    004700 000000 011322 170000
001   000000 000000 000000 000000    004701 000000 011323 170000    004701 000000 011323 170000
002   000000 000000 000000 000000    004702 000000 011324 170000    004702 000000 011324 170000
003   000000 000000 000000 000000    004703 000000 011325 170000    004703 000000 011325 170000
004   000000 000000 000000 000000    004704 000000 011326 170000    004704 000000 011326 170000
005   000000 000000 000000 000000    004705 000000 011327 170000    004705 000000 011327 170000
006   000000 000000 000000 000000    004706 000000 011330 170000    004706 000000 011330 170000
007   000000 000000 000000 000000    004707 000000 011331 170000    004707 000000 011331 170000
010   000000 000000 000000 000000    004710 000000 011332 170000    004710 000000 011332 170000
011   000000 000000 000000 000000    004711 000000 011333 170000    004711 000000 011333 170000
012   000000 000000 000000 000000    004712 000000 011334 170000    004712 000000 011334 170000
013   000000 000000 000000 000000    004713 000000 011335 170000    004713 000000 011335 170000
014   000000 000000 000000 000000    004714 000000 011336 170000    004714 000000 011336 170000
015   000000 000000 000000 000000    004715 000000 011337 170000    004715 000000 011337 170000
016   000000 000000 000000 000000    004716 000000 011340 170000    004716 000000 011340 170000
017   000000 000000 000000 000000    004717 000000 011341 170000    004717 000000 011341 170000
```

6 CRAY C90 CBP RUNTIME MODULE

The command buffer parser (CBP) application includes different runtime modules used to troubleshoot the different types of hardware CBP supports. This section describes the CRAY C90 CBP runtime module that enables automation of troubleshooting tasks performed by MME environment 1, MME environment 2, MCE, and LME for the CRAY C90 series mainframes.

This section describes the procedure to start CBP with the CRAY C90 runtime module, the command buffer programs that are available for use with the CRAY C90 runtime module, and the command buffer parser commands that are specific to the CRAY C90 runtime module.

For general information about the CBP application, including descriptions of the interface, CBP programming, and general-purpose commands, refer to the *Command Buffer Parser User Guide*, publication number HDM-076-0.

Starting the CRAY C90 CBP Runtime Module

The CRAY C90 CBP runtime module can be started from MME environment 1, MME environment 2, or LME. To start CBP with the CRAY C90 CBP runtime module, choose **`Utilities --> Command Buffer`** in the MME environment 1 base window, the MME environment 2 base window, or the LME base window.

The CRAY C90 CBP runtime module cannot be started from MME environment 0 because environment 0 is not supported by the CRAY C90 CBP runtime module.

CRAY C90 Command Buffer Programs and Files

When the CRAY C90 CBP runtime module is loaded with the CBP application, several command buffer programs specific to CRAY C90 mainframe troubleshooting can be accessed through the (Dir. ▽) menu button in the `CBP:  Load/Save` window. This menu structure is shown in Figure 6-1.

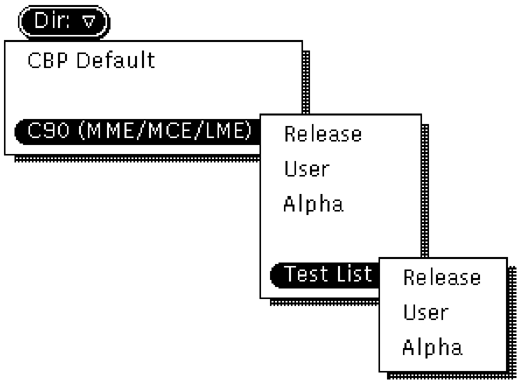


Figure 6-1. CBP Dir Menu Structure

The following options of the `Dir --> C90 (MME/MCE/LME)` menu provide access to files used with the CRAY C90 CBP runtime module:

Menu Option	Description
`Release`	CRAY C90 command buffer programs included in the current offline diagnostic release; refer to Table 6-1.
`User`	CRAY C90 command buffer programs that the user has modified/created and saved.
`Alpha`	CRAY C90 command buffer programs that have not been officially released.
`Test List --> Release`	CRAY C90 test lists that are included in the current offline diagnostic release.
`Test List --> User`	CRAY C90 test lists that the user has modified/created and saved.
`Test List --> Alpha`	CRAY C90 series test lists that have not been officially released.

Table 6-1. CRAY C90 Command Buffer Programs

Program	Description
`CONFIDENCE.cmd`	Performs comprehensive multi-CPU system test (uses the RUN system and runs with the IOS-E `hst.conf` command buffer)
`CPUQA.cmd`	Performs single-CPU (TV11) quality assurance (QA)
`CRASH_X.cmd`	Runs the `CRASH_X` utility. For more information about the `CRASH_X` utility, refer to "Running the CRASH_X Utility" on page 5-35.
`HMMQA.cmd`	Performs half-memory mode multi-CPU QA
`MCPU.cmd`	Performs multi-CPU confidence test
`MCPUQA.cmd`	Performs multi-CPU QA
`MCPUS.cmd`	Performs multi-CPU functionality test (This is a quick version of `MCPU.cmd`.)
`MILE.cmd`	Performs single-CPU confidence test
`PINT.cmd`	Performs programmable interrupt test (Run `PINT.cmd` with the IOS-E `_pint` command buffer.)
`ROA_CHECK.cmd`	Checks RAM on array
`SHRQA.cmd`	Performs shared reference QA

Several files stored in the `/cri/cme/c90/rel/cmd` directory are used to support the CRAY C90 CBP runtime module. These files are:

File	Description
`readme`	Description of the files in this directory
`std_def.h`	Standard variable definitions
`std.h`	Standard function definitions (Refer to Table 6-2 for descriptions of these functions.)

Table 6-2. Standard Function Descriptions

Function	Description
mme_clr( )	Multi-CPU clear routine
mme_clrssd( )	SSD/VHISP clear routine
mme_menu_enable( )	Enable interrupt on register parity error (IRP), interrupt on uncorrectable memory error (IUM), and interrupt on correctable memory error (ICM) routine
mmerun( )	Load and run a test in a CPU routine
mme_cycle_cpu( )	Cycle environment 1 tests through each available CPU routine

Table 6-2. Standard Function Descriptions (continued)

Function	Description
mme_runtest( )	Run a test to completion or error routine
mme_fail_test( )	Failure handler routine for command buffers
mme_mme_set( )	Set parameters in all loaded diagnostics routine
set_logger_init( )	Initialize set_logger( ) routine
set_logger( )	Enable or disable the error logger routine

Another CBP-related file is the `newcbp` file located in the `/cri/cme/c90/bin` directory. This script converts revision 1.0 through 3.0 CBP command buffers to the format used with CBP 4.0 and above.

CAUTION

Be sure to make a backup copy of a file before converting it. Failure to do so can result in loss of the file.

Use the following command to start this script:

`newcbp [–m | –f] <f`*`ilename1`*`> [<`*`filename2`*`> ...]`

The `-m` option indicates the input file is a mainline program (default); the `-f` option indicates the input file is a function.

Global variables are placed into a file named `x<`*`filename`*`>.h`; global `.h` files are included in the output file to eliminate undefined errors.

The `input` commands sent to this filter must have the following form:

```
<blank line>
input(EXCLUSIVE,“string”,1,
<blank line>
               “string”,
               “string”,
<blank line>
               variable)
<blank line>
```

All `goto` commands are left intact by the `newcbp` filter. You should replace `goto` commands with the more powerful commands available in the new version of CBP (`for`, `while`, `if`, `switch`, etc.).

CRAY C90 CBP Runtime Module Commands

The CRAY C90 CBP runtime module includes commands that are used only for the CRAY C90 series mainframes. These are active CBP commands that perform functions specific to the MME, LME, and MCE applications used to maintain and troubleshoot the CRAY C90 series mainframes.

You can use these commands in offline or concurrent mode. Use caution in concurrent mode because you could crash UNICOS.

MME-specific Command Descriptions

Use the following commands in your command buffer programs to manipulate the MME interface and to automate CRAY C90 mainframe testing:

MMEalloc( )

The `MMEalloc` command is used to set up MME memory allocation. This command uses the following forms:

`MMEalloc(CPUMODE,AUTO|MANUAL);`

This command enables or disables automatic CPU assignment when a control point is loaded.

`MMEalloc(IOICM,`*`value`*`|ALL,ENABLE|DISABLE);`

This command enables or disables interrupt on correctable memory error (ICM) for the CPU specified by *`value`* or for all CPUs; this sets or clears the ICM flag in the exchange package for the specified CPUs.

`MMEalloc(IOCPU,`*`value`*`);`

This command sets the I/O CPU (which CPU path is used to write to memory and read from memory) to *`value`*; *`value`* can be a constant or variable.

`MMEalloc(IOIRP,`*`value`*`|ALL,ENABLE|DISABLE);`

This command enables or disables interrupt on register parity error (IRP) for the CPU specified by *`value`* or for all CPUs; this sets or clears the IRP flag in the exchange package for the specified CPUs.

`MMEalloc(IOIUM,`*`value`*`|ALL,ENABLE|DISABLE);`

This command enables or disables interrupt on uncorrectable memory error (IUM) for the CPU specified by *`value`* or for all CPUs; this sets or clears the IUM flag in the exchange package for the specified CPUs.

`MMEalloc(MEMMODE,BOTTOM_UP|RANDOM|TOP_DOWN);`

This command sets the memory mode to bottom up, random, or top down.

`MMEalloc(MEMMODE,PARTITION_UP,SIZE|COUNT,`*`value`*`);`

This command sets the memory mode to bottom up with partitions that have the size specified by *`value`* or with *`value`* number of partitions, depending on whether size or count was specified.

`MMEalloc(MEMMODE,PARTITION_DOWN,SIZE|COUNT,`
`value``);`

This command sets the memory mode to top down with partitions that have the size specified by *`value`* or with *`value`* number of partitions, depending on whether size or count was specified.

`MMEalloc(PCI,`*`value`*`);`

This command sets the programmable-clock interrupt to the specified *`value`*.

`MMEalloc(SSD_MEMMODE,PARTITION_UP,SIZE|COUNT,`
`value``);`

This command sets the SSD memory mode to bottom up with partitions that have the size specified by *`value`* or with *`value`* number of partitions, depending on whether size or count was specified.

MMEassign( )

The `MMEassign` command assigns the control point referenced by *`tag`* (set by the `MMEload` command) to the specified CPU(s). This command uses the following form:

`MMEassign(`*`tag`*`,`*`value`*`|ALL);`

This command assigns the control point referenced by *`tag`* to the CPU specified by *`value`* or to all CPUs.

MMEautorestart( )

The `MMEautorestart` command enables or disables the auto restart function in environment 1. This command uses the following forms:

`MMEautorestart(DISABLE);`

This command disables the auto restart function in environment 1.

`MMEautorestart(ENABLE[,`*`interval`*`]);`

This command enables the auto restart function in environment 1. If you specify an *`interval`* (in milliseconds), the swap interval is set to the specified *`interval`*.

MMEcond( )

The `MMEcond` command sets the pass limit or time limit for each control point referenced by the specified *`tag`*. The `MMEcond` command must be used before the control points it references have been started with the `MMEgo` command. This command uses the following forms:

`MMEcond(`*`tag`*`|ALL,[`*`error_addr`*`|`*`pass_addr`*`],PASS|TIME,` *`value`*`);`

This command sets the pass limit or time limit for the control point referenced by *`tag`* or for all current control points to *`value`* and sets the address of the error count to *`error_addr`* and the address of the pass count to *`pass_addr`*. If *`error_addr`* and *`pass_addr`* are not specified, the address of the error count defaults to 0303, and the address of the pass count defaults to 0304.

MMEdeassign( )

The `MMEdeassign` command deassigns the control point referenced by *`tag`* from the specified CPU(s). This command uses the following forms:

`MMEdeassign(`*`value`*`);`

This command deassigns the CPU specified by *`value`* from whatever control point is assigned to it.

`MMEdeassign(ALL,tag);`

This command deassigns the control point referenced by *`tag`* from all CPUs it is assigned to.

MMEerror( )

The `MMEerror` command tests the control points referenced by a specified tag for an error (memory at error address was not zero). If an error is found, the command returns a logical true (nonzero) value. If no error is found, the command returns a logical false (zero) value. This command uses the following form:

`MMEerror(`*`tag`*`|ALL);`

> This command returns a nonzero (true) value if the control point referenced by *`tag`* (or any control point) has an error and a zero (false) value otherwise.

MMEgo( )

The `MMEgo` command starts a control point referenced by a specified tag. All active control points are monitored until their pass count addresses set by the `MMEcond` or `MMEgo` commands are reached or until a control point finds an error (error address is set by the `MMEgo` or `MMEcond` command). If an error is found, the error flag is set. This error flag can be examined with the `MMEerror` command. This command uses the following forms:

`MMEgo(`*`tag`*`|ALL);`

> This command starts the control point referenced by *`tag`* or all loaded control points. If no conditions have been previously set for the specified control points (by previous `MMEcond` or `MMEgo` commands), the pass and error counts default to –1 (enables the control point to run indefinitely), and the address of the error count defaults to 0303, and the address of the pass count defaults to 0304.

`MMEgo(`*`tag`*`|ALL,[`*`error_addr`*`|`*`pass_addr`*`],PASS|TIME);`

> This command performs the `MMEcond` command and then starts the control point referenced by *`tag`* or all loaded control points. (Refer to the description of the `MMEcond` command for more information.)

MMEhalt( )

The `MMEhalt` command stops a control point referenced by a specified tag. This command uses the following form:

`MMEhalt(`*`tag`*`|ALL);`

This command stops the control point referenced by *`tag`* or stops all control points.

MMEload( )

The `MMEload` command loads a control point into MME from the specified file. This command uses the following form:

`MMEload(“`*`string`*`”|ALL);`

This command loads the control point contained in the file specified by *`string`*. This command returns an integer value called a tag, which is a handle for the control point that other commands use to reference it.

For example, *`var`* `= MMEload(“aab.c”);` loads the `aab.c` control point and returns an integer value (a tag) to the variable *`var`*.

MMEloadat( )

The `MMEloadat` command loads a control point from the a specified file to a specified address. This command uses the following forms:

`MMEloadat(“`*`string`*`”,`*`iba`*`,`*`ila`*`);`

This command loads the control point into memory, where *`iba`* is the instruction base address and *`ila`* is the instructions limit address. This command returns an integer value called a tag, which is the handle for the loaded control point so other commands can reference it. This command returns a –1 value for the tag, indicating an error occurred, if there is not memory available to load the specified control point.

For example, *`var`* `= loadat(“aab.c”,10000,300000);` loads the control point `aab.c` into memory between the specified addresses and returns an integer value (a tag) to the variable *`var`*.

`MMEloadat("`*`string`*`",`*`iba,ila,dba,dla`*`);`

NOTE: Use this form of `MMEloadat` for segmented diagnostics only.

This command loads the control point into memory, where *`iba`* is the instruction base address, *`ila`* is the instruction limit address, *`dba`* is the data base address, and *`dla`* is the data limit address. This command returns an integer value called a tag, which is the handle for the loaded control point so other commands can reference it. This command returns a –1 value for the tag, indicating an error occurred, if there is not memory available to load the specified control point.

MMEread( )

The `MMEread` command reads a block of mainframe memory. The block may be one or more words in length and may begin at an absolute address or at an address relative to the starting address of a specified control point. This command uses the following form:

`MMEread([`*`tag`*`,] `*`buffer`*`,`*`address`*`,`*`length`*`);`

This command reads a block of data from mainframe memory. The block begins at the word address specified by *`address`* and ends with the address specified by *`length`*. If *`tag`* is specified, addresses are relative to the beginning of the control point specified by *`tag`*, otherwise the addresses are absolute. The data is copied into *`buffer`*, which must be an array of byte, short, uint, or long data and must be large enough to store the amount of data requested.

MMEreload( )

The `MMEreload` command reloads a control point. This command uses the following form:

`MMEreload(`*`tag`*`|ALL);`

This command reloads the control point referenced by *`tag`* or reloads all control points.

MMEreset( )

The `MMEreset` command resets the MME environment. This command uses the following form:

```
MMEreset([1|2]);
```

This command resets MME to the current environment, or to the environment specified.

NOTE: Environment 0 is not supported by CBP.

MMErunsys( )

The `MMErunsys` command enables or disables the run system. This command uses the following forms:

```
MMErunsys(DISABLE);
```

This command disables the run system.

`MMErunsys(ENABLE[,`*`value`*`]);`

This command enables the run system and specifies *`value`* as the swap interval.

MMEtimeout( )

The `MMEtimeout` command tests control points referenced by a specified tag for a timeout (the wait limit is reached before the pass count or time limit was reached). If a timeout is found, the command returns a logical true (nonzero) value. If no timeout is found, the command returns a logical false (zero) value. This command uses the following form:

`MMEtimeout(`*`tag`*`, `*`label`*`);`

This command returns a nonzero (true) value if the control point referenced by tag (or any control point) timed out and a zero (false) value otherwise.

MMEunload( )

The `unload` command unloads control points from memory. This command uses the following form:

`MMEunload(`*`tag`*`|ALL);`

> This command unloads the control point referenced by *`tag`* or all control points.

MMEwait( )

The `MMEwait` command waits for a specified amount of time for all started control points to reach their pass or time limits or to find an error. If any control points are active when this time limit is reached, all active control points are stopped and the timeout flag is set. The timeout flag can be examined with the `MMEtimeout` command. This command uses the following form:

`MMEwait(`*`limit`*`);`

> This command waits for *`limit`* seconds and returns logical false (zero) if any control points reach their time limits or find an error and logical true (nonzero) if the control points reach their pass limits or the user clicks on Continue in the CBP base window to stop the `MMEwait` command

MMEwrite( )

The `MMEwrite` command writes a block of data to mainframe memory. The block can be one or more words in length and can be written to an absolute address or to an address relative to the starting address of a specified control point. This command uses the following form:

`MMEwrite([`*`tag`*`,] `*`buffer`*`,`*`address`*`,`*`length`*`);`

> This command writes a block of data to mainframe memory. The data is read from *`buffer`*, which must be an array of byte, short, uint, or long data and must contain at least as much data as was requested. The data is written to the word address specified by *`address`* and ends with the address specified by *`length`*. If *`tag`* is specified, addresses are relative to the beginning of the control point specified by *`tag`*, otherwise the addresses are absolute.

MCE-specific Command Descriptions

Use the following commands in your command buffer programs to manipulate the MCE interface and to automate configuring a CRAY C90 series mainframe.

MCEconfig( )

The `MCEconfig` command is used to set up MCE mainframe configuration. This command uses the following forms:

```
MCEconfig(CTRL_CABLE_0|CTRL_CABLE_1,ENABLE|
DISABLE);
```

This command enables or disables the specified control cable (0 or 1).

```
MCEconfig(IOECC,value|ALL,ENABLE|DISABLE);
```

This command enables or disables I/O ECC generation for the CPU specified by *`value`* or for all CPUs.

```
MCEconfig(MAINTMODE,value|ALL,ENABLE|DISABLE);
```

This command enables or disables maintenance mode for the CPU specified by *`value`* or for all CPUs.

```
MCEconfig(MASTER_CPU,value);
```

This command sets the master CPU in the mainframe to the CPU specified by *`value`*.

```
MCEconfig(MEMMODE,value|ALL,HALF_UPPER|
HALF_LOWER|FULL);
```

This command sets the memory mode for the CPU specified by *`value`* or for all CPUs to upper half, lower half, or full memory.

MCEgetconfig( )

The `MCEgetconfig` command returns a numeric value corresponding to the requested machine configuration attribute for the current configuration. This command uses the following forms:

`MCEgetconfig(BMM_ENABLED_CPUS_MASK);`

This command returns a mask of the CPUs that have bit matrix multiple (BMM) present and enabled. For example, `var = MCEgetconfig(BMM_ENABLED_CPUS_MASK);` returns the value to the variable *`var`*.

`MCEgetconfig(BMM_PRESENT_CPUS_MASK);`

This command returns a mask of the CPUs that have BMM present. For example, `var = MCEgetconfig (BMM_PRESENT_CPUS_MASK);` returns the value to the variable *`var`*.

`MCEgetconfig(IDLE_CPUS_MASK);`

This command returns a mask of the CPUs that are present and idle (1 = CPU is present and idle).

`MCEgetconfig(LOSP_IOS_MASK);`

This command returns a mask of the LOSP channels cabled to the IOS-E. Add this mask to 040_8 to get a value corresponding to the LOSP channel numbers. For example, `var = MCEgetconfig(LOSP_IOS_MASK);` returns the value to the variable *`var`*.

`MCEgetconfig(LOSP_LOOP_MASK);`

This command returns a mask of the LOSP channels cabled for loopback. Add this mask to 040_8 to get a value corresponding to the LOSP channel numbers. For example, `var = MCEgetconfig(LOSP_LOOP_MASK);` returns the value to the variable *`var`*.

`MCEgetconfig(MAINTMODE_CPUS_MASK);`

This command returns a mask of the CPUs that have maintenance mode enabled. For example, `var = MCEgetconfig (MAINTMODE_CPUS_MASK);` returns the value to the variable *`var`*.

`MCEgetconfig(MEMMODE);`

This command returns a value representing the memory mode of the current configuration (0 = half upper, 1 = half lower, and 2 = full memory mode). For example, `*var* = MCEgetconfig (MEMMODE);` returns the value to the variable *`var`*.

`MCEgetconfig(NUM_CPUS);`

This command returns the maximum number of CPUs for the current configuration. For example, `*var* = MCEgetconfig (NUM_CPUS);` returns the value to the variable *`var`*.

`MCEgetconfig(NUM_USABLE_CPUS);`

This command returns the number of usable CPUs. For example, `*var* = MCEgetconfig(NUM_USABLE_CPUS);` returns the value to the variable *`var`*.

`MCEgetconfig(PRESENT_CPUS_MASK);`

This command returns a mask of the CPUs present. For example, `*var* = MCEgetconfig(PRESENT_CPUS_MASK);` returns the value to the variable *`var`*.

`MCEgetconfig(SSD_LIMIT);`

This command returns the size of the SSD (the last available address in SSD memory + 1). For example, `*var* = MCEgetconfig(SSD_LIMIT);` returns the value to the variable *`var`*.

`MCEgetconfig(SYSTEM_TYPE);`

This command returns one of the following symbols: MF4000, MF4200, MF4300, MF4400, MF4600, MF4700, MF4800, or MF4900 for the mainframes; TV11, TV12, TV22, or TV12DRAM for the testers; SIMULATOR; or UNKNOWN. The symbol indicates the system type that is currently configured. For example, `*var* = MCEgetconfig (SYSTEM_TYPE);` returns the symbol to the variable *`var`*.

`MCEgetconfig(USABLE_CPUS_MASK);`

This command returns a mask of the usable CPUs. For example, `*var* = MCEgetconfig(USABLE_CPUS_MASK);` returns the value to the variable *`var`*.

```
MCEgetconfig(VHISP_SSD_MASK);
```

This command returns a mask of the VHISP channels cabled to the SSD. The bit numbers correspond to the VHISP channels. For example, `var = MCEgetconfig(VHISP_SSD_MASK);`

MCEreset( )

The `MCEreset` command resets MCE, reinitializes the hardware, and applies the configuration.

MCEstatus( )

The `MCEstatus` command reads the system status or the status of an individual CPU. This command uses the following form:

```
MCEstatus(SYSTEM|cpu,buffer);
```

This command returns four parcels of status data for the system or the specified *`cpu`* into the variable *`buffer`*.

LME-specific Command Descriptions

Use the following commands in your command buffer programs to manipulate the CRAY C90 LME interface and to automate use of the diagnostic monitors (DMs).

LMEactive( )

The `LMEactive` command tests the diagnostic monitors of the CPU(s) assigned to a specified parameter set (*`pset`*) for being active (still recording) or inactive (triggered or stopped). If the specified CPU's DM is active, the command returns a logical true (nonzero) value; otherwise, a logical false (zero) is returned. This command uses the following form:

```
LMEactive(cpu|ALL);
```

This command returns a nonzero (true) value if the DM of the specified *`cpu`* is active and returns a zero (false) otherwise.

LMEassign( )

The `LMEassign` command assigns the specified *`pset`* to the specified CPU(s). This command uses the following forms:

`LMEassign(`*`pset`*`,`*`value`*`|ALL);`

> This command assigns the *`pset`* to the CPU specified by *`value`* or to all CPUs. For example, `LMEassign(P00,ALL);` assigns parameter set P00 to all CPUs.

`LMEassign(`*`pset`*`,MASK,`*`cpu_mask`*`);`

> This command assigns the *`pset`* to multiple CPUs specified in the *`cpu_mask`* bit mask (bit 2^0 of *`cpu_mask`* corresponds to CPU 0, bit 2^1 corresponds to CPU 1, etc.). For example, `LMEassign(P00,MASK,100001);` assigns parameter set P00 to CPU 0 and CPU 17.

LMEcpudata( )

The `LMEcpudata` command copies the data from the specified CPU DM data buffer to a specified user buffer. This command uses the following form:

`LMEcpudata(`*`value`*`,A|B|C|D|CURRENT,`*`buffer`*`);`

> This command copies the data from a data buffer of the CPU specified by *`value`* to the user buffer. The data is copied from one of the CPU's four buffers (A, B, C, D, or the current target data buffer assigned to the corresponding *`pset`*). The *`buffer`* must be an array of byte, short, uint, or long data and must be able to hold 0100_8 words of DM data.

`LMEcpudata(`*`value`*`,TPDUMP,`*`buffer`*`);`

> This command copies test point dump data for the CPU specified by *`value`* to a specified user buffer. The *`buffer`* must be an array of byte, short, uint, or long data and must be able to hold 040_8 words of test point dump data.

`LMEcpudata(`*`value`*`,IBDUMP,`*`ib#`*`|ALL,`*`buffer`*`);`

This command copies data from an instruction dump buffer of the CPU specified by *`value`* to the user buffer. If an instruction buffer number (*`ib#`*) 0 through 7 is specified, data from a specific buffer is copied to the user buffer (buffer must be at least 040_8 words long). If ALL is specified, all 8 instruction buffers are dumped to the user buffer (buffer must be at least 0400_8 words long).

LMEdeassign( )

The `LMEdeassign` command deassigns the *`pset`* from the specified CPU(s). This command uses the following forms:

`LMEdeassign(`*`value`*`);`

This command deassigns the CPU specified by *`value`* from the *`pset`* assigned to it.

`LMEdeassign(ALL,`*`pset`*`);`

This command deassigns the *`pset`* from all CPUs it is assigned to.

LMEgo( )

The `LMEgo` command starts the diagnostic monitors on all CPUs assigned to a specified *`pset`*. All active CPUs are monitored until their DM active flags are cleared (when the DMs trigger or are halted using the `LMEhalt` command). The `LMEactive` command can be used to test whether a specified CPU's DM is active. This command uses the following form:

`LMEgo(`*`pset`*`|ALL);`

This command starts the DMs of all CPUs that are assigned to the specified *`pset`* or starts the DMs of all assigned CPUs.

LMEhalt( )

The `LMEhalt` command stops the diagnostic monitors on all CPUs assigned to a specified *`pset`*. This command uses the following form:

`LMEhalt(`*`pset`*`|ALL);`

This command stops the DMs of all CPUs that are assigned to the specified *`pset`* or stops the DMs of all assigned CPUs.

LMEibdump( )

The `LMEibdump` command dumps instruction buffers. This command uses the following form:

`LMEibdump(`*`cpu`*`);`

This command dumps instruction buffers for the specified CPU. Use the `LMEcpudata` command to retrieve the data.

LMEmaint( )

The `LMEmaint` command enables specialized maintenance channel function sequences to be performed. This command uses the following form:

`LMEmaint(`*`func`*`,`*`destination`*`);`

This command enables specialized maintenance channel function sequences to be performed. Currently, the only valid *`func`* is IOMC, which causes I/O master clear to be set and then cleared. *`Destination`* is either SYSTEM, ALL (broadcast to all CPUs), or a CPU number.

LMEparams( )

The `LMEparams` command reads and sets DM parameters in the specified *`pset`*. Individual parcels in the *`pset`* can be read and set, and the entire 12-parcel DM parameter list in the *`pset`* can be read or set in one operation. This command uses the following forms:

`LMEparams(`*`pset`*`,`*`parcel_number`*`[,`*`new_value`*`]);`

This command returns the value of the parcel specified by *`parcel_number`* of the specified *`pset`*; if *`new_value`* is specified, the value of the parcel is set to that value.

`LMEparams(`*`pset`*`,GETBUF|SETBUF,`*`buffer`*`);`

If `GETBUF` is specified, this command copies the entire 12-parcel DM parameter list from the *`pset`* to the user buffer specified by *`buffer`*. If `SETBUF` is specified, this command copies the parameter list from the user buffer to the *`pset`*. In either case, *`buffer`* must be an array of byte, short, uint, or long data and must be at least 12 parcels.

LMEread( )

The `LMEread` command reads a block of mainframe memory. The block may be 1 or more words in length. This command uses the following form:

```
LMEread(buffer,address, length,cpu);
```

This command reads a block of data from mainframe memory. The block begins at the absolute word address specified by *`address`*. The number of words written is specified by *`length`*. The data is copied into *`buffer`*, which must be an array of byte, short, uint, or long data and must be large enough to store the amount of data requested. The specified *`cpu`* performs the I/O operations.

LMEreset( )

The `LMEreset` command resets the LME environment and server.

LMEsetbuf( )

The `LMEsetbuf` command sets the target data buffer for the specified *`pset`*. The target data buffer is the buffer (A, B, C, or D) that the DM data is recorded in for each CPU assigned to the *`pset`*. This command uses the following form:

```
LMEsetbuf(pset,A|B|C|D);
```

This command sets the target data buffer for the diagnostic monitor specified by *`pset`*.

LMEtpdump( )

The `LMEtpdump` command dumps test points. This command uses the following form:

```
LMEtpdump(cpu);
```

This command dumps test points for the specified CPU. Use the `LMEcpudata( )` command to retrieve the data.

LMEwait( )

The `LMEwait` command waits for a specified amount of time for active DMs on all CPUs to trigger and stop recordings. The command can be set up to return when either the first active CPU triggers or when all active CPUs trigger. This command uses the following form:

```
LMEwait(limit, FIRST | ALL);
```

This command waits for the number of seconds specified by *`limit`* and returns logical false (zero) if the time-out value is reached before the wait condition is met. If FIRST is specified, the command waits for the first CPU's DM to trigger. If ALL is specified, the command waits for the DMs on all CPUs to trigger.

LMEwrite( )

The `LMEwrite` command writes a block of data to mainframe memory. The block can be 1 or more words in length and can be written to an absolute address or to an address relative to the starting address of a specified control point. This command uses the following form:

```
LMEwrite(buffer,address,length,cpu);
```

This command writes a block of data to mainframe memory. The data is read from *`buffer`*, which must be an array of byte, short, uint, or long data and must contain at least as much data as was requested. The data is written to the absolute word address specified by `address` . The number of words written is specified by *`length`*. The specified *`cpu`* performs the I/O operations.

7 DIAGNOSTIC TESTS AND UTILITIES

This section describes the diagnostic tests and utilities you can use with MME. It includes descriptions of environment 0 tests, customizing control points, environment 1 and 2 control points, and testing strategies.

Environment 0 Tests

Environment 0 tests test the mainframe from the MWS-E through a series of maintenance channel functions. The six environment 0 tests are described below.

Diagnostic Monitor Test

The diagnostic monitor test checks the data path from the MWS-E to the clock module through the DC option and the HF/HR10 option (diagnostic monitor) of the selected CPUs and back again. The diagnostic monitor test generates sequences that test the following areas of the diagnostic monitor hardware: diagnostic monitor loopback with 0’s and 1’s, odd and even, address, random, and user-defined patterns; event recording; and triggering. Automatic mode tests all areas except loopback with a user-defined pattern. Manual mode enables selection of the testing area.

Exchange Test

The exchange test checks the HA, HB, HC, JA, JB, and JC options of the exchange logic. The exchange test generates sequences that test the exchange hardware paths with the following data patterns: 0’s, 1’s, address, random, and user-defined patterns. Automatic mode uses all the patterns except the user-defined pattern. Manual mode enables selection of the pattern or compare mask.

Instruction Buffer Test

The instruction buffer test checks the data path from the MWS-E through the maintenance channel to the clock module, selected CPUs, memory, exchange logic, instruction buffers, DC logic, and diagnostic monitor and back again. The instruction buffer test generates sequences that test the instruction buffers with the following patterns: 0’s, 1’s, address, random, and user-defined patterns. Automatic mode uses all patterns except a user-defined pattern. The CPU being tested is the same CPU whose path to memory was used to write the initial exchange package. Manual mode enables selection of the patterns, compare mask, and write path to memory.

Maintenance Channel Test

The maintenance channel test checks the data path from the MWS-E to the clock module, to the selected CPU’s DC option, and back again. The maintenance channel test generates sequences that test the following areas of the maintenance channel hardware: maintenance channel loopback with 1’s and 0’s, odd and even, random, and user-defined patterns; CPU master clear (set); CPU master clear (clear) and exchange; half-memory mode; 256K mode; memory read and write (address 0_8 to 020000_8), CA and CL; and master CPU selection. Automatic mode tests all areas except loopback with a user-defined pattern. Manual mode enables selection of the testing area.

Memory Data Pattern Test

The memory data pattern test checks the data path from the MWS-E to the clock module through the DC option of the selected CPUs to memory and back again. The memory data pattern test generates sequences that test memory with the following data patterns: 0’s, 1’s, odd bits, even bits, address, complement address, sliding 0’s, sliding 1’s, random, and user-defined patterns. Automatic mode tests the first 020000_8 words of memory with all the patterns except the user-defined pattern. Memory error correction is disabled, and the same CPU’s path is used to write memory that is used to read memory. Manual mode enables selection of the pattern, address range, I/O error correction, compare mask, and write CPU path to memory.

Miscellaneous Test

The miscellaneous test checks memory addressing by a CPU for all available memory; tests spare chip functionality; tests several test points (located on the HA0, HC0, JA0, JB0, JC0, JQ0, JQ1, VQ0, VQ1, YE0, YF0, YF1, YF2, YG0, YH0, YJ, YK0, and YK1 options); verifies that the correct logical processor number is created and stored in the exchange package for a given CPU; and verifies that a CPU can set the interprocessor interrupt on other CPUs. The miscellaneous test is structured as follows:

- An address-bit test sequence performs a series of 1-word reads and writes. One 1-word read and one 1-word write is performed per address bit.
- A spare-chip test sequence simultaneously checks the spare chip selections. A data pattern is written into the default selected data chips, each bit group is selected to be spared, and the data is checked to verify that the data pattern changed appropriately.
- The test point test sequences check the specified options' test points by verifying that the diagnostic monitor (DM) can trigger off the test point or by recording the test point and verifying the test point for a specific number of clock periods.
- A logical processor number (LPN) test sequence performs an exclusive OR (XOR) of a CPU's physical processor number and the master CPU's physical processor number. One or more CPUs can be selected for this test sequence. When a CPU is under test, its LPN is tested with each selected CPU as the master.
- A set interprocessor interrupt (SIPI) test sequence sets the CPU being tested to the master CPU and deadstarts code that causes the CPU being tested to set an interprocessor interrupt for the other CPU(s). The first status register (SR0) for each of these CPUs is written to a specific memory location. The contents of the memory location(s) are verified to ensure that the correct CPU(s) received the SIPI command and executed the correct code. At least two CPUs must be selected for the SIPI test sequence.

Automatic mode performs the address-bit test sequence with all patterns except a user-defined pattern and performs the spare-chip test sequence. For the address-bit test sequence, the selected CPU uses the same path that is used to write data to memory and read data from memory. Manual mode enables selection of the address-bit sequence, spare-chip sequence, logical processor number sequence, set interprocessor interrupt request sequence, test point sequences, last address, compare mask, and CPU write path to memory and read path from memory.

Customizing Control Points

When you select a diagnostic program or utility in environment 1 or 2 and load it into memory, it becomes known as a control point. By default, when you select a diagnostic program and load it as a control point, all the sections run. Using the `Standard Locations` window, you can customize the control point such as specifying that fewer sections run. If you specify that fewer sections run, then fewer functions are tested. However, if you have nearly isolated the error and need to run only some of the sections, you can complete the testing more quickly.

MME Memory Layout

Table 7-1 explains the MME memory layout in environment 1 or 2. Major headings are shown in **bold**.

Table 7-1. MME Memory Layout

Address	Label	Explanation
0000 – 0017	**DEXP**	**CPU 0 initial deadstart exchange package area**
0000	P-Reg	P-register value
0001	IBA	Instruction base address
0002	ILA	Instruction limit address
0003	DBA	Data base address
0004	DLA	Data limit address
0005		Interrupt modes, status, and modes
0006		Interrupt flags
0007		Processor number (PN), cluster number (CLN), exchange address, and vector length register (VL)
0020 – 0037	**SEXP**	**Starting exchange package**
0020	P-Reg	P-register value
0021	IBA	Instruction base address
0022	ILA	Instruction limit address
0023	DBA	Data base address
0024	DLA	Data limit address
0025		Interrupt modes, status, and modes
0026		Interrupt flags
0027		Processor number (PN), cluster number (CLN), exchange address, and vector length register (VL)
0040 – 0177	**IEXP**	**Interrupt handler exchange package**

Table 7-1. MME Memory Layout (continued)

Address	Label	Explanation
0040	P-Reg	P-register value
0041	IBA	Instruction base address
0042	ILA	Instruction limit address
0043	DBA	Data base address
0044	DLA	Data limit address
0045		Interrupt modes, status, and modes
0046		Interrupt flags
0047		Processor number (PN), cluster number (CLN), exchange address, and vector length register (VL)
0200 – 0377	**STDLOC**	**Standard locations**
0200	LPASS	Last pass to be executed (0 = forever)
0201	SECS	Section selection bit mask
0202	CONDS	Conditions select bit mask
0204	STOP	Stop flag bit mask, where: 00 = Continue (update CPU information and continue processing) 01 = Stop (update CPU information and stop processing) 02 = Loop (loop when an error occurs) 04 = Log (write data to the error information block) 10 = Isolate (restart and isolate the error)
0205	MRSTOP	Memory and register error bit mask (stop and log): 000001 = Log correctable memory errors 000002 = Log uncorrectable memory errors 000004 = Log register parity errors 000010 = Stop on a correctable memory error 000020 = Stop on an uncorrectable memory error 000040 = Stop on a register parity error 200000 = Disable error correction
0206	PCITIME	Programmable-clock interrupt time interval
0207	PCILOG	Programmable-clock interrupt counter
0210	CPUN	Number of CPUs
0211	CPUM	Master CPU number
0212	CPUS	Bit mask of CPUs to be tested
0214	CLNN	Number of clusters
0215	CLNU	Cluster number used or cluster under test
0216	CLNS	Bit mask of the clusters to be tested
0217	CLNB	Monitor background cluster number: 0 = Do not use semaphores
0220	DIBA	Diagnostic program instruction base address

Table 7-1. MME Memory Layout (continued)

Address	Label	Explanation
0221	DILA	Diagnostic program instruction limit address
0222	DDBA	Diagnostic program data base address
0223	DDLA	Diagnostic program data limit address
0224	MFIRST	First memory word to be tested (BSS)
0225	MLAST	Last memory address (like the data limit address)
0226	BANKS	Number of bank bits and number of memory banks
0227	MCFG	Memory configuration, memory modes, and number of memory sections
0230	SSDBA	SSD base address
0231	SSDL	SSD limit address
0232	CRMASK	Channel reserve mask, where the bit equals the channel number
0233	CIMASK	Channel interrupt mask, where the bit equals the channel number
0234	DIFM	Diagnostic program interrupt handled flag mask
0235	SIFM	System interrupt flag mask
0236	SIFR	System interrupt flag return
0240	DMPMASK	Dump register for hIDLE: 00001 = V registers 00002 = B registers 00004 = T registers 00010 = BMM registers 00020 = Shared B registers 00040 = Shared T registers 00100 = Semaphore registers 00200 = A registers (WEXP) 00400 = S registers (WEXP) 01000 = Status registers 02000 = VM registers 04000 = VL register 10000 = Channel CA and status register
0241	dmpAREA	Dump area
0242	dmpJUMP	Dump and idle routine address
0245	LASTREQ	Copy of the last diagnostic controller request
0246	LASTRET	Copy of controller return status
0247	EIBPTR	EIB pointer to next free entry
0250 – 0257		MME request port
0300 – 0307	DIAGINFO	Diagnostic information
0300	DIF	Difference between expected and actual diagnostic information
0301	ACT	Actual diagnostic program information
0302	EXP	Expected diagnostic program information

Table 7-1. MME Memory Layout (continued)

Address	Label	Explanation
0303	ERROR	Number of errors
0304	PASS	Number of passes
0305	ERA	Error return address
0306	INFOa	Diagnostic program specific information A
0307	INFOb	Diagnostic program specific information B
0310	SUT	Section under test
0311	CUT	Condition under test
0340–0377	HARDWARE	Asymmetric hardware information
0400 – 0777	**CPUINFO**	**CPU information for multi-CPU diagnostic programs**
0400	DIF0	Difference between expected and actual control point diagnostic information
0401	ACT0	Actual control point diagnostic information
0402	EXP0	Expected control point diagnostic information
0403	ERROR0	Number of errors
0404	PASS0	Number of passes
0405	ERA0	Error return address
0406	INFOa0	Control point specific information A
0407	INFOb0	Control point specific information B
1000 – 1577	**PARAM**	**Parameter block** (diagnostic-specific parameters)
1600 – 1777	**ELOG**	**Error log**
1600–1677	pARELOG	Parity error log
1700–1777	mEMELOG	Memory error log
2000 – 2377	**WEXP**	**Working exchange packages for all CPUs**
2400 – 2777	**CEXP**	**Current exchange packages for all CPUs**
3000 – 3377	**TEXP**	**Trap exchange packages for all CPUs**
3000	P-Reg	P-register value
3001	IBA	Instruction base address
3002	ILA	Instruction limit address
3003	DBA	Data base address
3004	DLA	Data limit address
3005		Interrupt modes, status, and modes
3006		Interrupt flags
3007		Processor number (PN), cluster number (CLN), exchange address, and vector length register (VL)
3010		Boundary symbol address

Table 7-1. MME Memory Layout (continued)

Address	Label	Explanation
4000 – 5000+		**Code block (text area)**
4000–4777	STDCODE	Control point standard code that includes the interrupt router and normal exit router
6000+	MAIN	Control point program code
†		**Data area**
	dmpAREA	Dump area for CPU registers
	dmpVEC	Vector register
	dmpB	B register
	dmpT	T register
	dmpBMM	Bit matrix multiply (BMM)
	dmpSHR	Shared B, T, and semaphore registers on 40_8-word boundaries
	dmpEXP	Exchange package
	dmpSTAT	Status register
	dmpVM	Vector mask
	dmpVL	Vector length
	IDATA	Initialized data and may have optional error information block (EIBK) data
	BSS	**Block storage segment (BSS)**
	UDATA	Uninitialized data

† All locations after the control point code are dependent on the size of the control point code. Refer to a control point's listing for specific addresses.

Environment 1 and 2 Control Points

Environments 1 and 2 move the testing into the mainframe. The tests and utilities that run in the mainframe are called control points. The following subsections describe the control point tests and utilities you can run under MME environment 1 or 2.

Control Point Tests

Several control point diagnostic tests are available in environments 1 and 2. Table 7-2 lists the diagnostic test sections and describes the functions each section tests. To view the complete listing for any of these tests, perform the following steps:

1. Choose **`View --> Listing --> Other`** in the `Mainframe Maintenance Environment` base window; the `MME View Listing Setup` window appears.

2. Double click on the test you want to view, or click on the test and click on View... . The listing window appears.

Table 7-2. Environment 1 and 2 Control Point Tests

		Sections	
Test † ‡	Description	Section Selection	Function
`aab.c`	Address register basic test	Condition 1	Tests 022*ijk* instructions
		Condition 2	Tests 020*ijk* instructions
		Condition 3	Tests 021*ijk* instructions
		Condition 4	Tests 030*ijk* instructions (A*i* = A*j* + A*k)*
		Condition 5	Tests 030*ijk* instructions (path test)
		Conditions 6 – 10	Tests special case instructions (030*i*0*k*, 030*ij*0, 031*i*00, 031*i*0*k)*
		Condition 11	Tests 020 and 021 instructions (walks a 1 through the unused *jk* fields)
`aht.c`	A*h* addressing test	0	Uses a different A register as A*h* to read a 100_8 word buffer; all other A registers are set to 0

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

		Sections	
Test [†] [‡]	Description	Section Selection	Function
`aht.c` (cont.)	A*h* addressing test	1	Uses a different A register as A*h* to read a 100_8 word buffer; all other A registers are set to –1
		2	Uses a different A register as A*h* to write a 100_8 word buffer; all other A registers are set to 0
		3	Uses a different A register as A*h* to write a 100_8 word buffer; all other A registers are set to –1
		4	Uses the A1 register as A*h* to write all memory with an address pattern; all other A registers are 0 Uses different A registers as A*h* to read all of memory
		5	Uses the A1 register as A*h* to write all memory with a reverse address pattern; all other A registers are –1 Uses different A registers as A*h* to read all of memory
		6	Reads and writes a fixed address while incrementing all A registers
		7	Writes an address pattern to all of memory; all enabled CPUs read memory using the A1 register as A*h* (will not run unless CPU 0 enabled in location CPUS)
		10	Reads memory with various data base addresses starting with 10000_8 (automatically disabled if MS not used)
		11	Reads memory with A*h*+*nm* fields
`amb.c` `amb.y`	Address multiply basic test	0	Tests k = 1 and j = sliding 1’s
		1	Tests j = 1 and k = sliding 1’s

[†] A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

[‡] indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

		Sections	
Test † ‡	Description	Section Selection	Function

`amb.c`
`amb.y`
(cont.)
	Address multiply basic test	2	Tests predetermined operands for enables and satisfies

		3	Tests predetermined operands for carries
		4	Tests random increasing operands
		5	Tests paths
`ars.c`	Address (A) and scalar (S) register add and multiply test	0	Tests A register subtraction (all sections use toggle numbers of a 1, a 3, a 5, and a 4-bit random number in sequence)
		1	Tests A register addition
		2	Tests A register multiplication
		3	Tests S register subtraction
		4	Tests S register addition
		5	Tests S register multiplication
`ave.c`	Vector register test	0	Tests vector register 0
		1	Tests vector register 1
		2	Tests vector register 2
		3	Tests vector register 3
		4	Tests vector register 4
		5	Tests vector register 5
		6	Tests vector register 6
		7	Tests vector register 7
`bmm.c`	Bit matrix multiply (BMM) functional unit test	0	Tests basic scalar BMM data
		1	Tests basic vector BMM data
		2	Tests single matrix of random data
		3	Tests dual matrices of random data
		4	Tests vector population/parity data

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

Test † ‡	Description	Sections	
		Section Selection	Function
`bmm.c` (cont.)	Bit matrix multiply functional unit test	5	Tests instruction timing
		6	Tests chaining
		7	Tests the B matrix loaded status bit
`bp.c`	Breakpoint interrupt test	0	Tests scalar write references
		1	Tests vector write references
`brt.c`	Block transfer register test (Sections 0, 1, and 2 do not use B00 for return jumps; can detect erratic B00 operation)	0	Tests B register basic and block transfers
		1	Tests T register basic and block transfers
		2	Tests V register basic and block transfers
		3	Tests B, T, and V registers with comprehensive block transfers
`bsr.c`	Basic shared/semaphore registers test	0	Tests B, T, and semaphore registers
`btas.c`	B to A register transfer and T to S register transfer test	0	Tests 025*ijk* instruction
		1	Tests 025*ijk* instruction
		2	Tests 075*ijk* instruction
		3	Tests 024*ijk* instruction
		4	Tests random combinations of the four instructions
`cat.c`	Comprehensive abort test	None	Tests memory reference range errors
`cbit.c`	Check-bit test (runs in maintenance mode)	0	Sets the check bits to all 0's (all memory may be selected from 0 to the maximum)
		1	Sets the check bits to all 1's

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

		Sections	
Test †‡	Description	Section Selection	Function
`cbit.c` (cont.)	Check-bit test (runs in maintenance mode)	2	Sets the check bits to a 1 in a 0's field (all memory may be selected from 0 to the maximum)
		3	Sets the check bits to a 0 in a 1's field
		4	Sets the check bit to the address
		5	Sets the check bits randomly
		6	Uses check-bit generation
`cfpt.c`	Comprehensive floating-point test	0	Tests floating-point instructions comprehensively
`cm.c`	Central memory test	0	Tests central memory storage and S register paths
		1	Tests central memory storage and T register paths
		2	Tests central memory storage and B register paths
		3	Uses a simplified algorithm for quick memory address testing
		4	Tests central memory storage and V register paths using both vector logical units
		5	Tests central memory using random data
		6	Tests central memory using conflicts
`crit.c`	Comprehensive random instruction test	0	Detects data-sensitive and instruction-sensitive sequence failures
`csr.c`	Multi-CPU shared registers test	0	Tests shared B and shared T register addressing and control
		1	Tests shared B and shared T register data
		2	Fetches and increments SB*jk* and runs the A*j* test

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

		Sections	
Test † ‡	Description	Section Selection	Function
`csr.c` (cont.)	Multi-CPU shared registers test	3	Tests semaphore registers with set and clear operations and tests semaphore registers with broadcast read and write operations
		4	Tests error reporting jumps and sets *jk* and the A*j* test
		5	Tests semaphore registers
		6	Tests semaphore registers with a multi-CPU deadlock timing test
		7	Tests foreground and background clusters
`csvc.c`	Comprehensive scalar and vector compare test	0	Tests vector registers, paths, and functional units
`ejt.c`	Multi-CPU exchange jump test	0	Tests A0 through S7 registers using scalar memory instructions
		1	Tests (*mn* + A*h* + DBA), (A0 + V*j* + DBA), and (A0 + A*k* + DBA) address adders (uses sliding 1's address pattern)
		2	Tests DBA range error
		3	Tests DLA range error
		4	Tests the instruction's ability to set and clear mode bits in the exchange package and status registers
		5	Tests 076*ijk*, 077*ijk*, and 0014*j*3 instructions
		6	Tests each bit of an exchange address for the ability to exchange
		7	Tests address adders using random A*k* and P register addresses as operands
		10	Tests address adders using random operands
`etem.c`	End-to-end LOSP channel test (runs with the IOS-E ETEI test)	None	Produces random activity on LOSP channel pairs connected to an IOS-E

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

		Sections	
Test † ‡	Description	Section Selection	Function
`fdrec.c`	FDR-4 error-correction test	0	Forces single-bit errors in CPU memory
		1	Forces uncorrectable errors into CPU memory
		2	Writes data with single-bit errors forced on the FDR (going into the SSD)
		3	Forces uncorrectable errors in the SSD by setting the force 0 check-bit switch in remote FDR
		4	Writes data with the SSD 0 check bit on
		5	Forces uncorrectable errors in the SSD by setting the force check-bit switch in the SSD
		6	Forces correctable errors in the local FDR by setting the force check-bit switch
		7	Forces uncorrectable errors in the local FDR by setting the force check-bit switch
`fdrei.c`	FDR-4 error injection feature test	0	Tests injection of data errors
		1	Tests injection of address, block-length, and control compare errors
		2	Tests injection of synchronization errors
		3	Tests injection of framing errors
		4	Tests injection of block-length errors
		5	Tests injection of high bias errors
		6	Tests injection of low lamp errors

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

		Sections	
Test [†] [‡]	Description	Section Selection	Function
`fdrlb.c`	FDR-4 loop-back test	0	Runs a local FDR loop-back test
		1	Runs a remote FDR loop-back test
		2	Runs data through the address loop-back mode
`fpt.c`	Floating-point unit test	0	Tests floating-point addition (canned answers)
		1	Tests floating-point multiplication (canned answers)
		2	Tests floating-point reciprocal unit (canned answers)
		3	Tests floating-point addition (simulated answers)
		4	Tests floating-point integer multiplication (simulated answers)
		5	Tests floating-point multiplication (simulated answers)
		6	Tests floating-point reciprocal unit (simulated answers)
		7	Tests chained-functional units (simulated answers)
		10	Tests floating-point error checking (simulated results)
		11	Tests vector and scalar back-to-back control
`ibba.y`	Instruction buffer test (parcels A and B)	0	Tests parcels A and B using parcels C and D
`ibbc.y`	Instruction buffer test (parcels C and D)	0	Tests parcels C and D using parcels A and B
`ibta.y`	Multi-CPU instruction buffer comprehensive test (parcels A and B)	0	Tests parcels A and B comprehensively using parcels C and D
`ibtc.y`	Multi-CPU instruction buffer comprehensive test (parcels C and D)	0	Tests parcels C and D comprehensively using parcels A and B

[†] A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

[‡] indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

		Sections	
Test † ‡	Description	Section Selection	Function
`ibuf.c`	Instruction buffer test	0	Tests instruction buffers with a series of jump instructions that provide both in- and out-of-buffer conditions
`int.c`	Multi-CPU interrupt test (runs in maintenance mode)	0	Tests a normal exit, an error exit, and a program range error
		1	Tests the IMI mode bit and the MII (monitor mode) interrupt
		2	Tests the deadlock interrupt where the current logical master CPU is rotated in test condition 4
		3	Tests the programmable clock interrupt
		4	Tests the I/O interrupts and the LOSP channel priority on the channel pairs selected in the LOSP channel's table parameter; the current logical master CPU, CPUMc, is rotated until all the CPUs have been the master CPU
		5	Tests interprocessor interrupts; the current logical master CPU, CPUMc location 7657, is rotated until all the CPUs have been the master
		6	Tests correctable-memory-error interrupts and uncorrectable–memory-error interrupts
		7	Tests breakpoint interrupts
		10	Tests operand range error interrupts
		11	Tests floating-point error interrupts
		12	Tests register parity error interrupts
`jib.c`	Jump instruction buffer test (runs in maintenance mode)	0	Tests 0051*jk* instruction to jump within and between instructions
		1	Tests 005*ijk* instruction to flush all instruction buffers and does a flush

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

Test † ‡	Description	Sections	
		Section Selection	Function
`jpt.c` `jpt.y`	Jump/branch test (tests instruction buffers)	0	Tests 006*ijkm* instruction (runs only in CPU selected as master)
		1	Tests A0 conditional branch instructions (S4 = condition counter; A0 = failing data pattern)
		2	Tests S0 conditional branch instructions (A1 = condition counter; S0 = failing data pattern)
		3	Tests P to B00 path (A1 = condition counter; S0 = bits in error; S1 = actual data; S2 = expected data)
		4	Tests B00 to P (A1 = condition counter; S0 = bits in error; S1 = actual data; S2 = expected data)
		5	Tests B*xx* to P (A1 = B*xy*; S0 = bits in error; S1 = actual data; S2 = expected data)
		6	Tests various IBA jumps
		7	Times in-buffer jumps; confirms that no fetch was done (A1 = condition counter; A7 = loop counter; B00 = address of failing code)
		10	Tests parcel 1 of instruction in one instruction buffer or parcel 2 in a different instruction buffer [A1 = condition counter (equals instruction buffer under test); A7 = loop counter]
`jpt256.c` `jpt256.y`	Jump test (with a minimum of 256 Kwords of memory)	0	Tests 006*ijkm* instruction (runs only in CPU selected as master)

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

Test †‡	Description	Sections	
		Section Selection	Function
`jpt256.c` `jpt256.y` (cont.)	Jump test (with a minimum of 256 Kwords of memory)	1	Tests A0 conditional branch instructions (S4 = condition counter; A0 = failing data pattern)
		2	Tests S0 conditional branch instructions (A1 = condition counter; S0 = failing data pattern)
		3	Tests P to B00 path (A1 = condition counter; S0 = bits in error; S1 = actual data; S2 = expected data)
		4	Tests B00 to P (A1 = condition counter; S0 = bits in error; S1 = actual data; S2 = expected data)
		5	Tests B*xx* to P (A1 = B*xy*; S0 = bits in error; S1 = actual data; S2 = expected data)
		6	Tests various IBA jumps
		7	Times in-buffer jumps; confirms that no fetch was done (A1 = condition counter; A7 = loop counter; B00 = address of failing code)
		10	Tests parcel 1 of instruction in one instruction buffer or parcel 2 in a different instruction buffer [A1 = condition counter (equals instruction buffer under test); A7 = loop counter]
`losp.c`	LOSP channel test	0	Performs basic test
		1	Performs basic address test
		2	Performs data test
		3	Performs uneven transfer length test

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

Test † ‡	Description	Sections	
		Section Selection	Function
`losp.c` (cont.)	LOSP channel test	4	Performs CPU CA/CL path test
		5	Performs interrupt test
		6	Performs random address and data test
		7	Performs multi-CPU and concurrent LOSP I/O test
		10	Executes loop specified by the user loop control parameter
`lsc.c`	LOSP channel comprehensive test	0	Produces random activity to test all selected LOSP channel pairs
		1	Patterns read buffer with random data
		2	Tests random data (address limited to MLIMT)
		3	Performs gather/scatter with random data
		4	Aborts on operand-range errors
`mem3.c`	Multi-CPU vector registers/memory test (relies on SBCDBD for error correction)	0	Tests for vector register and memory conflicts
`mem4.c`	CPU memory (tests data paths and addressing) – [can be run with error correction on or off (off is suggested)]	0	Tests data paths between A registers and memory
		1	Tests data paths between B registers and memory
		2	Tests data paths between T registers and memory
		3	Tests data paths: V registers to memory to memory banks
		4	Tests data paths: V registers to memory to memory banks
		5	Checks A*h* addressing

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

Test † ‡	Description	Sections	
		Section Selection	Function
`mem4.c` (cont.)	CPU memory (tests data paths and addressing) – [can be run with error correction on or off (off is suggested)]	6	Checks addressing for B register memory references
		7	Checks T register addressing
		10	Checks V register memory addressing using register A0
		11	Checks V register addressing using V*k*
		12	Checks V register addressing using V*k*
		13	Checks data while trying to cause port A, B, and C conflicts
`mmat.c`	Multi-CPU monitor mode protection test (uses WAIT/RESUME)	0	Verifies 0010*jk* instruction (CA,A*j* A*k*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		1	Verifies 0011*jk* instruction (CL,A*j* A*k*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		2	Verifies 0012*j*0 instruction (CI,A*j*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		3	Verifies 0012*j*1 instruction (MC,A*j*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		4	Verifies 0012*j*2 instruction (DI,A*j*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		5	Verifies 0012*j*3 instruction (EI,A*j*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

Test † ‡	Description	Sections	
		Section Selection	Function
`mmat.c` (cont.)	Multi-CPU monitor mode protection test (uses WAIT/RESUME)	6	Verifies 0013*j*0 instruction (XA A*j*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		7	Verifies 0013*j*2 instruction (EMI) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		10	Verifies 0013*j*3 instruction (DMI) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		11	Verifies 0014*j*0 instruction (RT S*j*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		12	Verifies 0014*j*1 instruction (SIPI A*j*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		13	Verifies 0014*j*2 instruction (CIPI) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		14	Verifies 0014*j*3 instruction (CLN A*j*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		15	Verifies 0014*j*4 instruction (PCI S*j*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		16	Verifies 0014*j*5 instruction (CCI) issues as a no operation while the CPU is in user mode and interrupt mode is disabled

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

		Sections	
Test †‡	Description	Section Selection	Function
`mmat.c` (cont.)	Multi-CPU monitor mode protection test (uses WAIT/RESUME)	17	Verifies 0014*j*6 instruction (ECI) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		20	Verifies 0014*j*7 instruction (DCI) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		21	Verifies 001500 instruction (CPM) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		22	Verifies 001600 instruction (ESI) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		23	Verifies 0017*jk* instruction (BP,*k* A*j*) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		24	Verifies 073i*j*1 instruction (S*i* PM) issues as a no operation while the CPU is in user mode and interrupt mode is disabled
		25	Verifies maintenance modes are disabled
`mpxres.c`	MPX-24 port reservation test (runs in monitor mode)	0	Performs a read-only test (diagnostic mode loop-back read from the MPX devices)
		1	Forces a port reservation
`pave.c` `pave.y`	Memory addressing, conflicts, and port select test	0	Tests 1 and 0 data; address limited to 11,000 words
		1	Tests random data; address limited to 11,000 words
		2	Tests random data; address unlimited

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

		Sections	
Test †‡	Description	Section Selection	Function
`pave.c` `pave.y` (cont.)	Memory addressing, conflicts, and port select test	3	Tests random data; gather/scatter
		4	Tests abort on operand range error
`pint.c`	Programmable interrupt test (runs with IOS-E PINT test)	0	Checks the ability of a CPU to exchange on a programmable interrupt (RTI interrupt) received from the IOS-E
`pmt.c`	Multi-CPU performance monitor	0	Verifies PM counters can be cleared
		1	Verifies counters can be incremented
		2	Verifies performance monitoring of user mode
		3	Verifies performance monitoring of I/O memory references and I/O (port d) conflicts
		4	Verifies counter 015 (clock periods holding issue on semaphores)
`prtc.c`	Multi-CPU programmable-clock and real-time clock (RTC) test	0	Tests RTC data integrity
		1	Tests RTC propagate carry
		2	Tests RTC path
		3	Tests basic PCI mechanisms (forces time-out condition)
		4	Tests basic CCI and DCI mechanisms
		5	Tests basic ICD register test
		6	Tests PCI II register timing
		7	Tests RTC fanout
		10	Performs basic ECI/DCI mechanism check

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

Test † ‡	Description	Sections	
		Section Selection	Function
`rpt.c`	Multi-CPU register parity test	0	Tests register parity
`sab.c`	Scalar basic register test	0	Checks 0’s, 1’s, and alternate patterns (tests S registers and S adders)
		1	Complement patterns *mn* to S
		2	Tests 060 instructions (predetermined operands)
		3	Tests paths
		4	Tests 023*ij*0 instruction
		5	Tests 071*i*0*k* instruction
`sas.c`	A and S register test	0	Tests address addition
		1	Tests scalar addition
		2	Tests address multiplication
		3	Tests address subtraction
		4	Tests scalar subtraction
		5	Tests population count and leading zero
		6	Tests A to S floating-point
		7	Tests A to vector length (VL) register
`scl.c`	Scalar logical basic test	Condition 0	Tests 042 instruction
		Condition 1	Tests 043 instruction
		Condition 2	Tests 044 instruction (conditions 2 through 7 are tested with no S0 operands)
		Condition 3	Tests 045 instruction
		Condition 4	Tests 046 instruction
		Condition 5	Tests 047 instruction
		Condition 6	Tests 051 instruction
		Condition 7	Tests 050 instruction

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

Test † ‡	Description	Sections	
		Section Selection	Function
`scl.c` (cont.)	Scalar logical basic test **NOTE:** Conditions 10 through 15 are tested with S0 operands	Condition 10	Tests 044 instruction
		Condition 11	Tests 045 instruction
		Condition 12	Tests 046 instruction
		Condition 13	Tests 047 instruction
		Condition 14	Tests 051 instruction
		Condition 15	Tests 050 instruction
`scs.c`	Scalar shift test	0	Shifts all S registers left and right by a count of 1, 2, 4, 10, 20, and 40
		1	Shifts an S register left and right by A*j* = 1, 2, 4, and so on until all bits of A*j* have been tested
		2	Tests double shift left and right; tests random shift count and data
		3	Tests selectable parcels; tests random shift instruction with random data and shift counts
`sdt.c`	Storage error correction/detection test (runs in maintenance mode)	0	Tests for single-byte errors
		1	Tests for double-byte errors
		2	Tests for vector read errors
		3	Tests for fetch errors
		4	Tests for B register read errors
		5	Tests for T register read errors
		6	Tests for A register read errors
		7	Tests for exchange error
		10	Tests for I/O (input/output) errors but the LOSP channel must be in loop-back test mode
		11	Tests for back-to-back errors
		12	Tests maintenance modes

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

		Sections	
Test † ‡	Description	Section Selection	Function
`sfa.c`	Floating-point add functional unit test	0	Checks unit reservation timing
		1	Test transmits floating-point constant
		2	Tests scalar path
		3	Tests scaling shifts
		4	Tests subtraction with small differences
		5	Performs FA adder and normalize shifts
		6	Performs sign check
		7	Tests underflow
		10	Tests vector paths
		11	Tests random scalar FA
		12	Tests random vector FA
		13	Tests vector adder
		14	Tests random scalar floating subtract
		15	Tests random vector floating subtract
`sfm.c`	Floating-point multiply functional unit test	0	Tests unit reservation timing
		1	Tests scalar path
		2	Tests exponent –1
		3	Tests multiply without changing the exponent (straight through exponent)
		4	Performs sign check
		5	Tests coefficient matrix
		6	Tests FM adders
		7	Performs strong/weak/2minus test
		10	Tests 0 times 0

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

		Sections	
Test †‡	Description	Section Selection	Function
`sfm.c` (cont.)	Floating-point multiply functional unit test	11	Tests underflow
		12	Tests vector paths
		13	Tests random vector
		14	Performs adder check with vectors
		15	Tests split vector
`sfr.c`	Floating-point reciprocal functional unit test	0	Tests user-selected operand
		1	Tests unit reservation
		2	Tests sliding scalar
		3	Tests sequential scalar
		4	Tests random scalar
		5	Tests vector path
		6	Tests random vector
`smp2.c`	Semaphore box test	0	Tests basic functions
		1	Tests heartbeat semaphore arrays
		2	Tests command compare logic
		3	Tests test mode echo
`sr2.c` `sr2.y`	Random scalar conflict test	0	Tests random scalar conflicts using 1 and 2 parcels
`sr3.c` `sr3.y`	Register and scalar/vector instruction (3-parcel) conflicts test	0	Tests registers and scalar and vector conflicts using 3-parcel instructions
`ssd.c`	Solid-state storage device (SSD) test	0	Tests the SSD operations using a data integrity test of the VHISP-to-SSD channel

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

Test † ‡	Description	Sections	
		Section Selection	Function
`vbt.c`	Vector register basic test 2001 - failing section 2002 - failing condition 2003 - failing subcondition 2004 - failing vector 2005 - failing element 2006 - vector length	0	Tests scalar-to-vector transfers (contains some special error locations)
		1	Tests vector logic
		2	Tests vector addition
		3	Tests vector shift
		4	Tests vector mask/ compressed index
		5	Tests vector population/parity
		6	Tests second-vector timing
`vhc.c`	VHISP channel comprehensive test	0	Tests the VHISP channel and the SSD
`vhec.c`	VHISP error correction/detection test	0	Forces single-bit errors on VHISP output
		1	Forces uncorrectable errors on VHISP output
		2	Forces correctable errors in SSD by setting the force 0 check-bit switch
		3	Forces uncorrectable errors in SSD by setting the force 0 check-bit switch
`vhx.c`	Multi-CPU VHISP multiple channel test	0	Tests the very high-speed channel
`vpt.c`	Vector path test	0	Tests vector registers
		1	Tests vector paths
		2	Tests different paths
		3	Tests the bit counter
		4	Tests 140 – 177 instructions
`vsg.c`	Gather/scatter test	0	Gathers using port B
		1	Scatters using port C
		2	Gathers using port B (port A conflicts)

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-2. Environment 1 and 2 Control Point Tests (continued)

Test † ‡	Description	Sections	
		Section Selection	Function
`vsg.c` (cont.)	Gather/scatter test	3	Scatters using port C (port A conflicts)
		4	Gathers using port B (port C conflicts)
		5	Scatters using port C (port B conflicts)
		6	Gathers using port A (port B conflicts)
		7	Scatters using port C (port A and port B conflicts)
		10	Gathers using port A (port B and port C conflicts
		11	Scatters using port C (port A and port B conflicts)
		12	Gathers using port A (port B chaining and port C conflicts)
		13	Scatters using port C and gathers using port B

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Control Point Utilities

Several control point utilities are available in environments 1 and 2. Table 7-3 provides a quick reference description of these utilities. To view the complete listing for any of these utilities, perform the following steps:

1. Choose **`View --> Listing --> Other`**; the `MME View Listing Setup` window appears.

2. Choose **`Dir --> Utility --> Release`**; the `Dir` field displays `rel/util/lst/*.` and the scroll box shows the utility listings that are available.

3. Double click on the utility you want to view, or click on the utility and click on View... . The listing window appears.

Table 7-3. Environment 1 and 2 Control Point Utilities

Utility	Description	Sections	
		Section Selection	Function
`bat.c`	Basic abort test utility (runs in a single CPU)	0	Checks vector and vector chain memory
`chan.c`	Basic LOSP loop-back test/utility (runs in multiple CPUs)	0	Tests the low-speed channel using the loop-back test
		1	Tests channel data
		2	Tests multi-channels
		3	Tests multi-CPU channels
`clr.c`	Clear utility (runs in multiple CPUs)	0	Clears registers, memory, and shared registers with scalar pattern (clears with zeros; no user-selectable pattern)
		1	Clears registers, memory, and shared registers with vector pattern
`clrfdr.c`	Clear FDR-to-VHISP channel utility (runs in maintenance mode)	0	Clears and checks selected FDR-to-VHISP channels
`clrmem.c`	Clear memory utility (runs in a single CPU)	0	Clears memory; this is the single-CPU inline code version of `clr.c`
`clrreg.c`	Clear registers utility (runs in multiple CPUs)	0	Clears registers; this is the single-CPU inline code version of `clr.c`
`clrshr.c`	Clear shared registers utility (runs in multiple CPUs)	0	Clears shared registers; this is the single-CPU inline code version of `clr.c`
`clrssd.c`	Clear solid-state storage device (SSD) utility (runs in a single CPU)	0	Clears the VHISP channels and from 1 to selected SSDs (solid-state storage devices)
`clrv.c`	Clears interrupts	1	Writes a vector pattern to memory
`dc.c`	Diagnostic controller utility	0	Tests the diagnostic controller

† indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.
‡ These utilities hang on termination.

Table 7-3. Environment 1 and 2 Control Point Utilities (continued)

Utility	Description	Sections	
		Section Selection	Function
`diag.c`	Diagnostic utility (runs in multiple CPUs)	0	Provides a loop frame for writing loops in binary
`find.c`	Find utility (runs in a single CPU)	0	Locates a word, parcel, 9-bit pattern or check-bit pattern in the MWS buffer
`hisp.c`	HISP test utility (No CPU should ever execute in this control point.)	0	Reserves a block storage segment
`iocon.c`	I/O conflict utility (runs in a single CPU)	0	Helps user with I/O conflict problem when port A, B, or C is conflicting with port D; you must supply the code with an error-checking routine
`loop.c`	Loop frame utility (runs in a single CPU)	0	Provides a frame for writing a loop in binary in environment 1
`patt00.c`	Simplified pattern utility (runs in a single CPU)	0	Writes a constant scalar data pattern to memory
		1	Writes a halfword scalar logical address pattern to memory
`patt01.c`	Memory pattern utility (runs in multiple CPUs)	0	Writes a constant scalar data pattern to memory
		1	Writes a halfword scalar logical address pattern to memory
		2	Writes a scalar addition bit field address pattern to memory
		3	Writes a constant vector data pattern to memory
		4	Writes a vector addition bit field address pattern to memory
`scan.c`	Memory scan utility (runs in multiple CPUs)	0	Reads memory and logs SBC and DBD errors with full address, data, and processor number
`ssdio.c`	SSD I/O test (runs in a single CPU)	0	Tests the SSD I/O operations using a data integrity test of the VHISP-to-SSD channel
`ssdsz.c`	SSD memory size test (runs in a single CPU)	0	Checks an SSD attached to a fiber box

† indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.
‡ These utilities hang on termination.

Diagnostic Programs by Functional Groups

Table 7-4 shows which diagnostic programs you use to test specific areas.

Table 7-4. Diagnostic Programs by Functional Groups

Functional Group	Diagnostic Program †‡	Description
A and S registers	`aab.c`	Address register basic test
	`ave.c`	Vector register test
	`sab.c`	Scalar basic register test
	`vbt.c`	Vector register basic test
	`vpt.c`	Vector path test
B, T, and V registers	`brt.c`	Block transfer register test
	`btas.c`	B to A register transfer and T to S register transfer test
	`rpt.c`	Multi-CPU register parity test
Confidence	`cfpt.c`	Comprehensive floating-point test
	`cm.c`	Central memory test
	`crit.c`	Comprehensive random instruction test
	`csvc.c`	Comprehensive scalar and vector compare test
	`ibuf.c`	Instruction buffer test

† A `.y` extension means the diagnostic program was assembled in Y-MP mode, and a `.c` extension means that the diagnostic program was assembled in C90 mode.

‡ indicates the diagnostic program runs in environment 1 only; indicates the diagnostic program runs in environment 2 only; indicates the diagnostic program runs in environment 1 or 2.

Table 7-4. Diagnostic Programs by Functional Groups (continued)

Functional Group	Diagnostic Program † ‡	Description
Control	`bp.c`	Breakpoint interrupt test
	`bsr.c`	Basic shared/semaphore registers test
	`cat.c`	Comprehensive abort test
	`csr.c`	Multi-CPU shared registers test
	`ejt.c`	Multi-CPU exchange jump test
	`ibba.y`	Instruction buffer test (parcels A and B)
	`ibbc.y`	Instruction buffer test (parcels C and D)
	`ibta.y`	Multi-CPU instruction buffer comprehensive test (parcels A and B)
	`ibtc.y`	Multi-CPU instruction buffer comprehensive test (parcels C and D)
	`jib.c`	Jump instruction buffer test (runs in maintenance mode)
	`jpt.c` `jpt.y`	Jump/branch test
	`jpt256.c` `jpt256.y`	Jump test (with a minimum of 256 Kwords of memory)
	`mmat.c`	Multi-CPU monitor mode protection test
	`pmt.c`	Multi-CPU performance monitor
	`prtc.c`	Multi-CPU programmable-clock and real-time clock (RTC) test

† A `.y` extension means the diagnostic program was assembled in Y-MP mode, and a `.c` extension means that the diagnostic program was assembled in C90 mode.

‡ indicates the diagnostic program runs in environment 1 only; indicates the diagnostic program runs in environment 2 only; indicates the diagnostic program runs in environment 1 or 2.

Table 7-4. Diagnostic Programs by Functional Groups (continued)

Functional Group	Diagnostic Program †‡	Description
Control (continued)	`sr2.c` `sr2.y`	Random scalar conflict test
	`sr3.c` `sr3.y`	Register and scalar/vector instruction (3-parcel) conflicts test
External devices	`smp2.c`	Semaphore box test
	`ssd.c`	Solid-state storage device (SSD) test
Functional units	`amb.c` `amb.y`	Address multiply basic test
	`ars.c`	Address (A) and scalar (S) register add and multiply test
	`bmm.c`	Bit matrix multiply (BMM) functional unit test
	`fpt.c`	Floating-point unit test
	`sas.c`	A and S register test
	`scl.c`	Scalar logical basic test
	`sfa.c`	Floating-point add functional unit test
	`sfm.c`	Floating-point multiply functional unit test
	`sfr.c`	Floating-point reciprocal functional unit test
I/O	`chan.c`	Multi-CPU basic LOSP loop-back test/utility
	`etem.c`	End-to-end LOSP channel test (runs with IOS-E ETEI and CRAY T3D ETET3D tests)
	`fdrec.c`	FDR-4 error-correction test

† A `.y` extension means the diagnostic program was assembled in Y-MP mode, and a `.c` extension means that the diagnostic program was assembled in C90 mode.

‡ indicates the diagnostic program runs in environment 1 only; indicates the diagnostic program runs in environment 2 only; indicates the diagnostic program runs in environment 1 or 2.

Table 7-4. Diagnostic Programs by Functional Groups (continued)

Functional Group	Diagnostic Program † ‡	Description
I/O (continued)	`fdrei.c`	FDR-4 error injection feature test
	`fdrlb.c`	FDR-4 loop-back test
	`int.c`	Multi-CPU interrupt test (runs in maintenance mode)
	`losp.c`	LOSP channel test
	`lsc.c`	LOSP channel comprehensive test
	`pint.c`	Programmable interrupt test (runs with IOS-E PINT test)
	`sdt.c`	Storage error correction/detection test (runs in maintenance mode)
	`vhc.c`	VHISP channel comprehensive test
	`vhec.c`	VHISP error correction/detection test
	`vhx.c`	Multi-CPU VHISP multiple channel test
Memory	`aht.c`	A*h* addressing test
	`cbit.c`	Check-bit test (runs in maintenance mode)
	`mem3.c`	Multi-CPU vector registers/memory test (relies on SBCDBD for error correction)
	`mem4.c`	CPU memory (tests data paths and addressing) – [can be run with error correction on or off (off is suggested)]
	`mpxres.c`	MPX-24 port reservation test (runs in monitor mode)
	`pave.c` `pave.y`	Memory addressing, conflicts, and port select test

† A `.y` extension means the diagnostic program was assembled in Y-MP mode, and a `.c` extension means that the diagnostic program was assembled in C90 mode.

‡ indicates the diagnostic program runs in environment 1 only; indicates the diagnostic program runs in environment 2 only; indicates the diagnostic program runs in environment 1 or 2.

Table 7-4. Diagnostic Programs by Functional Groups (continued)

Functional Group	Diagnostic Program †‡	Description
Memory (continued)	`port.c`	Multi-CPU memory port test
	`vsg.c`	Gather/scatter test

† A `.y` extension means the diagnostic program was assembled in Y-MP mode, and a `.c` extension means that the diagnostic program was assembled in C90 mode.

‡ indicates the diagnostic program runs in environment 1 only; indicates the diagnostic program runs in environment 2 only; indicates the diagnostic program runs in environment 1 or 2.

Testing Strategies

Two testing strategies you can use with MME are the top-down approach and the bottom-up approach. The top-down approach operates on the premise that all the diagnostic functions are operable and have not been disabled by the problem. The bottom-up approach operates on the premise that nothing is functional.

Top-down Approach to Problem Solving

Using the top-down approach, you begin in environment 2 with the run system, command buffers, diagnostic programs, and loops. If the diagnostic controller or other hardware prevents you from running tests, you move to environment 1 and run command buffers, diagnostic programs, and loops. If environment 1 is not functional, you move to environment 0 to run tests. Figure 7-1 shows a top-down approach to problem solving. In Figure 7-1, ENV0 indicates that you should perform the action in environment 0, ENV1 indicates environment 1, and ENV2 indicates environment 2.

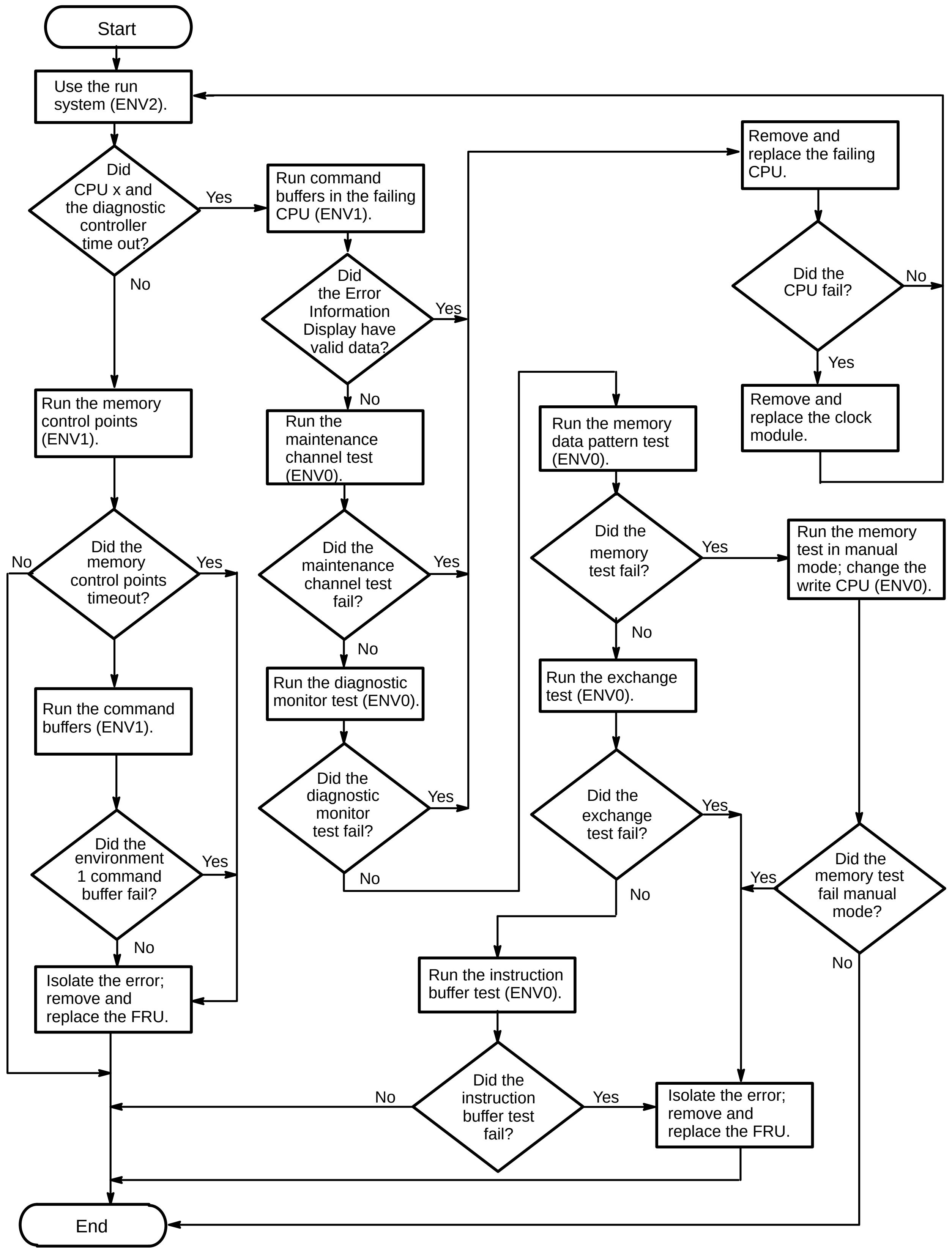

Start
Use the run system (ENV2).
Did CPU x and the diagnostic controller time out?
Yes
No
Run command buffers in the failing CPU (ENV1).
Did the Error Information Display have valid data?
Yes
No
Remove and replace the failing CPU.
Did the CPU fail?
No
Yes
Remove and replace the clock module.
Run the memory control points (ENV1).
Run the maintenance channel test (ENV0).
Run the memory data pattern test (ENV0).
Did the memory control points timeout?
No
Yes
Did the maintenance channel test fail?
Yes
No
Did the memory test fail?
Yes
No
Run the memory test in manual mode; change the write CPU (ENV0).
Run the command buffers (ENV1).
Run the diagnostic monitor test (ENV0).
Run the exchange test (ENV0).
Did the diagnostic monitor test fail?
Yes
No
Did the exchange test fail?
Yes
No
Did the environment 1 command buffer fail?
Yes
No
Did the memory test fail manual mode?
Yes
No
Isolate the error; remove and replace the FRU.
Run the instruction buffer test (ENV0).
Did the instruction buffer test fail?
No
Yes
Isolate the error; remove and replace the FRU.
End

Figure 7-1. Using a Top-down Approach to Problem Solving

Bottom-up Approach to Problem Solving

Using the bottom-up approach, you begin testing the most basic functions in environment 0. If they are functional, use environment 1 to test those functions. Then if they are operable, use environment 2 to test those functions.

Run these environment 0 tests in the order shown below:

1. Run the maintenance channel test.
2. Run the diagnostic monitor test.
3. Run the memory data pattern test.
4. Run the exchange test.
5. Run the instruction buffers test.
6. Run the miscellaneous test.

Table 7-5 shows diagnostic programs listed that require the fewest functions operable (shown at the top of the table) to the most functions operable (shown at the bottom of the table), according to Systems Test and Check-out (STCO). Run them in the order shown in the table in the appropriate environment.

Table 7-5. Solving Problems from the Bottom Up

Control Point [†] [‡]	Area Tested	Control Point Description
`mem1.c`	Memory	Tests memory
`ibba.y`	Instruction buffer	Tests the instruction buffer using parcels A and B
`ibbc.y`	Instruction buffer	Tests the instruction buffer using parcels C and D
`aab.c`	Address (A) registers	Tests A registers and the address adder
`sab.c`	S registers	Tests scalar registers
`sas.c`	S registers	Tests scalar addition operations
`scl.c`	S registers	Tests scalar logic
`scs.c`	S registers	Tests scalar shifting

[†] A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

[‡] indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-5. Solving Problems from the Bottom Up (continued)

Control Point † ‡	Area Tested	Control Point Description
`jpt256.c` `jpt256.y`	Branch control	Tests conditional and unconditional jumps
`jpt.c` `jpt.y`	Branch control	Tests conditional and unconditional jumps
`cbit.c`	Check bit	Tests check-bit storage and generation
`sdt.c`	Memory error correction	Tests for single-byte and double-byte errors; A, B, and T register read errors; exchange errors; fetch errors; and I/O errors; tests maintenance mode
`pmt.c`	Performance monitor	Tests system performance
`jib.c`	Instruction buffers	Tests the instruction buffers using jump instructions
`int .c`	Programmable-clock interrupt (PCI)	Tests interrupts, exits, and program range interrupts
`rpt.c`	Register parity	Tests register parity
`btas.c`	A, B, S, and T registers	Tests the transfer of data from B to A registers and from T to S registers
`brt.c`	Shared registers	Tests basic shared registers
`aht.c`	A registers	Tests A*h* index memory references
`amb.c` `amb.y`	A registers	Tests A registers using a basic multiply operation
`ars.c`	A registers	Tests A and scalar (S) registers using integer add and multiply operations
`mem3.c`	Memory	Tests memory (multi-CPU test)

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-5. Solving Problems from the Bottom Up (continued)

Control Point [†] [‡]	Area Tested	Control Point Description
`mem4.c`	Memory	Tests memory
`fdrlb.c`	Fiber-optic box (FDR-4)	Forces uncorrectable errors in the local FDR by setting the force check-bit switch
`fdrec.c`	Fiber-optic box (FDR-4)	Runs local and remote loop-back tests and runs data through the address loop-back mode for error-correction testing
`cm.c`	Central memory	Tests B, S, T, and V register paths; logical vector units; and central memory conflicts
`vbt.c`	V registers	Tests V register basic operations
`ave.c`	V register chip	Tests the V register chip
`vpt.c`	V registers	Tests V register paths
`ibta.y`	Instruction buffers	Tests the instruction buffers comprehensively using parcels A and B (multi-CPU test)
`ibtc.y`	Instruction buffers	Tests the instruction buffers comprehensively using parcels C and D (multi-CPU test)
`ibuf.c`	Instruction buffers	Tests instruction buffers with a series of jump instructions for in- and out-of-buffer conditions (multi-CPU test)
`ejt.c`	Exchange and jump	Runs an exchange and jump test
`vsg.c`	V registers	Tests vector registers using scatter/gather
`fpt.c`	Floating-point functional unit	Tests memory with floating-point instructions
`cfpt.c`	Floating-point functional unit	Tests memory with comprehensive floating-point instructions

[†] A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

[‡] indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-5. Solving Problems from the Bottom Up (continued)

Control Point † ‡	Area Tested	Control Point Description
`csr.c`	Shared registers	Tests shared registers comprehensively
`sr2.c` `sr2.y`	S registers	Tests for random scalar conflicts
`sr3.c` `sr3.y`	S and V registers	Tests for scalar and vector conflicts
`bp.c`	Break point	Tests scalar and vector write references
`csvc.c`	S and V registers, paths, and functional units	Tests using comprehensive scalar and vector compare operations
`bsr.c`	B, T, and V registers	Tests shared registers with a basic test (multi-CPU test)
`crit .c`	Comprehensive random instructions	Detects data-sensitive and instruction-sensitive sequence failures
`bmm.c`	Bit matrix multiply functional unit	Tests the bit matrix multiply functional unit (for CPU revision 5 and greater)
`clrssd.c`	SSD	Clears one or more solid-state storage devices (SSDs) and the very high-speed (VHISP) channels
`ssd.c`	SSD	Tests the solid-state storage device (SSD) using a data integrity test of the VHISP-to-SSD channel
`ssdio.c`	SSD and VHISP	Tests the I/O of the solid-state storage device (SSD) and the very high-speed (VHISP) channels
`vhc.c`	SSD and VHISP	Tests the solid-state storage device (SSD) and the very high-speed (VHISP) channels
`vhx.c`	VHISP	Tests the very high-speed (VHISP) channels

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

Table 7-5. Solving Problems from the Bottom Up (continued)

Control Point † ‡	Area Tested	Control Point Description
`chan.c` `chan.y`	Channels	Tests the low-speed channel; channel data; and multiple channels
`losp.c`	Low-speed channel	Tests all low-speed channel functions
`lsc.c`	Low-speed channel pairs	Tests all low-speed channel pairs using random activity

† A `.y` extension means the control point was assembled in Y-MP mode, and a `.c` extension means that the control point was assembled in C90 mode.

‡ indicates the control point runs in environment 1 only; indicates the control point runs in environment 2 only; indicates the control point runs in environment 1 or 2.

8 SIMULATOR AND BUGGER/DEBUGGER

This section provides information about using the mainframe simulator (MSIM) to practice using MME, diagnostic programs, and the debug (MDB) control program. MSIM runs much like the actual hardware. You do not control MSIM directly but instead use MME and the MDB control program. You can use MME and the MDB control program to re–create a situation the customer may have encountered. This helps you to understand problems and to identify and solve future problems more quickly. Using this section, you can perform the following tasks:

- Start and use MSIM
- Use the MDB control program to simulate hardware errors
- Use the MDB control program to debug and investigate diagnostic programs in detail
- Use the MDB control program to single-step CPUs
- Use the MDB control program to examine and change active registers in the simulator

Starting the Mainframe Simulator with the Bugger/Debugger

You can start MSIM and MDB from the OpenWindows workspace menu or from a UNIX command prompt.

To start MME with MSIM and MDB from the workspace menu, perform one of the following actions:

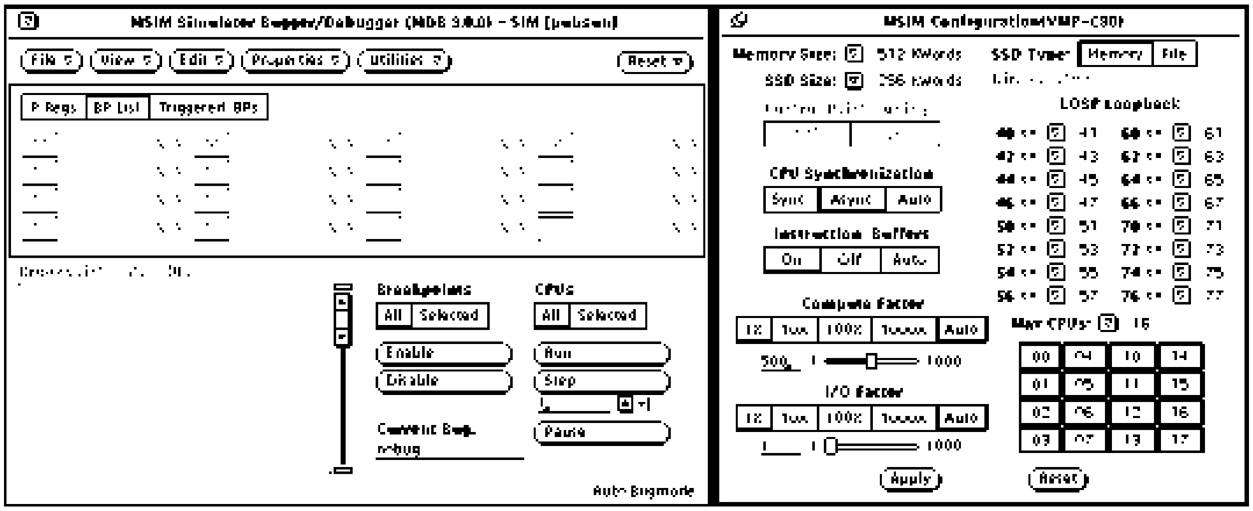
- Choose **`Maintenance Tools ––> MME Simulator ––> MME env 0 ––> Simulator with Debugger`** to start MME environment 0 with the simulator and debugger.
- Choose **`Maintenance Tools ––> MME Simulator ––> MME env 1 ––> Simulator with Debugger`** to start MME environment 1 with the simulator and debugger.
- Choose **`Maintenance Tools ––> MME Simulator ––> MME env 2 ––> Simulator with Debugger`** to start MME environment 2 with the simulator and debugger.

To start MME with MSIM from the workspace menu, perform one of the following actions:

- Choose **`Maintenance Tools --> MME Simulator --> MME env 0 --> Simulator`** to start MME environment 0 with the simulator.
- Choose **`Maintenance Tools --> MME Simulator --> MME env 1 --> Simulator`** to start MME environment 1 with the simulator.
- Choose **`Maintenance Tools --> MME Simulator --> MME env 2 --> Simulator`** to start MME environment 2 with the simulator.

To start MME with MSIM and MDB from a UNIX command prompt, use the `–debug` command line option. To start MME with MSIM, use the `–sim` command line option. Refer to the information that describes starting MME from a UNIX command prompt in Sections 3 and 4 for more information about command line options.

MME displays an environment window, and MDB displays the `MSIM Simulator Bugger/Debugger` and `MSIM Configuration` windows, shown in Figure 8-1.



Current Bug: mbug

Memory Size: 512 Kwords
SSD Size: 256 Kwords
SSD Type: Memory File
Compute Factor: 500 (1–1000)
I/O Factor: 1 (1–1000)
Max CPUs: 16

Figure 8-1. MSIM Simulator Bugger/Debugger and Configuration Windows

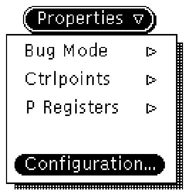
Configuring the Mainframe Simulator

After you have started the mainframe simulator, you can configure the mainframe simulator or use the default configuration. To configure the mainframe simulator, perform the following procedure:

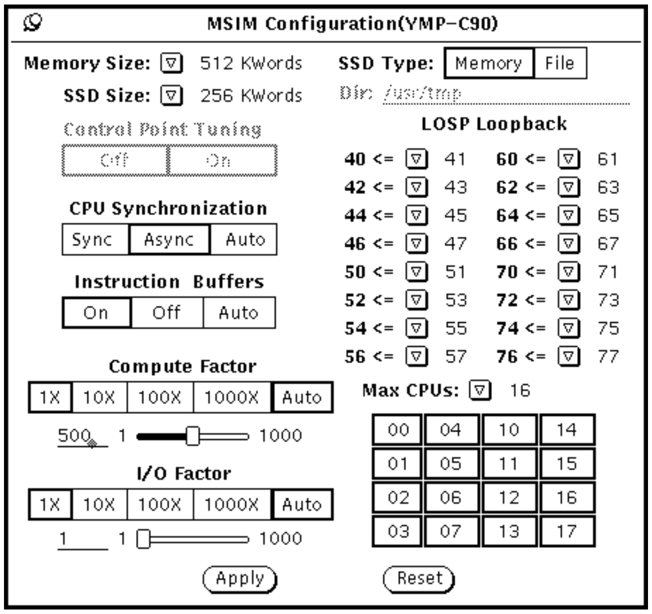
1. Perform one of the following actions:

   - If MDB displays the `MSIM Configuration` window, go to Step 2. This window appears automatically when you start MDB.

   - If MDB does not display the `MSIM Configuration` window, choose **`Properties --> Configuration`**, as shown at the left.



   The `MSIM Configuration` window appears:



MSIM Configuration(YMP–C90)

Memory Size: ▽ 512 KWords
SSD Size: ▽ 256 KWords
SSD Type: Memory | File
Dir: /usr/tmp

Control Point Tuning
Off | On

CPU Synchronization
Sync | Async | Auto

Instruction Buffers
On | Off | Auto

Compute Factor
1X | 10X | 100X | 1000X | Auto
500 1 1000

I/O Factor
1X | 10X | 100X | 1000X | Auto
1 1 1000

LOSP Loopback

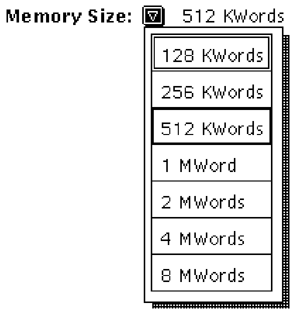
40 <=	▽ 41	60 <=	▽ 61
42 <=	▽ 43	62 <=	▽ 63
44 <=	▽ 45	64 <=	▽ 65
46 <=	▽ 47	66 <=	▽ 67
50 <=	▽ 51	70 <=	▽ 71
52 <=	▽ 53	72 <=	▽ 73
54 <=	▽ 55	74 <=	▽ 75
56 <=	▽ 57	76 <=	▽ 77

Max CPUs: ▽ 16

00	04	10	14
01	05	11	15
02	06	12	16
03	07	13	17

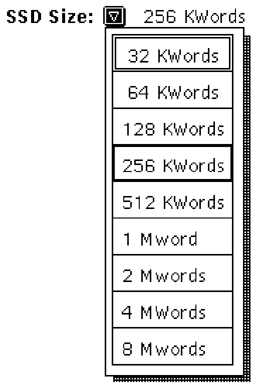
Apply Reset

2. Choose the memory size from the `Memory Size:` ▽:



Memory Size: ▽ 512 KWords

3. Choose the SSD (solid-state storage device) memory size from the `SSD Size:` ▽:



SSD Size: ▽ 256 KWords

4. Click on `CPU Synchronization:` Sync (synchronous) or Async (asynchronous). The Sync setting forces all CPUs to execute the same instructions at the same time; the CPUs are synchronized. The Async setting does not force the CPUs to execute the same instructions at the same time; the CPUs may be executing different instructions.

   Currently, Auto (automatic CPU synchronization) is not implemented. If you click on Auto, the `CPU Synchronization` setting option defaults to the setting of the previously applied configuration.

5. Click on `Instruction Buffers:` [On] or [Off]. Currently, [Auto] is not implemented. If you set `Instruction Buffers` to [Auto], the `Instruction Buffers` setting option defaults to the setting of the previously applied configuration.

   If you use `Instruction Buffers:` [On], the instruction buffers work like a CRAY C90 series computer system. That is, if you know the instructions are in instruction buffers, you can overwrite them in memory.

   If you use `Instruction Buffers:` [Off], the instructions are pulled from simulated mainframe memory. You must not assume that the instructions are in the instruction buffers.

   NOTE: In a CRAY C90 series computer system, the instruction buffers improve performance; however, instruction buffers actually slow down the simulator performance significantly.

6. Specify the compute factor by performing one of the following actions:

   - Type the value you want in the `Compute Factor` field. Press and release the ↵ key to change the value.

   - Click on the `Compute` slider and drag the slider to the value you want.

   The compute factor specifies the number of instructions the simulator performs before MME and MDB are updated. The higher the compute factor, the higher the pass counts. The lower the compute factor, the more often MME and MDB are updated with data.

7. Click on the [1x], [10x], [100x], [1000x], or [Auto] `Compute Factor` by which you want to multiply the value in the `Compute Factor` field. Currently, [Auto] is not implemented.

8. Specify the I/O factor by performing one of the following actions:

   - Type the value you want in the `I/O Factor` field. Press and release the ↵ key to change the value.

   - Click on the `I/O Factor` slider and drag the slider to the value you want.

   The I/O factor determines the speed at which the simulator updates channel activity. The higher the specified number, the faster the channel activity is updated.

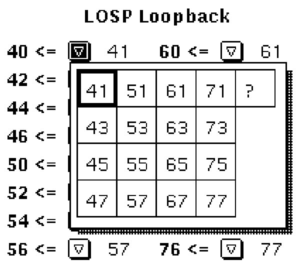
9. Click on the [1x], [10x], [100x], [1000x], or [Auto] `I/O Factor` by which you want to multiply the value in the `I/O` field. Currently, [Auto] is not implemented.

10. Use the `SSD Type` selection to specify the type of simulated SSD created for running SSD diagnostic programs. Perform one of the following actions:

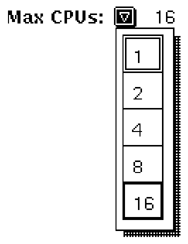
    - Click on [Memory] to use the MWS-E memory. Your workstation should have at least 12 megabytes of memory for good performance.

    - Click on [File] to use a file on the hard disk. Triple click on the `Dir` field and type the name of the directory you want to use. In this case, the simulated SSD is created in a file on disk in the specified directory.

11. Choose the appropriate `LOSP Loopback:` ▽ to specify the LOSP (low-speed channel number) you want to configure:



12. Choose `Max CPUs:` ▽ to specify the number of CPUs you want to configure:



If you click on an individual CPU under `Max CPUs`, you configure the CPU. If you deselect an individual CPU under `Max CPUs`, you remove the CPU from the configuration. If you deselect a CPU, the effect is the same as removing the physical CPU module from the machine, as far as MME/MCE is concerned. Then you can click on Reset in the `MCE` base window to force MME/MCE to check the installed CPUs for the configuration you just specified.

13. Perform one of the following steps:

    - Click on Apply to change the values in MSIM to the values you specified in this `MSIM Configuration` window.

    - Click on Reset to restore the previous values.

Using Bugs

With MSIM, you can load and remove the system-supplied bugs to help you understand how to solve problems.

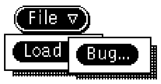
The following bug files are available: `bugmch1`, `bugmctl1`, `bugmem1`, `bugmem2`, `bugmem3`, `bugmem4`, `bugmem5`, `bugmem6`, `bugcpu1`, `bugcpu2`, `bugcpu3`, `bugcpu4`, `bugcpu5`, `bugshr1`, `bugshr2`, and `bugshr3`. Refer to the files in `/cri/cme/c90/rel/msim/bugdoc` for detailed information about the bugs and for troubleshooting hints.

The following subsections explain these tasks:

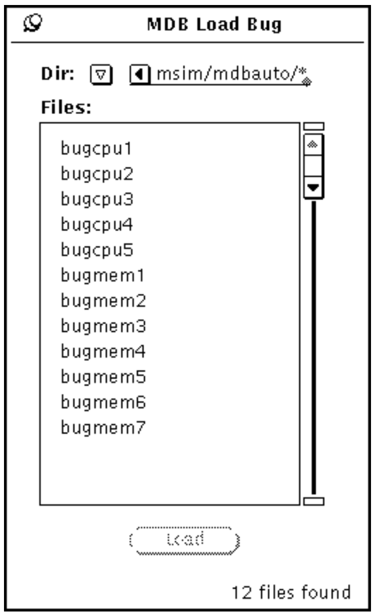
- Loading a bug
- Removing a bug

Loading a Bug

To load a bug, perform the following procedure:



1. Choose **`File --> Load --> Bug`**, as shown at the left. The `MDB Load Bug` window appears:

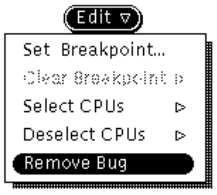


2. Choose the `Dir:` ▽ to specify the directory you want to use or triple click on the `Dir` field, type the directory name, and press the return (↵) key.

3. Click on the bug you want to load in the `Files` scroll box.

4. Click on (Load); the simulator loads the bug. MDB loads the bug. The currently loaded bug is indicated in the `Current Bug` field.

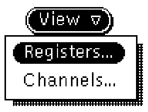
Removing a Bug



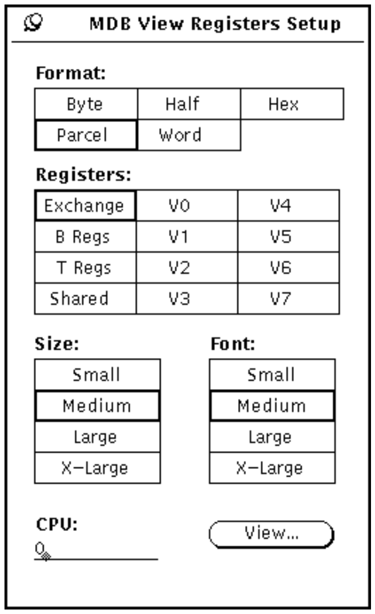
To remove a bug, choose **`Edit --> Remove Bug`**, as shown at the left. MDB removes the bug, which is indicated by `nobug` in the `Current Bug` field.

Viewing the Contents of Registers

To view the contents of registers, perform the following procedure:



1. Choose **`View --> Registers`**, as shown at the left. MDB displays the `MDB View Registers Setup` window:



MDB View Registers Setup

Format:

Byte	Half	Hex
Parcel	Word	

Registers:

Exchange	V0	V4
B Regs	V1	V5
T Regs	V2	V6
Shared	V3	V7

Size: Small, Medium, Large, X–Large

Font: Small, Medium, Large, X–Large

CPU: 0

View...

2. Click on a `Format` [Byte, Half (halfword), Hex (hexadecimal), Parcel, or Word] to indicate which format the register contents should be displayed in.

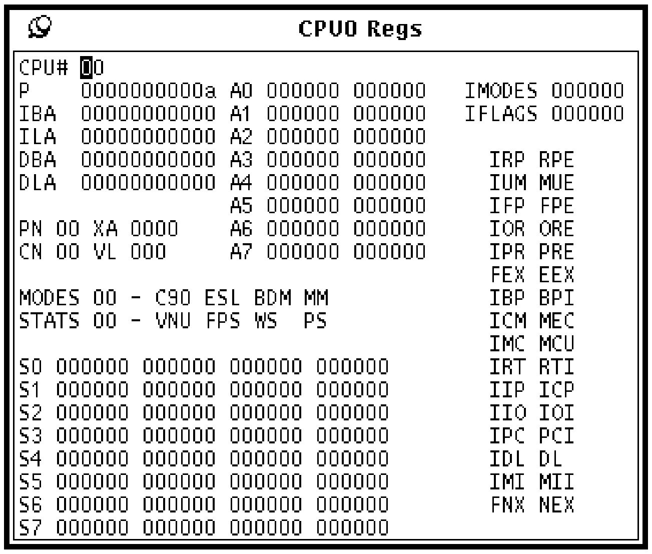
3. Specify which `Registers` you want to view by clicking on one of the following: Exchange, B Regs, T Regs, Shared, V0, V1, V2, V3, V4, V5, V6, or V7.

4. Click on a `Size` [Small, Medium, Large, or X–Large (extra large)] to specify the size of the window that will be displayed.

5. Click on a `Font` [Small, Medium, Large, or X–Large (extra large)] to specify the size of the font that will be displayed.

6. Double click on the `CPU` field. Type the number of the CPU you want to use.

7. Click on `View..`. MDB displays the specified register. For example, if you select CPU 0 exchange registers, the following window appears:

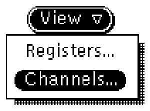


CPU0 Regs

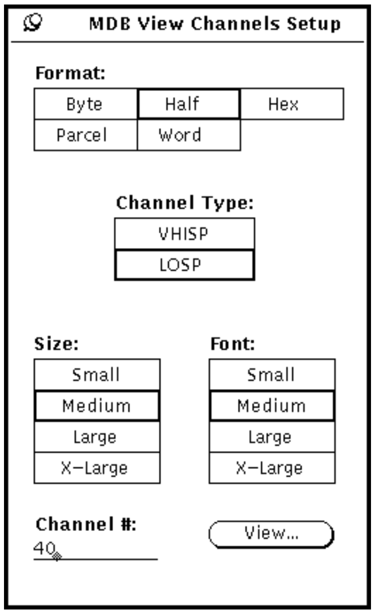
```
CPU# 00
P    0000000000a A0 000000 000000   IMODES 000000
IBA  00000000000 A1 000000 000000   IFLAGS 000000
ILA  00000000000 A2 000000 000000
DBA  00000000000 A3 000000 000000     IRP RPE
DLA  00000000000 A4 000000 000000     IUM MUE
                 A5 000000 000000     IFP FPE
PN 00 XA 0000    A6 000000 000000     IOR ORE
CN 00 VL 000     A7 000000 000000     IPR PRE
                                      FEX EEX
MODES 00 - C90 ESL BDM MM             IBP BPI
STATS 00 - VNU FPS WS  PS             ICM MEC
                                      IMC MCU
S0 000000 000000 000000 000000        IRT RTI
S1 000000 000000 000000 000000        IIP ICP
S2 000000 000000 000000 000000        IIO IOI
S3 000000 000000 000000 000000        IPC PCI
S4 000000 000000 000000 000000        IDL DL
S5 000000 000000 000000 000000        IMI MII
S6 000000 000000 000000 000000        FNX NEX
S7 000000 000000 000000 000000
```

Viewing Channel Data

To view channel data, perform the following procedure:



1. Choose **`View --> Channels`**, as shown at the left. MDB displays the `MDB View Channels Setup` window:



2. Click on a `Format` [Byte, Half (halfword), Hex (hexadecimal), Parcel, or Word] to indicate which format the register contents should be displayed in.

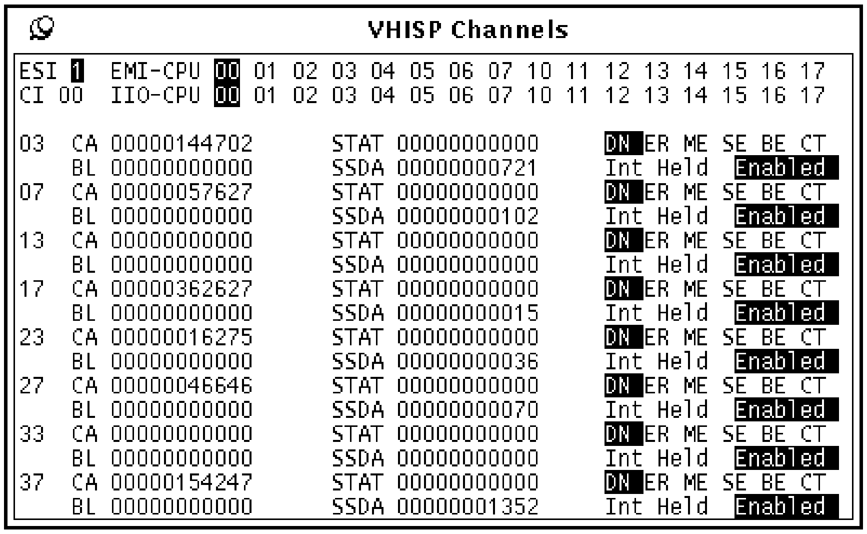
3. Click on a `Channel` [VHISP (very-high speed channel) or LOSP (low-speed channel)] to specify the type of channel data you want to view.

4. Click on a `Size` [Small, Medium, Large, or X-Large (extra large)] to specify the size of the window that will be displayed.

5. Click on a `Font` [Small, Medium, Large, or X-Large (extra large)] to specify the size of the font that will be displayed.

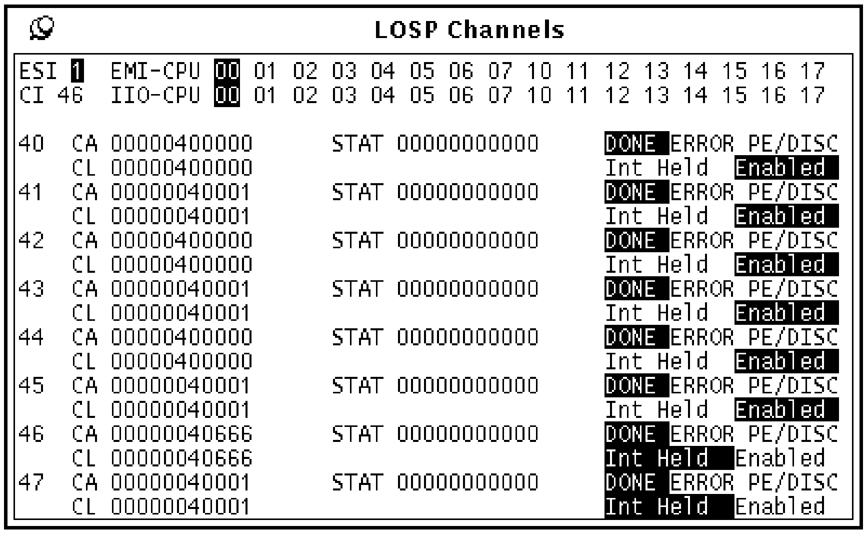
6. Double click on the `Channel #` field. Type the number of the channel you want to use.

7. Click on View... . The channel data is displayed in a window. Refer to Figure 8-2 for an example `VHISP Channels` window or to Figure 8-3 for an example `LOSP Channels` window.



```
                        VHISP Channels
ESI 1   EMI-CPU 00 01 02 03 04 05 06 07 10 11 12 13 14 15 16 17
CI 00   IIO-CPU 00 01 02 03 04 05 06 07 10 11 12 13 14 15 16 17

03   CA 00000144702      STAT 00000000000     DN ER ME SE BE CT
     BL 00000000000      SSDA 00000000721     Int Held  Enabled
07   CA 00000057627      STAT 00000000000     DN ER ME SE BE CT
     BL 00000000000      SSDA 00000000102     Int Held  Enabled
13   CA 00000000000      STAT 00000000000     DN ER ME SE BE CT
     BL 00000000000      SSDA 00000000000     Int Held  Enabled
17   CA 00000362627      STAT 00000000000     DN ER ME SE BE CT
     BL 00000000000      SSDA 00000000015     Int Held  Enabled
23   CA 00000016275      STAT 00000000000     DN ER ME SE BE CT
     BL 00000000000      SSDA 00000000036     Int Held  Enabled
27   CA 00000046646      STAT 00000000000     DN ER ME SE BE CT
     BL 00000000000      SSDA 00000000070     Int Held  Enabled
33   CA 00000000000      STAT 00000000000     DN ER ME SE BE CT
     BL 00000000000      SSDA 00000000000     Int Held  Enabled
37   CA 00000154247      STAT 00000000000     DN ER ME SE BE CT
     BL 00000000000      SSDA 00000001352     Int Held  Enabled
```

Figure 8-2. VHISP Channel Data Display



```
                        LOSP Channels
ESI 1   EMI-CPU 00 01 02 03 04 05 06 07 10 11 12 13 14 15 16 17
CI 46   IIO-CPU 00 01 02 03 04 05 06 07 10 11 12 13 14 15 16 17

40   CA 00000400000      STAT 00000000000     DONE ERROR PE/DISC
     CL 00000400000                           Int Held  Enabled
41   CA 00000040001      STAT 00000000000     DONE ERROR PE/DISC
     CL 00000040001                           Int Held  Enabled
42   CA 00000400000      STAT 00000000000     DONE ERROR PE/DISC
     CL 00000400000                           Int Held  Enabled
43   CA 00000040001      STAT 00000000000     DONE ERROR PE/DISC
     CL 00000040001                           Int Held  Enabled
44   CA 00000400000      STAT 00000000000     DONE ERROR PE/DISC
     CL 00000400000                           Int Held  Enabled
45   CA 00000040001      STAT 00000000000     DONE ERROR PE/DISC
     CL 00000040001                           Int Held  Enabled
46   CA 00000040666      STAT 00000000000     DONE ERROR PE/DISC
     CL 00000040666                           Int Held  Enabled
47   CA 00000040001      STAT 00000000000     DONE ERROR PE/DISC
     CL 00000040001                           Int Held  Enabled
```

Figure 8-3. LOSP Channel Data Display

Using CPUs and Breakpoints

The total number of available CPUs, numbered 00 to 17, for the debug control program appears on the `MSIM Simulator Bugger/ Debugger` window. If a CPU does not have its status set to N/A, then it is available. If the debugger is tracking MME control points (**`Properties --> Control Points --> Track`**), the CPUs assigned to the current control point are available and all the other CPUs are marked as N/A (not available). If control point tracking is not on (**`Properties --> Control Points --> Ignore`**), all CPUs are automatically available to MDB.

A breakpoint consists of a P register address value and a list of CPUs assigned to that breakpoint. When you use a breakpoint and run Cray Research code, the CPU stops issuing instructions when the P register reaches the breakpoint for each assigned CPU. The CPU does not begin issuing instructions again until you press Stop or Run. The address is an absolute memory address or an address based on the instruction base address (IBA) register in the selected CPUs. The simulator recalculates the relative address each time an exchange occurs, so a breakpoint at relative address 5000a, for example, matches the diagnostic listing regardless of where the diagnostic program is loaded in simulated memory.

To stop processing instructions from Cray Research memory when the P register reaches a specified address, you can set a breakpoint. If no breakpoints are in the `MSIM Simulator Debugger/Bugger` window, you cannot select any, but you can set a breakpoint instead. If, however, more than one breakpoint appears in the breakpoint list in the `MSIM Simulator Debugger/Bugger` window, the selected one appears in a rectangular box.

Using breakpoints, you can perform the following tasks:

- Set a breakpoint for one or more CPUs.
- Replace an existing breakpoint with a new P register value or assigned CPU(s).
- Enable a breakpoint for each assigned CPU; when you set a breakpoint, the simulator automatically enables the breakpoint.
- Disable a breakpoint for each assigned CPU. This maintains the breakpoint and the assigned CPU relationship, but the breakpoint will not trigger in the assigned CPUs.
- Clear or erase a breakpoint.
- Assign one CPU to one breakpoint.

- Assign one CPU to more than one breakpoint.
- Assign more than one CPU to one breakpoint, and all the breakpoints have the same assigned CPUs.
- Assign more than one CPU to more than one breakpoint, and some of the breakpoints have the some of the CPUs assigned, while other breakpoints have other CPUs assigned.
- Select all CPUs for all breakpoints.
- Specify selected CPUs for all breakpoints.
- Select all CPUs for the currently selected breakpoint.
- Specify selected CPUs for the currently selected breakpoint.

Selecting and Deselecting CPUs

If any `CPU` 00 through 17 appears in bold on a monochrome display or in this manual, or appears pressed in on a color display, it is selected. If a CPU is not selected (it is deselected), the CPU does not appear in bold on a monochrome display or in this manual, or does not appear pressed in on a color display.

All and Selected `CPUs` determine which CPUs are affected when you use Run, Stop, Pause, Enable, or Disable. If you use All CPUs, then all CPUs not marked as '`N/A`' are affected by the Run, Stop, and Pause; and all CPUs assigned to the currently selected breakpoint are affected by Enable and Disable.

All and Selected `Breakpoints` determine which breakpoints are affected when you use Enable and Disable. The selected breakpoint refers to a breakpoint enclosed in a box in the `Breakpoint List`.

Selecting a CPU

To select a CPU, perform the following procedure:

1. Click on any `CPU` 00 through 17 you want to use. The CPU is selected.
2. Repeat Step 1 until you have selected all the CPUs you want.

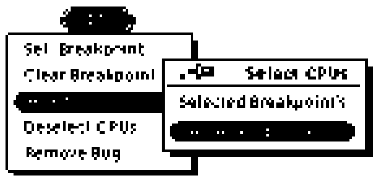
Deselecting a CPU

To deselect a CPU that is selected, perform the following procedure:

1. Click on any selected CPU $\boxed{00}$ through $\boxed{17}$ you want to deselect. The CPU is deselected.

2. Repeat Step 1 until you have deselected all the CPUs that you want to deselect.

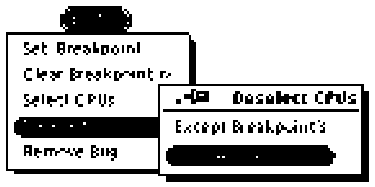
Selecting All Available CPUs



To select all available CPUs, choose **Edit --> Select CPUs --> All Available CPUs**, as shown at the left.

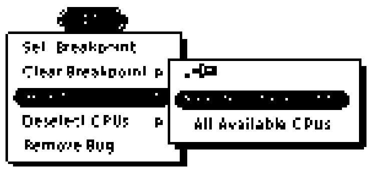
NOTE: If control point tracking is on, all available CPUs means all CPUs assigned to the current MME control point. If control point tracking is off, all available CPUs means all of the CPUs $\boxed{00}$ through $\boxed{17}$ in octal.

Deselecting All CPUs



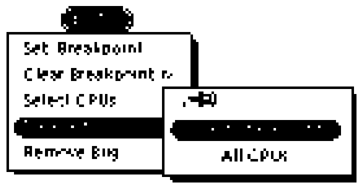
To deselect all CPUs, choose **Edit --> Deselect CPUs --> All CPUs**, as shown at the left.

Selecting CPUs Assigned to the Current Breakpoint



To select those CPUs assigned to the current breakpoint (so you can perform an operation on all the CPUs associated with the breakpoint), choose **Edit --> Select CPUs --> Selected Breakpoint's**, as shown at the left. The MSIM selects all CPUs assigned to the current breakpoint.

Deselecting CPUs Except for CPUs Assigned to the Current Breakpoint



To deselect CPUs except for those CPUs with the current breakpoint assigned to them (to choose only the CPUs affected by the breakpoint), choose **`Edit --> Deselect CPUs --> Except Breakpoint’s`**, as shown at the left. The MSIM deselects any CPUs not assigned to the current breakpoints.

Understanding the Breakpoint Status Field

The breakpoint status field is to the right of its corresponding CPU 00 through 17 in the `MSIM Simulator Bugger/Debugger` window. The letters in the breakpoint status field identify the status of each CPU regarding the currently selected breakpoint. The letters have the following meanings:

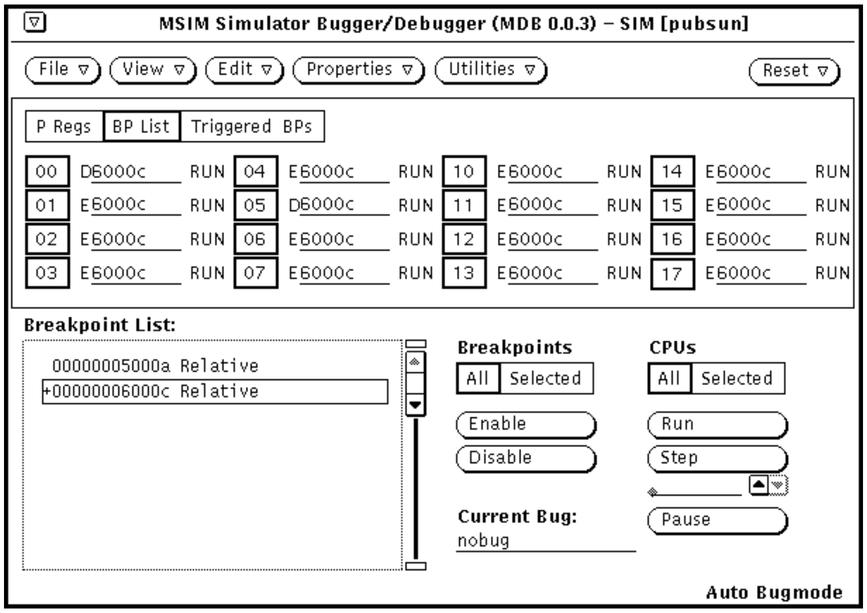
- If a `B` is in the first position of the status field, the breakpoint is currently triggered in the specific CPU.
- If a `D` is in the first position of the status field, the breakpoint is currently disabled for the specific CPU.
- If an `E` is in the first position of the status field, the breakpoint is currently enabled for the specific CPU.
- If the status field is blank, the CPU is not assigned to the current breakpoint.

The P registers (P Regs), breakpoint list (BP List), and triggered breakpoints (Triggered BPs) are used as follows:

- If you use P Regs, MDB periodically reads the current P register values from the MSIM and displays them next to CPUs 00 through 17.
- If you use BP List, any CPUs assigned to the current breakpoint have the breakpoint address displayed. This identifies which CPUs are associated with which breakpoints.
- If you use Triggered BPs, MDB displays a value for all CPUs that have reached a breakpoint.

NOTE: The breakpoint status field containing a `B`, a `D`, an `E`, or a blank) always refers to the currently selected breakpoint. However, the address shown with this selection is independent of the breakpoint selection. You can use it to see the overall breakpoint status of the system.

Figure 8-4 shows a sample breakpoint status field.



```
00 D6000c RUN  04 E6000c RUN  10 E6000c RUN  14 E6000c RUN
01 E6000c RUN  05 D6000c RUN  11 E6000c RUN  15 E6000c RUN
02 E6000c RUN  06 E6000c RUN  12 E6000c RUN  16 E6000c RUN
03 E6000c RUN  07 E6000c RUN  13 E6000c RUN  17 E6000c RUN
```

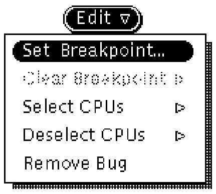
```
 00000005000a Relative
+00000006000c Relative
```

Current Bug: nobug

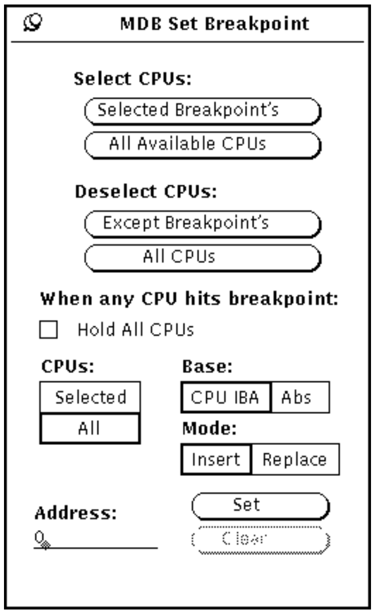
Figure 8-4. Breakpoint Status Field

Setting a Breakpoint

To set a breakpoint, perform the following procedure:



1. Choose **`Edit --> Set Breakpoint`**, as shown at the left. The `MDB Set Breakpoint` window appears:



2. Perform one of the following actions:

   - Click on Selected Breakpoint's to select CPUs with the currently selected breakpoint (in the breakpoint list); use this button to set a new breakpoint with the same CPU assignments as the currently selected breakpoint.

   - Click on All Available CPUs to select all available CPUs

   NOTE: Use Selected Breakpoint's, All Available CPUs, Except Breakpoint's, or All CPUs to specify which CPUs will or will not be affected by the breakpoint you are creating. If you click on All CPUs for CPUs, MDB ignores the settings for the other `Selected` or `Deselect CPUs` buttons.

   - Click on Except Breakpoint's to deassign the CPUs without the currently selected breakpoint

   - Click on All CPUs to deassign all CPUs

   - Click on the CPUs (00 through 17) in the `MSIM Simulator Bugger/Debugger` base window to select (or deselect) them

3. Perform one of the following actions:

   - To hold all CPUs when the breakpoint is reached in any CPU, click on the `Hold All CPUs` box. Go to Step 4.

   NOTE: The `Hold All CPUs` feature is provided to help you debug multi-CPU code with tight timing considerations. This feature enables you to examine *all* CPUs when any *one* CPU reaches a certain point in the code.

   - To allow other CPUs to keep running when a CPU reaches a breakpoint, go to Step 4.

4. Click on one of the following selections:

   - Selected to assign only the highlighted CPUs
   - All to assign all available CPUs to the breakpoint

5. Click on one of the following selections:

   - CPU IBA to specify the address relative to the current value in the instruction base address and to have it recalculated after every exchange

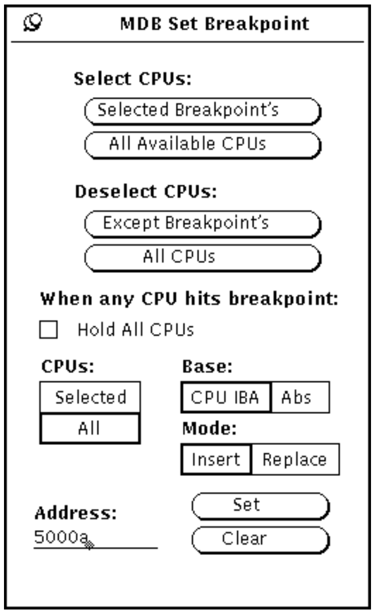
   - Abs (absolute) to use a fixed address in memory

6. Click on one of the following selections:

   - Insert to create a new breakpoint

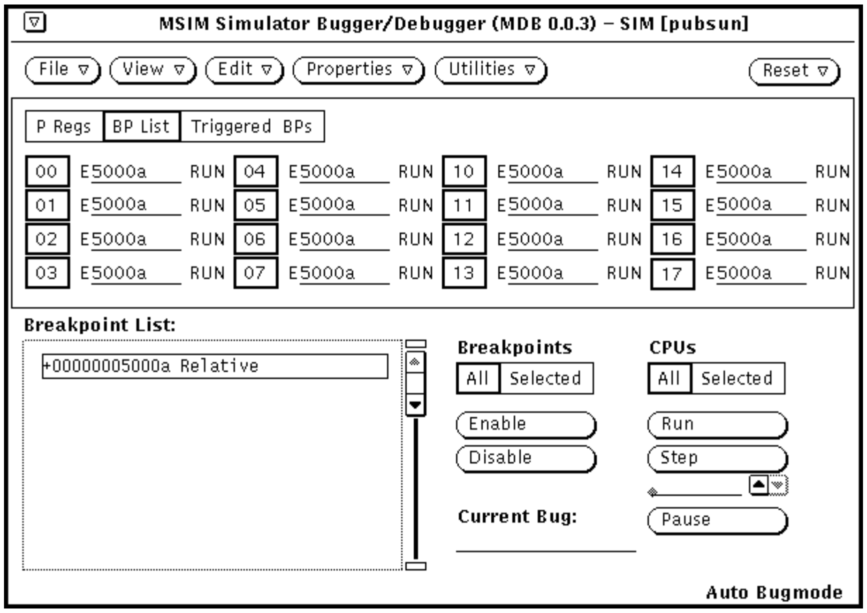
   - Replace to replace the currently selected breakpoint, including its CPU assignments

7. Click on the `Address` field, and type the value you want. In this example, use 5000a in the `Address` field to set the breakpoint at 5000a, as shown:



Address: 5000a

8. Click on Set . Refer to Figure 8-5.

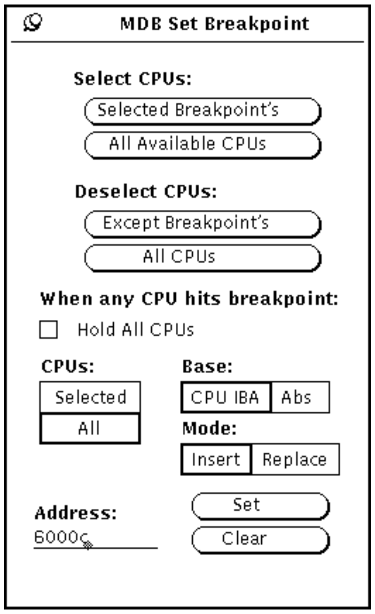

MSIM Simulator Bugger/Debugger (MDB 0.0.3) – SIM [pubsun]

00	E5000a	RUN	04	E5000a	RUN	10	E5000a	RUN	14	E5000a	RUN
01	E5000a	RUN	05	E5000a	RUN	11	E5000a	RUN	15	E5000a	RUN
02	E5000a	RUN	06	E5000a	RUN	12	E5000a	RUN	16	E5000a	RUN
03	E5000a	RUN	07	E5000a	RUN	13	E5000a	RUN	17	E5000a	RUN

Breakpoint List:
+00000005000a Relative

Figure 8-5. Breakpoint Set at Location 5000a

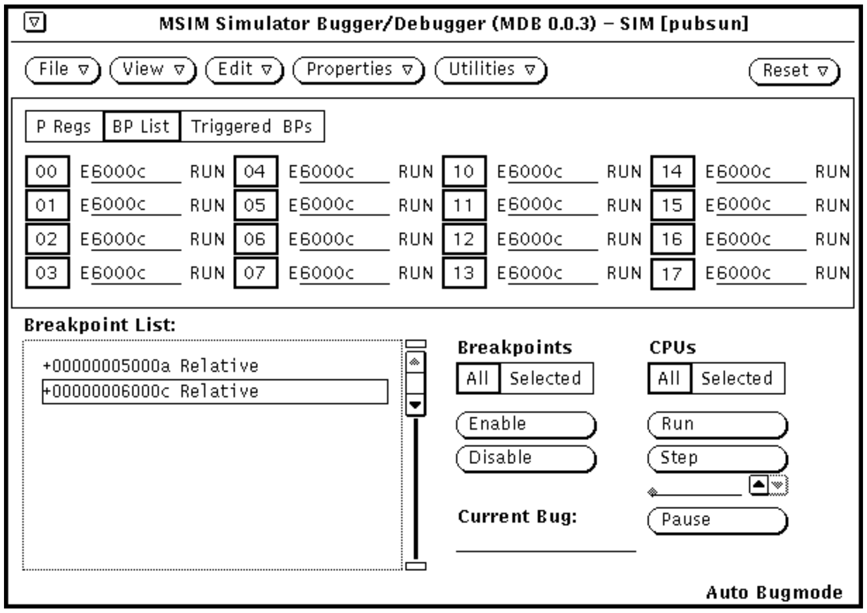
9. Repeat Steps 2 through 8, but in Step 7, type 6000c in the `Address` field to set the next breakpoint at 6000c. Refer to Figure 8-6.


MDB Set Breakpoint

Address: 6000c

Figure 8-6. Breakpoint at Location 6000c

After you have set the breakpoint to address 6000c, the following window is shown. Refer to Figure 8-7.



```
00 E6000c RUN   04 E6000c RUN   10 E6000c RUN   14 E6000c RUN
01 E6000c RUN   05 E6000c RUN   11 E6000c RUN   15 E6000c RUN
02 E6000c RUN   06 E6000c RUN   12 E6000c RUN   16 E6000c RUN
03 E6000c RUN   07 E6000c RUN   13 E6000c RUN   17 E6000c RUN
```

```
+00000005000a Relative
+00000006000c Relative
```

Figure 8-7. Breakpoint Set at Location 6000c

Using All or Selected Breakpoints

To select all or selected breakpoints, click on All or Selected. In this example, Figure 8-8 shows that the user selected All.

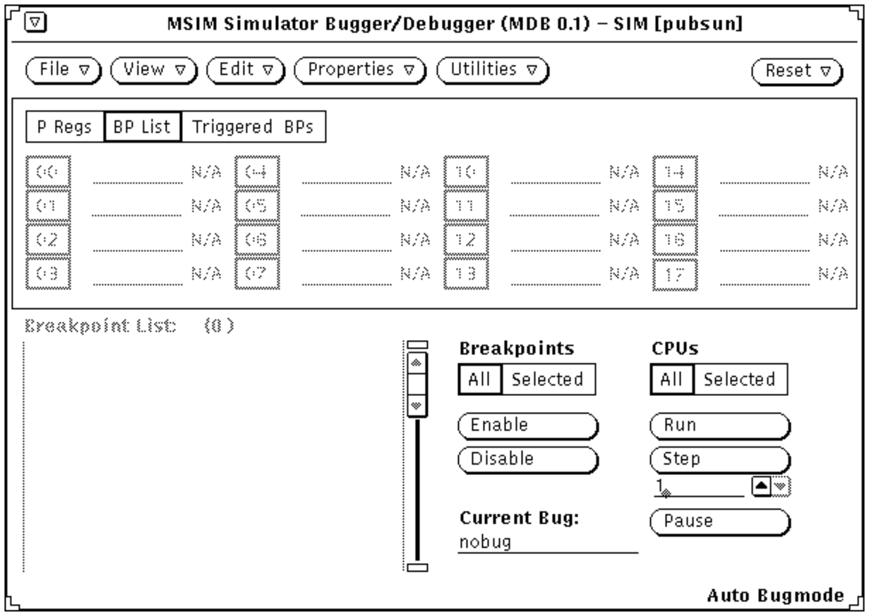
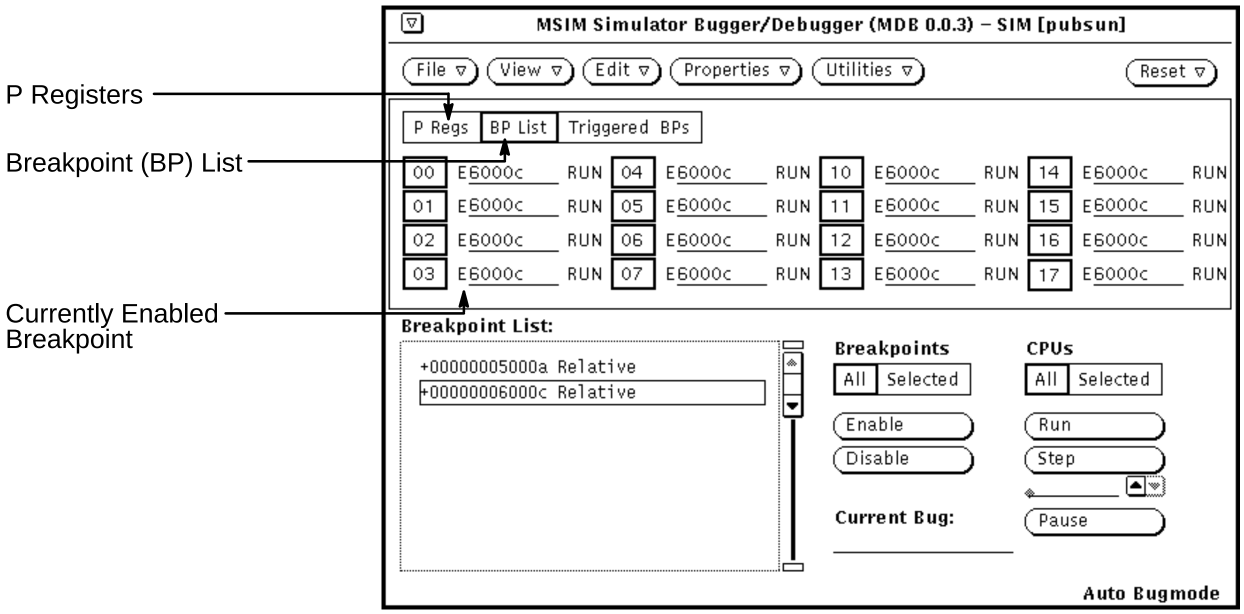


Figure 8-8. All or Selected Breakpoints

In this example, the MSIM displays the All `Breakpoints` in bold on a monochrome display or as pressed in on a color display.

Displaying P Registers

To display the P registers, click on P Regs. Refer to Figure 8-9.

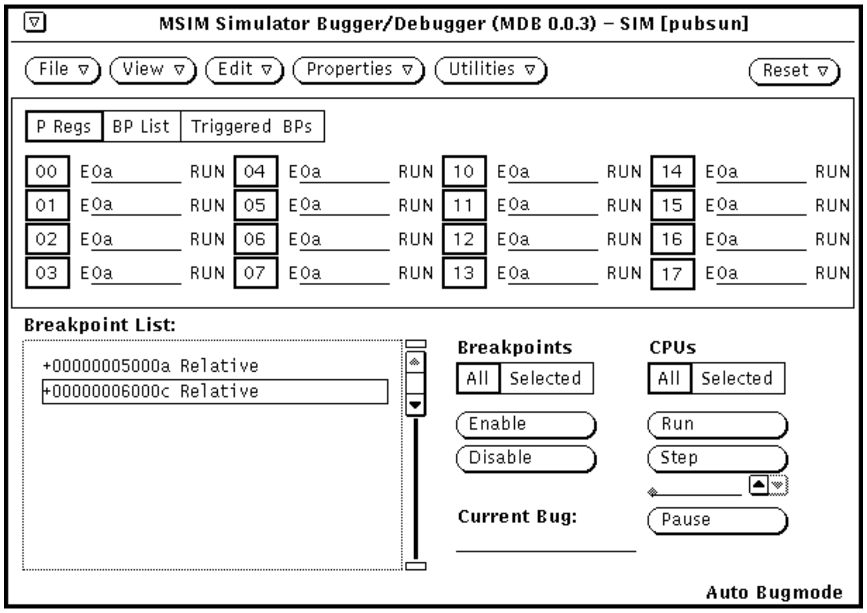


00	E6000c	RUN	04	E6000c	RUN	10	E6000c	RUN	14	E6000c	RUN
01	E6000c	RUN	05	E6000c	RUN	11	E6000c	RUN	15	E6000c	RUN
02	E6000c	RUN	06	E6000c	RUN	12	E6000c	RUN	16	E6000c	RUN
03	E6000c	RUN	07	E6000c	RUN	13	E6000c	RUN	17	E6000c	RUN

```
+00000005000a Relative
+00000006000c Relative
```

Figure 8-9. Displaying a P Register

The P register address appears. Refer to Figure 8-10. Notice that the corresponding P register location is to the right of each CPU 00 through 17.



```
00 E0a RUN   04 E0a RUN   10 E0a RUN   14 E0a RUN
01 E0a RUN   05 E0a RUN   11 E0a RUN   15 E0a RUN
02 E0a RUN   06 E0a RUN   12 E0a RUN   16 E0a RUN
03 E0a RUN   07 E0a RUN   13 E0a RUN   17 E0a RUN
```

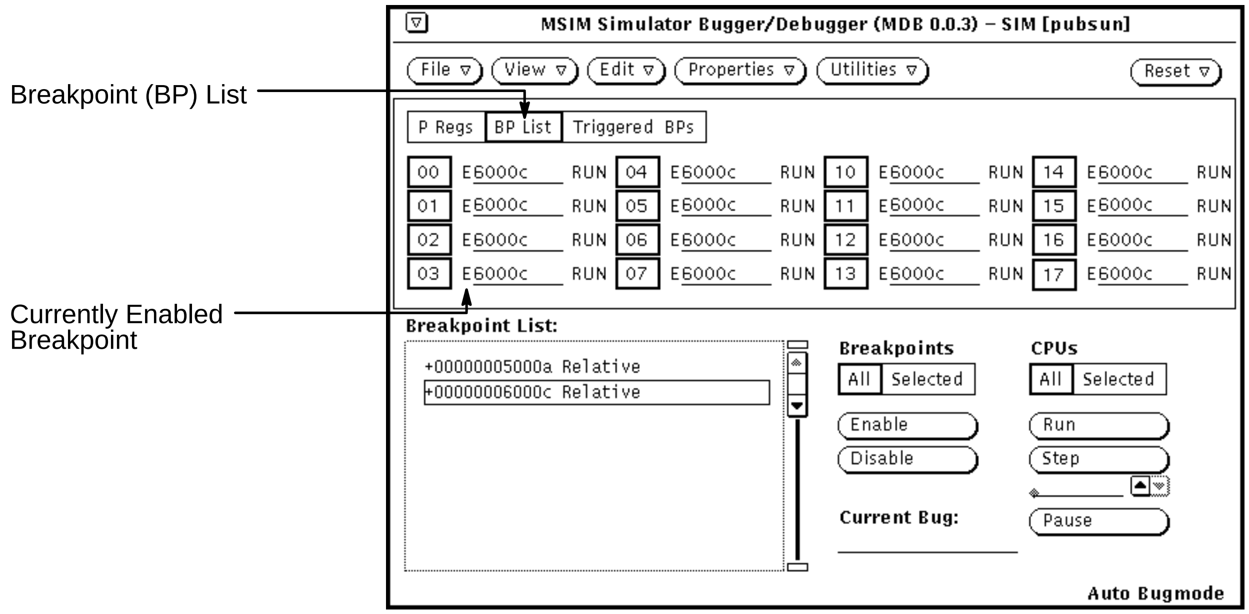
```
+00000005000a Relative
+00000006000c Relative
```

Figure 8-10. P Register Displayed

Displaying the Breakpoint List

To display the breakpoint list, click on BP List. The breakpoint list appears. Refer to Figure 8-11. Notice that the breakpoint location is shown to the right of each CPU 00 through 17.

Selecting the individual breakpoints listed in the `Breakpoint List` enables you to quickly see which CPUs are assigned to each breakpoint.



```
00 E6000c RUN   04 E6000c RUN   10 E6000c RUN   14 E6000c RUN
01 E6000c RUN   05 E6000c RUN   11 E6000c RUN   15 E6000c RUN
02 E6000c RUN   06 E6000c RUN   12 E6000c RUN   16 E6000c RUN
03 E6000c RUN   07 E6000c RUN   13 E6000c RUN   17 E6000c RUN
```

Breakpoint List:

```
+00000005000a Relative
+00000006000c Relative
```

Figure 8-11. Displayed Breakpoint List

Displaying Triggered Breakpoints

To display triggered breakpoints, click on Triggered BPs. The P register values for all CPUs that have reached a breakpoint are displayed to the right of the CPUs (00 through 17). Note that several or no breakpoints may appear. When Triggered BPs is selected, the fields to the right of the CPU 00 through 17 indicators are not tied to the breakpoint list: this enables you to see a broader view of the breakpoint status across all CPUs, rather than for one particular breakpoint selected from the `Breakpoint List`. The breakpoint status field (containing `E`, `D`, or `B`) always refers to the currently selected breakpoint in the breakpoint list, regardless of the setting of P Regs, BP List, and Triggered BPs. This ensures there is always a form of visual feedback to help you see how the breakpoint list and CPUs are related.

Pausing CPUs

Before performing this procedure, ensure that All or Selected `CPUs` is in the proper position.

To pause selected CPUs or all CPUs, click on Pause. A pause occurs for each CPU or all CPUs (depending on whether Selected or All is specified for the `CPUs`). In this example, All causes the CPUs to stop running. When `PAU` is shown to the right of a CPU and its breakpoint, it indicates the CPU has stopped running (no instructions are being issued for the CPU).

Running Selected CPUs or All CPUs

Before performing this procedure, ensure that All or Selected `CPUs` is in the proper position.

To run selected CPUs or all CPUs, click on Run. The specified CPU(s) (All or Selected) begin issuing instructions until one of the following things occurs:

- A breakpoint is reached
- You click Pause to pause the CPU(s)
- You click Step to step the CPU(s)
- The simulated mainframe is master cleared by MME

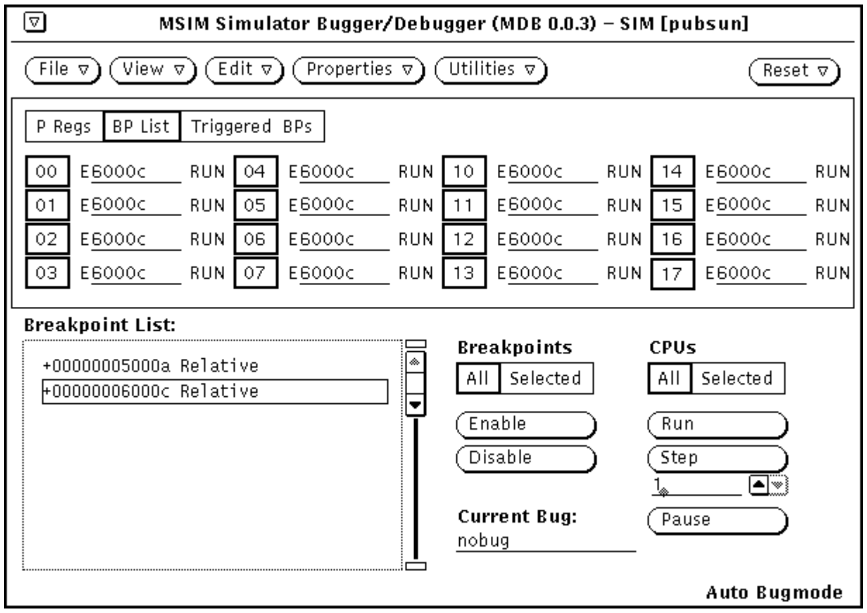
Using Step Mode to Run a CPU or All CPUs

Before starting this procedure, ensure that you have selected one or more CPUs and that All or Selected `CPUs` is in the proper position.

To use step mode to run a selected CPU or all CPUs, perform the following procedure:

1. Click on ▼ to decrease the value or ▲ to increase the value in the `Step` field.

2. Click on Step. The selected CPU or all CPUs are now in step mode (depending on whether Selected or All is specified for the `CPUs` selection). Refer to Figure 8-12. The CPU(s) in step mode issue the specified number of instructions (unless a breakpoint is reached) and then pause.



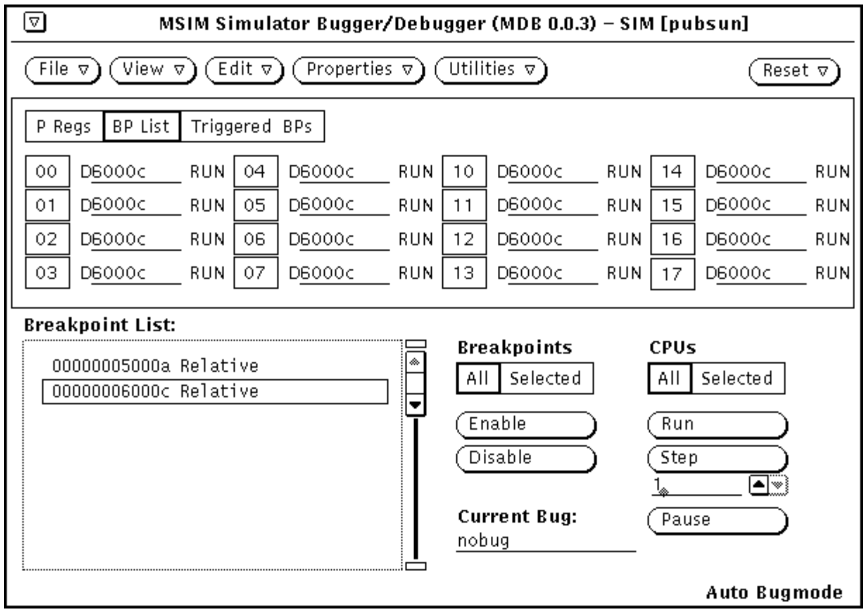
```
00 E6000c RUN  04 E6000c RUN  10 E6000c RUN  14 E6000c RUN
01 E6000c RUN  05 E6000c RUN  11 E6000c RUN  15 E6000c RUN
02 E6000c RUN  06 E6000c RUN  12 E6000c RUN  16 E6000c RUN
03 E6000c RUN  07 E6000c RUN  13 E6000c RUN  17 E6000c RUN
```

```
+00000005000a Relative
+00000006000c Relative
```

Figure 8-12. Step Mode Used to Run a Breakpoint

Disabling One or More Breakpoints

When one or more CPUs and breakpoints are enabled and Selected or All `Breakpoints` are in the proper position, click on Disable. The breakpoints are disabled as determined by their Selected or All settings. Refer to Figure 8-13.



```
00  D6000c  RUN   04  D6000c  RUN   10  D6000c  RUN   14  D6000c  RUN
01  D6000c  RUN   05  D6000c  RUN   11  D6000c  RUN   15  D6000c  RUN
02  D6000c  RUN   06  D6000c  RUN   12  D6000c  RUN   16  D6000c  RUN
03  D6000c  RUN   07  D6000c  RUN   13  D6000c  RUN   17  D6000c  RUN
```

Breakpoint List:

```
00000005000a Relative
00000006000c Relative
```

Step: 1

Current Bug: nobug

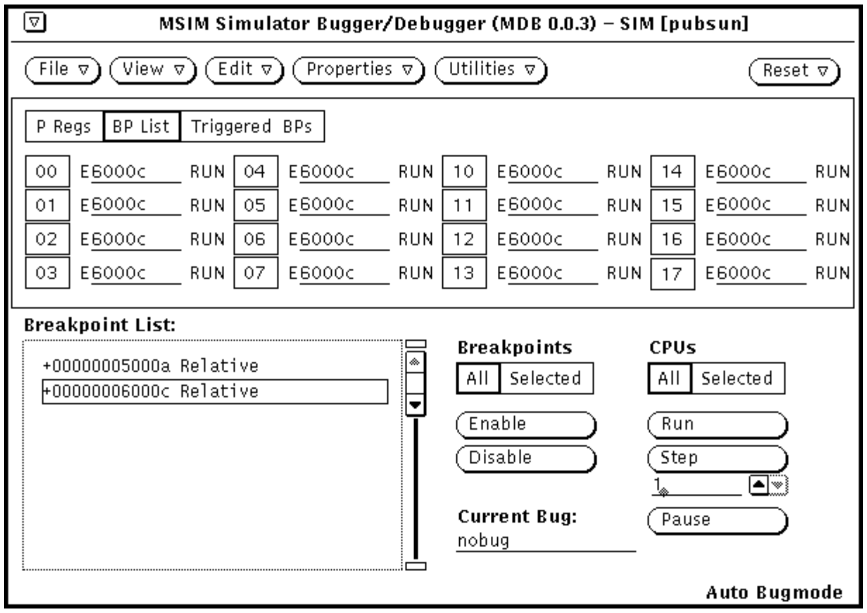
Figure 8-13. Disabled Breakpoints

NOTE: An enabled breakpoint has an `E` in the first position of the status column, followed by its address. A disabled breakpoint has a `D` in the first position of the status column, followed by its address. A triggered breakpoint has a `B` in the first position of the status column, followed by its address. A blank in the status column indicates the CPU is not assigned to a breakpoint.

Enabling One or More Breakpoints

NOTE: Before starting this procedure, ensure that you have selected one or more CPUs and that the Selected or All `Breakpoint` setting is in the proper position. Refer to "Using All or Selected Breakpoints" earlier in this section.

When one or more CPUs and breakpoints are disabled, click on Enable. The CPUs and breakpoints are enabled as determined by their Selected or All settings. Refer to Figure 8-14.



00	E6000c	RUN	04	E6000c	RUN	10	E6000c	RUN	14	E6000c	RUN
01	E6000c	RUN	05	E6000c	RUN	11	E6000c	RUN	15	E6000c	RUN
02	E6000c	RUN	06	E6000c	RUN	12	E6000c	RUN	16	E6000c	RUN
03	E6000c	RUN	07	E6000c	RUN	13	E6000c	RUN	17	E6000c	RUN

Breakpoint List:

```
+00000005000a Relative
+00000006000c Relative
```

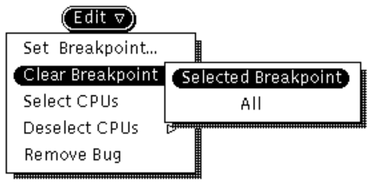
Step: 1

Current Bug: nobug

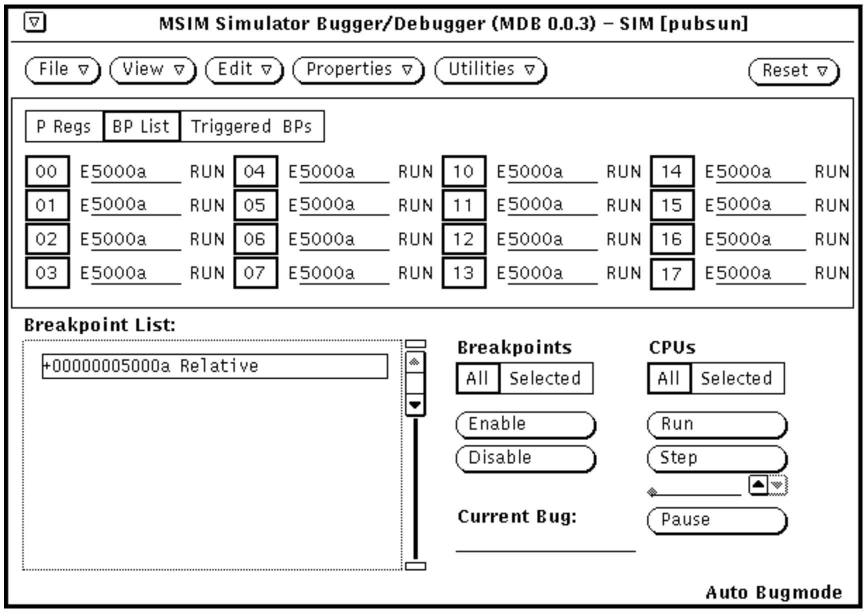
Figure 8-14. Enabled Breakpoints

NOTE: An enabled breakpoint has an `E` in the first position of the status column, followed by its address. A disabled breakpoint has a `D` in the first position of the status column, followed by its address. A triggered breakpoint has a `B` in the first position of the status column, followed by its address. A blank in the status column indicates the CPU is not assigned to a breakpoint.

Clearing Selected Breakpoints



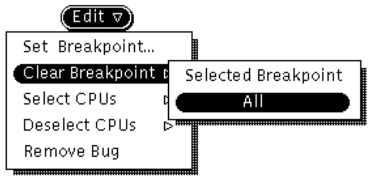
To clear selected breakpoints, choose **`Edit --> Clear Breakpoint --> Selected Breakpoint`**, as shown at the left. The selected breakpoints are cleared:



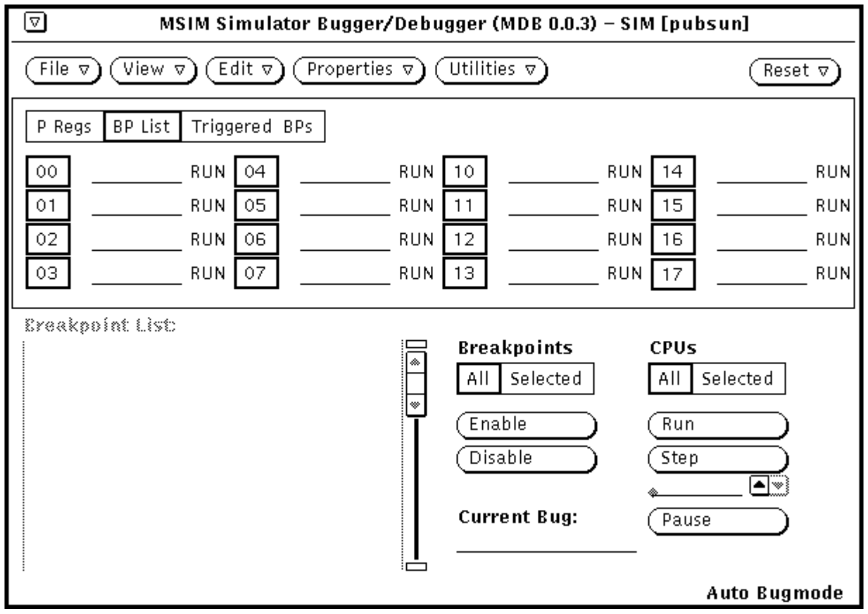
```
00 E5000a RUN  04 E5000a RUN  10 E5000a RUN  14 E5000a RUN
01 E5000a RUN  05 E5000a RUN  11 E5000a RUN  15 E5000a RUN
02 E5000a RUN  06 E5000a RUN  12 E5000a RUN  16 E5000a RUN
03 E5000a RUN  07 E5000a RUN  13 E5000a RUN  17 E5000a RUN
```

```
+00000005000a Relative
```

Clearing All Breakpoints



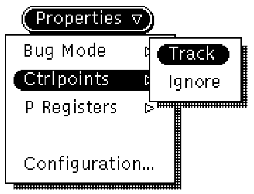
To clear all breakpoints, choose the **`Edit --> Clear Breakpoint --> All`**, as shown at the left. All the breakpoints are cleared:



Using Control Points in the MSIM

Using the MSIM, you can track or ignore control points you defined in the `Mainframe Maintenance Environment` window.

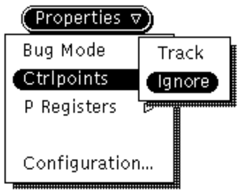
Tracking Control Points



You can track MME control points while you manipulate CPUs and breakpoints. To track control points, choose **`Properties --> Ctrlpoints --> Track`**, as shown at the left.

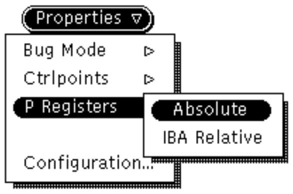
`N/A` appears to the right of each CPU that is not assigned to the current control point in MME. Selecting a control point in MME enables you to concentrate on the CPUs assigned to the control point.

Ignoring Control Point Tracking



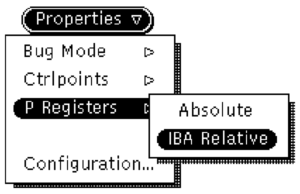
To ignore control point tracking, choose **`Properties --> Ctrlpoints --> Ignore`**, as shown at the left. All CPUs become available for manipulation by MDB, regardless of CPU assignments to MME control points (if any exist).

Displaying P Registers in Absolute Format



You can display the memory locations from program address (P) registers in either absolute or instruction base address (IBA) format. To display addresses from P registers in absolute format, choose **`Properties --> P Registers --> Absolute`**, as shown at the left. The addresses from the P registers appear in absolute format.

Displaying P Registers in Relative Instruction Base Address Format

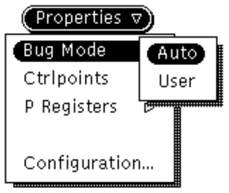


You can display the memory locations from program address (P) registers in either absolute or instruction base address (IBA) format. To display the addresses from P registers in IBA format, choose **`Properties --> P Registers --> IBA`**, as shown at the left. The addresses from the P registers appear in instruction base address format.

Using a Bug Mode

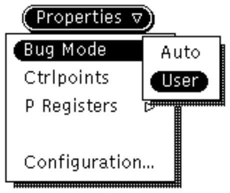
Using bug mode, you can specify whether you want to use the defaults the system supplies in automatic bug mode or whether you want the bugs to run using your own user bug mode.

Using Auto Bug Mode



You can use automatic (`Auto`) bug mode to have MSIM specify the way you want bugs to be loaded and run. To use `Auto` bug mode, choose **`Properties --> Bug Mode --> Auto`**, as shown at the left. `Auto` bug mode is enabled.

Using User Bug Mode



You can select user bug mode to have more control over how the bugs are loaded. To use `User` bug mode, choose **`Properties --> Bug Mode --> User`**, as shown at the left. `User` bug mode is enabled.

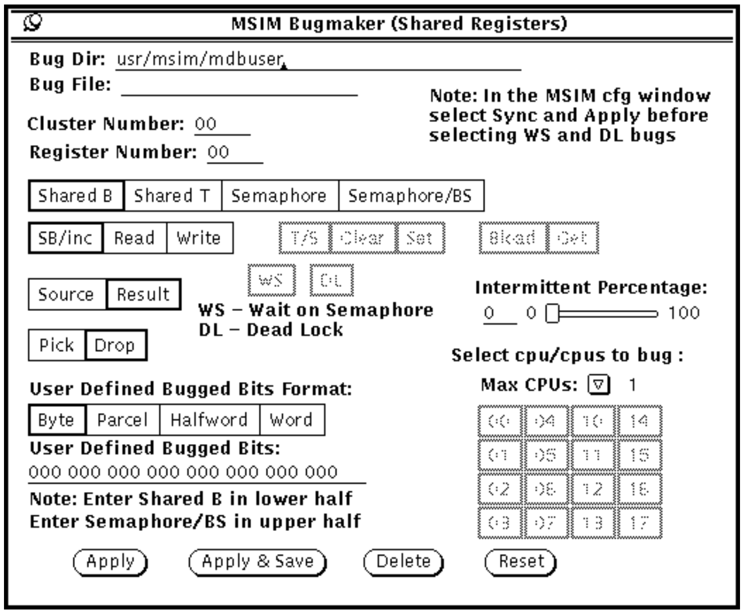
Creating User Bugs

You can create user bugs using the bugmaker utility in user mode. Currently, you can create shared register bugs only.

Creating Shared Register Bugs

To create shared register bugs, you must use the bugmaker utility. To use `Bugmaker`, choose **`Utilities --> Bugmaker`**. The `MSIM Bugmaker (Shared Registers)` window appears.

NOTE: To access the bugmaker utility, the bugger/debugger program must be in user bug mode. For more information on user bug mode, refer to the previous subsection “Using User Bug Mode.”


MSIM Bugmaker (Shared Registers)

Bug Dir: usr/msim/mdbuser

Bug File:

Cluster Number: 00

Register Number: 00

Note: In the MSIM cfg window select Sync and Apply before selecting WS and DL bugs

WS – Wait on Semaphore
DL – Dead Lock

Intermittent Percentage: 0 0 100

Select cpu/cpus to bug :

Max CPUs: 1

User Defined Bugged Bits Format:

User Defined Bugged Bits:
000 000 000 000 000 000 000 000

Note: Enter Shared B in lower half
Enter Semaphore/BS in upper half

You can use this window to create a shared B register or shared T register bug, a semaphore bug, or a semaphore/broadside bug.

Creating a Shared B Register or Shared T Register Bug

To create a shared B register or shared T register bug, perform the following procedure:

1. In the `Bug File` field, enter a name for the bug you are creating.

2. In the `Cluster Number` field, enter the cluster number for the shared register cluster you want to bug. (Use an octal number 0 through 21.)

3. In the `Register Number` field, enter the shared B or shared T register number you want to bug.

4. Click on Shared B or Shared T to specify a shared B or shared T bug.

5. Click on SB/inc, Read, or Write to select the type of function you want to bug.

NOTE: Currently, the Source and Result options do not work.

6. Click on Pick or Drop to pick or drop a bit.

7. Click on Byte, Parcel, Halfword, or Word to specify the format for the bit mask of bits you want to bug.

8. In the `User Defined Bugged Bits` field, enter a bit mask indicating the bits you want to bug.

NOTE: Enter shared B register bugs in the lower half of the bit mask.

9. In the `Intermittent Percentage` field, use the slide bar or enter the percentage of time you want the bug to be intermittent. This value specifies how often the bug will occur.

10. Select the maximum number of available CPUs from the `Max CPUS:` ▽.

11. Click on the CPUs you want to bug.

12. To apply the bug without saving it, click on Apply ▽. To apply and save the bug, click on Save ▽. If there is a bug saved with the same name, a popup message appears that asks if you want to overwrite the bug that has the same name.

NOTE: To delete the bug selected in the bug file, click on Delete. To reset the window to the last loaded bug, click on Reset.

Creating a Semaphore Bug

To create a semaphore bug, follow the following procedure:

1. Click on Semaphore.

2. Click on T/S, Clear, or Set to specify the type of function you want to bug. (T/S represents a test and set instruction bug.)

3. If you selected T/S, click on WS (wait on semaphore) or DL (dead lock).

NOTE: Currently, the Source and Result options do not work.

4. Click on Pick or Drop to pick or drop a bit.

5. Click on Byte, Parcel, Halfword, or Word to specify the format for the bit mask.

6. In the `User Defined Bugged Bits` field, enter a bit mask indicating the bits you want to bug.

7. In the `Intermittent Percentage` field, use the slide bar or enter the percentage of time you want the bug to be intermittent. This value specifies how often the bug will occur.

8. Select the maximum number of available CPUs from the `Max CPUS:` ▽ menu.

9. Click on the CPUs you want to bug.

10. To apply the bug without saving it, click on (Apply ▽). To apply and save the bug, click on (Save ▽). If there is a bug saved with the same name, a popup message appears that asks if you want to overwrite the bug that has the same name.

NOTE: To delete the bug selected in the bug file, click on (Delete). To reset the window to the last loaded bug, click on (Reset).

Creating a Semaphore/Broadside Bug

To create a semaphore/broadside bug, perform the following procedure:

1. Click on Semaphore/BS.

2. Click on BLoad or Get to specify that you want to bug a broadside load or a get function.

NOTE: Currently, the Source and Result options do not work.

3. Click on Pick or Drop to pick or drop a bit.

4. Click on Byte, Parcel, Halfword, or Word to specify the format for the bit mask.

5. In the `User Defined Bugged Bits` field, enter a bit mask indicating the bits you want to bug.

6. In the `Intermittent Percentage` field, use the slide bar or enter the percentage of time you want the bug to be intermittent. This value specifies how often the bug will occur.

7. Select the maximum number of available CPUs from the `Max CPUS:` ▽.

8. Click on the CPUs you want to bug.

9. To apply the bug without saving it, click on (Apply ▽). To apply and save the bug, click on (Save ▽). If there is a bug saved with the same name, a popup message appears that asks if you want to overwrite the bug that has the same name.

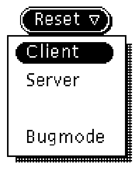
NOTE: To delete the bug selected in the bug file, click on (Delete). To reset the window to the last loaded bug, click on (Reset).

Resetting MDB

If you find problems with the debugger software or you want to delete the breakpoint and reset the CPUs, you can reset the MSIM software in one of the following ways:

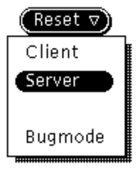
- Reset the MDB client
- Reset the MDB server
- Reset the bug mode

Resetting the MDB Client



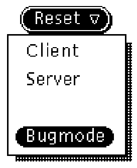
Choose **`Reset --> Client`**, as shown at the left, to reset the client portion of the MDB application.

Resetting the MDB Server



Choose **`Reset --> Server`**, as shown at the left, to reset the server portion of the MDB application.

Resetting Bug Mode



If you find an error from which you cannot recover, you can reset bug mode. To reset bug mode, choose **`Reset --> Bugmode`**, as shown at the left. This resets bug mode. Any CPUs and breakpoints you selected or bugs you loaded are not lost. You do not need to specify whether you want to track control points.

9 TROUBLESHOOTING

This section describes how to create a loop for troubleshooting a hardware problem. In this section, you are shown how to load a bug and troubleshoot it using the mainframe simulator (MSIM).

Creating a Loop for Troubleshooting

When you are attempting to identify a problem in a specific field-replaceable unit, you can create a loop to isolate a problem to a register, bit, or functional unit. While this is not required of field engineers, STCO personnel routinely perform loop operations in order to isolate a problem to a register, bit, or functional unit. To create a loop, you can either change an existing control point or modify a template. Two templates are provided: `diag.c` and `loop.c`.

`Diag.c` is the template used for all diagnostic programs. However, `diag.c` has no code in the main block and runs in multiple CPUs. `Diag.c` has all the standard locations, exchange package tables, routers, and handlers that all diagnostics have. It runs in either environment 1 or environment 2.

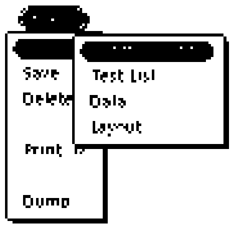
`Loop.c` is a type of program known to MME as a loop. Loop-type programs are handled differently by MME than diag-type programs. A loop is not configured by MME. It runs only in a single CPU and runs only in environment 1. However, you can load multiple loops in environment 1, whereas you can load only one diag-type program in environment 1.

MME does not configure loop-type programs because of the loop layout. For example, a loop has no standard locations to set up and no working exchange package table to configure. This loop layout allows you to write extremely small code segments. However, it also requires you to set up every aspect of the code.

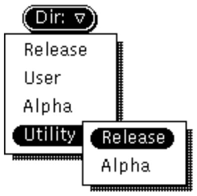
The advantage of loops is that any number of them can be loaded anywhere in memory, and they can be set up to test anything you choose. The disadvantage is that loops are very rigid. Once they are set up to run at one memory location, they must be changed in order to run at another location. That is, you must set up the base, limit, and exchange addresses correctly in order for the loop to run.

In this example, you are shown how to create your own loop using the `diag.c` template in environment 2. To create a loop in mainframe memory, perform the following procedure:

1. Load the `diag.c` template in environment 2: perform the following steps.

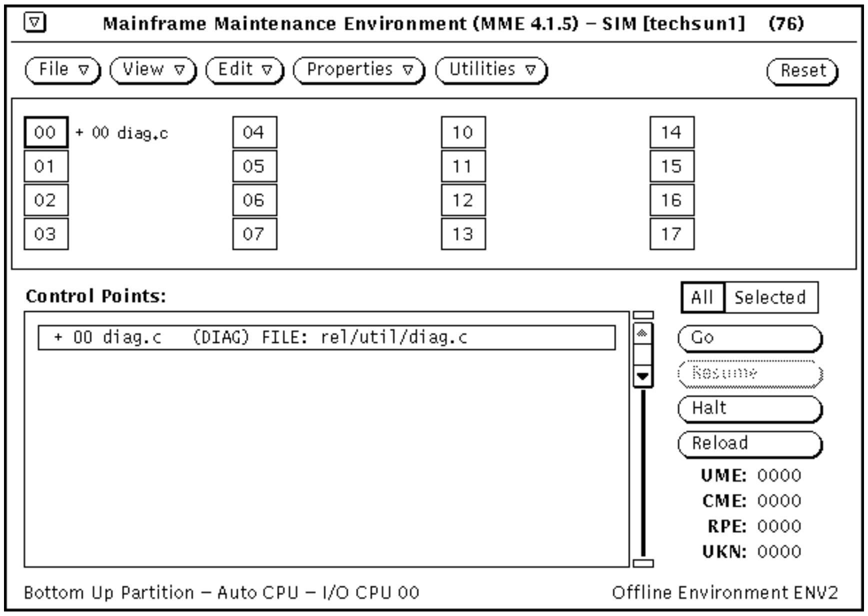


Step 1: Choose **`File --> Load --> Control Point`** in the `Mainframe Maintenance Environment` window, as shown at the left.



Step 2: Choose **`Dir --> Utility --> Release`** in the `MME Load Control Point` window, as shown at the left, to access the directory where `diag.c` is located.

Step 3: Click on `diag.c` in the `Files` scroll box in the `MME Load Control Point` window, and click on . The `diag.c` control point is loaded in environment 2; refer to Figure 9-1.

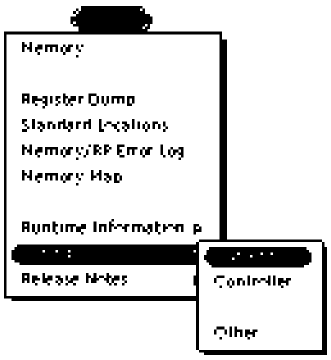


```
+ 00 diag.c   (DIAG) FILE: rel/util/diag.c
```

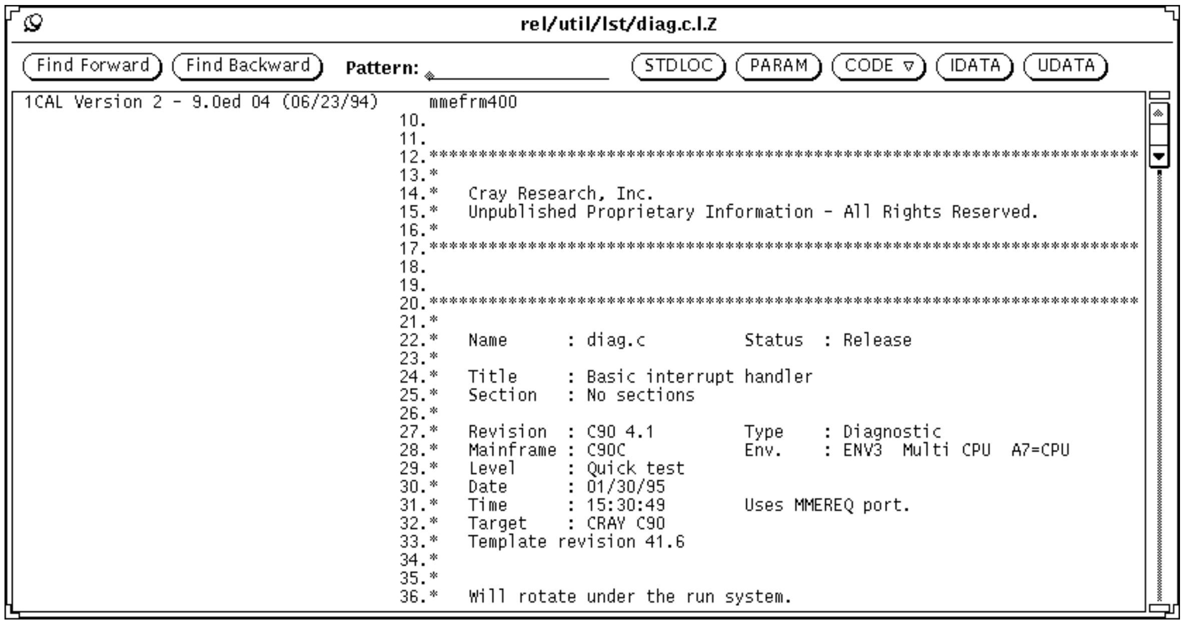
```
UME: 0000
CME: 0000
RPE: 0000
UKN: 0000
Bottom Up Partition – Auto CPU – I/O CPU 00          Offline Environment ENV2
```

Figure 9-1. Using diag.c in Environment 2

Now that you have the `diag.c` template loaded, you need to determine where to start writing your loop code. To do this, view the listing and locate the `MAIN` portion of the code, as described in the following steps.

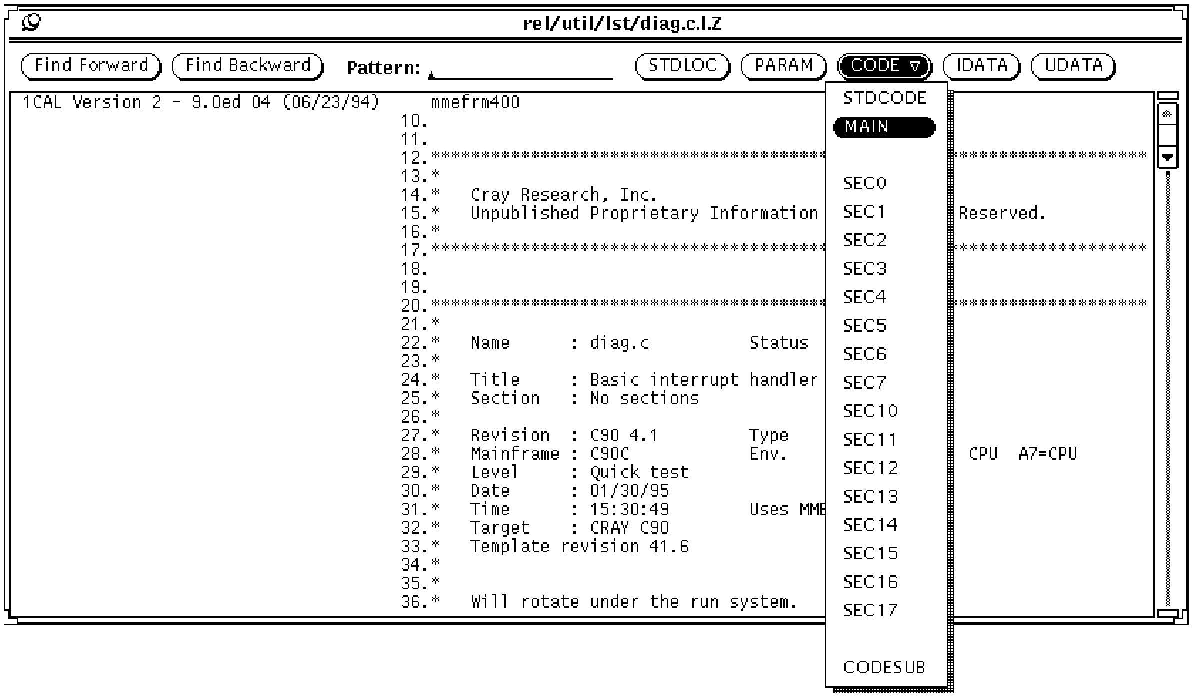


2. In the `Mainframe Maintenance Environment` window, choose **`View --> Listing --> Current`**, as shown at the left. MME displays the `diag.c` utility listing.



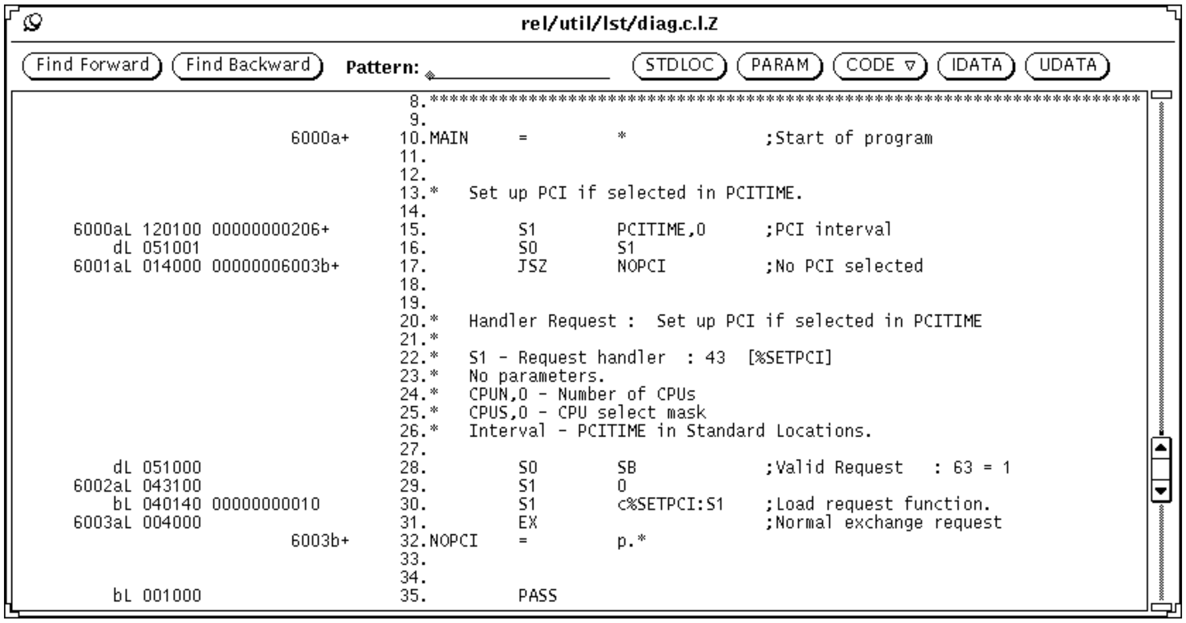
```
1CAL Version 2 - 9.0ed 04 (06/23/94)     mmefrm400


*************************************************************************
*
*   Cray Research, Inc.
*   Unpublished Proprietary Information - All Rights Reserved.
*
*************************************************************************


*************************************************************************
*
*   Name      : diag.c          Status  : Release
*
*   Title     : Basic interrupt handler
*   Section   : No sections
*
*   Revision  : C90 4.1         Type    : Diagnostic
*   Mainframe : C90C            Env.    : ENV3  Multi CPU  A7=CPU
*   Level     : Quick test
*   Date      : 01/30/95
*   Time      : 15:30:49        Uses MMEREQ port.
*   Target    : CRAY C90
*   Template revision 41.6
*
*
*   Will rotate under the run system.
```

3. Choose **`CODE --> MAIN`**, as shown in Figure 9-2. Figure 9-3 shows that the code is located at address 6000.



```
1CAL Version 2 - 9.0ed 04 (06/23/94)     mmefrm400


                                          ****************************************            ********************
                                          *
                                          *   Cray Research, Inc.
                                          *   Unpublished Proprietary Information             Reserved.
                                          *
                                          ****************************************            ********************


                                          ****************************************            ********************
                                          *
                                          *   Name      : diag.c         Status
                                          *
                                          *   Title     : Basic interrupt handler
                                          *   Section   : No sections
                                          *
                                          *   Revision  : C90 4.1        Type
                                          *   Mainframe : C90C           Env.                   CPU   A7=CPU
                                          *   Level     : Quick test
                                          *   Date      : 01/30/95
                                          *   Time      : 15:30:49       Uses MME
                                          *   Target    : CRAY C90
                                          *   Template revision 41.6
                                          *
                                          *
                                          *   Will rotate under the run system.
```

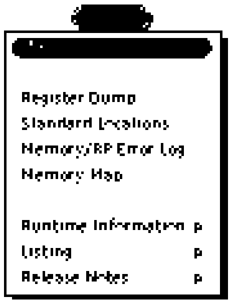
Figure 9-2. CODE Menu Button for Displaying the Loop Code



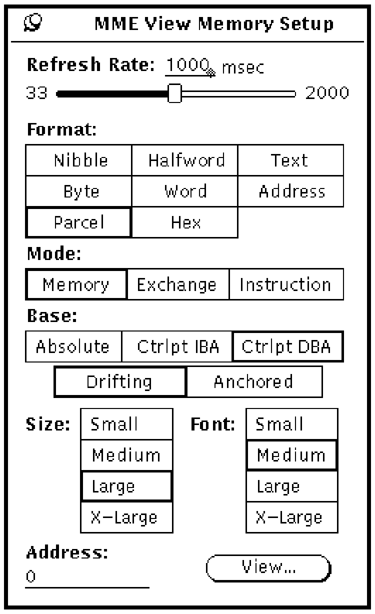
```
                                          ************************************************************************

                         6000a+           MAIN     =         *             ;Start of program


                                          *   Set up PCI if selected in PCITIME.

     6000aL 120100 00000000206+                    S1        PCITIME,0     ;PCI interval
         dL 051001                                 S0        S1
     6001aL 014000 00000006003b+                   JSZ       NOPCI         ;No PCI selected


                                          *   Handler Request :  Set up PCI if selected in PCITIME
                                          *
                                          *   S1 - Request handler  : 43  [%SETPCI]
                                          *   No parameters.
                                          *   CPUN,0 - Number of CPUs
                                          *   CPUS,0 - CPU select mask
                                          *   Interval - PCITIME in Standard Locations.

         dL 051000                                 S0        SB            ;Valid Request   : 63 = 1
     6002aL 043100                                 S1        0
         bL 040140 00000000010                     S1        c%SETPCI:S1   ;Load request function.
     6003aL 004000                                 EX                      ;Normal exchange request
                         6003b+           NOPCI    =         p.*


         bL 001000                                 PASS
```

Figure 9-3. Current diag.c Listing Showing Line 6000

Before you begin to modify the template, you should view the working exchange package (WEXP) for CPU 0. This enables you to watch what happens as your loop runs. The WEXP for CPU 0 is located at address 2000; the following steps describe how to view the WEXP for CPU 0.



4. In the `Mainframe Maintenance Environment` window, choose **`View --> Memory`**, as shown at the left. MME displays the `MME View Memory Setup` window:



MME View Memory Setup

Refresh Rate: 1000 msec (33 – 2000)

Format: Nibble | Halfword | Text | Byte | Word | Address | [✓ Parcel] | Hex

Mode: [✓ Memory] | Exchange | Instruction

Base: Absolute | Ctrlpt IBA | [✓ Ctrlpt DBA]

[✓ Drifting] | Anchored

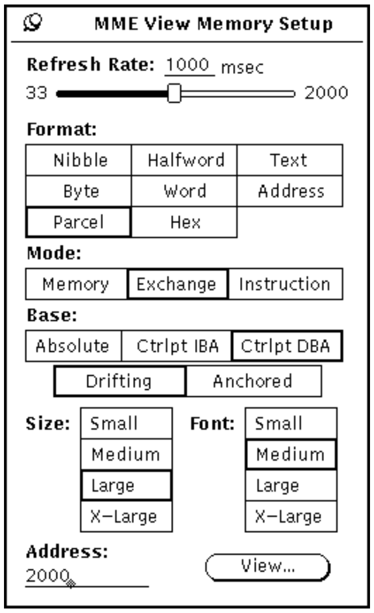
Size: Small | Medium | [✓ Large] | X-Large

Font: Small | [✓ Medium] | Large | X-Large

Address: 0

View...

5. To view memory in exchange package mode, click on Exchange mode and click on Ctrlpt DBA. Double click on the `Address` field and type 2000. Figure 9-4 shows the `MME View Memory Setup` window you use.



MME View Memory Setup

Refresh Rate: 1000 msec (33 – 2000)

Format: Nibble Halfword Text Byte Word Address [✓ Parcel] Hex

Mode: Memory [✓ Exchange] Instruction

Base: Absolute Ctrlpt IBA [✓ Ctrlpt DBA]

[✓ Drifting] Anchored

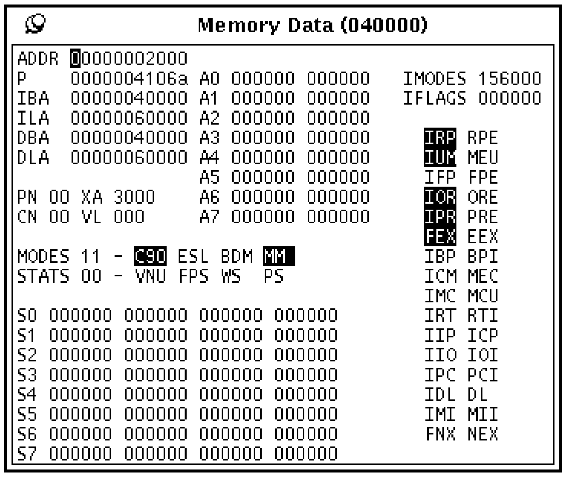
Size: Small Medium [✓ Large] X–Large

Font: Small [✓ Medium] Large X–Large

Address: 2000

Figure 9-4. MME View Memory Setup Window for an Exchange Package

6. Click on View... . MME displays the relative memory `(Memory Data)` window:



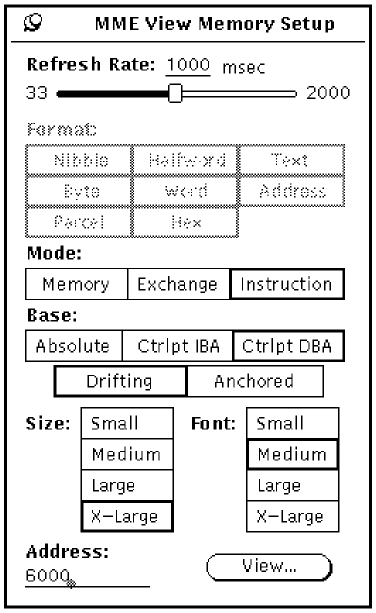
Memory Data (040000)

```
ADDR 00000002000
P    0000004106a A0 000000 000000   IMODES 156000
IBA  00000040000 A1 000000 000000   IFLAGS 000000
ILA  00000060000 A2 000000 000000
DBA  00000040000 A3 000000 000000     IRP RPE
DLA  00000060000 A4 000000 000000     IUM MEU
                 A5 000000 000000     IFP FPE
PN 00 XA 3000    A6 000000 000000     IOR ORE
CN 00 VL 000     A7 000000 000000     IPR PRE
                                      FEX EEX
MODES 11 - C90 ESL BDM MM             IBP BPI
STATS 00 - VNU FPS WS  PS             ICM MEC
                                      IMC MCU
S0 000000 000000 000000 000000        IRT RTI
S1 000000 000000 000000 000000        IIP ICP
S2 000000 000000 000000 000000        IIO IOI
S3 000000 000000 000000 000000        IPC PCI
S4 000000 000000 000000 000000        IDL DL
S5 000000 000000 000000 000000        IMI MII
S6 000000 000000 000000 000000        FNX NEX
S7 000000 000000 000000 000000
```

This is the WEXP for CPU 0 relative memory window.

Remember that the code you want to modify begins at address 6000. To view the instructions starting at this address, perform the following step.

7. In the `MME View Memory Setup` window, click on Instruction to view the `diag.c` code in instruction mode, and click on Ctrlpt DBA. Click on `Size:` X-Large. Double click on the `Address` field, and type 6000. Refer to the `MME View Memory Setup` window shown in Figure 9-5.



MME View Memory Setup

Refresh Rate: 1000 msec

33 2000

Format:

Nibble Halfword Text
Byte Word Address
Parcel Hex

Mode: Memory Exchange [✓ Instruction]

Base: Absolute Ctrlpt IBA [✓ Ctrlpt DBA]

[✓ Drifting] Anchored

Size: Small Medium Large [✓ X-Large]

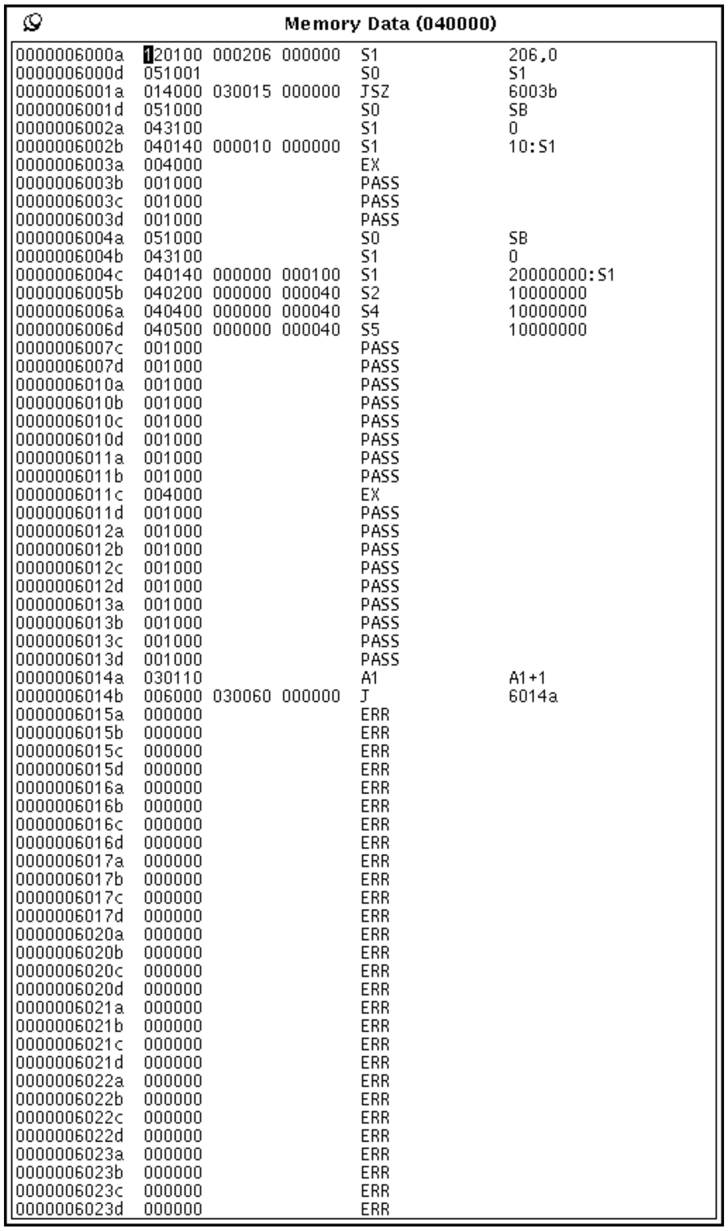
Font: Small [✓ Medium] Large X-Large

Address: 6000

View...

Figure 9-5. Viewing diag.c in Instruction Mode

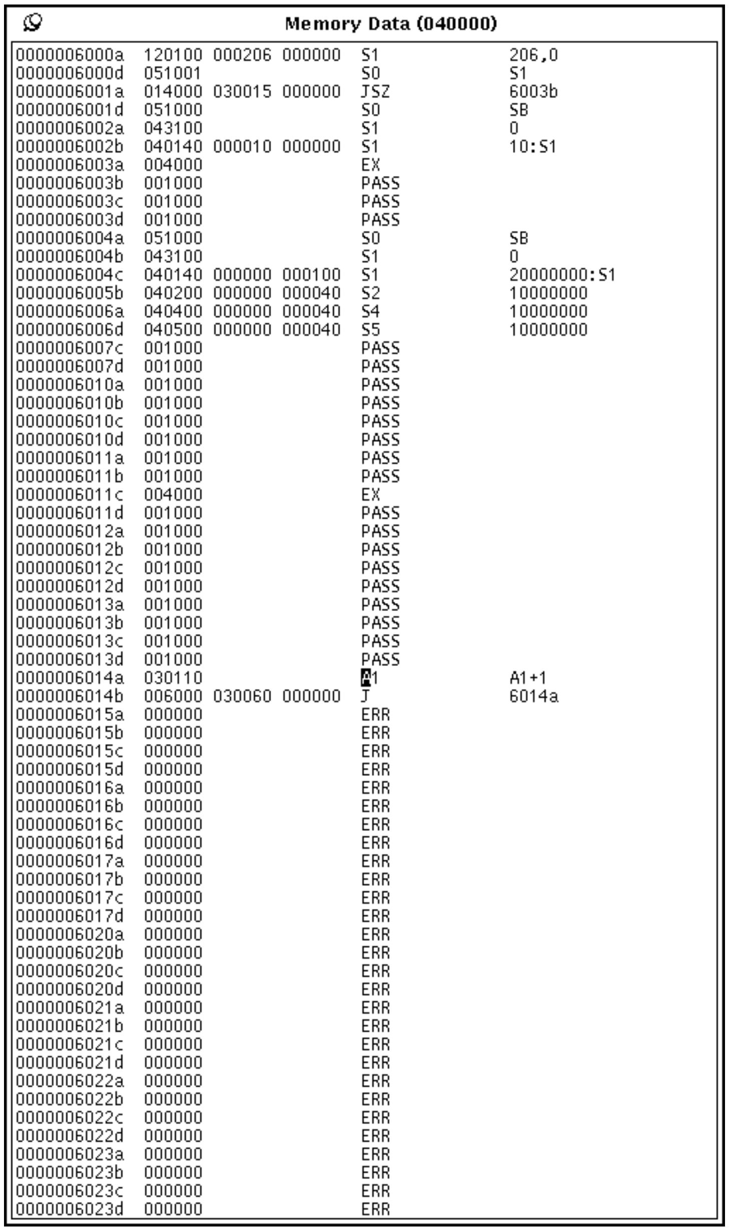
8. Click on View... . MME displays the `Memory Data` window at address 6000:



```
Memory Data (040000)
0000006000a  120100 000206 000000  S1                  206,0
0000006000d  051001                S0                  S1
0000006001a  014000 030015 000000  JSZ                 6003b
0000006001d  051000                S0                  SB
0000006002a  043100                S1                  0
0000006002b  040140 000010 000000  S1                  10:S1
0000006003a  004000                EX
0000006003b  001000                PASS
0000006003c  001000                PASS
0000006003d  001000                PASS
0000006004a  051000                S0                  SB
0000006004b  043100                S1                  0
0000006004c  040140 000000 000100  S1                  20000000:S1
0000006005b  040200 000000 000040  S2                  10000000
0000006006a  040400 000000 000040  S4                  10000000
0000006006d  040500 000000 000040  S5                  10000000
0000006007c  001000                PASS
0000006007d  001000                PASS
0000006010a  001000                PASS
0000006010b  001000                PASS
0000006010c  001000                PASS
0000006010d  001000                PASS
0000006011a  001000                PASS
0000006011b  001000                PASS
0000006011c  004000                EX
0000006011d  001000                PASS
0000006012a  001000                PASS
0000006012b  001000                PASS
0000006012c  001000                PASS
0000006012d  001000                PASS
0000006013a  001000                PASS
0000006013b  001000                PASS
0000006013c  001000                PASS
0000006013d  001000                PASS
0000006014a  030110                A1                  A1+1
0000006014b  006000 030060 000000  J                   6014a
0000006015a  000000                ERR
0000006015b  000000                ERR
0000006015c  000000                ERR
0000006015d  000000                ERR
0000006016a  000000                ERR
0000006016b  000000                ERR
0000006016c  000000                ERR
0000006016d  000000                ERR
0000006017a  000000                ERR
0000006017b  000000                ERR
0000006017c  000000                ERR
0000006017d  000000                ERR
0000006020a  000000                ERR
0000006020b  000000                ERR
0000006020c  000000                ERR
0000006020d  000000                ERR
0000006021a  000000                ERR
0000006021b  000000                ERR
0000006021c  000000                ERR
0000006021d  000000                ERR
0000006022a  000000                ERR
0000006022b  000000                ERR
0000006022c  000000                ERR
0000006022d  000000                ERR
0000006023a  000000                ERR
0000006023b  000000                ERR
0000006023c  000000                ERR
0000006023d  000000                ERR
```

Now you can modify the individual instructions in the code. The following steps describe how to enter an example loop.

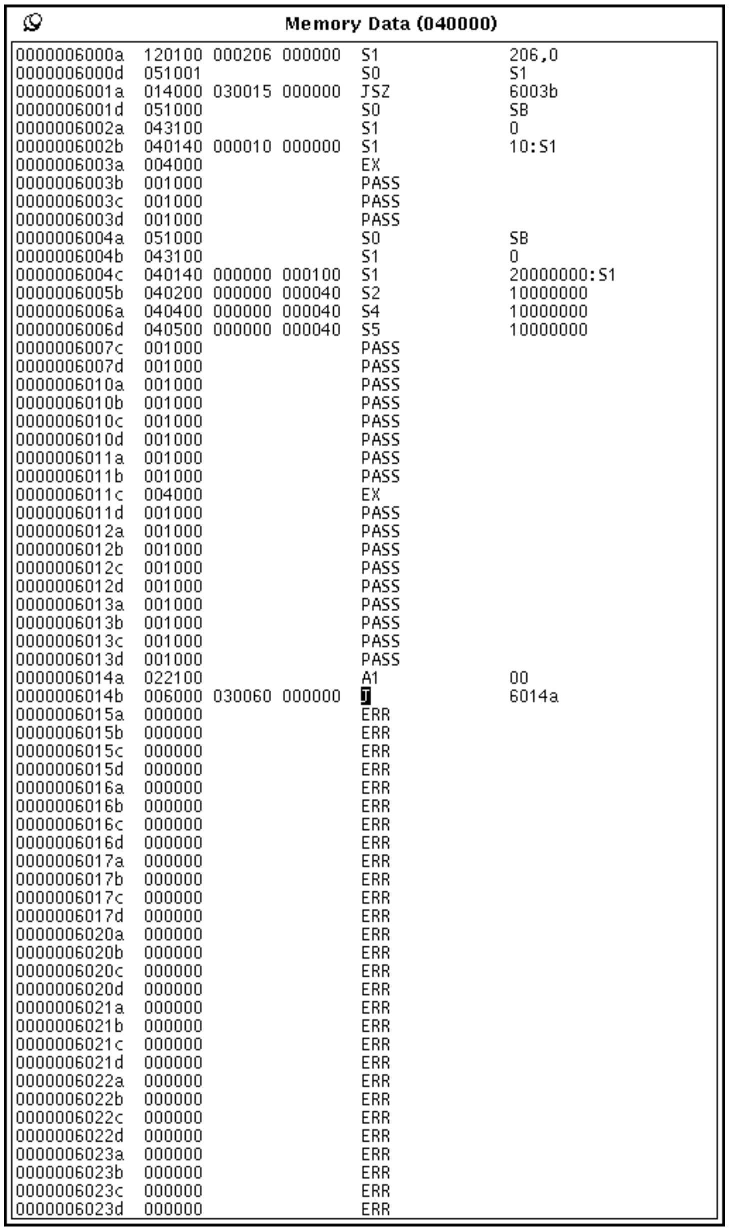
9. Click on `A1` at memory location 6014a. Refer to Figure 9-6.



```
Memory Data (040000)
0000006000a  120100 000206 000000  S1             206,0
0000006000d  051001                S0             S1
0000006001a  014000 030015 000000  JSZ            6003b
0000006001d  051000                S0             SB
0000006002a  043100                S1             0
0000006002b  040140 000010 000000  S1             10:S1
0000006003a  004000                EX
0000006003b  001000                PASS
0000006003c  001000                PASS
0000006003d  001000                PASS
0000006004a  051000                S0             SB
0000006004b  043100                S1             0
0000006004c  040140 000000 000100  S1             20000000:S1
0000006005b  040200 000000 000040  S2             10000000
0000006006a  040400 000000 000040  S4             10000000
0000006006d  040500 000000 000040  S5             10000000
0000006007c  001000                PASS
0000006007d  001000                PASS
0000006010a  001000                PASS
0000006010b  001000                PASS
0000006010c  001000                PASS
0000006010d  001000                PASS
0000006011a  001000                PASS
0000006011b  001000                PASS
0000006011c  004000                EX
0000006011d  001000                PASS
0000006012a  001000                PASS
0000006012b  001000                PASS
0000006012c  001000                PASS
0000006012d  001000                PASS
0000006013a  001000                PASS
0000006013b  001000                PASS
0000006013c  001000                PASS
0000006013d  001000                PASS
0000006014a  030110                A1             A1+1
0000006014b  006000 030060 000000  J              6014a
0000006015a  000000                ERR
0000006015b  000000                ERR
0000006015c  000000                ERR
0000006015d  000000                ERR
0000006016a  000000                ERR
0000006016b  000000                ERR
0000006016c  000000                ERR
0000006016d  000000                ERR
0000006017a  000000                ERR
0000006017b  000000                ERR
0000006017c  000000                ERR
0000006017d  000000                ERR
0000006020a  000000                ERR
0000006020b  000000                ERR
0000006020c  000000                ERR
0000006020d  000000                ERR
0000006021a  000000                ERR
0000006021b  000000                ERR
0000006021c  000000                ERR
0000006021d  000000                ERR
0000006022a  000000                ERR
0000006022b  000000                ERR
0000006022c  000000                ERR
0000006022d  000000                ERR
0000006023a  000000                ERR
0000006023b  000000                ERR
0000006023c  000000                ERR
0000006023d  000000                ERR
```

Figure 9-6. Moving Cursor to Address 6000 to Write a Loop

10. In memory location 6014a, type **`A1 0`** to initialize address register A1 to 0. Press and release the **`Control`** and **`Return`** keys simultaneously. This truncates the line at the cursor. Refer to Figure 9-7.



Memory Data (040000)

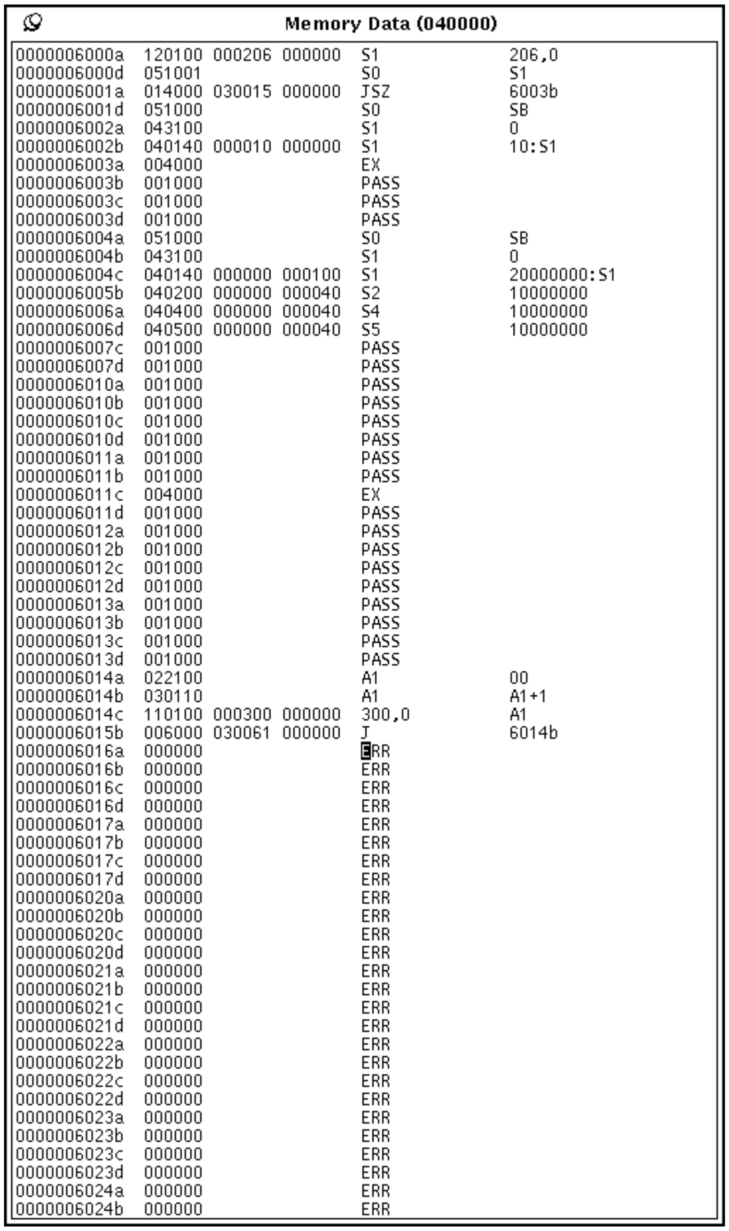
```
0000006000a  120100 000206 000000  S1              206,0
0000006000d  051001                S0              S1
0000006001a  014000 030015 000000  JSZ             6003b
0000006001d  051000                S0              SB
0000006002a  043100                S1              0
0000006002b  040140 000010 000000  S1              10:S1
0000006003a  004000                EX
0000006003b  001000                PASS
0000006003c  001000                PASS
0000006003d  001000                PASS
0000006004a  051000                S0              SB
0000006004b  043100                S1              0
0000006004c  040140 000000 000100  S1              20000000:S1
0000006005b  040200 000000 000040  S2              10000000
0000006006a  040400 000000 000040  S4              10000000
0000006006d  040500 000000 000040  S5              10000000
0000006007c  001000                PASS
0000006007d  001000                PASS
0000006010a  001000                PASS
0000006010b  001000                PASS
0000006010c  001000                PASS
0000006010d  001000                PASS
0000006011a  001000                PASS
0000006011b  001000                PASS
0000006011c  004000                EX
0000006011d  001000                PASS
0000006012a  001000                PASS
0000006012b  001000                PASS
0000006012c  001000                PASS
0000006012d  001000                PASS
0000006013a  001000                PASS
0000006013b  001000                PASS
0000006013c  001000                PASS
0000006013d  001000                PASS
0000006014a  022100                A1              00
0000006014b  006000 030060 000000  J               6014a
0000006015a  000000                ERR
0000006015b  000000                ERR
0000006015c  000000                ERR
0000006015d  000000                ERR
0000006016a  000000                ERR
0000006016b  000000                ERR
0000006016c  000000                ERR
0000006016d  000000                ERR
0000006017a  000000                ERR
0000006017b  000000                ERR
0000006017c  000000                ERR
0000006017d  000000                ERR
0000006020a  000000                ERR
0000006020b  000000                ERR
0000006020c  000000                ERR
0000006020d  000000                ERR
0000006021a  000000                ERR
0000006021b  000000                ERR
0000006021c  000000                ERR
0000006021d  000000                ERR
0000006022a  000000                ERR
0000006022b  000000                ERR
0000006022c  000000                ERR
0000006022d  000000                ERR
0000006023a  000000                ERR
0000006023b  000000                ERR
0000006023c  000000                ERR
0000006023d  000000                ERR
```

Figure 9-7. Writing the Code for a Loop

11. In memory location 6014b, type **`A1 A1 + 1`** to increment the address register by 1. Press and release the **`Control`** and **`Return`** keys simultaneously. This truncates the line at the cursor.

12. In memory location 6014c, type **`300,0 A1`** to place the results at address 300. Press and release the **`Control`** and **`Return`** keys simultaneously. This truncates the line at the cursor.

13. In memory location 6015b, type **`J 6014b`** to jump to address 6001b to continue the loop. Press and release the **`Control`** and **`Return`** keys simultaneously. This truncates the line at the cursor. Your loop code should look like the code shown in Figure 9-8.

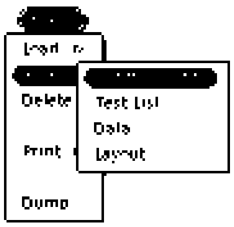


Memory Data (040000)

```
0000006000a  120100 000206 000000  S1              206,0
0000006000d  051001                S0              S1
0000006001a  014000 030015 000000  JSZ             6003b
0000006001d  051000                S0              SB
0000006002a  043100                S1              0
0000006002b  040140 000010 000000  S1              10:S1
0000006003a  004000                EX
0000006003b  001000                PASS
0000006003c  001000                PASS
0000006003d  001000                PASS
0000006004a  051000                S0              SB
0000006004b  043100                S1              0
0000006004c  040140 000000 000100  S1              20000000:S1
0000006005b  040200 000000 000040  S2              10000000
0000006006a  040400 000000 000040  S4              10000000
0000006006d  040500 000000 000040  S5              10000000
0000006007c  001000                PASS
0000006007d  001000                PASS
0000006010a  001000                PASS
0000006010b  001000                PASS
0000006010c  001000                PASS
0000006010d  001000                PASS
0000006011a  001000                PASS
0000006011b  001000                PASS
0000006011c  004000                EX
0000006011d  001000                PASS
0000006012a  001000                PASS
0000006012b  001000                PASS
0000006012c  001000                PASS
0000006012d  001000                PASS
0000006013a  001000                PASS
0000006013b  001000                PASS
0000006013c  001000                PASS
0000006013d  001000                PASS
0000006014a  022100                A1              00
0000006014b  030110                A1              A1+1
0000006014c  110100 000300 000000  300,0           A1
0000006015b  006000 030061 000000  J               6014b
0000006016a  000000                ERR
0000006016b  000000                ERR
0000006016c  000000                ERR
0000006016d  000000                ERR
0000006017a  000000                ERR
0000006017b  000000                ERR
0000006017c  000000                ERR
0000006017d  000000                ERR
0000006020a  000000                ERR
0000006020b  000000                ERR
0000006020c  000000                ERR
0000006020d  000000                ERR
0000006021a  000000                ERR
0000006021b  000000                ERR
0000006021c  000000                ERR
0000006021d  000000                ERR
0000006022a  000000                ERR
0000006022b  000000                ERR
0000006022c  000000                ERR
0000006022d  000000                ERR
0000006023a  000000                ERR
0000006023b  000000                ERR
0000006023c  000000                ERR
0000006023d  000000                ERR
0000006024a  000000                ERR
0000006024b  000000                ERR
```

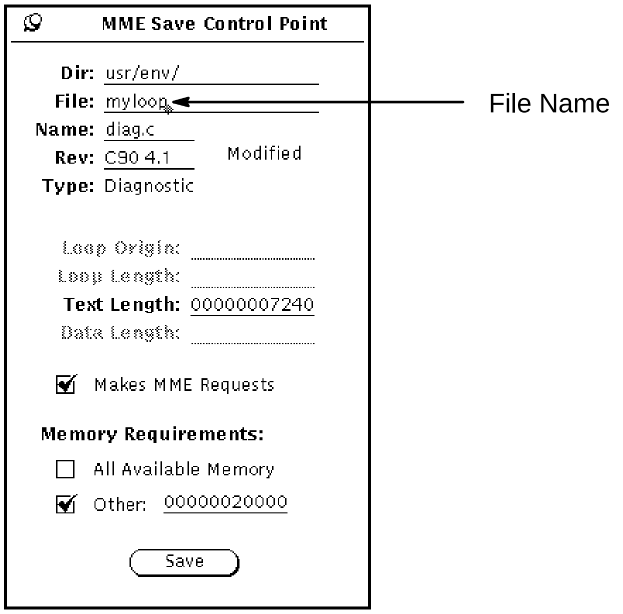
Figure 9-8. Loop Code

You should save any loops before you run them. The following steps describe how to save the example loop you created.



14. In the `Mainframe Maintenance Environment` window, choose **`File --> Save --> Control Point`**, as shown at the left.

15. In the `MME Save Control Point` window, type a file name in the `File` field; this example uses the file name `myloop`. Click on Save. Refer to Figure 9-9.


MME Save Control Point

Dir: usr/env/
File: myloop
Name: diag.c
Rev: C90 4.1 Modified
Type: Diagnostic

Loop Origin:
Loop Length:
Text Length: 00000007240
Data Length:

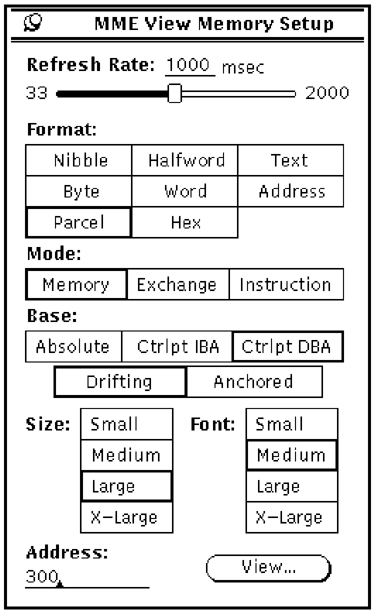
☑ Makes MME Requests

Memory Requirements:
☐ All Available Memory
☑ Other: 00000020000

Figure 9-9. Specifying the Way You Want Your Loop Saved

Before you run the loop, view the memory that this example loop modifies. To do this, perform the following steps.

16. In the `MME View Memory Setup` window, click on Memory and Parcel. Double click on the `Address` field, and type 300. Refer to Figure 9-10.



MME View Memory Setup

Refresh Rate: 1000 msec

33 2000

Format: Nibble Halfword Text Byte Word Address [✓ Parcel] Hex

Mode: [✓ Memory] Exchange Instruction

Base: Absolute Ctrlpt IBA [✓ Ctrlpt DBA]

[✓ Drifting] Anchored

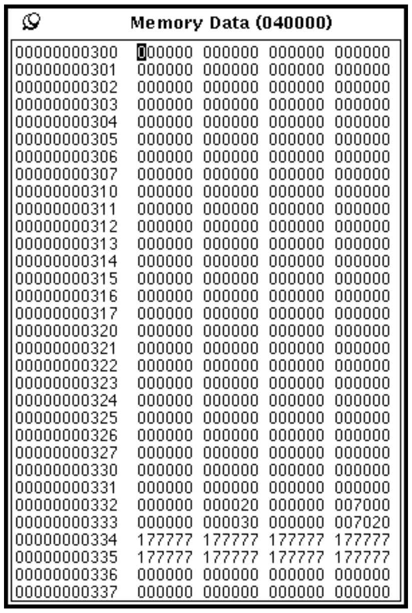
Size: Small Medium [✓ Large] X-Large

Font: Small [✓ Medium] Large X-Large

Address: 300

Figure 9-10. Viewing Mainframe Memory at Location 300

17. Click on; MME displays the `Memory Data` window:



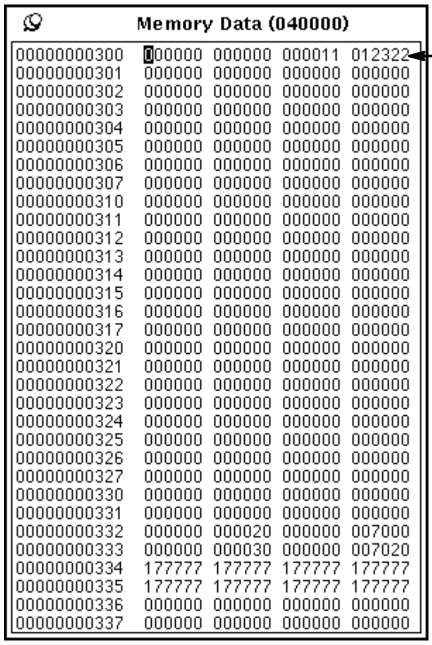
Memory Data (040000)

```
00000000300  000000  000000  000000  000000
00000000301  000000  000000  000000  000000
00000000302  000000  000000  000000  000000
00000000303  000000  000000  000000  000000
00000000304  000000  000000  000000  000000
00000000305  000000  000000  000000  000000
00000000306  000000  000000  000000  000000
00000000307  000000  000000  000000  000000
00000000310  000000  000000  000000  000000
00000000311  000000  000000  000000  000000
00000000312  000000  000000  000000  000000
00000000313  000000  000000  000000  000000
00000000314  000000  000000  000000  000000
00000000315  000000  000000  000000  000000
00000000316  000000  000000  000000  000000
00000000317  000000  000000  000000  000000
00000000320  000000  000000  000000  000000
00000000321  000000  000000  000000  000000
00000000322  000000  000000  000000  000000
00000000323  000000  000000  000000  000000
00000000324  000000  000000  000000  000000
00000000325  000000  000000  000000  000000
00000000326  000000  000000  000000  000000
00000000327  000000  000000  000000  000000
00000000330  000000  000000  000000  000000
00000000331  000000  000000  000000  000000
00000000332  000000  000020  000000  007000
00000000333  000000  000030  000000  007020
00000000334  177777  177777  177777  177777
00000000335  177777  177777  177777  177777
00000000336  000000  000000  000000  000000
00000000337  000000  000000  000000  000000
```

Now you can run the loop; perform the following step.

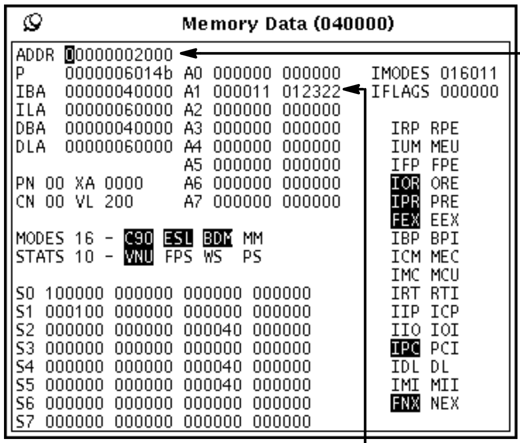
18. Click on in the `Mainframe Maintenance Environment` window.

Notice that the data is placed at address 300:



```
00000000300   000000  000000  000011  012322
00000000301   000000  000000  000000  000000
00000000302   000000  000000  000000  000000
00000000303   000000  000000  000000  000000
00000000304   000000  000000  000000  000000
00000000305   000000  000000  000000  000000
00000000306   000000  000000  000000  000000
00000000307   000000  000000  000000  000000
00000000310   000000  000000  000000  000000
00000000311   000000  000000  000000  000000
00000000312   000000  000000  000000  000000
00000000313   000000  000000  000000  000000
00000000314   000000  000000  000000  000000
00000000315   000000  000000  000000  000000
00000000316   000000  000000  000000  000000
00000000317   000000  000000  000000  000000
00000000320   000000  000000  000000  000000
00000000321   000000  000000  000000  000000
00000000322   000000  000000  000000  000000
00000000323   000000  000000  000000  000000
00000000324   000000  000000  000000  000000
00000000325   000000  000000  000000  000000
00000000326   000000  000000  000000  000000
00000000327   000000  000000  000000  000000
00000000330   000000  000000  000000  000000
00000000331   000000  000000  000000  000000
00000000332   000000  000020  000000  007000
00000000333   000000  000030  000000  007020
00000000334   177777  177777  177777  177777
00000000335   177777  177777  177777  177777
00000000336   000000  000000  000000  000000
00000000337   000000  000000  000000  000000
```

19. Click on Halt in the `Mainframe Maintenance Environment` window; MME places the exchange package for CPU 0 at memory address 2000. Refer to Figure 9-11.



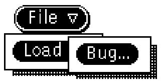
```
ADDR 00000002000
P    0000006014b A0 000000 000000   IMODES 016011
IBA  00000040000 A1 000011 012322   IFLAGS 000000
ILA  00000060000 A2 000000 000000
DBA  00000040000 A3 000000 000000     IRP RPE
DLA  00000060000 A4 000000 000000     IUM MEU
                 A5 000000 000000     IFP FPE
PN 00 XA 0000    A6 000000 000000     IOR ORE
CN 00 VL 200     A7 000000 000000     IPR PRE
                                      FEX EEX
MODES 16 - C90 ESL BDM MM             IBP BPI
STATS 10 - VNU FPS WS  PS             ICM MEC
                                      IMC MCU
S0 100000 000000 000000 000000        IRT RTI
S1 000100 000000 000000 000000        IIP ICP
S2 000000 000000 000040 000000        IIO IOI
S3 000000 000000 000000 000000        IPC PCI
S4 000000 000000 000040 000000        IDL DL
S5 000000 000000 000040 000000        IMI MII
S6 000000 000000 000000 000000        FNX NEX
S7 000000 000000 000000 000000
```

Figure 9-11. Updated Exchange Package at Memory Address 2000

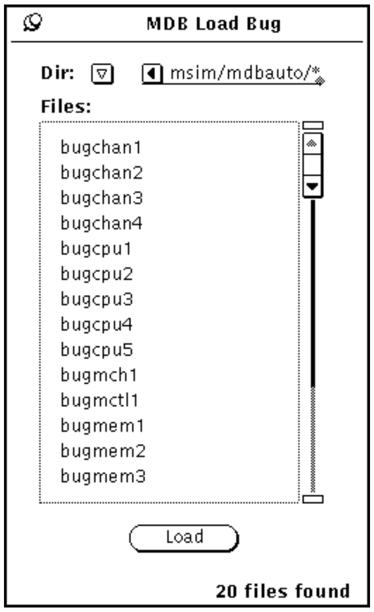
Troubleshooting a Mainframe Simulator Bug

This subsection provides an example of troubleshooting a mainframe simulator bug. The following procedure shows how to load the `bugcpu2` mainframe bug and use control points to detect the error.

NOTE: Before you start this procedure, MME environment 2 must be running with the simulator and debugger. If you need help starting environment 2 with the simulator and debugger, refer to "Starting MME Environment 1 or Environment 2" in Section 4 of this manual.



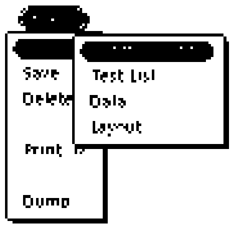
1. Choose **`File --> Load --> Bug`**, as shown at the left, in the `MSIM Simulator Bugger/Debugger` window. The `MDB Load Bug` window appears:



2. Click on the `bugcpu2` bug to select it.

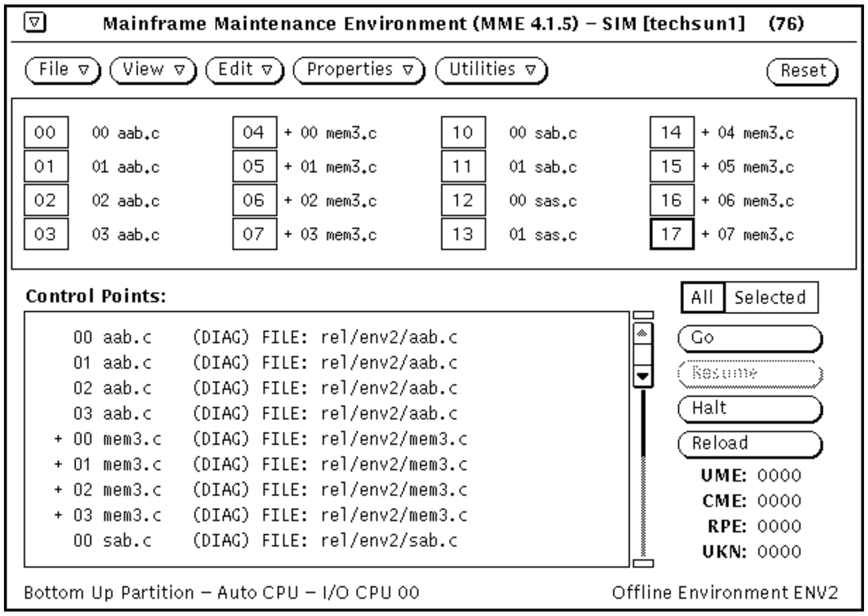
3. Click on Load; the bug is loaded.

   Now that the bug is loaded, start troubleshooting in environment 2 with multiple control points. You can load different control points or several copies of the same control point.



4. To load control points in environment 2, ensure MME is in environment 2 and choose **`File --> Load --> Control Point`**, as shown at the left, in the `Mainframe Maintenance Environment` window. (For more information about loading control points in environment 2, refer to “Loading Control Points in Environment 1 or 2” in Section 4 of this manual.)

   For this example, load the following control points: four copies of `aab.c`, four copies of `mem3.c`, two copies of `sab.c`, two copies of `sas.c`, and four more copies of `mem3.c`, as shown in the following snap:


Mainframe Maintenance Environment (MME 4.1.5) – SIM [techsun1] (76)
File
View
Edit
Properties
Utilities
Reset
All
Selected
Go
Resume
Halt
Reload

```
00  00 aab.c      04  + 00 mem3.c    10  00 sab.c    14  + 04 mem3.c
01  01 aab.c      05  + 01 mem3.c    11  01 sab.c    15  + 05 mem3.c
02  02 aab.c      06  + 02 mem3.c    12  00 sas.c    16  + 06 mem3.c
03  03 aab.c      07  + 03 mem3.c    13  01 sas.c    17  + 07 mem3.c
```

Control Points:

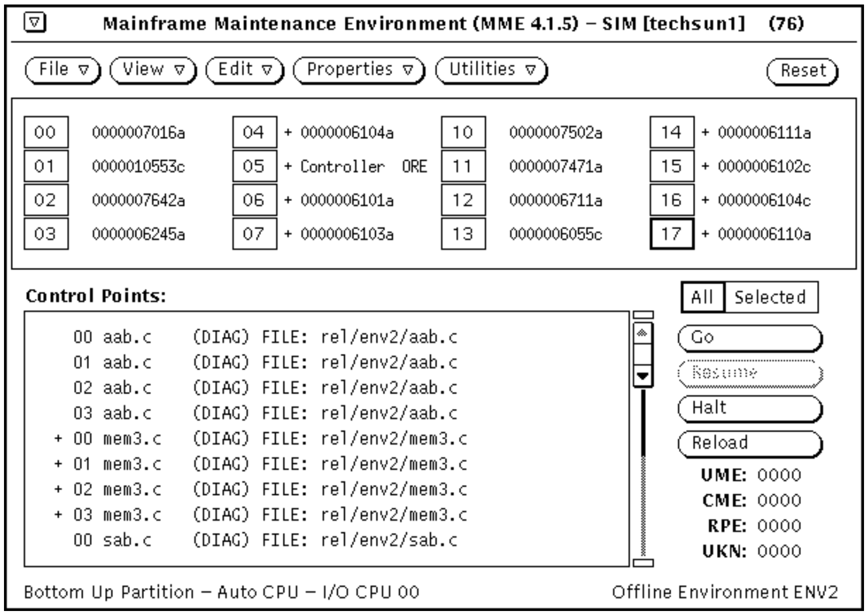
```
   00 aab.c    (DIAG) FILE: rel/env2/aab.c
   01 aab.c    (DIAG) FILE: rel/env2/aab.c
   02 aab.c    (DIAG) FILE: rel/env2/aab.c
   03 aab.c    (DIAG) FILE: rel/env2/aab.c
 + 00 mem3.c   (DIAG) FILE: rel/env2/mem3.c
 + 01 mem3.c   (DIAG) FILE: rel/env2/mem3.c
 + 02 mem3.c   (DIAG) FILE: rel/env2/mem3.c
 + 03 mem3.c   (DIAG) FILE: rel/env2/mem3.c
   00 sab.c    (DIAG) FILE: rel/env2/sab.c
```

UME: 0000
CME: 0000
RPE: 0000
UKN: 0000

Bottom Up Partition – Auto CPU – I/O CPU 00 Offline Environment ENV2

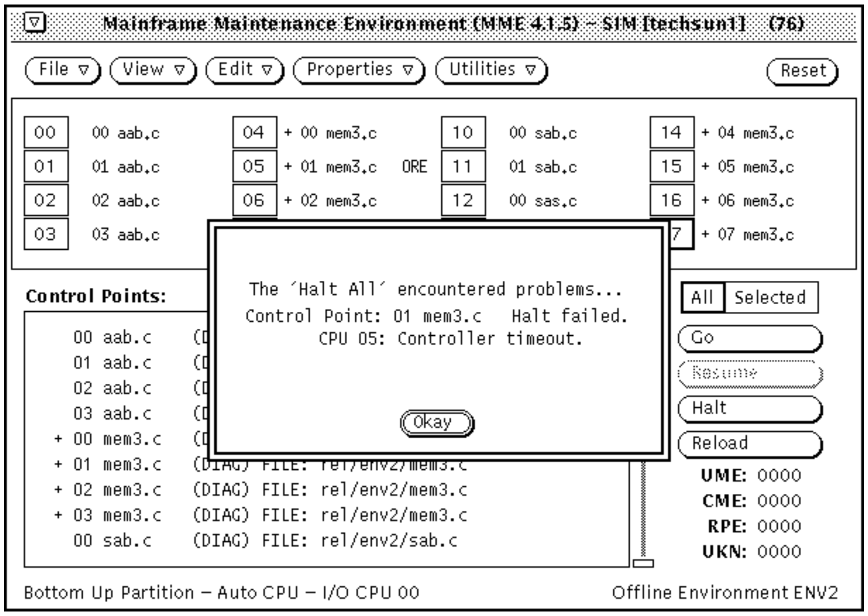
5. Click on in the `Mainframe Maintenance Environment` window to start the control points.

After the control points have run a while, an operand range-error interrupt occurs in CPU 5, as shown in the following snap:

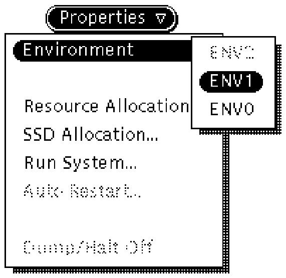


```
Mainframe Maintenance Environment (MME 4.1.5) – SIM [techsun1]   (76)

File ▽  View ▽  Edit ▽  Properties ▽  Utilities ▽                    Reset

00  0000007016a    04  + 0000006104a     10  0000007502a   14  + 0000006111a
01  0000010553c    05  + Controller  ORE 11  0000007471a   15  + 0000006102c
02  0000007642a    06  + 0000006101a     12  0000006711a   16  + 0000006104c
03  0000006245a    07  + 0000006103a     13  0000006055c   17  + 0000006110a

Control Points:                                            All  Selected
    00 aab.c    (DIAG) FILE: rel/env2/aab.c                Go
    01 aab.c    (DIAG) FILE: rel/env2/aab.c                Resume
    02 aab.c    (DIAG) FILE: rel/env2/aab.c                Halt
    03 aab.c    (DIAG) FILE: rel/env2/aab.c                Reload
  + 00 mem3.c   (DIAG) FILE: rel/env2/mem3.c                UME: 0000
  + 01 mem3.c   (DIAG) FILE: rel/env2/mem3.c                CME: 0000
  + 02 mem3.c   (DIAG) FILE: rel/env2/mem3.c                RPE: 0000
  + 03 mem3.c   (DIAG) FILE: rel/env2/mem3.c                UKN: 0000
    00 sab.c    (DIAG) FILE: rel/env2/sab.c

Bottom Up Partition – Auto CPU – I/O CPU 00          Offline Environment ENV2
```

This is an unexpected interrupt, so the control point does not provide any error information about the bug. You should try a different control point to gain error information. Click on Halt to stop the control points so you can assign CPU 5 to a different control point. A message should appear that indicates the controller has timed out for CPU 5, as shown in the following snap:

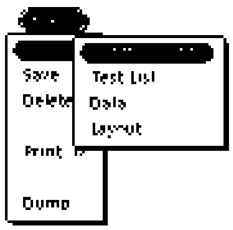


```
Mainframe Maintenance Environment (MME 4.1.5) – SIM [techsun1]   (76)

File ▽  View ▽  Edit ▽  Properties ▽  Utilities ▽                    Reset

00  00 aab.c    04  + 00 mem3.c      10  00 sab.c    14  + 04 mem3.c
01  01 aab.c    05  + 01 mem3.c  ORE 11  01 sab.c    15  + 05 mem3.c
02  02 aab.c    06  + 02 mem3.c      12  00 sas.c    16  + 06 mem3.c
03  03 aab.c                                        7  + 07 mem3.c

        The ´Halt All´ encountered problems...
        Control Point: 01 mem3.c   Halt failed.
              CPU 05: Controller timeout.

                        Okay

Control Points:                                            All  Selected
    00 aab.c    (D                                         Go
    01 aab.c    (D                                         Resume
    02 aab.c    (D                                         Halt
    03 aab.c    (D                                         Reload
  + 00 mem3.c   (D                                          UME: 0000
  + 01 mem3.c   (DIAG) FILE: rel/env2/mem3.c                CME: 0000
  + 02 mem3.c   (DIAG) FILE: rel/env2/mem3.c                RPE: 0000
  + 03 mem3.c   (DIAG) FILE: rel/env2/mem3.c                UKN: 0000
    00 sab.c    (DIAG) FILE: rel/env2/sab.c

Bottom Up Partition – Auto CPU – I/O CPU 00          Offline Environment ENV2
```



This indicates a problem with CPU 5 which cannot be tested further in environment 2. To determine the specific error in CPU 5, switch to environment 1 to isolate troubleshooting to CPU 5.

6. To switch to environment 1, choose **`Properties -->`** **`Environment --> ENV1`**, as shown at the left, in the `Mainframe Maintenance Environment` window.

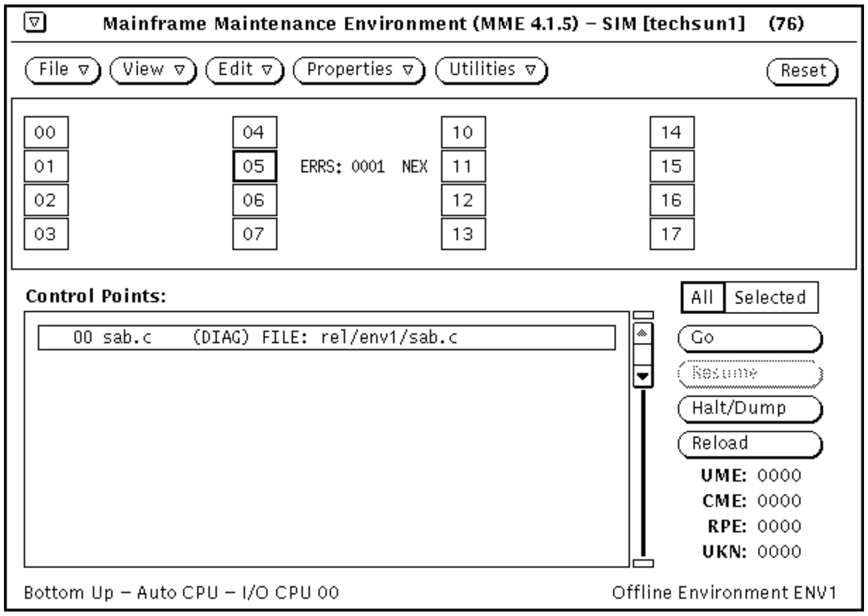


7. To load a control point in environment 1, choose **`File -->`** **`Load --> Control Point`**, as shown at the left, in the `Mainframe Maintenance Environment` window. (For more information about loading a control point in environment 1, refer to “Loading Control Points in Environment 1 or 2” in Section 4, “MME Environments 1 and 2,” of this manual.)

   For this example, load the `sab.c` control point. (Remember, if you need help choosing control points to run, refer to Table 7-5, “Solving Problems from the Bottom Up,” for a suggested list of control points to run.)

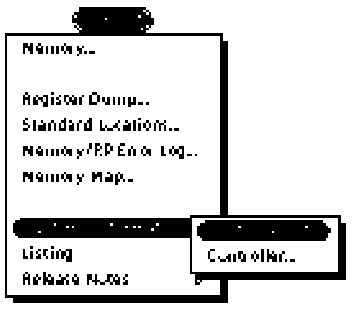
8. Click on 05 to assign CPU 5 to the control point.

9. Click on Go in the `Mainframe Maintenance Environment` window to start running the control point. An error occurs in CPU 5, as shown in the following snap:

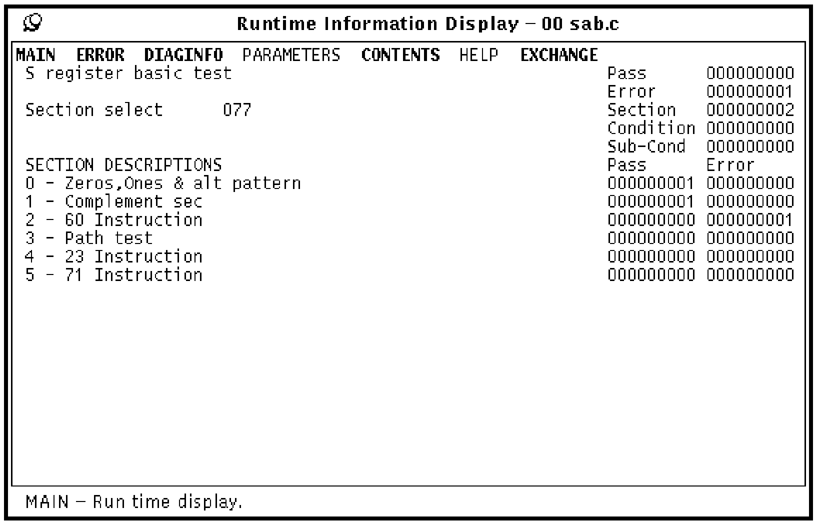


```
ERRS: 0001  NEX
00 sab.c    (DIAG) FILE: rel/env1/sab.c
UME: 0000
CME: 0000
RPE: 0000
UKN: 0000
Bottom Up – Auto CPU – I/O CPU 00          Offline Environment ENV1
```

To get detailed information about the error that occurred, you should view the runtime information displays.

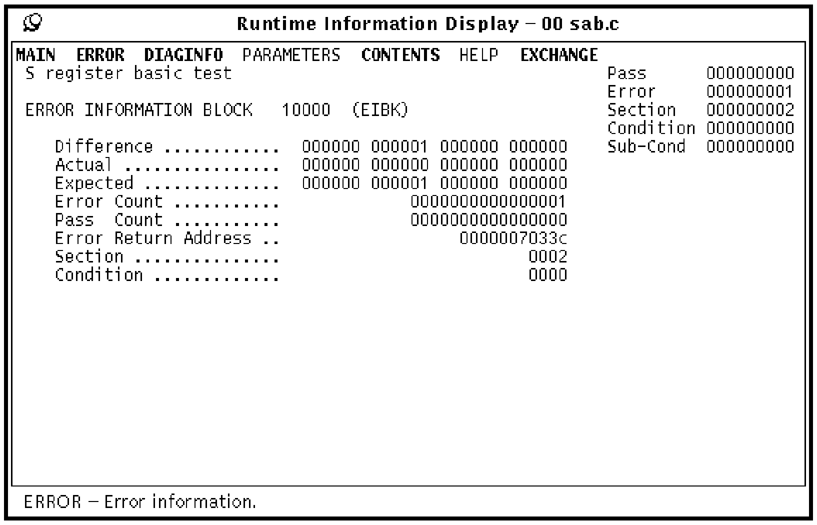


10. Choose **`View --> Runtime Information --> Control Point`**, as shown at the left. MME displays the `MME Runtime Information Display`, as shown in the following snap:



```
Runtime Information Display – 00 sab.c
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
 S register basic test                                       Pass      000000000
                                                             Error     000000001
 Section select      077                                     Section   000000002
                                                             Condition 000000000
                                                             Sub-Cond  000000000
 SECTION DESCRIPTIONS                                        Pass      Error
 0 - Zeros,Ones & alt pattern                                000000001 000000000
 1 - Complement sec                                          000000001 000000000
 2 - 60 Instruction                                          000000000 000000001
 3 - Path test                                               000000000 000000000
 4 - 23 Instruction                                          000000000 000000000
 5 - 71 Instruction                                          000000000 000000000

 MAIN – Run time display.
```

11. Click on **`ERROR`**. MME displays the error information on the `MME Runtime Information Display`, as shown in the following snap:



```
Runtime Information Display – 00 sab.c
MAIN  ERROR  DIAGINFO  PARAMETERS  CONTENTS  HELP  EXCHANGE
 S register basic test                                       Pass      000000000
                                                             Error     000000001
 ERROR INFORMATION BLOCK   10000  (EIBK)                     Section   000000002
                                                             Condition 000000000
    Difference ...........  000000 000001 000000 000000      Sub-Cond  000000000
    Actual ...............  000000 000000 000000 000000
    Expected .............  000000 000001 000000 000000
    Error Count ..........             0000000000000001
    Pass  Count ..........             0000000000000000
    Error Return Address ..                  0000007033c
    Section ..............                          0002
    Condition ............                          0000

 ERROR – Error information.
```

The `MME Runtime Information Display` shows errors in CPU 5 and bit 2^{32} (using the `Difference` information).

APPENDIX: CRAY C90 REMOTE SUPPORT

This appendix contains two procedures that describe how to start MME, MCE, or LME from a service center where you must first go through a hub and then log in to the on-site MWS-E (refer to Figure A-1). One procedure describes the commands used with the ME-C2.3.1 offline diagnostic release; the other procedure describes the commands used with the ME-C2.3 offline diagnostic release. Use the `release` command to determine which release you are using.

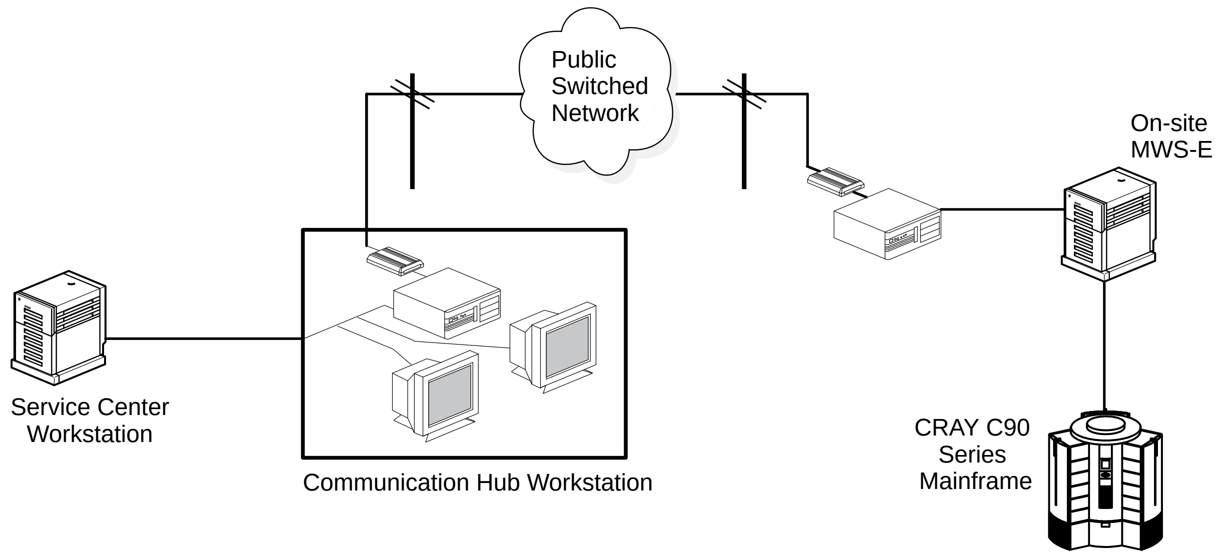


Figure A-1. Starting MME, MCE, or LME through a Hub

These procedures are written with the assumption that the service center workstation has a network connection to the hub workstation and that the on-site MWS-E is connected to the hub workstation with modems and NetBlazer routers. When you use a procedure, the application's server runs on the on-site MWS-E, and the application's client runs on the hub workstation and displays the interface on the service center workstation.

To perform these procedures, you must have a login on the hub, have a login on the MWS-E, and be in the `mws` group on the MWS-E. Refer to the *Remote Support System Guide*, CDM-1125-000, for more information about Remote Support.

Remote Support Procedure (ME-C2.3.1 Offline Diagnostic Release)

1. Open two windows on your service center workstation. You will use these windows in Steps 2 through 11 to start the server and client applications.

   Throughout this procedure, one window is called the *MWS-E window*, and one window is called the *hub window*. The MWS-E window will be used to start the server on the on-site MWS-E. The hub window will be used to start the client on the hub workstation.

2. In the hub window, enter the following command to enable the hub workstation display to be redirected to the service center workstation.

   Replace *`<hub>`* with the hostname of the hub workstation.

   ```
   xhost + <hub> ↵
   ```

3. In the hub window, enter the following command to log in to the hub workstation (log in as yourself).

   Replace *`<hub>`* with the hostname of the hub workstation.

   ```
   telnet <hub> ↵
   ```

4. In the hub window, enter the following commands to set the display variable to your service center workstation.

   Replace *`<servicecenter>`* with the hostname of the service center workstation.

   ```
   DISPLAY=<servicecenter>:0.0 ↵
   export DISPLAY ↵
   ```

5. In the MWS-E window, enter the following command to log in to the hub workstation to access the on-site MWS-E.

   Replace *`<hub>`* with the hostname of the hub workstation.

   ```
   telnet <hub> ↵
   ```

6. In the MWS-E window, enter the following command to log in to the on-site MWS-E (log in as yourself) from the hub workstation.

   Replace *`<onsitemws>`* with the hostname of the on-site MWS-E.

   ```
   telnet <onsitemws> ↵
   ```

7. In the MWS-E window, enter the following command to change to the directory used by the MME, MCE, and LME applications:

   `cd /cri/cme/c90` ↵

8. In the MWS-E window, enter the following command to find out whether any server processes are already running:

   `bin/psmme`

   If there are no processes displayed, the specified server (`mme0s`, `mmes`, `mces`, or `lmes`) is not running on the on-site MWS-E. Continue with Step 9 to start the server on the on-site MWS-E.

   If the command displays a process (`mme0s`, `mmes` , `mces`, or `lmes`), this indicates that the desired application is already running on the on-site MWS-E. Go to Step 10 to start the client application on the hub workstation.

9. In the MWS-E window, enter the command or commands shown in Table A-1 followed by ↵ to start the server on the on-site MWS-E for the application you want to use.

Table A-1. Server Commands on the On-site MWS-E (ME-C2.3.1 Only)

Application	Command(s)
MME environment 0	**`bin/mme –0 –server &`**
MME environment 0 (with the simulator)	**`bin/mme –0 –server –sim &`**
MME environment 1	**`bin/mme –1 –server &`**
MME environment 1 (with the simulator)	**`bin/mme –1 –server –sim &`**
MME environment 2	**`bin/mme –2 –server &`**
MME environment 2 (with the simulator)	**`bin/mme –2 –server –sim &`**
MCE	**`bin/mce –server &`**
MCE (with the simulator)	**`bin/mce –server –sim &`**
LME	**`bin/lme –server &`**
LME (with the simulator)	**`bin/lme –server –sim &`**

10. In the hub window, enter the following command to change to the directory used by the MME, MCE, and LME applications:

    `cd /cri/ME-C2.3.1/cme/c90` ↵

11. In the hub window, enter the command shown in Table A-2 followed by ↵ to start the client on the hub workstation for the application you want to use.

    Replace `<onsitemws>` in the commands with the hostname of the on-site MWS-E (this is the hostname used in Step 5).

Table A-2. Client Commands on the Hub Workstation (ME-C2.3.1 Only)

Application	Command
MME environment 0	**bin/mme –0 –remote <*onsitemws*> &**
MME environment 0 (with the simulator)	**bin/mme –0 –remote <*onsitemws*> &**
MME environment 1	**bin/mme –1 –remote <*onsitemws*> &**
MME environment 1 (with the simulator)	**bin/mme –1 –remote <*onsitemws*> &**
MME environment 2	**bin/mme –2 –remote <*onsitemws*> &**
MME environment 2 (with the simulator)	**bin/mme –2 –remote <*onsitemws*> &**
MCE	**bin/mce –remote <*onsitemws*> &**
MCE (with the simulator)	**bin/mce –remote <*onsitemws*> &**
LME	**bin/lme –remote <*onsitemws*> &**
LME (with the simulator)	**bin/lme –remote <*onsitemws*> &**

The interface appears on the local workstation in a few seconds; then, you can troubleshoot the CRAY C90 series mainframe connected to the remote MWS-E.

Remote Support Procedure (ME-C2.3 Offline Diagnostic Release)

1. Open two windows on your service center workstation. You will use these windows in Steps 2 through 11 to start the server and client applications.

   Throughout this procedure, one window is called the *MWS-E window*, and one window is called the *hub window*. The MWS-E window will be used to start the server on the on-site MWS-E. The hub window will be used to start the client on the hub workstation.

2. In the hub window, enter the following command to enable the hub workstation display to be redirected to the service center workstation.

   Replace *<hub>* with the hostname of the hub workstation.

   ```
   xhost + <hub> ↵
   ```

3. In the hub window, enter the following command to log in to the hub workstation (log in as yourself).

   Replace *<hub>* with the hostname of the hub workstation.

   ```
   telnet <hub> ↵
   ```

4. In the hub window, enter the following commands to set the display variable to your service center workstation.

   Replace *<servicecenter>* with the hostname of the service center workstation.

   ```
   DISPLAY=<servicecenter>:0.0 ↵
   export DISPLAY ↵
   ```

5. In the MWS-E window, enter the following command to log in to the hub workstation to access the on-site MWS-E.

   Replace *<hub>* with the hostname of the hub workstation.

   ```
   telnet <hub> ↵
   ```

6. In the MWS-E window, enter the following command to log in to the on-site MWS-E (log in as yourself) from the hub workstation.

   Replace *<onsitemws>* with the hostname of the on-site MWS-E.

   ```
   telnet <onsitemws> ↵
   ```

7. In the MWS-E window, enter one of the following commands to see if any server processes are already running:

   - For environment 0: **`ps –auxc | grep mme0s`**↵
   - For environment 1 or 2: **`ps –auxc | grep mmes`**↵
   - For MCE: **`ps –auxc | grep mces`**↵
   - For LME: **`ps –auxc | grep lmes`**↵

   If there are no processes displayed, the specified server (`mme0s`, `mmes`, `mces`, or `lmes`) is not running on the on-site MWS-E. Continue with Step 8 to start the server on the on-site MWS-E.

   If the command displays a process (`mme0s`, `mmes` , `mces`, or `lmes`), this indicates that the desired application is already running on the on-site MWS-E. Go to Step 10 to start the client application on the hub workstation.

8. In the MWS-E window, enter the following command to change to the directory used by the MME, MCE, and LME applications:

   `cd /cri/cme/c90` ↵

9. In the MWS-E window, enter the command or commands shown in Table A-3 followed by ↵ to start the server on the on-site MWS-E for the application you want to use.

   NOTE: For more information about the command line options available for the commands shown in Table A-3, refer to Table A-5 later in this section.

Table A-3. Server Commands on the On-site MWS-E (ME-C2.3 Only)

Application	Command(s)
MME environment 0	**`bin/mme0s &`**

MME environment 0 (with the simulator)	**`bin/msim &`**
	`bin/mme0s –sim &`

MME environment 1	**`bin/mmes –1 &`**

MME environment 1 (with the simulator)	**`bin/msim &`**
	`bin/mmes –1 –sim &`

MME environment 2	**`bin/mmes –2  &`**

MME environment 2 (with the simulator)	**`bin/msim &`**
	`bin/mmes –2 –sim &`

MCE	**`bin/mces  &`**

MCE (with the simulator)	**`bin/msim &`**
	`bin/mces –sim &`

Table A-3. Server Commands on the On-site MWS-E (ME-C2.3 Only) (continued)

Application	Command(s)
LME	**bin/lmes &**
LME (with the simulator)	**bin/msim &** **bin/lmes –sim &**

10. In the hub window, enter the following command to change to the directory used by the MME, MCE, and LME applications:

```
cd /cri/ME-C2.3/cme/c90 ↵
```

11. In the hub window, enter the command shown in Table A-4 followed by ↵ to start the client on the hub workstation for the application you want to use.

    Replace *<onsitemws>* in the commands with the hostname of the on-site MWS-E (this is the hostname used in Step 5).

    NOTE: For more information about the command line options available for the commands shown in Table A-4, refer to Table A-5 later in this section.

Table A-4. Client Commands on the Hub Workstation (ME-C2.3 Only)

Application	Command
MME environment 0	**bin/mme0c –remote <*onsitemws*> &**
MME environment 0 (with the simulator)	**bin/mme0c –remote <*onsitemws*> &**
MME environment 1	**bin/mmec –remote <*onsitemws*> &**
MME environment 1 (with the simulator)	**bin/mmec –remote <*onsitemws*> &**
MME environment 2	**bin/mmec –remote <*onsitemws*> &**
MME environment 2 (with the simulator)	**bin/mmec –remote <*onsitemws*> &**
MCE	**bin/mcec –remote <*onsitemws*> &**
MCE (with the simulator)	**bin/mcec –remote <*onsitemws*> &**
LME	**bin/lmec –remote <*onsitemws*> &**
LME (with the simulator)	**bin/lmec –remote <*onsitemws*> &**

The interface appears on the local workstation in a few seconds; then, you can troubleshoot the CRAY C90 series mainframe connected to the remote MWS-E.

Remote Support Command Line Options (ME-C2.3 Only)

When you start the server and client programs on different workstations to use MME, MCE, and LME for remote support, there are several command line options available. Table A-5 shows the command line options for the different server and client programs. This information is provided for reference; usually, you should start the programs with the default settings, as shown in the previous procedure.

Table A-5. Command Line Options (ME-C2.3 Only)

Command	Option	Description
mme0s	–mmeport <*port*>	Forces the MME environment 0 server to use a port other than the default; be sure the server and client are set to the same *port*
	–mceport <*port*>	Forces the MME environment 0 server to connect to an MCE port other than the default
	–lmeport <*port*>	Forces the MME environment 0 server to connect to an LME port other than the default
	–chn [0|1|2|3|4|5|6|7]	Specifies the slot to which the maintenance channel is connected
	–sim [<*host*> [<*port*>]]	Starts the MME environment 0 server with the simulator; optionally, you can specify the *host* the simulator is running on and a specific *port* if you do not want to use the default
	–concurrent	Forces the MME environment 0 server into concurrent mode; this is the default mode for the server
	–offline	Forces the MME environment 0 server into offline mode; this will crash the operating system in the mainframe if the operating system is running
mme0c	–mmeport <*port*>	Forces the MME environment 0 client to use a port other than the default; be sure the server and client are set to the same *port*
	–remote <*host*>	Forces the MME environment 0 client to connect to a MME environment 0 server on a remote workstation
	–server <*server_path*>	Specifies a different directory path for the server

Table A-5. Command Line Options (ME-C2.3 Only) (continued)

Command	Option	Description
mmes	–1	Starts MME in environment 1
	–2	Starts MME in environment 2
	–mmeport <*port*>	Forces the MME server to use a port other than the default; be sure the server and client are set to the same *port*
	–mceport <*port*>	Forces the MME server to connect to an MCE port other than the default
	–lmeport <*port*>	Forces the MME server to connect to an LME port other than the default
	–chn [0|1|2|3|4|5|6|7]	Specifies the slot to which the maintenance channel is connected
	–sim [<*host*> [<*port*>]]	Starts the MME server with the simulator; optionally, you can specify the *host* the simulator is running on and a specific *port* if you do not want to use the default
	–concurrent	Forces the MME server into concurrent mode; this is the default mode for the server
	–offline	Forces the MME server into offline mode; this will crash the operating system in the mainframe if the operating system is running
mmec	–mmeport <*port*>	Forces the MME client to use a port other than the default; be sure the server and client are set to the same *port*
	–remote <*host*>	Forces the client to connect to a MME server on a remote workstation
	–server <*server_path*>	Specifies a different directory path for the server
mces	–offline	Forces the MCE server into offline mode; this will cause the operating system in the mainframe to crash if the operating system is running
	–concurrent	Forces the MCE server into concurrent mode; this is the default mode for the server
	–mceport <*port*>	Forces the MCE server to use a port other than the default; be sure the server and client are set to the same *port*
	–chn [0|1|2|3|4|5|6|7]	Specifies the slot to which the maintenance channel is connected
	–sim <*host*> [<*port*>]]	Starts the MCE server with the simulator; optionally, you can specify the *host* the simulator is running on and a specific *port* if you do not want to use the default

Table A-5. Command Line Options (ME-C2.3 Only) (continued)

Command	Option	Description
mcec	–mceport <*port*>	Forces the MCE client to use a port other than the default; be sure the client and server are set to the same *port*
	–remote <*host*>	Forces the client to connect to a MCE server on a remote workstation
	–server <*server_path*>	Specifies a different directory path for the server
lmes	–lmeport <*port*>	Forces the LME server to use a port other than the default; be sure to set the LME server and client to the same port
	–chn [0|1|2|3|4|5|6|7]	Specifies the slot to which the maintenance channel is connected
	–sim [<*host*> [<*port*>]]	Starts the LME server with the simulator; optionally, you can specify the *host* the simulator is running on and a specific *port* if you do not want to use the default
	–mmeport <*port*>	Forces the LME server to connect to an MME port other than the default
	–mceport <*port*>	Forces the LME server to connect to an MCE port other than the default
lmec	–lmeport<*port*>	Forces the LME client to use a port other than the default; be sure to set the LME client and server to the same *port*
	–remote <*host*>	Forces the client to connect to a LME server on a remote workstation

GLOSSARY

A

A register	Address register. A registers are primarily used as address storage for memory references and as index registers.
Abbreviated menu button	A ▽ button that is displayed as a small square with a hollow triangle inside the border. The triangle points downward when the menu is displayed below the menu button; it points to the right when the menu is displayed to the right. The current setting is usually displayed to the right of the abbreviated menu button. Abbreviated menu buttons function the same way as menu buttons.
Automatic mode	One of three modes available in the mainframe maintenance environment. This automatic mode appears as the default mode when you do not specify a mode. Automatic mode uses the system defaults to run a diagnostic program.

B

B register	Intermediate scalar register. The B registers are used as intermediate storage for the A registers.
Banks	Sectioned portions of central memory that provide improved memory access speeds. CRAY C90 series mainframe memory is divided into 8 sections; each section is divided into 8 subsections. Each subsection is divided into groups and banks; the number of banks depends on the system.
Base window	The primary window for an application.
Breakpoint	A specified point within a computer program at which the program may be interrupted.
Buffer memory	In MME, an area of memory in the MWS used for result checking.
Bug	A simulated hardware failure that you use with the simulator and bugger/debugger. Several bugs are included in the offline diagnostic release, and you create your own bugs with the bugmaker utility.

B (continued)

Bugmaker utility	The utility that you use to create user bugs for use with the simulator and bugger/debugger. Use the `Utilities --> Bugmaker` in the `MSIM Simulator Bugger/Debugger` window to access the bugmaker utility.
Button	A one-choice element of a menu. Buttons are used to execute commands (command button) and to display popup windows (window button) and menus (menu button).
Button menu	A menu that is displayed when the pointer is on either a menu button or an abbreviated menu button and you press the MENU button.

C

CA	Current address register**.** The CA register contains the initial address for a channel transfer. The contents of the CA register are incremented until the transfer is finished.
CAL	Cray assembly language. A symbolic language that generates machine instructions on a one-for-one basis and enables programs to call subroutines from the library through the use of macros.
Canned answers	A list of problems and a solution to each of those problems.
Caret	A symbol within a window or screen that indicates where keyboard input is placed. An active caret is a solid triangle (▲) that may blink. An inactive caret is a dimmed diamond (◊).
CBP	*See* Command buffer parser.
Central memory	Memory in the CRAY C90 series mainframe shared by all central processing units.
Check box	A box representing an option you can enable or disable. Press and release (click) the SELECT mouse button when the mouse pointer is over a box to toggle on or toggle off the check in the box. A check mark (√) in a box enables the desired option, and an empty box means that the option is disabled.
CIP	Current instruction parcel register. The CIP register holds the next instruction waiting to be issued.
CL	Channel limit register. The contents of the CL register are one greater than the last address of the CPU’s cluster.

C (continued)

Clear	The clear utility clears or erases the contents of registers, memory, and shared registers.
CLN	The CLN register contents determine which set of SB, ST, and SM registers the CPU can access.
Command buffer parser	An X Window System based application that automates troubleshooting with the offline diagnostic tools.
Command button	A button used to execute application commands. *See also* Button.
Command window	A popup window that is used to execute application commands or set parameters. *See also* Window.
Completion message	A status message in the footer of a window that identifies when a task is finished.
Compose mode	One of three test modes available in the mainframe maintenance environment. As the name implies, you can use this mode to compose tests for identifying a problem. You can create your own tests or change existing tests to identify and solve the problem. *See also* Manual mode and Auto (automatic) mode.
Concurrent Mode	One of two modes available with the mainframe configuration environment, mainframe maintenance environment, and logic monitor environment. In this mode, the operating system has control over the mainframe and the mainframe maintenance environment and logic monitor environment can run in specified CPUs.
Continue on error	Enables a diagnostic program to continue processing after an error has been found.
Control area	An unbordered region of a window where controls such as buttons, settings, sliders, gauges, text fields, and check boxes are displayed.
Control point	A diagnostic program, loop, or utility that is loaded in memory in environment 1 or 2.
Control point listing	A listing that contains control point code, information about how to use a control point, and what the program tests. A diagnostic program listing is the same as a control point listing.
CPU	Central processing unit. The CPU is the primary functioning unit of the computer system. It consists of a computational section and a control section.

C (continued)

CRASH_X program	A program that collects CPU data from multiple DM dumps to create an analysis of CPU status.
Current item	An active item, shown with a box around it, in a scrolling list.

D

Data base address register	The DBA register is part of the exchange package and holds the base address of the user's data range.
Data base limit register	Data limit address register. The DLA register holds the upper limit address of the user's data range.
DBA	*See* Data base address register.
Deadstart	The sequence of operations required to start an operating system running in a Cray Research computer system.
Deadstart exchange package	Located in standard location 00 through 17 in octal, this exchange package initially starts CPU 0.
Diagnostic	*See* Diagnostic program.
Diagnostic program	Program supplied by Cray Research, Inc. that is used to identify and solve hardware errors in environment 1 or 2.
Diagnostic monitor	Two options (HR10 and HF0) on each CPU used to record test point and control information at specified times to indicate the state of the CPU.
Diagnostic program listing	A listing that contains diagnostic program code, information about how to use a diagnostic program, and what the program tests. A diagnostic program listing is the same as a control point listing.
Diagnostic test	Test supplied by Cray Research, Inc. that is used to identify and solve hardware errors in environment 0. These tests are available: the maintenance channel test, the instruction buffer test, memory data pattern test, the exchange test, the diagnostic monitor test, and the miscellaneous test.
Dimmed control	An inactive control that is dimmed to show it cannot accept input from the mouse or keyboard.
Disable	To prevent a function from being used. The opposite of Enable.
Display	To show; to make viewing possible.

D (continued)

DLA	*See* Data base limit register.
DM	*See* Diagnostic monitor.

E

ECC	Error-correction code.
Enable	To cause a function to be used. The opposite of Disable.
Environment	*See* Environment 0 (ENV0), Environment 1 (ENV1), and Environment 2 (ENV2).
Environment 0 (ENV0)	Environment 0 tests the mainframe from the MWS-E through a sequence of maintenance channel functions. Three modes – automatic, manual, and compose – are available to test the basic functions of the mainframe. [*See also* Auto (automatic) mode, Manual mode, and Compose mode.] Automatic mode runs a predefined series of sequences against user-selected areas and CPUs. Manual mode runs user selected sequences against a single user-selected area and CPUs. Compose mode runs a user-defined sequence. In compose mode, a customized sequence may be generated or one of the predefined sequences may be edited. Various displays show information necessary to analyze hardware failures. This environment replaces the level 0 and boot tests used in the previous CRAY Y-MP computer systems. Environment 0 tests specific areas of the mainframe to a degree that ensures environment 1 functions. The areas tested are: the maintenance channel, the diagnostic monitor, the mainframe memory, the CPU exchange mechanism, and the CPU instruction buffers. These areas are tested from the MWS-E through a sequence of maintenance channel functions. Environment 0 runs in offline mode.

E (continued)

Environment 1 (ENV1)	Testing is done from the mainframe. Control points, a generic name used to reference diagnostic programs, utilities, or loops, are loaded into mainframe memory. Memory allocation and control point configuration are controlled by MME through user settings and information available from the maintenance channel. Control points are assigned to CPUs, and MME provides users control over starting and stopping of the CPU execution of the control point. Various displays show information necessary to analyze hardware failures. Environment 1 enables one diagnostic program and/or one or more loops to be loaded into memory. Generally, the diagnostic or utility has access to all of the resources available in the mainframe. All diagnostics are loaded into memory at address 0; loops are loaded at an origin. Generally, because only one control point resides in the mainframe at a given time, it has access to all the resources such as memory, I/O channels, and shared registers. Because the maintenance channel functions are available, such as individual CPU control and direct memory access, monitors are no longer required to perform these functions. In environment 1, all control points (except loops) contain a segment of code referred to as the interrupt router. The interrupt router provides one method of handling interrupts among all the control points. Environment 1 replaces the MM and MI monitors that were used in the previous CRAY Y-MP computer systems. Environment 1 runs in concurrent or offline mode.
Environment 2 (ENV2)	Testing is done from the mainframe. Control points, generic names used to reference diagnostic programs, utilities, or loops, are loaded into mainframe memory. Memory allocation and control point configuration are controlled by MME through user settings and information available from the maintenance channel. Control points are assigned to the CPUs, and MME provides users control over starting and stopping of the CPU execution of the control points. Various displays show information necessary to analyze hardware failures. Environment 2 enables one or more diagnostics or utilities to be loaded into memory. This may be multiple copies of the same diagnostic or utility or different diagnostics or utilities. Several memory allocation schemes are available; they cause the control points to be loaded starting from the lowest memory location (bottom up), starting from the highest memory location (top down), randomly, or equally (partitioned). A small code segment referred to as the controller resides in the lower 040000_8 words of memory. Because there may be more than one control point in memory, the controller negotiates the sharing of resources such as I/O channels and shared registers. Environment 2 controls run system operation. The run system automatically rotates the CPUs among various control points. Environment 2 replaces the M8 and run system monitors used in the previous CRAY Y-MP computer systems. Environment 2 runs in concurrent or offline mode.

E (continued)

Error mode	Enables MME to stop on an error or continue processing when an error occurs while running a diagnostic program such as a test.
Event	An event occurs when conditions are met and recording starts; this causes 64 bits of data to enter the data buffer. Events are recorded until the set of conditions causes recording to stop or until another event occurs.
Exchange mechanism	The technique used in a CRAY C90 series computer system for switching instruction execution from program to program. *See also* Exchange package.
Exchange package	A 16-word block of data is reserved in memory for exchange packages. The exchange package contains the necessary registers and flags associated with a particular program. Each program has its own exchange package.
Exclusive scrolling list	A scrolling list from which users can choose only one item at a time.
Exclusive setting	A control that is used for mutually exclusive settings and is shown by rectangles or boxes. The chosen setting is shown with a bold border around it.

F

Find	A utility used to locate a word, parcel, 9-bit pattern, or check-bit pattern.
Footer	The bottom area of a window. The footer is used by an application for information and error messages.
Functional unit	Hardware within a CRAY C90 series mainframe that performs specialized functions. Functional units perform arithmetic, logical, shift, and other functions. All functions can operate concurrently.

G

Gauge	A read-only control that shows the percent of use or the portion of the region that defines a selected graphic object.
Go	A command button that starts processing tests, control points, and sequences.

H

Halt	A command button that causes command processing to stop.
Header	The band across the top of every window. Each header has a centered title. Base windows have a window menu button on the left; popup windows have a pushpin on the left.
Help	An OPEN LOOK user-interface implementation that provides on-screen help for each element in a window. The application provides help for application functions and elements. The user can press a function key (F1) to display help information.
Highlighting	A visual indication that an object is in a special state. In this manual and on monochrome displays, the image appears in reverse video. On the color displays, the highlighting color is a slightly darker hue of the window color.

I

IBA	Instruction base address register. The IBA register is in the exchange package. The IBA register holds the base address of the user’s instruction range.
Icon	A small pictorial representation of a base window. Displaying objects as icons conserves screen space while keeping the window available for easy access.
ILA	Instruction limit address register. The ILA register is in the exchange package. The ILA register holds the limit address of the user’s instruction field.
Inactive control	A dimmed control that cannot accept input from the mouse or keyboard.
Input area	The place on the screen that accepts keyboard input. Press and release the SELECT mouse button to set the insert point in the input area (click-to-type).
Input/Output	The data or code that is sent to and from mainframe memory.
Input/Output Error-correction Code	The code that identifies parity errors for data.
Insert point	The specific location in the input area where keyboard input is displayed. When users set the insert point, an active caret is displayed.

I (continued)

Instruction buffer	A set of registers in a CRAY C90 series mainframe used for temporary storage of instructions before each issue. Each instruction buffer can hold 128 consecutive instruction parcels.
Instruction message	A prompting message that informs users of the next logical step.
I/O	*See* Input/Output.
I/O ECC	*See* Input/Output Error-correction Code.
Items	Menu controls that start actions. This can also refer to choices found in a scrolling list.

K

Keyboard processor utility	A Cray maintenance system (CMS)-style interface that provides control over one diagnostic test, utility, or loop.

L

Label	The title of a button, item, or setting that describes its function.
Listing	*See* Diagnostic program listing or Control point listing.
LME	*See* Logic monitor environment.
Logic monitor environment	An X Window System based application that provides an interface to the CRAY C90 diagnostic monitors.
Long-term message	Text that is displayed in a window header following the window title and two hyphens (--) or a single en dash (–).

M

Mainframe configuration environment	An X Window System based application that enables you to configure the mainframe maintenance environment and logic monitor environment. This application also configures the soft switches; mainframe, memory, SSD, and CPU parameters; and the channels used by the mainframe maintenance environment and logic monitor environment. This application also enables you to view the status of the system and a CPU, view and edit a table of flawed-chip data, and specify the mode in which the maintenance software should run (concurrent or offline).

M (continued)

Mainframe maintenance environment	An X Window System based application that enables you to run diagnostic tests and utilities to troubleshoot a CRAY C90 series mainframe. This application includes three environments: environment 0, environment 1, and environment 2.
Manual mode	One of three test modes available in the mainframe maintenance environment. As the name implies, you must select the manual mode. In this mode, you can specify which tests are run and how they are run instead of using the defaults that the system selects in Auto mode in Environment 0. *See also* Automatic mode and Compose mode.
Master CPU	The controlling CPU.
MCE	*See* Mainframe configuration environment.
Menu	A rectangle containing a group of controls. Menus are displayed from a menu button with choices appropriate to the menu button (button menu).
Menu button	A multiple-choice control. A menu button always has a menu mark and is used to display a menu. *See also* Button and Window button.
Menu button command	A command accessed through a menu button.
Menu group	A menu and its associated submenus.
Menu item	An item on a menu with a menu mark pointing to the right that is used to display a submenu.
Menu mark	A hollow triangle in the border of a button or following a menu item that has a submenu attached to it. The triangle points toward the location of where the menu or submenu will be displayed.
Message log	The list that identifies errors from a diagnostic program, identifies any errors that occurred during system processing, and ensures that processing is running properly.
MME	*See* Mainframe maintenance environment.

N

Notice	A window displayed when an application generates warning and error messages that require an action before users can proceed. A notice blocks input to the application until a user clicks on one of its buttons.
Numeric field	Text input field, with increment and decrement buttons, that is used for numeric input.

O

Offline Mode	One of two modes available with the mainframe configuration environment, mainframe maintenance environment, and logic monitor environment. In this mode, the mainframe maintenance environment and logic monitor environment have complete control over the mainframe, and the operating system cannot be running in any CPUs. Stop the operating system in the mainframe before using offline mode.

P

P register	Program address register. The P register selects an instruction parcel from one of the instruction buffers. The contents of the P register are stored in the exchange package. The instruction at this location is the first instruction issued when the program begins.
Pane	A bordered rectangle in a window where the active application displays data.
Parameter set	A collection of data used by the logic monitor environment application. It includes a 12-parcel block of diagnostic monitor data, a 64-word block of expected data, and a word-long mask used to determine which bits from the actual data of a DM run are compared to the corresponding expected data.
Parcel	A 16-bit portion of a word that is addressable for instruction execution but not for operand references.
Pattern utility	A program that locates data by memory address, up or down keys, or true and false statements. It is available in a simple (patt) and a comprehensive (patt+) version.
Popup menu	A menu that users access by pressing the MENU mouse button on any area of the workspace that is not a control. The menu that is displayed depends on what the user chooses with the mouse pointer.
Popup window	A window that pops up to perform specific functions and that is then dismissed. Command windows, property windows, help windows, and notices are all popup windows.
Popup window menu	The window menu that is displayed when users press the MENU mouse button of a popup window.
Previous	The settings or values prior to the current settings or values in a window or a field.
Progress	Information generated by an application that informs users about the status of a process.

P (continued)

Properties	Characteristics of an object that users can set, such as the color of a window.
Property window	A popup window that is used to set properties associated with an object, an application, or a window.
Pushpin	A symbol (📌) that keeps a window displayed on the screen. Click on the pushpin to close the window; the window disappears from the screen.

R

Reload	Loads the control point or test point again in order to start processing again from the beginning of the test sequence rather than from where the control point or test point found the last error.
Reset	A control in a property window that restores the property window controls to their previous state.
Reset driver	Returns the driver or application program to its original or default values that existed when the product was shipped.
Resource allocation	Differs by environment. In environment 1 or 2, you can set memory allocation (in environment 2 only), CPU allocation (in environment 1 or 2), and execution mode allocation (in environment 1 or 2). Memory allocation includes bottom up, top down, random, or partitioned memory (with count and size specified if memory is partitioned). CPU allocation is automatic or manual. In execution mode allocation, you can view or specify the following selections: system programmable-clock interrupt, I/O CPU, memory priority, I/O error-correction code, disabling of interrupts for register parity errors, disabling of interrupts for uncorrectable memory errors, disabling of interrupts for correctable memory errors, and maintenance mode. In environment 0, you can set the device timeout, specify the master CPU, and specify the memory priority.
Runtime information	Data provided while a control point is running.

S

S register	Scalar register. The S registers are the source and destination registers for operands executing scalar arithmetic and logical instructions.

S (continued)

SBCDBD	Single-byte correction/double-byte detection. SBCDBD ensures that data written into central memory is read with consistent precision. If a single bit of a data word is altered, alteration is automatically corrected when the word is read from memory. If 2 bits of the same data word are altered, the error is detected but cannot be corrected.
Scrollbar	A device used to move the view of the data displayed in a pane.
Scrolling	The process of moving through data that cannot be viewed entirely in a pane.
Scrolling button	An abbreviated button with a solid triangle arrowhead inside the border that is used for scrolling.
Scrolling list	A pane containing a list of text fields; the list can be read-only, or it can be edited.
Select	To choose an object or objects on the screen or in the window.
Settings	Rectangular boxes used to choose a particular value. A chosen setting is shown with a bold border around it.
Short-term messages	Text that is displayed in the left portion of a window footer, usually displaying instructions, progress, or error messages.
Shrink	To resize a window so that its area is reduced.
Slider	An object used to set a value and give a visual indication of the setting.
SMON	*See* System monitor utility.
Software switches	Software used to change diagnostic program settings for which there are also hardware switches.
Spot help	A brief message pertaining to the object under the pointer when users press the F1 key.
SSD	SSD solid-state storage device. The SSD is a high-performance device used for temporary data storage. SSD is a federally registered trademark of Cray Research, Inc.
ST register	Shared scalar register. The ST register is a shared register used for passing scalar information from one CPU to another.
Standard locations	Normal memory locations for a diagnostic program or utility.

S (continued)

Step mode	A function that, when enabled, causes a diagnostic program to process one instruction and then stop for manual intervention.
Stop on error	A function that, when enabled, causes a diagnostic program to stop when the first error occurs.
Status message	Information generated by an application that informs users about the progress of a process.
Submenu	A menu that displays additional choices under a menu item on a menu.
System monitor utility	An extension to the logic monitor environment that automatically acquires CPU information during hardware and software failures by using the diagnostic monitor hardware.

T

T register	Intermediate scalar register. The T registers are used as intermediate storage for the S registers.
Test mode	The test mode selected determines the amount of control a field engineer has over the tests that are run in environment 0. *See also* Automatic mode and Compose mode.
Text field	An area in a window into which users enter text from the keyboard.
Title	The name of the application or function that is displayed at the top of a window or a popup menu.
Trigger	A set of conditions that causes recording to stop until another event occurs. *See also* Event.

U

User bugs	Bugs you create to use with the simulator and bugger/debugger. You can use the bugmaker utility to create user bugs.

V

V register	Vector register. Each V register contains sixty-four 64-bit elements.

W

Window	A graphical screen display. Multiple windows can reside on the screen at one time. *See also* Base window, Command window, and Popup window.
Window border	The part of the window, including the header, footer, and sides of the window, that you can use to select a window, set the input area, move the window by dragging, and display the `Window` menu.
Window button	A button that is used to display a window containing additional controls.
Window item	An item on a menu that is used to display a window containing additional controls.
Window mark	The three dots (...) that are displayed following both the button label on the window buttons and a window item on a menu.
Window menu button	The abbreviated menu button that is always displayed at the left of the header in each base window. This button can be used to execute the default setting on the window menu (by pressing and releasing the SELECT mouse button) and to display the window menu (by pressing and releasing the MENU mouse button). *See also* Button and Menu button.
Workspace	The background screen area where windows and icons are displayed.
Workspace menu	The menu that controls global functions. You can use this menu to start the maintenance tools.

X

XA	Exchange address register. The XA register in the exchange package specifies the first word address of a 16-word exchange package loaded by an exchange operation.
X-Large	Selection used to specify a font or the memory size as extra large

BIBLIOGRAPHY

Refer to the following CRI and vendor publications for more information about the CRAY C90 series computer system hardware, OpenWindows software, CRI diagnostic software, and the MWS-E:

Command Buffer Parser User Guide, publication number HDM-076-0.

This document contains information about the command buffer parser (CBP), including but not limited to discussions of the CBP interface, command buffer structure and use, operation and data types, command descriptions, and CBP's interaction with the C preprocessor.

CRAY C90 Series Hardware Maintenance Manual, publication number CMM-0502-0A0.

This manual is written to assist field engineers in maintaining the CRAY C90 series computer systems. It is also designed to assist product specialists in repairing system problems in the field and to assist system test technicians in diagnosing problems and modules at the chip level.

This manual is a three-volume set whose organization is described as follows: Volume 1 describes the CRAY C90 computer system; 10-K gate array macrocells; online documentation; and memory modules MEM1M, HM4M, HDRM, and MEM4M. Volume 2 describes the memory control logic, error reporting, shared registers, maintenance features, and I/O channels. Volume 3 describes the hardware registers and functional units and includes timing diagrams.

CRAY C90 Series LME System Monitor Utility, publication number HDM-120-0.

This document describes the LME System Monitor (SMON) utility. This document describes how to the run SMON and interpret the output that SMON returns.

CRAY C90 Series Mainframe Offline Diagnostic Booklet, publication number CQH-0509-0A0.

This quick-reference booklet contains MCE startup and chip flawing information; MME startup information, diagnostic program and utility layout information, tests, and utilities; LME startup and interface information; and runtime module information, including command buffer programs and command buffer parser commands.

CRAY C90 Series Memory Chip Flawing, publication number HMM-104-A.

This document describes memory chip flawing using the spare chip hardware. The discussion includes but is not limited to required software and hardware, software problems and corrections, the spare map file, using the MCE Apply and Save buttons, and using the MCE Flaw Chip Management window.

CRAY C90 Series MME Reference, publication number HDM-081-0.

This document provides detailed information about MME environments 1 and 2.

CRAY Y-MP C90 System Programmer Reference Manual, publication number CSM-0500-000.

This manual provides a detailed architectural overview of the mainframe from a programmer's perspective. It is written to help programmers and analysts write and optimize program code.

OpenWindows Version 3 End User's Manuals, publication number 851-1390-01.

This set includes the following manuals:

OpenWindows Version 3 User's Guide, publication number 800-6618-10.

This manual describes how to use OpenWindows features for common user tasks.

Desk Set Environment Reference Guide, publication number 800-6231-10.

This manual is a concise reference that describes ways you can customize OpenWindows software on your MWS-E; however, you should be aware that the *CRAY C90 Series Mainframe Offline Diagnostic Manual* describes the default environment.

OpenWindows Version 3 Release Notes, publication number 800-6006-10.

This manual provides an additional theory of operation for OpenWindows software.

OpenWindows Version 3 Installation & Start Up Guide, publication number 800-6029-10.

This manual describes how to install, configure, and start using OpenWindows software.

Remote Support System Guide, publication number CDM-1125-000.

This manual describes the steps required to set up and install phase 1 of the Remote Support 3.0 release. The Remote Support environment, which includes Communication Hubs and Services Centers, is described. Sections describing network security and pager use are also included.

INDEX

The term *environment* is abbreviated in this index as follows:

env. 0 (environment 0)
env. 1 (environment 1)
env. 2 (environment 2)
env. 1/2 (environments 1 and 2)